Proposal for C2x
WG14 N2333

Title:	Querying attribute support
Author, affiliation:	Aaron Ballman, GrammaTech
Date:	2019-01-22
Proposal category:	New features
Target audience:	General developers, compiler/tooling developers

Abstract: Users with cross-platform code bases need the ability to interrogate a given implementation to determine whether an attribute is supported or not. This provides a preprocessor mechanism to perform that interrogation.

Prior art: C++ standardized this feature in C++2a, spelled __has_cpp_attribute. Clang is shipping this feature under the proposed spelling.

Querying attribute support

Reply-to: Aaron Ballman (aaron@aaronballman.com)
Document No: N2333
Date: 2019-01-22

Summary of Changes

N2333

- Original proposal.

Introduction

To keep code portable, especially for library code which may be consumed by implementations unknown or unavailable to the author of the code, users need the ability to interrogate a compiler to determine whether an attribute [N2269] is supported or not. The proposed `__has_c_attribute` function-like macro provides users with a way to query whether an implementation supports a given attribute.

Rationale

The list of supported attributes changes with each revision of the C Standard and implementations are allowed to provide their own attributes. Users writing portable code may wish to guard against quality implementations diagnosing use of unknown or unsupported attributes while still preserving the functionality provided by the attribute where possible. Additionally, some implementations may provide a non-attribute fallback for the desired functionality, possibly allowing a user's code base to degrade more gracefully in the absence of support for an attribute.

Proposal

C++ uses the `__has_cpp_attribute` function-like macro to interrogate an implementation's support for an attribute [P0941R2], and this document proposes adding the `__has_c_attribute` function-like macro for the same purposes. Separate macros are useful to allow independent queries for both C and C++ code, as an attribute may be supported in only one of the two languages for a given implementation.

This function-like macro is intended to be used in conjunction with other user-defined macros exposing the attribute, as in the following example.

```
/* Fallback for compilers not yet implementing this feature. */
#ifndef __has_c_attribute
#define __has_c_attribute(x)  0
#endif /* __has_c_attribute */

#if __has_c_attribute(fallthrough)
/* Attribute is available, use it. */
#define FALLTHROUGH [[fallthrough]]
#else
/* Fallback implementation. */
```

```
#define FALLTHROUGH
#endif
```

This feature-test macro can be used with either a standards-based attribute or with a vendor-supplied attribute. The result of the macro expansion will be 0 if the attribute is unknown or unsupported, and will return nonzero if the attribute is supported for that build configuration. Standards-based attributes will return the latest date of modification to the standard for that attribute (e.g., `201811L` if an attribute was voted into the standard in Nov 2018) in order to allow more fine-grained feature testing capabilities should an attribute evolve over time. For example, imagining a hypothetical situation in which the deprecated attribute was standardized as not taking a string argument until Nov 2018:

```
#if __has_c_attribute(deprecated) >= 201811L
#define DEPRECATED(MSG) [[deprecated(MSG)]]
#elif __has_c_attribute(deprecated)
#define DEPRECATED(MSG) [[deprecated]]
#else
/* Fallback implementation; perhaps use an alternative impl. */
#define DEPRECATED(MSG)
#endif
```

C++ lists attribute feature test macro values alongside a list of other, non-attribute feature testing macros. Given that C does not have a similar list serving the same purpose, this document proposes adding the feature testing macro value for standards-based attributes as a normative paragraph attached to each attribute.

C++ based the behavior of `__has_cpp_attribute` on the `defined` preprocessor token, where it is only available for use within a conditional inclusion expression. This proposal is consistent with the C++ treatment and does not propose adding this as a predefined macro available outside of the preprocessor.

Proposed Wording

The wording proposed is a diff from ISO/IEC 9899-2018. Green text is new text, while ~~red~~ text is deleted text.

Modify 6.10.1p1, splitting it into two paragraphs:

1 The expression that controls conditional inclusion shall be an integer constant expression except that: identifiers (including those lexically identical to keywords) are interpreted as described below.~~;~~167) ~~and it~~

2 The conditional inclusion expression may contain unary operator expressions of the form

`defined` *identifier*

or

`defined (` *identifier* `)`

which evaluate to 1 if the identifier is currently defined as a macro name (that is, if it is predefined or if it has been the subject of a `#define` preprocessing directive without an intervening `#undef` directive with the same subject identifier), 0 if it is not.

Add a third paragraph after the new 2nd paragraph from above:

3 The conditional inclusion expression may contain unary operator expressions of the form

`__has_c_attribute (` *pp-tokens* `)`

which are replaced by a nonzero *pp-number* matching the form of an *integer-constant* if the implementation supports an attribute with the name specified by interpreting the *pp-tokens* as an *attribute-token*, and by 0 otherwise. The *pp-tokens* shall match the form of an *attribute-token*.

Add new paragraphs after 6.10.1p6:

8 EXAMPLE

```
/* Fallback for compilers not yet implementing this feature. */
#ifndef __has_c_attribute
#define __has_c_attribute(x)  0
#endif /* __has_c_attribute */

#if __has_c_attribute(fallthrough)
/* Standard attribute is available, use it. */
#define FALLTHROUGH [[fallthrough]]
#elif __has_c_attribute(vendor::fallthrough)
/* Vendor attribute is available, use it. */
#define FALLTHROUGH [[vendor::fallthrough]]
#else
/* Fallback implementation. */
#define FALLTHROUGH
#endif
```

Add a new paragraph after 6.7.11.1p3, to be applied only if the `deprecated` attribute is adopted. The editors are expected to replace the given value with the appropriate date of adoption for the attribute:

4 The `__has_c_attribute` conditional inclusion expression (6.10.1) shall return the value `201811L` when given `deprecated` as the *pp-tokens* operand.

Add a new paragraph after 6.7.11.2p1, to be applied only if the `fallthrough` attribute is adopted. The editors are expected to replace the given value with the appropriate date of adoption for the attribute:

2 The `__has_c_attribute` conditional inclusion expression (6.10.1) shall return the value `201811L` when given `fallthrough` as the *pp-tokens* operand.

Add a new paragraph after 6.7.11.3p2, to be applied only if the `maybe_unused` attribute is adopted. The editors are expected to replace the given value with the appropriate date of adoption for the attribute:

3 The `__has_c_attribute` conditional inclusion expression (6.10.1) shall return the value `201811L` when given `maybe_unused` as the *pp-tokens* operand.

Add a new paragraph after 6.7.11.4p1, to be applied only if the `nodiscard` attribute is adopted. The editors are expected to replace the given value with the appropriate date of adoption for the attribute:

2 The `__has_c_attribute` conditional inclusion expression (6.10.1) shall return the value `201811L` when given `nodiscard` as the *pp-tokens* operand.

References

[P0941R2]
Integrating feature-test macros into the C++WD (rev. 2). Ville Voutilainen, Jonathan Wakely. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2018/p0941r2.html

[N2269]
Attributes in C. Aaron Ballman. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2269.pdf