Proposal for C2x
WG14 N2334

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

Abstract: Sometimes an API no longer meets the developer's needs and should be retired from use. Lacking such machinery, the producer of the code must rely on unreliable sources such as documentation to convey the information to the consumer of the API. This paper proposes a new attribute, `deprecated`, to convey this information.

Prior art: C++ has this feature using the same syntax. Various vendors have vendor-specific extensions such as `__attribute__((deprecated))` and `__declspec(deprecated)` that provide a similar mechanism.

The deprecated attribute

Reply-to: Aaron Ballman (aaron@aaronballman.com; aballman@grammatech.com)
Document No: N2334
Revises Document No: N2266
Date: 2019-01-22

Summary of Changes

N2334

- Updated examples to demonstrate passing a string literal argument.
- Changed footnote to make it more clear that there are many reasons one might wish to deprecate an API that are not related to safety or obsolescence.

N2266

- Added a new section to the paper on Implementation Divergence.
- Rearranged and renumbered paragraphs; the [Note] is now a Recommended Practice section.
- Changed 6.7.11.1p5 to clarify the desire to not diagnose uses of a deprecated entity within a context which is itself deprecated.
- Update the informative examples to demonstrate hoped-for behavior from implementations.

N2214

- Added the appropriate cover page.
- Rebased on top of N2165, the latest attributes syntax proposal
- Replaced “non-static data member” with “struct or union member”

N2050

- Original proposal.

Introduction

Sometimes an API no longer meets the developer's needs and should be retired from use. However, there is no standard mechanism by which an API designer can communicate this to the consumer of their API. Lacking such machinery, the producer of the code must rely on unreliable sources such as documentation to convey the information to the consumer of the API. One such example is the C Standard Library function `gets()`, which was deprecated in C99 and eventually removed in C11. However, this need is not limited to just the Standard Library, it is applicable to the producer of any library code, including in-house libraries.

Rationale

The `[[deprecated]]` attribute has considerable real-world use, being implemented by Clang and GCC as vendor-specific extensions under the name `__attribute__((deprecated))`, by Microsoft Visual Studio under the name `__declspec(deprecated)`, and was standardized under the name `[[deprecated]]` by WG21.

Unlike other proposed attributes like `[[nodiscard]]` and `[[maybe_unused]]`, what constitutes a "use" can be more concretely determined to be any naming of a deprecated entity other than in the entity's declaration (or redeclaration). For instance, this encourages an implementation to diagnose a deprecated variable that is named in a `sizeof` expression despite the operand being unevaluated. However, because the semantics of the attribute are informative rather than normative, the notion of what constitutes a "use" is still a matter of QoI.

Proposal

This document proposes the `[[deprecated]]` attribute as a way for a programmer to specify that an entity is discouraged from being used. This gives the developer a mechanism by which they can alert the consumer of their code to pending breaking changes, and can optionally provide the consumer with additional information such as alternatives superseding the deprecated functionality.

The `[[deprecated]]` attribute can be applied to the declaration of `struct`, `union`, `enum`, *typedef-name*, variable, non-static data member, function, or enumerator. It can optionally accept a string literal argument that an implementation is encouraged to display when the deprecated entity is used. For instance, if such an attribute were present in C99, the `gets()` function could have been declared:

```
[[deprecated("Consider using fgets() instead")]] char *gets(char *);
// Alternatively:
[[deprecated]] char *gets(char *);
```

Implementation Divergence

While the deprecated attribute exists in multiple different implementations, there is some implementation divergence in how certain constructs are diagnosed [N2236]. This section outlines the various behaviors discovered. The code examples use a macro, `DEP`, to signify deprecated entities. The implementations tested were Clang 6.0.0, GCC 8.1, ICC 18.0.0, and MSVC 2017. The comments indicate which implementations do and do not diagnose a construct with a deprecation diagnostic.

Example 1

```
struct DEP S {
  int a;
};

void g(struct S s); // Clang, GCC, ICC [not MSVC]
DEP void h(struct S s); // GCC [not Clang, ICC, or MSVC]
```

Example 2

```
struct DEP S {
  int a;
};

struct T {
  struct S s; // Clang, GCC, ICC [not MSVC]
};

struct DEP U {
```

```
  struct S s; // GCC [not Clang, ICC, or MSVC]
};
```

Example 3

```
struct DEP S {
  int a;
};

DEP void f(struct S s) { // GCC [not Clang, ICC, or MSVC]
  s.a = 12; // MSVC [not GCC, Clang, or ICC]
}
```

Example 4

```
DEP void f(void);

void g(void) {
  f(); // MSVC, Clang, GCC, and ICC
}

DEP void h(void) {
  f(); // MSVC and GCC [not Clang or ICC]
}
```

Example 5

```
enum DEP E1 {
  one
};

enum E2 {
  two DEP // [Not supported by MSVC or ICC]
};

void f(void) {
  int i = one; // Clang and ICC [not GCC or MSVC]
  int j = two; // Clang and GCC [not ICC or MSVC]
}
```

Example 6

```
DEP typedef int Foo;

Foo f1; // Clang, GCC, ICC, and MSVC

void g(Foo f2); // Clang, GCC, ICC, and MSVC
DEP void h(Foo f3); // GCC and MSVC [not Clang or ICC]

struct S {
  Foo f; // Clang, GCC, ICC, and MSVC
};

struct DEP T {
  Foo f; // GCC and MSVC [not Clang or ICC]
```

```
};
```

These examples demonstrate a wide degree of implementation divergence with the existing implementation-defined deprecated attribute. This paper does not propose putting normative requirements on implementations, but it does add informative examples recommending an approach where there appears to be consensus among implementations.

Proposed Wording

This proposed wording currently uses placeholder terms of art and it references a new subclause from WG14 N2269, 6.7.11, Attributes that describes the referenced grammar terms. The [Note] in paragraph 4 of the semantics is intended to convey informative guidance rather than normative requirements.

6.7.11.1 Deprecated attribute

Syntax

1 *deprecated-attr:*
deprecated deprecated-argument$_{opt}$

deprecated-argument:
(string-literal)

Constraints

2 The *attribute-token* `deprecated` can be used to mark names and entities whose use is still allowed, but is discouraged for some reason. [Footnote: in particular, `deprecated` is appropriate for names and entities that are obsolescent, insecure, unsafe, or otherwise unfit for purpose.] It shall appear at most once in each *attribute-list*.

3 The attribute shall be applied to the declaration of a struct, a union, a *typedef-name*, a variable, a struct or union member, a function, an enumeration, or an enumerator.

Semantics

4 A name or entity declared without the `deprecated` attribute can later be redeclared with the attribute and vice versa. An entity is considered marked after the first declaration that marks it.

Recommended Practice

5 Implementations should use the `deprecated` attribute to produce a diagnostic message in case the program refers to a name or entity other than to declare it, after a declaration that specifies the attribute, when the reference to the name or entity is not within the context of a related deprecated entity. The diagnostic message may include text provided within the *deprecated-argument* of any `deprecated` attribute applied to the name or entity.

6 EXAMPLE

```
struct [[deprecated]] S {
  int a;
};
```

```
enum [[deprecated]] E1 {
  one
};

enum E2 {
  two [[deprecated("use 'three' instead")]],
  three
};

[[deprecated]] typedef int Foo;

void f1(struct S s) { // Diagnose use of S
  int i = one; // Diagnose use of E1
  int j = two; // Diagnose use of two: "use 'three' instead"
  int k = three;
  Foo f; // Diagnose use of Foo
}

[[deprecated]] void f2(struct S s) {
  int i = one;
  int j = two;
  int k = three;
  Foo f;
}

struct [[deprecated]] T {
  Foo f;
  struct S s;
};
```

Implementations are encouraged to diagnose the use of deprecated entities within a context which is not itself deprecated.

Acknowledgements

Thank you to Martin Sebor for reviewing the paper and helping to unify existing practice. Additionally, I would like to recognize the following people for their help in this work: David Keaton and David Svoboda.

References

[N2236]
Using Attribute deprecated. Martin Sebor. http://www.open-std.org/jtc1/sc22/wg14/www/docs/n2236.htm

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

[N3394]
[[deprecated]] attribute. Alberto Ganesh Barbati. http://www.open-std.org/jtc1/sc22/wg21/docs/papers/2012/n3394.html