Any attribute may be qualified with an attribute namespace,[2] i.e., a single arbitrary identifier:

```
[[gnu::const]]  // (GCC-specific) namespace-gnu-qualified const attribute
[[my::own]]     // (user-specified) namespace-my-qualified own attribute
```

C++ attribute placement

Attributes can be placed in a variety of locations within the C++ grammar. For each such location, the Standard defines the entity or statement to which the attribute is said to *pertain*. For example, an attribute in front of a simple declaration statement pertains to each of the entities declared by the statement, whereas an attribute placed immediately after the declared name pertains only to that entity:

```
[[foo]] void f(), g();   // foo pertains to both f() and g().
void u(), v [[foo]] ();  // foo pertains only to v().
```

Attributes can apply to an entity without a name (e.g., anonymous **union** or **enum**):

```
struct S { union [[attribute_name]] { int a; float b; }; };
enum [[attribute_name]] { SUCCESS, FAIL } result;
```

~~The valid positions for any particular attribute are constrained to only those locations where the attribute pertains to the entity to which it applies.~~ That is, an attribute such as `noreturn`, which applies only to functions, would be valid syntactically but not semantically if it were used to annotate any other kind of entity or syntactic element. Misplacement of a standard attribute results in an ill-formed program[3]:

```
void [[noreturn]] x() {}      // Error, cannot be applied to a type
     [[noreturn]] int i;      // Error, cannot be applied to a variable
     [[noreturn]] { throw; }  // Error, cannot be applied to a statement
```

The empty attribute specifier sequence `[[]]` is allowed to appear anywhere the C++ grammar allows attributes.

Common compiler-dependent attributes

Prior to C++11, no standardized syntax for attributes was available and nonportable compiler intrinsics — such as `__attribute__((fallthrough))`, which is GCC-specific syntax — had to be used instead. Given the new standard syntax, vendors are now able to express

[2]Attributes having a namespace-qualified name — e.g., `[[gnu::const]]` — were only **conditionally supported** in C++11 and C++14 but were historically supported by all major compilers, including both Clang and GCC; all C++17-conforming compilers *must* support attribute namespaces.

[3]As of this writing, GCC is lax and merely warns when it sees the standard `noreturn` attribute in an unauthorized syntactic position, whereas Clang correctly fails to compile. Hence, using even a standard attribute might lead to different behavior on different compilers.