Document Number: P0609R2
Date: 2023-11-27
Author: Aaron Ballman <aaron@aaronballman.com>
Audience: Evolution Working Group

Attributes for Structured Bindings

Revision History

R0

- Original proposal

R1

- Updated motivation

R2

- Rebased proposed wording onto latest standard, added more motivation

Motivation

We added the ability to write structured binding declarations in C++17. The optional *attribute-specifier-seq* in such a declaration appertains to the hidden variable declared by the structured binding declaration. Despite the variable being hidden, this is still useful functionality (for instance, it allows the programmer to specify the alignment of the structured binding declaration itself, which may allow for useful compiler optimizations when loading from an array).

However, there is no way to specify attributes that appertain to the individual structured bindings. It is desirable to allow vendor-specific attributes to appertain to these bindings for attributes that would otherwise appertain to variables to enable better diagnostics, especially through static analysis. For instance, some implementations support thread-safety attributes (`guarded_by`, et al) which denote that a variable requires a particular locking primitive to be held before accessing the variable. Other implementations support an annotation which denotes that an object with an array of `char` or pointer to `char` type does not necessarily contain a terminating null character (`nonstring`). Given the prevalence of vendor-specific attributes, it is likely that other motivating use cases currently exist.

I propose to allow optional attributes for each of the introduced structured bindings, as in this example:

```
auto g() {
  auto [a, b [[vendor::attribute]], c] = f();
  return a + c;
}
```

While this may generate an overabundance of square brackets in a declaration, the syntax is consistent with our other treatments of attributes in declarations.

This proposal was seen by EWG at the November 2022 meeting in Kona where it was polled to forward to CWG pending potential updates to existing attributes and to ensure there is not a conflict with the syntax for structured binding packs in P1061.

There should be no conflict with P1061 as the two grammars can be unified in a straightforward manner and the semantics of an attribute on a structured binding pack should fall out naturally.

I investigated the existing standard attributes to see which ones, if any, should be changed to allow them to be applied to a structured binding. `assume`, `carries_dependency`, `fallthrough`, `likely`, `unlikely`, `nodiscard`, `noreturn`, and `no_unique_address` cannot sensibly apply to a structured binding because they do not apply to anything variable-like. It was questionable as to whether an alignment specifier or the `deprecated` attribute would make sense on a structured binding, so those were left for further exploration. The only standard attribute that had clear utility on a structured binding was `maybe_unused`.

Proposed Wording

Modify [dcl.pre]p1:

> ...
> *attributed-identifier:*
> *identifier attribute-specifier-seq*$_{opt}$
>
> *attributed-identifier-list:*
> *attributed-identifier*
> *attributed-identifier-list* , *attributed-identifier*
>
> *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*$_{opt}$ [*attributed-identifier-list*] *initializer* ;
> ...

Modify [dcl.pre]p6:

A *simple-declaration* with an *attributed-identifier-list* is called a *structured binding declaration*. …

Modify [dcl.struct.bind]p1:

A structured binding declaration introduces the *identifier*s v0, v1, v2, ... of the *attributed-identifier-list* as names of *structured bindings*. The optional *attribute-specifier-seq* of an *attributed-identifier* from the *attributed-identifier-list* appertains to the introduced structured binding. …

Modify [dcl.struct.bind]p3:

If E is an array type with element type T, the number of elements in the *attributed-identifier-list* shall be equal to the number of elements of E. ...

Modify [dcl.struct.bind]p4:

Otherwise, if the qualified-id std::tuple_size<E> names a complete class type with a member named value, the expression std::tuple_size<E>::value shall be a well-formed integral constant expression and the number of elements in the *attributed-identifier-list* shall be equal to the value of that expression. ...

Modify [dcl.struct.bind]p5:

Otherwise, all of E's non-static data members shall be direct members of E or of the same base class of E, well-formed when named as e.name in the context of the structured binding, E shall not have an anonymous union member, and the number of elements in the *attributed-identifier-list* shall be equal to the number of non-static data members of E. …

Modify [dcl.attr.unused]p2:

The attribute may be applied to the declaration of a class, a typedef-name, a variable (including a structured binding declaration), a structured binding, a non-static data member, a function, an enumeration, or an enumerator.

Acknowledgements

Thanks to Richard Smith and Jens Maurer for reviewing this paper, and to Corentin Jabot for presenting it on my behalf.