

# Proposal for attributes introspection

Document #:  
Date: 2024-04-20  
Project: Programming Language C++  
Audience: sg7  
Reply-to: Aurelien Cassagnes  
<[aurelien.cassagnes@gmail.com](mailto:aurelien.cassagnes@gmail.com)>

## Contents

|          |                               |          |
|----------|-------------------------------|----------|
| <b>1</b> | <b>Introduction</b>           | <b>1</b> |
| <b>2</b> | <b>Proposal</b>               | <b>2</b> |
| 2.1      | Reflection operator . . . . . | 2        |
| 2.2      | std::meta::info . . . . .     | 2        |
| 2.3      | Metafunctions . . . . .       | 2        |
| 2.4      | Queries . . . . .             | 2        |
| <b>3</b> | <b>Discussion</b>             | <b>2</b> |
| <b>4</b> | <b>References</b>             | <b>3</b> |

## 1 Introduction

Attributes are used to great extent and there likely will be attributes added as the language evolve. What is missing now is a way for generic code to look into the attributes related to an entity. A motivating example is following

```
[[nodiscard]] bool foo(int i) { return i % 2 ; }
```

```
template <class F, class... Args>  
constexpr std::invoke_result_t<F, Args...> logInvoke(F&& f, Args&&... args) {  
    // Do some extra work, log the call...  
    // Fwd call  
    return std::invoke(std::forward<F>(f), std::forward<Args>(args)...);  
}
```

```
int main() {  
    foo(0); // Warning on discarded return  
    logInvoke(foo, 0); // No warning on discarded return  
}
```

Other examples of wrapping around callables can be found, whether by closure or explicitly registering callbacks for dispatch, etc. Other applications can easily be thought of in the context of code injection [Metaprogramming] where one may want to skip over [[deprecated]] members

## 2 Proposal

We put ourselves in the context of [\[Reflection\]](#) for the current proposal to be more illustrative in terms of what is being proposed.

### 2.1 Reflection operator

If our understanding is correct, the proposition for `^` grammar does not cover attributes, as in `^[[deprecated]]` is not correct. We think this will limit the potential use of attributes introspection. The current proposal advocates for

```
^ attribute
```

to be well formed.

### 2.2 std::meta::info

We propose that attributes be a supported *reflectable* property of the expression that are reflected upon. That means `std::meta::info` should be augmented to represent an attribute in addition to what it can represent.

### 2.3 Metafunctions

We propose to add a function to what is discussed already in [\[Reflection\]](#)

```
template<typename E>
constexpr auto attributes_of(E entity) -> vector<info>;
```

This being applied to an entity E will yield a sequence of `std::meta::info` representing the attributes attached to E.

### 2.4 Queries

We do not think it is the best option to introduce a query per attribute such as

```
// ...
constexpr bool is_deprecated(info type);
constexpr bool is_nodiscard(info type);
// ...
```

We think query to filter in particular attribute scales better

```
auto fooAttributes = attributes_of([:foo(int):]);
if constexpr (is_attributed(fooAttributes, ^[[nodiscard]])) {

}
```

## 3 Discussion

Originally the idea of introducing a `declattr(Expression)` keyword seemed the most straightforward to tackle on this problem, but from feedback the concern of introspecting on expression attributes was a concern that belongs with the reflection SG. The current proposal shifted away from the original `declattr` idea to align better with the reflection toolbox.

Another item of discussion is whether vendor specific attributes should be supported or only the ones described by the standard.

## 4 References

[Metaprogramming] Andrew Sutton. Metaprogramming.

<https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2020/p2237r0.pdf>

[Reflection] Wyatt Childers, Peter Dimov, Dan Katz, Barry Revzin, Andrew Sutton, Faisal Vali, and Daveed Vandevoorde. Reflection for c++26.

<https://www.open-std.org/jtc1/sc22/wg21/docs/papers/2024/p2996r2.html>