[[=std::interop]] An Annotation for Interoperation with Targets

Document #:	D0000R0 [Latest] [Status]
Date:	unassigned
Project:	Programming Language C++
Audience:	N/A

Reply-to:	Adrian Johnston <ajohnston54637@gmail.com>

1 Abstract

This paper proposes `[[=std::interop]]`, a standard annotation marking a declaration for exposure to a foreign execution environment. The annotation has no effect on the program, and the meaning of target names is left unspecified. A reference code generator is provided for a number of languages.

2 Motivation

To propose a standard annotation, `[[=std::interop]]`, marking a declaration for exposure to a foreign execution environment, with an optional target list. It has no effect on the program and no target list semantics are to be specified. As context, [P3394R4] supplies the mechanism but no vocabulary, so each library will define its own `exposed_t`. By adding `[[=std::interop]]` different libraries and tools should not need to define their own `exposed_t`. This attribute is proposed with the intention of it being used as the basis for non-proliferation treaties around interoperability annotation. Although, that is hard to control.

3 Proposal

This proposal adds a class `interop_type` and a constant `interop` of that type to a new header, `<interop>`. The `interop` constant and the result of calling it with `operator()` are required to be valid annotation operands under [P3394R4]. The call operator accepts any number of arguments convertible to `string_view` and returns a new `interop_type` holding the targets in static storage and in call order. `targets()` returns that list unmodified, duplicates included, and empty for `interop` itself. Embedded nulls in a string literal argument are preserved. Whether equal strings in the list share a pointer is unspecified.

The proposed interface is as follows, however the implementation is for exposition only.

```
namespace std {
  struct interop_type {
    template<convertible_to<string_view>... S>
    consteval interop_type operator()(const S&... s) const {
      vector<const char*> list(list_, list_ + size_);
      (list.push_back(define_static_string(whole_(s))), ...);
      auto kept = define_static_array(list);
      return {kept.data(), kept.size()};
    }
    constexpr span<const char* const> targets(void) const {
      return {list_, size_};
    }
    template<class S>
    static consteval string_view whole_(const S& s) { // exposition only
      if constexpr (is_array_v<S>) return {s, extent_v<S> - 1};
      else return s;
    }
    const char* const* list_;                        // exposition only
    size_t size_;                                    // exposition only
```

```
  };
  inline constexpr interop_type interop{};
}
```

The annotation may appear wherever [P3394R4] allows one. `[[=std::interop]]` marks a declaration for exposure without naming a target, and `[[=std::interop("ECMAScript", "Python")]]` names a target list. A target list can also be built up using `std::interop` and reused as is the `[[=script]]` attribute below. It is unspecified if annotating a class or namespace indicates anything about its members.

What follows is an example involving a thermostat.

```
inline constexpr auto script = std::interop("ECMAScript", "Python");

class [[=std::interop]] thermostat_type {
public:
  [[=std::interop]] ~thermostat_type(void) = default;

  [[=std::interop]] double current_temp(void) const;
  [[=std::interop]] double temp_setting(void) const;

  [[=script]] bool furnace_on(void) const;
  [[=script]] bool fan_on(void) const;

  [[=std::interop("Rust")]] void set_temp_setting(double t_);

private:
  // ...
};

[[=std::interop]] thermostat_type thermostat_factory(double current,
                                                     double setting);
```

No change to the existing standard is proposed beyond the inclusion of `<interop>`, `interop_type` and `interop`.

4 Reference Implementation

In order to validate the design, the above code was used to generate a C header and to generate bindings for C, ECMAScript, Python and Rust.

Here is the website for the outputs. It provides an index of the files generated by the repository.

The code generator is `interop.cpp` and it emits 4 bindings and a C header in under 400 LOC. Everything passes tests and is formatted correctly.

5 Design Decisions

— Providing a naming system for foreign languages in the C++ standard is recognized as unmaintainable. Other standards are welcome to make recommendations for their own names.
— The reference implementation of `interop_type` uses `define_static_string` and `define_static_array` from [P3491R3].

6 Future Work

Any modifications to the standard will be described in detail in later revisions of this proposal.

[P3394R4] permits annotations on namespaces, but the meaning of `std::interop` on a namespace for its members is left open. Discussion is needed with independent interoperability project maintainers once this proposal is concrete.

7 References

[P3394R4] Daveed Vandevoorde, Wyatt Childers, Dan Katz, Barry Revzin. 2025-06-20. Annotations for Reflection.
https://wg21.link/p3394r4
[P3491R3] Barry Revzin, Wyatt Childers, Peter Dimov, Daveed Vandevoorde. 2025-06-20. define_static_{string,object,array}.
https://wg21.link/p3491r3