JUNE 10TH, 2026

P4261R0

PRESENTATION ON P4189R0:

GET()ING THE POINTER FROM STD::OPTIONAL

NEVIN ":-)" LIBER
nliber@anl.gov

U.S. DEPARTMENT of ENERGY
Argonne National Laboratory is a U.S. Department of Energy laboratory managed by UChicago Argonne, LLC.

Argonne NATIONAL LABORATORY
Argonne Leadership Computing Facility

GET()ING THE POINTER FROM STD::OPTIONAL

- During the discussions around

```
T* optional<T&> inplace_vector<T, N>::try_*_back(/* ... */)
```

  - It was noted that there is no easy way to convert an `optional<T&>` to a `T*`

GET()ING THE POINTER FROM STD::OPTIONAL

- With some help from Claude Sonnet 4.6:

```
optional<T&>::transform([](auto& r) {
    return std::addressof(r); }).value_or(nullptr);
```

GET()ING THE POINTER FROM STD::OPTIONAL

- With help from Steve Downey:

```
constexpr auto get_ptr = [](auto&& o) {
    return (!o.has_value()) ? nullptr : o.operator->();
};
```

GET()ING THE POINTER FROM STD::OPTIONAL

Motivation

- Why do we want this?
  - To interface with C and older C++, as they have no suitable vocabulary type other than T*
    - `optional<T&>` —> T*
      - pre-C++26 APIs
      - C APIs
    - `optional<T>` —> T*
      - pre-C++17 APIs
      - C APIs

GET()ING THE POINTER FROM STD::OPTIONAL

Proposal

Boost.Optional

```
constexpr T* optional<T&>::get_ptr() const noexcept;

    // T* to object referred to in the optional; disengaged: null
    return val;


constexpr       T* optional<T>::get_ptr()       noexcept;
constexpr const T* optional<T>::get_ptr() const noexcept;

    // T* to object stored in the optional; disengaged: null
    return has_value() ? std::addressof(val) : nullptr;
```

GET()ING THE POINTER FROM STD::OPTIONAL

Why get_ptr instead of a different name/method?

`std::to_address(/* ... */)`

- Good
  - Just works when `optional<T&>` or `optional<T>` is engaged
    - Because `std::to_address(o)` falls back to using `optional::operator->()`
- *(Note: this discussion on `std::to_address` has new, more important lines of reasoning compared with the discussion on `std::to_address` in P4189R0; will be changed in P4189R1)*

GET()ING THE POINTER FROM STD::OPTIONAL

Why get_ptr instead of a different name/method?

`std::to_address(/* ... */)`

- Bad
  - *UB when disengaged* (violates `has_value()` precondition on `operator->()`)
  - Possible fixes *(for all of these, pre-C++29 code would still be buggy)*
    - Provide `pointer_traits<optional<T>>::to_address(o)`
    - Remove precondition on `operator->()` by returning null value when disengaged
      - Required pessimization in non-*hardened* mode
      - Observable break to *hardened* precondition
    - Overload `std::to_address(o)` for `optional`

GET()ING THE POINTER FROM STD::OPTIONAL

Why get_ptr instead of a different name/method?

`.get()`

- It is obvious from the name of the type (`*_ptr`, `*_array`, etc.) that the type holds a pointer
- Doesn't generalize to `optional<T>`

`.data()`

- For an empty `span` or `basic_string_view` (the moral equivalent of a detached `optional`) or even `array<T,0>`, there is no guarantee that the pointer returned has a null pointer value
  - Typically used in conjunction with `.size()`
  - `.data()` guaranteed to return null pointer value, but only for some types, is strictly worse than never guaranteeing it

GET()ING THE POINTER FROM STD::OPTIONAL

Why get_ptr instead of a different name/method?

- More bike-shedding of even more obscure names and their drawbacks in P4189

GET()ING THE POINTER FROM STD::OPTIONAL

Proposal

Boost.Optional

```
constexpr T* optional<T&>::get_ptr() const noexcept;

    // T* to object referred to in the optional; disengaged: null
    return val;


constexpr       T* optional<T>::get_ptr()       noexcept;
constexpr const T* optional<T>::get_ptr() const noexcept;

    // T* to object stored in the optional; disengaged: null
    return has_value() ? std::addressof(val) : nullptr;
```

GET()ING THE POINTER FROM STD::OPTIONAL

Acknowledgements

- Thanks to the discussion on national body comments for C++26: PL-006, US 150-228, GB 08-225, US 68-122 and US 149-226 for showing this obvious deficiency in `optional`

- Nevin Liber was supported by the Office of Science, U.S. Department of Energy, under contract DE-AC02-06CH11357


Argonne
NATIONAL LABORATORY
Argonne Leadership
Computing Facility