How can I box a std::optional into a C++/WinRT IInspectable?

devblogs.microsoft.com/oldnewthing/20230330-00

March 30, 2023

Raymond Chen

Say you have a `std::optional<T>` in your hand, and you want to convert it to an `IInspectable`, say because you want to put it inside a `PropertySet`.

```
void SaveHeight(winrt::PropertySet const& set)
{
    std::optional<int> height = GetHeight();

    set.Insert(L"height", /* what goes here? */);
}
```

Well, one thing you *shouldn't* do is go right for the `value`:

```
// Code in italics is wrong.
void SaveHeight(winrt::PropertySet const& set)
{
    std::optional<int> height = GetHeight();

    set.Insert(L"height",
        winrt::box_value(height.value()));
}
```

Calling `value()` on an empty `std::optional` throws the `std::bad_optional_access` exception. In that case, your `SaveHeight` method throws an exception instead of saving a `nullptr` into the property set. If this exception crosses an ABI boundary, C++/WinRT and C++/CX will convert it to `E_FAIL`, and WIL will convert it to `ERROR_UNHANDLED_EXCEPTION`. But really, it doesn't matter how the C++ exception is converted to an ABI `HRESULT` because you didn't want an exception in the first place. You wanted the `std::nullopt` to convert to a null pointer.

You could manually check for an empty `std::optional`:

```
void SaveHeight(winrt::PropertySet const& set)
{
    std::optional<int> height = GetHeight();

    set.Insert(L"height",
        height ? winrt::box_value(*height) : nullptr);
}
```

But there's an even easier way.

C++/WinRT provides a conversion operator from `std::optional<T>` to `IReference<T>` which does the obvious thing: An empty `std::optional` becomes `nullptr` and a `std::optional` with a value becomes an `IReference` that holds the value.

```
void SaveHeight(winrt::PropertySet const& set)
{
    std::optional<int> height = GetHeight();

    set.Insert(L"height",
        winrt::IReference(height));
}
```

We are taking advantage here of class template argument deduction (CTAD), which lets us write just `winrt::IReference(...)` and let the compiler infer that we are constructing a `winrt::IReference<int>`.