Producing an empty Windows Runtime type in C++/WinRT

devblogs.microsoft.com/oldnewthing/20220504-00

May 4, 2022

Raymond Chen

Windows Runtime types fall into two general categories: Reference types and value types.

Reference types are copied by reference. When you copy a reference type, you're really just incrementing the reference count on the underlying object. And Windows Runtime references can be null.

Value types are copied by value. When you copy a value type, you are making a shallow copy of everything inside it. Windows Runtime values cannot be null.

There are cases where you want to produce an empty Windows Runtime type. For reference types, this would be a null reference, and for value types, it would be a default-initialized value filled with zero, `false`, and null, as appropriate.

In other words, we want to replicate what C# `default(T)` does.

Using `{}` to generate an empty value works for delegates, interfaces, and value types, but not for runtime classes, because C++/WinRT expresses the default constructor of the runtime class as the default constructor of the projection. In other words, default-constructing a Windows Runtime class in C++/WinRT actually *creates an object*, so you don't get `nullptr` at all. You get a real live object.

We need to detect the case where we have a Windows Runtime runtime class and use the `nullptr` constructor in that case.

The shortcut here is that the `nullptr` constructor also works for delegates and interfaces, we can just use it for all reference types.

And that leads us to this:

```
template<typename T>
constexpr T winrt_empty_value() noexcept
{
    if constexpr (std::is_base_of_v<winrt::Windows::Foundation::IUnknown, T>) {
        return nullptr;
    } else {
        return {};
    }
}
```

Making the whole thing `constexpr` allows it to be used a lot of places that would normally be constrained to constants.

This function will come in handy later, but we'll have to get there first.

Raymond Chen

Follow