Usage patterns for winrt::unbox_value_or

devblogs.microsoft.com/oldnewthing/20210204-00

February 4, 2021

Raymond Chen

C++/WinRT's `unbox_value_or` template function takes an `IInspectable` (which represents an arbitrary object) and tries to unbox it to a specified target type. If unable, it returns a provided default value.

The most explicit way of using it is to specialize the type, and pass a default value of that same type.

```
auto value = unbox_value_or<Thing>(obj, Thing{ params });
```

This is basically the belt-and-suspenders method. You make sure the output type is a `Thing` by passing it as an explicit template type parameter, and you provide the default value as a `Thing`. There's no possible way anybody could misunderstand you.

But you can go with just the belt or just the suspenders.

If you choose the belt, you can pass the type explicitly and let the compiler infer the default value, assuming that the constructor is not explicit.

```
auto value = unbox_value_or<Thing>(obj, { params });
```

If there is a single parameter, you can omit the braces, again assuming the constructor is not explicit.

```
auto value = unbox_value_or<Thing>(obj, param);
```

Or you can go with the suspenders and pass the default value explicitly and let the compiler infer the type.

```
auto value = unbox_value_or(obj, Thing{ params });
```

In the special case that the thing you are trying to unbox is a string, the second parameter goes through `param::hstring` which will use string references where possible, thereby avoiding allocating memory for an actual string unless it turns out to be necessary. (In the other cases, you pass a constructed default object, even if it turns out never to be used.)

Of course, if you omit both the explicit template type parameter and an explicit type for the default value, then your pants fall down.

```
// Compiler can't deduce what you're trying to do.
auto value = unbox_value_or(obj, { params });
```

Raymond Chen

Follow