Using the classical model for linking to provide unit test overrides

devblogs.microsoft.com/oldnewthing/20250416-00/?p=111077

April 16, 2025

Some time ago, I discussed the classical model of linking and the fact that you can override a LIB with another LIB, and a LIB with an OBJ, but you can't override an OBJ. I noted in passing that "This lets you override a symbol in a library by explicitly placing it an OBJ."

An example where you can use this trick is in writing unit test hooks, as a form of global dependency injection.

```
// library/unittesthooks.h

extern std::optional<int>
    UnitTestHook_Widget_GetValue(Widget* widget);

// library/widget.cpp

int Widget::GetValue()
{
    auto hook = UnitTestHook_Widget_GetValue(this);
    if (hook) {
        return *hook;
    }

    〖 production code 〗
}

// library/unittesthookfallbacks.cpp

std::optional<int>
    UnitTestHook_Widget_GetValue(
        Widget* [[maybe_unused]] widget)
{
    return std::nullopt;
}

// unittests/unittest.cpp

Widget* g_widgetToOverride;
int g_overrideValue;

std::optional<int>
    UnitTestHook_Widget_GetValue(
        Widget* [[maybe_unused]] widget)
{
    if (widget == g_widgetToOverride) {
        return g_overrideValue;
    }
    return std::nullopt;
}

void TestWidgetValue()
{
    Widget widget;

    // Force GetValue to return 42
    g_widgetToOverride = std::addressof(widget);
    g_overrideValue = 42;

    // clean up the override when the test exits
    auto cleanup = wil::scope_exit([] {
        g_widgetToOverride = nullptr;
    });

    // This should do the thing because we
    // made the value report as 42
```

```
    DoTheThingIfValueIs42(widget);
}
```

The idea here is that in production, the `UnitTestHook_Widget_GetValue` function is provided by `unittesthookfallbacks.cpp`, and that version always says "Don't mind me, just go ahead and do your normal production code." If you enable link-time code generation, the entire call will be optimized out, and the only code that will be generated is your production code.

It is important that the fallback be in a separate .cpp file (and therefore compile to a separate .obj file) so that it doesn't get taken along for the ride. It is also important that the fallback be packaged in a library and not as a loose .obj, so that the unit test can provide its own implementation of the function.

In the unit test, the unit test provides its own version of the `UnitTestHook_Widget_GetValue` function, and it checks whether the widget is the one being overridden, and if so it returns the overridden value. Otherwise, it allows the normal production code to run.

I like this form of global dependency injection because it means that the production code is completely devirtualized. You aren't paying for virtual method calls on your injected interface, and once link time code generation kicks in, all of the unit testing hooks disappear, so your production code operates entire unencumbered.

(It also means that you can modify the code's behavior in a unit test without having to do detouring. Detouring comes with its own problems, such as the inability to detour functions that have been inlined.)

Bonus chatter: To help out the link time code generator, the override function should return an object with no destructor, so that the compiler doesn't have to construct an object (to represent the return value of the override function), then immediately destruct that value. Maybe the compiler can optimize out the constructor and destructor at sufficiently high optimization levels, but I like to avoid the problem entirely.