Using type aliasing to avoid the ODR problem with conditional compilation, part 2

devblogs.microsoft.com/oldnewthing/20250502-00/?p=111140

May 2, 2025

Last time, we tried using type aliasing to avoid the ODR problem with conditional compilation, but we ran into trouble with cases where a change in type does not participate in overload resolution.

But we can still salvage the situation.

We can follow the same basic pattern, but instead of allowing the debug and non-debug versions to coexist, we can simply prohibit them from coexisting.

First, let’s isolate the two versions into separate translation units.

```
// widgetimpl.h
#include "widget.h"

template<bool debug>
void Widget<debug>::Widget()
{
    ⟦ constructor stuff ⟧
}

template<bool debug>
void Widget<debug>::SetName(std::string const& name)
{
    m_name = name;
    Log("Name changed");
    Log(name);
}

// widgetdebug.cpp
#include "widgetimpl.h"

template struct WidgetT<true>;

// widgetnondebug.cpp
#include "widgetimpl.h"

template struct WidgetT<false>;
```

Now we can use the One Definition Rule to our advantage: Declare the same variable in each translation unit.

```
// widgetdebug.cpp
#include "widgetimpl.h"

template struct WidgetT<true>;

namespace Widgets::internal
{
    extern const bool is_debug = true;
}

// widgetnondebug.cpp
#include "widgetimpl.h"

template struct WidgetT<false>;

namespace Widgets::internal
{
    extern const bool is_debug = false;
}
```

If somebody uses a debugging-mode Widget, then that pulls in the explicit instantiation from `widgetdebug.obj`, and that in turn pulls in the debugging definition of `Widgets::internal::is_debug`. And if somebody else uses a non-debugging Widget, then that will pull in the explicit instantiation from `widgetnondebug.obj`, which in turn pulls in the non-debugging definition of `Widgets::internal::is_debug`.

Two definitions for the same thing is a diagnosable One Definition Rule violation, and the linker will complain.

Phew, we did it.

Now, if you are using the Microsoft Visual C++ compiler, you could have used `#pragma detect_mismatch` from the outset and avoided this hassle.

```
// widget.h

struct Widget
{
    Widget();

    void SetName(std::string const& name);

    ⟦ more stuff ⟧

#ifdef EXTRA_WIDGET_DEBUGGING
    Logger m_logger;

    void Log(std::string const& message) {
        m_logger.log(message);
    }
#else
    void Log(std::string const& message) {
        // no extra logging
    }
#endif

    std::string m_name;
};

#ifdef EXTRA_WIDGET_DEBUGGING
#pragma detect_mismatch("widget build mode", "debug")
#else
#pragma detect_mismatch("widget build mode", "nondebug")
#endif
```

If two clients set `EXTRA_WIDGET_DEBUGGING` differently, the linker will complain about the mismatch.