On trying to log an exception as it leaves your scope

devblogs.microsoft.com/oldnewthing/20250203-00

February 3, 2025

A customer wanted to log exceptions that emerged from a function, so they used the WIL `scope_exit` object to specify a block of code to run during exception unwinding.

```
void DoSomething()
{
    auto logException = wil::scope_exit([&] {
        Log("DoSomething failed",
            wil::ResultFromCaughtException());
    });

    ⟦ do stuff that might throw exceptions ⟧

    // made it to the end - cancel the logging
    logException.release();
}
```

They found, however, that instead of logging the exception, the code in the `scope_exit` was crashing.

They debugged into the `ResultFromCaughtException` function, which eventually reaches something like this:

```
try
{
    throw;
}
catch (⟦ blah blah ⟧)
{
    ⟦ blah blah ⟧
}
catch (⟦ blah blah ⟧)
{
    ⟦ blah blah ⟧
}
catch (...)
{
    ⟦ blah blah ⟧
}
```

The idea is that the code rethrows the exception, then tries to catch it in various ways, and when it is successful, it uses the caught object to calculate a result code.

And that's where the problem lies.

It's sort of implied by the name `ResultFromCaughtException` that it tries to calculate a result from an exception that was caught. But the `scope_exit` functor is called during unwinding that results from an *uncaught* exception.

There is no caught exception to get a result from!

The C++ language says that a rethrowing `throw` rethrows the exception that is being handled, where "being handled" roughly means "is executing the body of its `catch` clause". If you try a rethrowing throw when there is no exception being handled, then it's straight to jail. (Formally, `std::terminate`.)

The solution, then, is to put the `ResultFromCaughtException` somewhere inside a `catch` block, like perhaps this:

```
void DoSomething()
{
    try {
        ⟦ do stuff that might throw exceptions ⟧
    } catch (...) {
        Log("DoSomething failed",
            wil::ResultFromCaughtException());
        throw;
    })
}
```

After logging the exception, we rethrow it so that the search for a handler can continue.

Bonus chatter: You can avoid a layer of indentation by using function-try.

```
void DoSomething() try
{
    ⟦ do stuff that might throw exceptions ⟧
} catch (...) {
    Log("DoSomething failed",
        wil::ResultFromCaughtException());
    throw;
}
```