# Our first attempt to detect and report all unhandled C++ exceptions as well as all unhandled structured exceptions

devblogs.microsoft.com/oldnewthing/20250710-00/?p=111366

July 10, 2025


Last time, we saw that installing an unhandled structured exception filter prevents the Microsoft Visual C++ runtime from converting unhandled C++ exceptions into `std::terminate`, because the C++ runtime itself uses the unhandled structured exception filter to do that conversion.

One option is to detect that your unhandled structured exception handler is observing an unhandled C++ exception, in which case you allow the call to pass through to the C++ unhandled exception handler so it can turn it into a `std::terminate`.

```
#define MSVC_EXCEPTION_CODE 0xE06D7363U

LPTOP_LEVEL_EXCEPTION_FILTER previousFilter;
std::terminate_handler previousTerminate;

LONG CALLBACK MyUnhandledExceptionFilter(
    EXCEPTION_POINTERS *ExceptionInfo)
{
    if (ExceptionInfo->ExceptionRecord.ExceptionCode ==
            MSVC_EXCEPTION_CODE) {
        return previousFilter(ExceptionInfo);
    }
    CaptureDump(ExceptionInfo,
                UnhandledException::Structured);
    return EXCEPTION_EXECUTE_HANDLER;
}

void __cdecl my_terminate()
{
    CaptureDump(nullptr,
                UnhandledException::Cpp);
    if (previousTerminate) {
        previousTerminate();
    } else {
        std::abort();
    }
}

void InstallCustomUnhandledExceptionFilters()
{
    previousFilter = SetUnhandledExceptionFilter(
                         MyUnhandledExceptionFilter);

    previousTerminate = std::set_terminate(my_terminate);
}
```

The diagram looks like this:

| | |
|---|---|
| Exception raised | |
| ↓ | |
| Frame handler | (not handled) |
| ↓ | |
| Frame handler | (not handled) |
| ↓ | |
| ⋮ | |
| ↓ | |
| Frame handler | (not handled) |

| | | |
|---|---|---|
| ↓ | | |
| MyUnhandled-ExceptionFilter | not C++ → | CaptureDump(Structured) |
| ↓ | | |
| CppUnhandled-ExceptionFilter | | |
| ↓ | | |
| std::terminate | → | CaptureDump(Cpp) |

But wait, why pass the C++ exception through to the runtime's unhandled exception handler if we know that it's just going to call our our custom terminate handler?[1] We can call the custom terminate handler directly.

```
#define MSVC_EXCEPTION_CODE 0xE06D7363U

LONG CALLBACK MyUnhandledExceptionFilter(
    EXCEPTION_POINTERS *ExceptionInfo)
{
    if (ExceptionInfo->ExceptionRecord.ExceptionCode ==
            MSVC_EXCEPTION_CODE) {
        CaptureDump(ExceptionInfo,
                    UnhandledException::Structured);
    } else {
        CaptureDump(ExceptionInfo,
                    UnhandledException::Cpp);
    }
    return EXCEPTION_EXECUTE_HANDLER;
}

void InstallCustomUnhandledExceptionFilters()
{
    previousFilter = SetUnhandledExceptionFilter(
                        MyUnhandledExceptionFilter);
}
```

Now the diagram looks like this:

| | |
|---|---|
| Exception raised | |
| ↓ | |
| Frame handler | (not handled) |
| ↓ | |
| Frame handler | (not handled) |
| ↓ | |

| | | | |
|---|---|---|---|
| ⋮ | | | |
| ↓ | | | |
| Frame handler | | (not handled) | |
| ↓ | | | |
| MyUnhandled-ExceptionFilter | → | CaptureDump(Cpp) | If not C++ |
| | ↘ | | |
| | | CaptureDump(Cpp) | If C++ |

But wait, this is still not complete. There are ways for a program to terminate with an unhandled exception that don't go through the unhandled exception filter. We'll look at those next time.

[1] On reason is that from the custom terminate handler, you can use `std::current_exception()` to snoop on the exception that caused the termination (if there is one).