C++ allows for nonsense, and that’s great

Justin Tremblay

Who am I?

- 4th year software engineering student at McGill
- Interned at:
    - Eidos Montreal, Gameplay, Shadow of the Tomb Raider
    - Eidos Montreal, Tools, [insert game title]
    - Genetec, Omnicast
- First started using C++ in 2014, in CÉGEP
- Into Game Engines and Compilers
- This is my first tech talk!

Disclaimer

This code is my own creation, although it may be similar, it is not the same as the code that was written as part of my internship at Genetec.

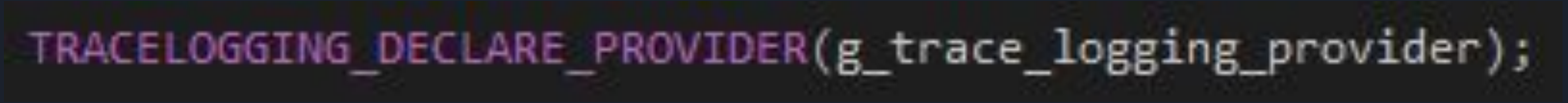
Some Background

- During my last internship, I was given the task to implement a library to log ETW traces.
- Introduced in Windows 10, the TraceLogging library builds on top of the ETW technology, which was introduced in Windows 2000.
- Very low impact on your program's runtime performance.

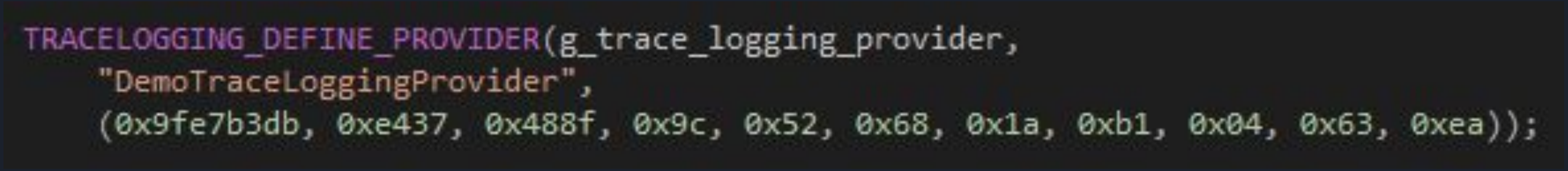
- **You said nonsense, where's the catch?**

It’s all macros!

- Most of the library is implemented as macros.
  - Most likely to preserve backward compatibility with other Windows versions.
- This is nonsense! Macros are evil! Why would anyone ever implement a whole library as macros?!



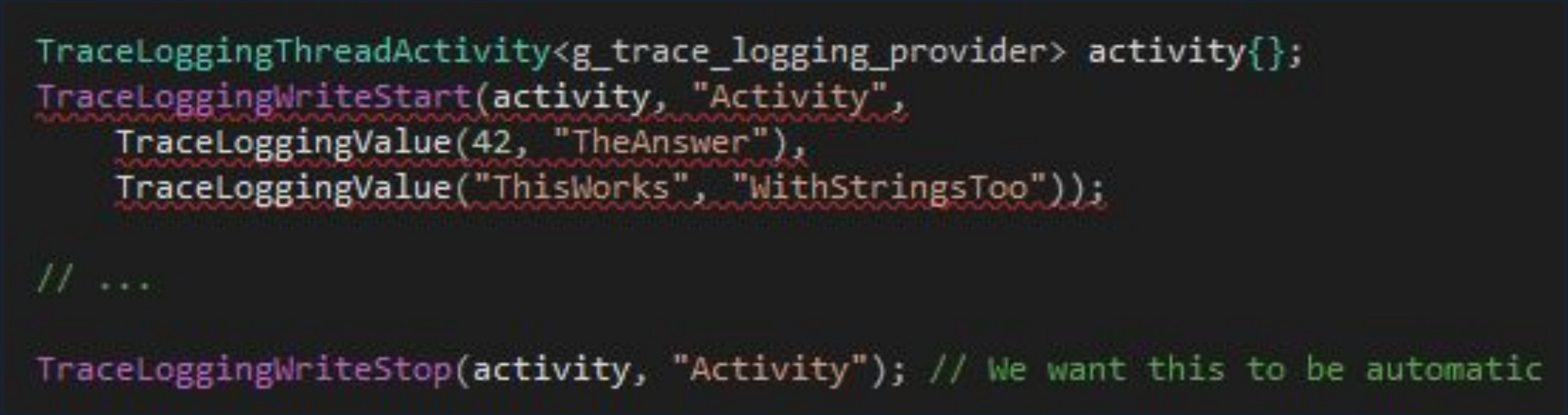
```
TRACELOGGING_DECLARE_PROVIDER(g_trace_logging_provider);
```



```
TRACELOGGING_DEFINE_PROVIDER(g_trace_logging_provider,
    "DemoTraceLoggingProvider",
    (0x9fe7b3db, 0xe437, 0x488f, 0x9c, 0x52, 0x68, 0x1a, 0xb1, 0x04, 0x63, 0xea));
```

It’s all macros!

- Another annoying restriction: whenever a name must be given, it <u>MUST</u> be a string literal, not a variable.



```
TraceLoggingThreadActivity<g_trace_logging_provider> activity{};
TraceLoggingWriteStart(activity, "Activity",
    TraceLoggingValue(42, "TheAnswer"),
    TraceLoggingValue("ThisWorks", "WithStringsToo"));

// ...

TraceLoggingWriteStop(activity, "Activity"); // We want this to be automatic
```

Also, resharper doesn’t like the TraceLoggingValue macro (even though it compiles).

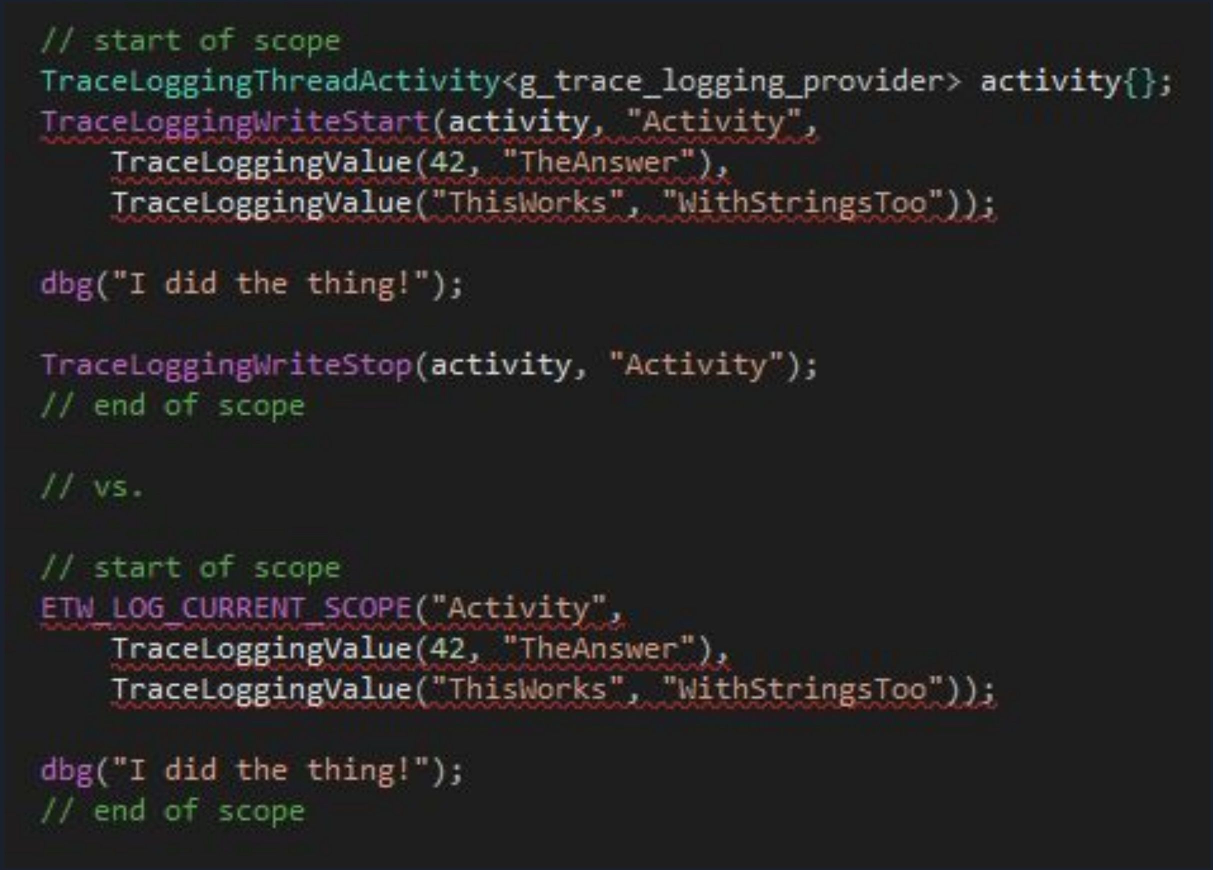
The Goal

- Make it easy to log an ETW activity.
  - It has to be a “one-liner” for the devs using it.
- Automate the “Stop” event (One-liner)
  - The type of activity I’m using (TraceLoggingThreadActivity) actually stops automatically at the end of the scope, but the event isn’t very descriptive.
- But how do you wrap around a macro that requires a string literal to be passed?
- How do you automate the call to TraceLoggingWriteStop at the end of the scope?

The Solution: More Nonsense!

Before:

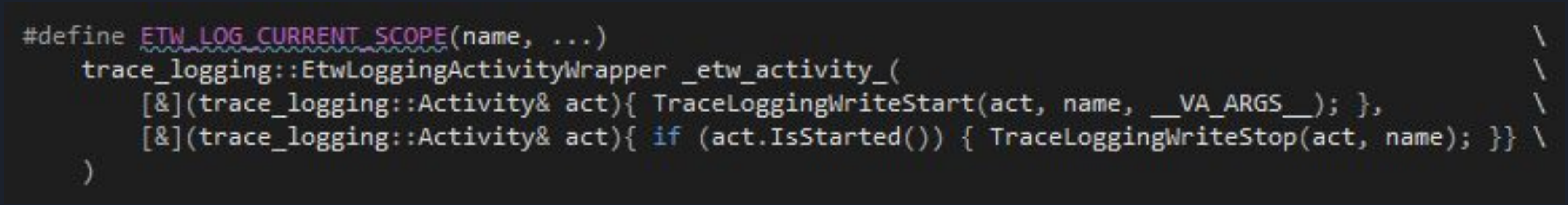
After:



```
// start of scope
TraceLoggingThreadActivity<g_trace_logging_provider> activity{};
TraceLoggingWriteStart(activity, "Activity",
    TraceLoggingValue(42, "TheAnswer"),
    TraceLoggingValue("ThisWorks", "WithStringsToo"));

dbg("I did the thing!");

TraceLoggingWriteStop(activity, "Activity");
// end of scope

// vs.

// start of scope
ETW_LOG_CURRENT_SCOPE("Activity",
    TraceLoggingValue(42, "TheAnswer"),
    TraceLoggingValue("ThisWorks", "WithStringsToo"));

dbg("I did the thing!");
// end of scope
```

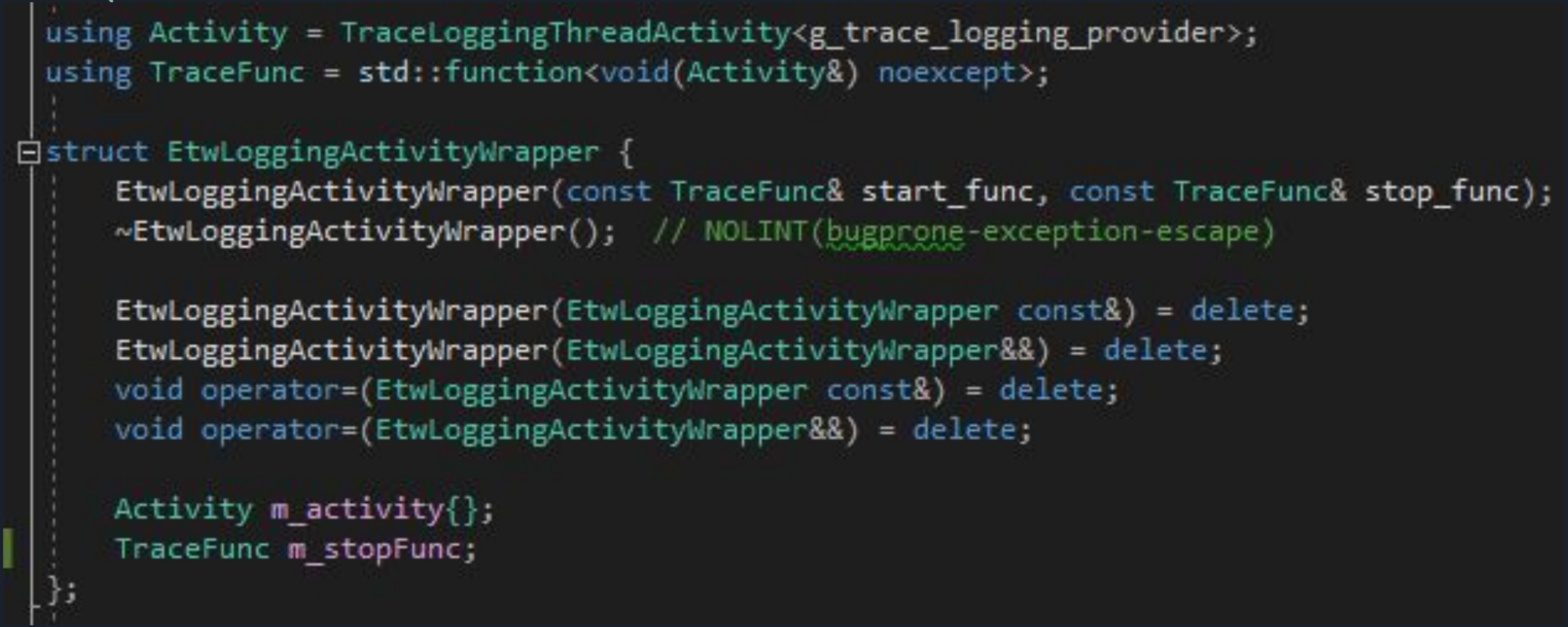
Implementation

- Passes the string literal “name” to the TraceLoggingWriteStart and TraceLoggingWriteStop calls.
- Captures the local scope by reference so that you can use local variables in the variadic arguments list.



```
#define ETW_LOG_CURRENT_SCOPE(name, ...)                                                   \
    trace_logging::EtwLoggingActivityWrapper _etw_activity_(                               \
        [&](trace_logging::Activity& act){ TraceLoggingWriteStart(act, name, __VA_ARGS__); },       \
        [&](trace_logging::Activity& act){ if (act.IsStarted()) { TraceLoggingWriteStop(act, name); }} \
    )
```

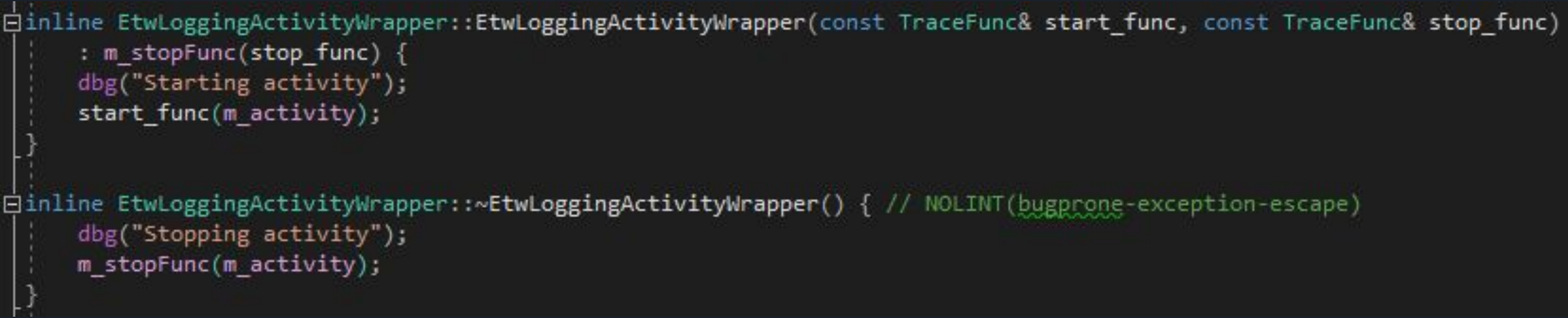
Implementation



```cpp
using Activity = TraceLoggingThreadActivity<g_trace_logging_provider>;
using TraceFunc = std::function<void(Activity&) noexcept>;

struct EtwLoggingActivityWrapper {
    EtwLoggingActivityWrapper(const TraceFunc& start_func, const TraceFunc& stop_func);
    ~EtwLoggingActivityWrapper();  // NOLINT(bugprone-exception-escape)

    EtwLoggingActivityWrapper(EtwLoggingActivityWrapper const&) = delete;
    EtwLoggingActivityWrapper(EtwLoggingActivityWrapper&&) = delete;
    void operator=(EtwLoggingActivityWrapper const&) = delete;
    void operator=(EtwLoggingActivityWrapper&&) = delete;

    Activity m_activity{};
    TraceFunc m_stopFunc;
};
```

Implementation

- Calls “start_func” upon construction and “stop_func” upon destruction



```
inline EtwLoggingActivityWrapper::EtwLoggingActivityWrapper(const TraceFunc& start_func, const TraceFunc& stop_func)
    : m_stopFunc(stop_func) {
    dbg("Starting activity");
    start_func(m_activity);
}

inline EtwLoggingActivityWrapper::~EtwLoggingActivityWrapper() { // NOLINT(bugprone-exception-escape)
    dbg("Stopping activity");
    m_stopFunc(m_activity);
}
```

What I’ve learned

- Macros are evil, but sometimes necessary.
- Sometimes the solution to a problem seems silly, but it’s the right one.
- C++ allows for complete nonsense, and that’s great!

Thank you for listening!

References

- TraceLogging quickstart and documentation:
  - https://docs.microsoft.com/en-us/windows/win32/tracelogging/tracelogging-native-quick-start
- “dbg” macro used in code:
  - https://github.com/sharkdp/dbg-macro
- Code and (soon) slides/write-up available at:
  - https://github.com/L3gume/ETWTracing