TypeLayout: Catching ABI Bugs at Compile Time with C++26 Reflection

Abstract

What if ABI mismatches could be caught **at compile time**, before your code ever runs? **TypeLayout** makes this possible by generating compile-time layout signatures using C++26 static reflection (P2996).

TypeLayout is a header-only library that generates **compile-time memory layout signatures**—human-readable strings that uniquely describe how a type is laid out in memory:

```
struct NetworkPacket { uint32_t id; uint64_t timestamp; };

constexpr auto sig = get_layout_signature<NetworkPacket>();
// "[64-le]struct[s:16,a:8]{@0[id]:u32[s:4,a:4],@8[timestamp]:u64[s:8,a:8]}"

constexpr auto hash = get_layout_hash<NetworkPacket>();  // 64-bit hash for fast comparison
```

The core guarantee: *Identical signature* ⟺ *Identical memory layout*. Two types with the same signature can safely share memory, no matter where or when they were compiled.

Embed the hash in your interface contracts, and mismatches become compile-time errors:

```
static_assert(get_layout_hash<NetworkPacket>() == EXPECTED_HASH, "ABI changed!");
```

This unlocks powerful patterns across all binary boundaries:

- **Shared Memory IPC**: Verify layout compatibility before mapping
- **Network Protocols**: Detect version mismatch at compile time or on receive
- **Plugin Systems**: Reject incompatible DLLs at load time
- **Binary Files**: Auto-detect schema changes on read

What makes TypeLayout unique:

Feature	TypeLayout	Protobuf/FlatBuffers
Runtime overhead	**Zero**	Serialization cost
Code generation	**None**	Required
IDL files	**None**	Required
Works with existing C++ types	**Yes**	No (must define in IDL)
Bit-field precision	**Bit-level**	Not supported

P2996 static reflection gives us compile-time access to *every* member—including private fields, base classes, virtual tables, and individual bits in bit-fields. The signature captures offsets, sizes, alignments, and platform characteristics (32/64-bit, endianness).

In this talk, you'll see:

- Live signature generation for complex types (inheritance, polymorphism, bit-fields)
- Robust hash verification with collision resistance
- Four production patterns with working code
- A plugin system demo that catches ABI mismatches at load time

This is what C++26 reflection was designed for—come see it in action.

Key Takeaways

1. **Layout signatures are the primitive** - A single string captures everything about memory layout
2. **Zero-cost verification** - All analysis happens at compile time
3. **No tooling required** - Write native C++ structs, get automatic compatibility checking
4. **Reflect existing code** - Works with your current types, no IDL migration needed

Target Audience

- Systems programmers working with binary interfaces (IPC, networking, plugins)
- Library authors concerned about ABI stability across versions
- Developers curious about practical applications of C++26 reflection
- Anyone tired of writing IDL files and running code generators

Prerequisites

- Basic understanding of C++ struct layout (padding, alignment)
- Familiarity with binary interface challenges (helpful but not required)
- No prior knowledge of P2996 required—we'll cover the essentials

Speaker Bio

[Your bio here]

Additional Notes for Reviewers

- **Working implementation**: https://github.com/ximicpp/TypeLayout
- **Live demo ready**: Compiles with Bloomberg Clang P2996 fork
- **Comprehensive examples**: Shared memory, network protocols, file formats, plugin systems
- **Full type coverage**: Classes, private members, inheritance, polymorphism, bit-fields, STL containers