# P3637R0 Inherit `std::meta::exception` from `std::exception`

2025-03-08

**Authors:**

Victor Zverovich
Nevin Liber
Michael Hava

**Audience:**

LEWG

**Project:**

ISO/IEC 14882 Programming Languages — C++, ISO/IEC JTC1/SC22/WG21

## Introduction

P3560R1 “Error Handling in Reflection” introduced `std::meta::exception` as a new exception type for handling errors in C++ reflection but made the unconventional choice of not deriving it from `std::exception`. This decision, influenced by outdated encoding concerns, creates inconsistencies within the standard and complicates generic exception handling.

This proposal aims to resolve the issue by inheriting `std::meta::exception` from `std::exception`, ensuring consistency with other exception types, including those used at compile time. This change simplifies error handling in generic code and aligns with existing exception design principles in C++.

## Problem

P3560R1 “Error Handling in Reflection” was reviewed by LEWG in Hagenberg and forwarded to LWG for C++26. The paper introduces a new exception class, `std::meta::exception` and uses it for reporting errors from reflection.

One novel design decision in this paper was not inheriting `std::meta::exception` from `std::exception`. To the best of our knowledge this is the first such case in the standard. There was interest in fixing this in LEWG but it narrowly didn’t get consensus.

| SF | F | N | A | SA |
|---|---|---|---|---|
| 4 | 3 | 5 | 1 | 3 |

Per email discussion that followed, it turned out that the main motivation for this decision was an encoding concern based on outdated information in LWG4087. Exception message encoding has been specified for the compile-time case ([exception]):

```
constexpr virtual const char* what() const noexcept;
```

> *Returns*: An implementation-defined NTBS, which during constant evaluation is encoded with the ordinary literal encoding ([lex.ccon]).

Encoding-wise this is compatible with the current design of P3560R1 which provides `what()` in the ordinary literal encoding which means that the motivation for not inheriting from `std::exception` no longer exists.

Not inheriting from `std::exception` is also inconsistent with P3557 that was reviewed by LEWG shortly after and also provided a new compile-time exception class.

Apart from consistency reasons, having a single exception hierarchy is beneficial for writing generic code. For example, consider tests for a facility that uses reflection and has other logic that may throw different exception types. You obviously want to see all failures in one compilation cycle. So test cases will be wrapped in a try-catch block that will catch exceptions and report them as test failures as it is normally done in test frameworks. And the common pattern of catching `std::exception` will no longer work, requiring users to catch both `std::exception` and `std::meta::exception` for no good reason.

Unlike other standard library types derived from `std::exception`, `std::meta::exception` does not have a non-throwing exception specification for its copy constructor and its copy assignment operator. This does not violate [exception]p2. Also, unlike the other standard library exception types, this type has a move constructor and a move assignment operator.

# Proposal

The current paper proposes inheriting `std::meta::exception` from `std::exception` to make it consistent with other exceptions including ones exclusively used at compile time and making generic exception handling easier.

Wording relative to P3560R1:

```
class exception : public std::exception
{
private:
  optional<string> what_;  // exposition only
  u8string u8what_;        // exposition only
  info from_;              // exposition only
  source_location where_;  // exposition only

public:
  consteval exception(u8string_view what, info from,
    source_location where = source_location::current()) noexcept;

  consteval exception(string_view what, info from,
    source_location where = source_location::current()) noexcept;

  exception(exception const&) = default;
  exception(exception&&) = default;

  exception& operator=(exception const&) = default;
  exception& operator=(exception&&) = default;

  constexpr const char* what() const noexcept override;
  consteval u8string_view u8what() const noexcept;
  ~~consteval string what() const noexcept;~~
  consteval info from() const noexcept;
  consteval source_location where() const noexcept;
};
```

Reflection functions throw exceptions of type `std::meta::exception` to signal an error. `std::meta::exception` is a consteval-only type.

```
  consteval exception(u8string_view what, info from,
      source_location where = source_location::current()) noexcept;
```

*Effects*: Initializes `u8what_` with `what`, `from_` with `from` and `where_` with `where`. If `what` can be represented in the ordinary literal encoding, initializes `what_` with `what`, transcoded from UTF-8 to the ordinary literal encoding.

```
  consteval exception(string_view what, info from,
      source_location where = source_location::current()) noexcept;
```

*Effects*: Initializes `what_` with `what`, `u8what_` with `what`, transcoded from the ordinary literal encoding to UTF-8, `from_` with `from` and `where_` with `where`.

```
consteval u8string_view u8what() const noexcept;
```

*Returns*: what_.

```
~~consteval string what() const noexcept;~~
constexpr const char* what() const noexcept override;
```

*Constant When*: ~~`what_` can be represented in the ordinary literal encoding.~~ `what_.has_value()` is `true`.

*Returns*: ~~`what_`, converted to the ordinary literal encoding.~~ `what_->c_str()`.

…

In the common case of ordinary literal encoding being UTF-8, implementations can have a single representation for the error message and transcoding is a noop. The optimization likely doesn't matter since it only affects error paths.

# Implementation

The current proposal is trivially implementable on top of P3560R1 and the demo implementation is available at https://compiler-explorer.com/z/nq63z918T using a fork of clang that supports compile-time exceptions.

# References


- Eric Niebler. P3557 High-Quality Sender Diagnostics with Constexpr Exceptions.
- Peter Dimov, Barry Revzin. P3560R1 Error Handling in Reflection.
- Victor Zverovich. LWG4087 Standard exception messages have unspecified encoding.