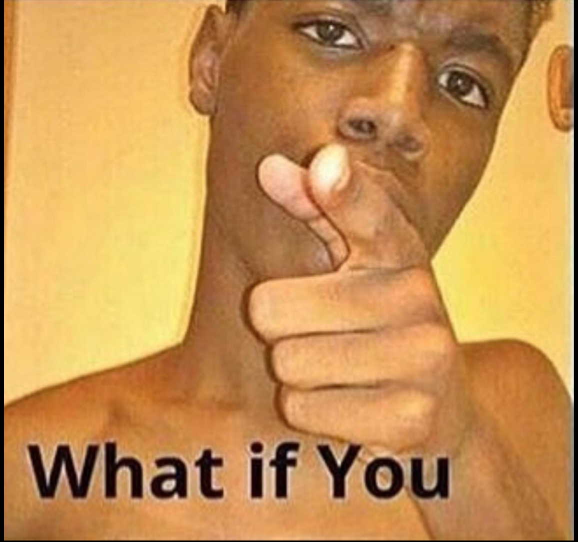
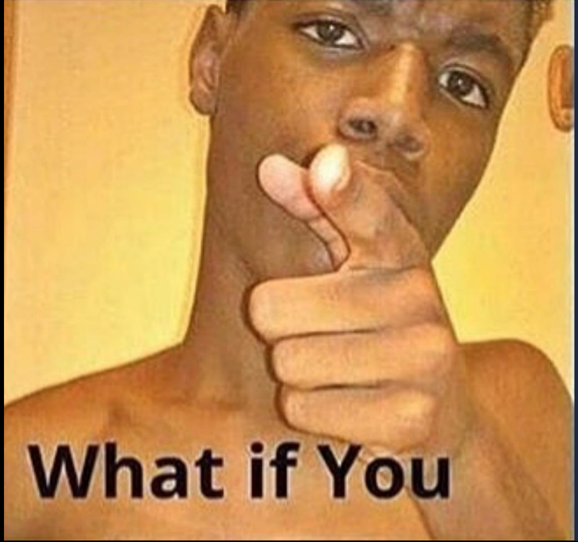
Can set_value Actually Throw?

Robert Leahy
Core Developer
rleahy@rleahy.ca


What if You



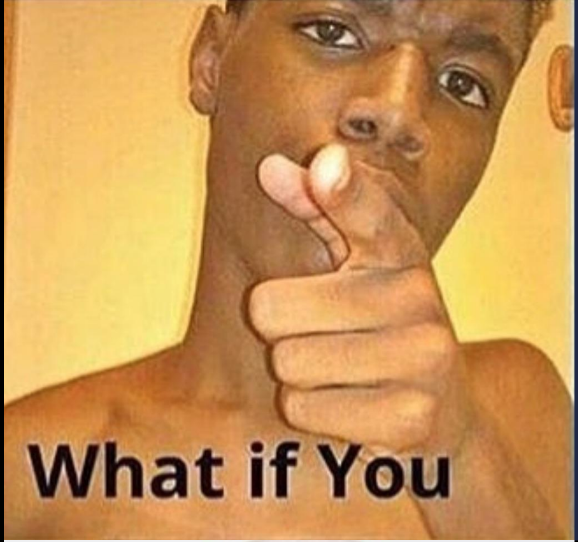
`std::execution` in Asio Codebases

Adopting Senders Without a Rewrite

ROBERT LEAHY

Cppcon
The C++ Conference

2025
September 13 - 19



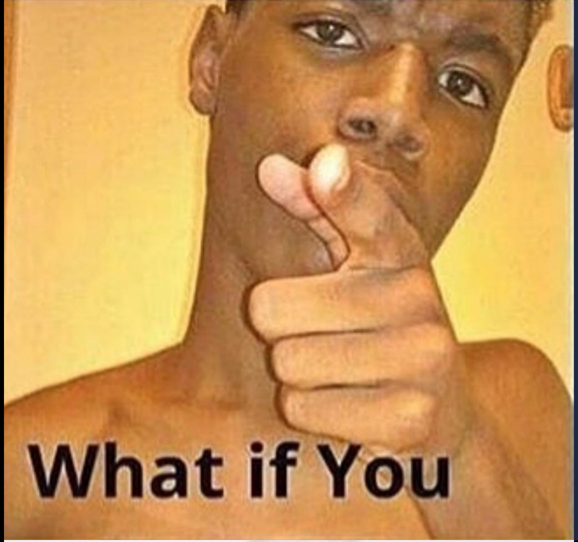
std::execution in Asio Codebases

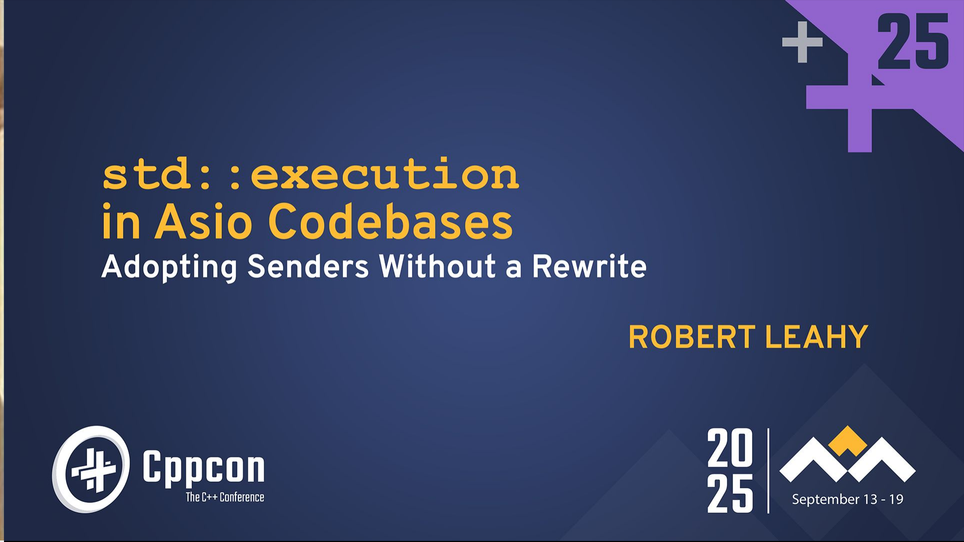
Adopting Senders Without a Rewrite

ROBERT LEAHY

Cppcon
The C++ Conference

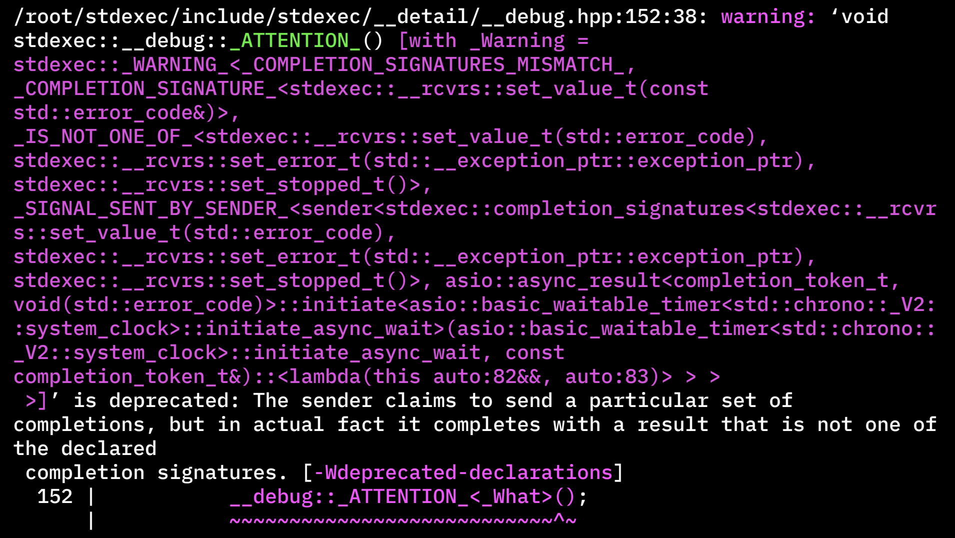
2025
September 13 - 19




std::execution
in Asio Codebases
Adopting Senders Without a Rewrite

ROBERT LEAHY

Cppcon
The C++ Conference
2025
September 13 - 19


```
/root/stdexec/include/stdexec/__detail/__debug.hpp:152:38: warning: ‘void
stdexec::__debug::_ATTENTION_() [with _Warning =
stdexec::_WARNING_<_COMPLETION_SIGNATURES_MISMATCH_,
_COMPLETION_SIGNATURE_<stdexec::__rcvrs::set_value_t(const
std::error_code&)>,
_IS_NOT_ONE_OF_<stdexec::__rcvrs::set_value_t(std::error_code),
stdexec::__rcvrs::set_error_t(std::__exception_ptr::exception_ptr),
stdexec::__rcvrs::set_stopped_t()>,
_SIGNAL_SENT_BY_SENDER_<sender<stdexec::completion_signatures<stdexec::__rcvr
s::set_value_t(std::error_code),
stdexec::__rcvrs::set_error_t(std::__exception_ptr::exception_ptr),
stdexec::__rcvrs::set_stopped_t()>, asio::async_result<completion_token_t,
void(std::error_code)>::initiate<asio::basic_waitable_timer<std::chrono::_V2:
:system_clock>::initiate_async_wait>(asio::basic_waitable_timer<std::chrono::
_V2::system_clock>::initiate_async_wait, const
completion_token_t&)::<lambda(this auto:82&&, auto:83)> > >
 >]’ is deprecated: The sender claims to send a particular set of
completions, but in actual fact it completes with a result that is not one of
the declared
 completion signatures. [-Wdeprecated-declarations]
  152 |           __debug::_ATTENTION_<_What>();
      |           ~~~~~~~~~~~~~~~~~~~~~~~~~~~~~~^~
```

```
/root/stdexec/include/stdexec/__detail/__debug.hpp:152:38: warning: ‘void
stdexec::__debug::_ATTENTION_() [with _Warning =
stdexec::_WARNING_<_COMPLETION_SIGNATURES_MISMATCH_,
_COMPLETION_SIGNATURE_<stdexec::__rcvrs::set_value_t(const std::error_code&)>,
_IS_NOT_ONE_OF_<stdexec::__rcvrs::set_value_t(std::error_code),
stdexec::__rcvrs::set_error_t(std::__exception_ptr::exception_ptr),
stdexec::__rcvrs::set_stopped_t()>,
_SIGNAL_SENT_BY_SENDER_<sender<stdexec::completion_signatures<stdexec::__rcvrs::set_val
ue_t(std::error_code),
stdexec::__rcvrs::set_error_t(std::__exception_ptr::exception_ptr),
stdexec::__rcvrs::set_stopped_t()>, asio::async_result<completion_token_t,
void(std::error_code)>::initiate<asio::basic_waitable_timer<std::chrono::_V2::system_cl
ock>::initiate_async_wait>(asio::basic_waitable_timer<std::chrono::_V2::system_clock>::
initiate_async_wait, const completion_token_t&)::<lambda(this auto:82&&, auto:83)> > >
 >]’ is deprecated: The sender claims to send a particular set of completions, but in
actual fact it completes with a result that is not one of the declared
 completion signatures. [-Wdeprecated-declarations]
  152 |           __debug::_ATTENTION_<_What>();
      |           ~~~~~~~~~~~~~~~~~~~~~~~~~~~^~
```

```
                       stdexec::__rcvrs::set_value_t(const std::error_code&)>,
_IS_NOT_ONE_OF_<stdexec::__rcvrs::set_value_t(std::error_code)
```

```
std::invoke(
  std::move(init_),
  [this](auto&&... args) noexcept {
    std::execution::set_value(
      std::move(r_),
      std::forward<decltype(args)>(args)...);
  });
```

```cpp
struct receiver {
  using receiver_concept = std::exception::receiver_t;
  void set_value(std::string) && noexcept;
};
```

```
std::execution::completion_signatures<
  std::execution::set_value_t(std::string),
  /* ... */>;
```

```
const std::string str("...");

std::execution::set_value(std::move(rcvr), str);
```

```
const std::string str("...");

std::move(rcvr).set_value(str);
```

```cpp
const std::string str("...");

std::move(rcvr).set_value(std::string(str));
```

From the Standard

“`set_value` is a value completion function. [...] The expression `set_value(rcvr, vs...)` for a subexpression `rcvr` and pack of subexpressions `vs` is [...] expression-equivalent to *`MANDATE-NOTHROW`*`(rcvr.set_value(vs...))`.”

“For a subexpression `expr`, let *`MANDATE-NOTHROW`*`(expr)` be expression-equivalent to `expr`.

“*Mandates*: `noexcept(expr)` is `true`.”

```
struct receiver {
  using receiver_concept = std::exception::receiver_t;
  void set_value(std::string) && noexcept;
};
```

```
struct receiver {
  using receiver_concept = std::exception::receiver_t;
  void set_value(std::string) && noexcept;
};
```

```
struct receiver {
  using receiver_concept = std::exception::receiver_t;
  void set_value(std::string&&) && noexcept;
};
```

```cpp
template <typename T, typename U>
  requires
    std::is_convertible_v<U&&, T&&>
constexpr T&& convert(U&& u) noexcept {

  return std::forward<U>(u);

}

template <typename T, typename U>
constexpr std::remove_cvref_t<T> convert(U&& u) {

  return std::forward<U>(u);

}
```

```cpp
static_assert(!noexcept(convert<std::string>("foo")));
```

```
template <typename T, typename U>
  requires
    std::is_convertible_v<U&&, T&&>
constexpr T&& convert(U&& u) noexcept {

  return std::forward<U>(u);

}
```

```
template <typename T, typename U>
  requires
    std::is_convertible_v<U&&, T&&>
constexpr T&& convert(U&& u) noexcept {

  return std::forward<U>(u);

}
```

```
template <typename T, typename U>
  requires (
    std::is_convertible_v<U&&, T&&> &&
    !std::reference_converts_from_temporary_v<T&&, U&&>)
constexpr T&& convert(U&& u) noexcept {

  return std::forward<U>(u);

}
```

```
template <
  typename Tuple,
  typename Receiver,
  std::size_t... Ns,
  typename... Args>
constexpr void set_value_impl(
  Receiver&& r,
  std::index_sequence<Ns...>,
  Args&&... args)
{

  std::execution::set_value(
    std::forward<Receiver>(r),
    ::convert<
      std::tuple_element_t<Ns, Tuple>>(
        std::forward<Args>(args))...);

}
```

```
template <
  typename Tuple,
  typename Receiver,
  std::size_t... Ns,
  typename... Args>
constexpr void set_value_impl(
  Receiver&& r,
  std::index_sequence<Ns...>,
  Args&&... args)
{

  std::execution::set_value(
    std::forward<Receiver>(r),
    ::convert<
      std::tuple_element_t<Ns, Tuple>>(
        std::forward<Args>(args))...);

}
```

```
template <typename>
struct has_function_call_operator {

  struct type {};

  static std::tuple<> operator()(type);

};

template <typename... Args>
struct has_function_call_operator<std::execution::set_value_t(Args...)> {

  static std::tuple<Args...> operator()(Args...);

};
```

```cpp
template <typename>
struct overload_set;

template <typename... Signatures>
struct overload_set<
  std::execution::completion_signatures<Signatures...>>
    : has_function_call_operator<Signatures>...
{

  using has_function_call_operator<Signatures>::operator()...;

};
```

```
template <typename Signatures, typename Receiver, typename... Args>
constexpr void set_value(Receiver&& r, Args&&... args) {

  using tuple = decltype(
    overload_set<Signatures>{}(
      std::declval<Args>()...));

  ::set_value_impl<tuple>(
    std::forward<Receiver>(r),
    std::make_index_sequence<std::tuple_size_v<tuple>>{},
    std::forward<Args>(args)...);

}
```

```
std::invoke(
  std::move(init_),
  [this](auto&&... args) noexcept {
    std::execution::set_value(
      std::move(r_),
      std::forward<decltype(args)>(args)...);
  });
```

```
std::invoke(
  std::move(init_),
  [this](auto&&... args) noexcept {
    std::execution::set_value(
      std::move(r_),
      std::forward<decltype(args)>(args)...);
  });
```

```
std::invoke(
  std::move(init_),
  [this](auto&&... args) noexcept {
    try {
      ::set_value(
        std::move(r_),
        std::forward<decltype(args)>(args)...);
    } catch (...) {
      std::execution::set_error(
        std::move(r_),
        std::current_exception());
    }
  });
```

Thank you!