

when_all_range Sender Adaptors

Document #: PXXXXR0
Date: 2025-10-08
Project: Programming Language C++
Audience: Library Working Group
Reply-to: Lixin Wei
<lixin.wei@gmail.com>

1 Abstract

This paper proposes to add a new sender adaptor algorithm `when_all_range` to the `std::execution` namespace, targeting C++29. It accepts a `std::ranges::range` of senders, allowing user to wait for multiple senders which are only known at runtime.

2 Description

2.1 The Problem Now

[P2300R10] provides `std::execution::when_all`, which allows user to wait multiple senders which are known at compile time. But there still lacks a way to wait for multiple senders which are **only known at runtime**.

For example:

```
int main(int argc, char** argv) {  
    int runtime_known_number = std::atoi(argv[1]);  
    std::vector<MySender> senders;  
    for (int i = 0; i < runtime_known_number; ++i) {  
        senders.push_back(CreateAsyncWork());  
    }  
  
    ex::when_all(senders); // this line will fail to compile  
}
```

In the above example, `when_all` doesn't work, because its signature is `when_all(ex::sender auto&&... senders)` which requires the number of senders to be known at compile time. We can't pass a dynamic container to it.

2.2 Proposed Solution

To resolve this problem, this paper proposes to add a new sender adaptor algorithm called `when_all_range` to the `std::execution` namespace, which accepts a `std::ranges::range` of senders.

```
struct when_all_range_t {  
    // qualifiers omitted for simplicity  
    ex::sender auto operator()(std::ranges::range Range auto&& range);  
};  
inline constexpr when_all_range_t when_all_range{};
```

Like `when_all`, `when_all_range` adapts all senders in the range into a sender that completes when all input senders have completed. `when_all_range` only accepts senders with a single value completion signature and on success concatenates all the input senders' value result datums into a new `std::ranges::range` and pass it to its own value completion operation.

Let's call the range passed to the value completion operation of the returned sender as **output range**.

The output range should conform `std::ranges::random_access_range`. The *i*-th element in the output range should be the value result datum of the *i*-th sender(iteration order) in the input range.

```
using SenderType = decltype(ex::just(1));
std::vector<SenderType> senders;
for (int i = 0; i < 10; ++i) {
    senders.emplace(ex::just(i));
}
ex::when_all_range(std::move(senders)) |
ex::then([](std::ranges::random_access_range auto&& res) {
    for (int i = 0; i < 10; ++i) {
        // the result order should be the same as the input order
        assert(res[i] == i);
    }
});
```

To emphasize, even though the input range is not a `std::ranges::random_access_range`, the output range should still conform `std::ranges::random_access_range`. Where the output order is the same as the input's iteration order.

```
using SenderType = decltype(ex::just(1));
// assume we have a comparator for `ex::just(i)` which sorts them by `i`.
std::set<SenderType> senders;
for (int i = 0; i < 10; ++i) {
    senders.emplace(ex::just(i));
}

// input range is not a `std::ranges::random_access_range` now
ex::when_all_range(std::move(senders)) |
ex::then([](std::ranges::random_access_range auto&& res) {
    for (int i = 0; i < 10; ++i) {
        // the result order should be the same as the input's iteration order
        assert(res[i] == i);
    }
});
```

3 Proposed Wording

[Editor's note: Change [execution.syn] as follows:]

```
struct when_all_t { unspecified };
struct when_all_with_variant_t { unspecified };
struct into_variant_t { unspecified };
struct when_all_range_t { unspecified };
```

```
inline constexpr when_all_t when_all{};
inline constexpr when_all_with_variant_t when_all_with_variant{};
inline constexpr into_variant_t into_variant{};
inline constexpr when_all_range_t when_all_range{};
```

4 References

[P2300R10] Eric Niebler, Michał Dominiak, Georgy Evtushenko, Lewis Baker, Lucian Radu Teodorescu, Lee Howes, Kirk Shoop, Michael Garland, Bryce Adelstein Lelbach. 2024-06-28. 'std::execution'.

<https://wg21.link/p2300r10>