

Project: ISO JTC1/SC22/WG21: Programming Language C++
 Doc No: WG21 **P2432R1**
 Date: 2021-09-24
 Reply to: Nicolai Josuttis (nico@josuttis.de)
 Audience: LEWG, LWG
 Issues:

Fix istream_view, Rev 1

This paper fixes a fundamental design problem with the current helper function `std::ranges::istream_view<>()` that cause multiple inconsistencies and unnecessary code overhead when declaring `istream_view` objects.

Tony Table:

Before	After
<code>std::ranges::istream_view<int> v{mystream}</code> // ERROR	<code>std::ranges::istream_view<int> v{mystream}</code> // OK
<code>std::ranges::istream_view<int>(mystream)</code> // OK	<code>std::ranges::istream_view<int>(mystream)</code> // still OK
// using input stream for wchar_t: <code>std::ranges::istream_view<int>{mywstream}</code>	// using input stream for wchar_t: <code>std::ranges::wistream_view<int>{mywstream}</code> // or: <code>std::views::istream<int>(mywstream)</code>
// using input stream for other char type: <code>std::ranges::istream_view<int>{u8stream}</code>	// using input stream for other char type: <code>std::views::istream<int>(u8stream)</code>

Rev1:

Small fixes on examples and wording.

Rev0:

First initial version.

Motivation

The current definition of `std::ranges::istream_view()` breaks several basic conventions:

- This would be the first type called `basic_xyz` that has a corresponding symbol `xyz` that is not a type.
- This would be the only symbol `xyz_view` that is not a view type, but a function (usually, we have corresponding adaptors in namespace `std::views` as functions).

It hinders to declare an `istream_view` just as follows:

```
std::istringstream mystream{"0 1 2 3 4"};
std::ranges::istream_view<int> v{mystream}; // ERROR
```

Instead, the programmer has to implement:

```
std::istringstream mystream{"0 1 2 3 4"};
std::ranges::basic_istream_view<int, char> v{mystream};
```

It also confuses programmers because using `{}` to create a temporary `istream_view` does not compile (and yields an even more confusing error message):

```
for (int val : std::ranges::istream_view<int>{mystream}) { // ERROR
```

```
...  
}
```

Instead, the programmer has to implement

```
for (int val : std::ranges::istream_view<int>(mystream)) {  
or:  
for (int val : std::ranges::basic_istream_view<int, char>{mystream}) {  
...  
}
```

Therefore, this paper proposes to fix this design mistake so that we follow the usual conventions. The fix should be a defect against C++20.

In addition, this view is the only type `xyz_view` without a adaptor in namespace `std::view`. So I propose to add it.

wistream_view

We have to decide whether also to support other char types with a corresponding convenience function: `wistream_view`, `u8istream_view`, `u16istream_view`, `u32istream_view`

In C++20, currently, We have full support for `char`, `wchar_t`, `char8_t`, `char16_t`, and `char32_t` only for:

- `basic_string`
- `basic_string_view`
- `streampos`

We only have support for `char` and `wchar_t` for

- `basic_istream`, `basic_ostream`, `basic_iostream`
- `basic_istringstream`, `basic_ostringstream`, `basic_stringstream`
- `basic_stringbuf`
- `basic_filebuf`
- `basic_streambuf`
- `basic_format`

As this feature belongs to the stream area, I propose only to standardize types `istream_view` and `wistream_view`.

Backward Compatibility

With the proposed fix, code using

```
for (int val : std::ranges::istream_view<int>(mywstream)) {  
...  
}
```

will still compile and work.

Code using this view for wide strings:

```
for (int val : std::ranges::istream_view<int>(mywstream)) {  
...  
}
```

will no longer compile, but can easily be converted to:

```
for (int val : std::ranges::wistream_view<int>(mywstream)) {  
...  
}
```

```

    }
or to:
    for (int val : std::views::istream<int>(mywstream)) {
        ...
    }

```

Code using this view for UTF strings:

```

    for (int val : std::ranges::istream_view<int>(ustream)) {
        ...
    }

```

will no longer compile, but can easily be converted to:

```

    for (int val : std::ranges::basic_istream_view<int, char8_t>(ustream)) {
        ...
    }

```

or to:

```

    for (int val : std::views::istream<int>(ustream)) {
        ...
    }

```

I don't assume that much code like that is written yet. And the way to perform the fix is easy.

Overall consistency is far more worth because otherwise programmers using char streams have to pay a significant price (plus confusion due to inconsistent design).

Proposed Solution

In 24.2 Header <ranges> synopsis [ranges.syn]

replace

```

template<class Val, class CharT, class Traits>
basic_istream_view<Val, CharT, Traits> istream_view(basic_istream<CharT, Traits>& s);

```

by

```

template<class Val>
using istream_view = basic_istream_view<Val, char>;
template<class Val>
using wistream_view = basic_istream_view<Val, wchar_t>;

namespace views { template<class T>
    inline constexpr unspecified istream = unspecified ; }

```

In 24.6.5.1 Overview [range.istream.overview]

insert after paragraph 1 before the example:

The name `views::istream<T>` denotes a customization point object (16.3.3.3.6). Given a type `T` and a subexpression `E` of type `U`, if `U` models `derived_from<basic_istream<typename U::char_type, typename U::traits_type>>`, then the expression `views::istream<T>(E)` is expression-equivalent to `basic_istream_view<T, typename U::char_type, typename U::traits_type>(E)`; otherwise, `views::istream<T>(E)` is ill-formed.

In 24.6.5.2 Class template `basic_istream_view` [range.istream.view]

strike:

```

template<class Val, class CharT, class Traits>
basic_istream_view<Val, CharT, Traits> istream_view(basic_istream<CharT, Traits>& s);
Effects: Equivalent to: return basic_istream_view<Val, CharT, Traits>(s);

```

Feature Test Macro

This should be a defect against C++20.

No feature test macro as `basic_istream_view` can be used with the old and new version.

Acknowledgements

Thanks to a lot of people who discussed the issue, proposed information and possible wording.
Especially: Barry Revzin, Tomasz Kamiński, Tim Song, Jonathan Wakely, Christopher Di Bella, Casey Carter.

Forgive me if I forgot anybody.