Subject: Re: 2022(?).2 beta
Posted by Lance on Mon, 19 Dec 2022 17:43:13 GMT
View Forum Message <> Reply to Message

mirek wrote on Mon, 19 December 2022 04:08Lance wrote on Mon, 19 December 2022 00:41Hello Mirek and Klugier:

The following code is an excerpt from /usr/include/c++/11/bits/stl_vector.h

```
      iterator
      begin() _GLIBCXX_NOEXCEPT
      { return iterator(this->_M_impl._M_start); }

      /**
       *  Returns a read-only (constant) iterator that points to the
       *  first element in the %vector.  Iteration is done in ordinary
       *  element order.
       */
      const_iterator
      begin() const _GLIBCXX_NOEXCEPT
      { return const_iterator(this->_M_impl._M_start); }

      /**
       *  Returns a read/write iterator that points one past the last
       *  element in the %vector.  Iteration is done in ordinary
       *  element order.
       */
      iterator
      end() _GLIBCXX_NOEXCEPT
      { return iterator(this->_M_impl._M_finish); }
```

Without looking into the definition of _GLIBCXX_NOEXCEPT, most experienced c++ users(, all participants of this thread for sure,) can tell that it will expand to noexcept when the -std version supports it and vanishes otherwise.

It might not be pleasant or pretty, but it certainly works and can be argued as the most reasonable solution in this particular situation.

It's a common problem that libraries with some history need to support different versions; maintaining backward compatibility should not mean stay backward. U++ necessarily has done similar thing for similar purposes, I believe.

Why is it so hard to swallow in this particular case?

BR,
Lance

Uhm, I guess we are presented with 2 more or less equivalently ugly options here. One of them requires a significant amount of work....

Hello Mirek and Klugier:

Another drawback for the disable-warnings option: Like Novo said, he and many similar-minded people are still using very old systems/compilers; people are different. What if in 4 years' horizon you decide that time has mature for switching to c++20 but there are still a significant number of users who wish to be able to have c++14 as an option?

From the point of smoother user experience, I am not quite sure if it's a good idea that the mainstream version today will become completely unusable the next day because it's upgraded. You likely need to deprecate it and keep it going for a couple of more years so people have time to transit to the new mainstream version/standard.

If this will be the case, we end up still need to be able to support both c++14 and c++20 at the same time for at least a period of time: trouble is deferred instead of solved. And as times goes, more [=] cases will be added to U++ (not in a significant number, but it's a non-decreasing function of time), chance is it will take more time and effort at the postponed switch date.

BR,
Lance