Document Number:	P4042R0
Date:	2026-03-18
Reply-to:	Matthias Kretz <m.kretz@gsi.de>
Audience:	LWG
Target:	C++26

Fix LWG4543: incorrect cast between simd::vec and simd::mask via conversion to and from impl-defined vector types

ABSTRACT

This paper resolves LWG4543 and includes a drive-by fix to [simd.mask.conv].

CONTENTS

1 CHANGELOG

(placeholder)

2 IMPLEMENTATION EXPERIENCE

This has been implemented and tested in the patch that's currently in review for libstdc++.

3 WORDING

In [simd.overview] add:

[simd.overview]

```
template<class U>
  constexpr explicit(see below) basic_vec(U&& value) noexcept;
template<class U, class UAbi>
  constexpr explicit(see below) basic_vec(const basic_vec<U, UAbi>&) noexcept;
template<class U, class UAbi>
  constexpr explicit(see below) basic_vec(const basic_vec<U, UAbi>&) noexcept = delete;
template<class G>
  constexpr explicit basic_vec(G&& gen);
```

In [simd.ctor] add:

[simd.ctor]

```
template<class U, class UAbi>
  constexpr explicit(see below) basic_vec(const basic_vec<U, UAbi>& x) noexcept;
```

5 *Constraints*:

- *simd-size-v*<U, UAbi> == size() is true, and
- U satisfies *explicitly-convertible-to*<T>.

6 *Effects*: Initializes the i^{th} element with static_cast<T>(x[i]) for all i in the range of [0, size()).

7 *Remarks:* The expression inside explicit evaluates to true if either

- the conversion from U to value_type is not value-preserving, or
- both U and value_type are integral types and the integer conversion rank ([conv.rank]) of U is greater than the integer conversion rank of value_type, or
- both U and value_type are floating-point types and the floating-point conversion rank ([conv.rank]) of U is greater than the floating-point conversion rank of value_type.

```
template<class U, class UAbi>
  constexpr explicit(see below) basic_vec(const basic_vec<U, UAbi>&) noexcept = delete;
```

-?- *Constraints*:

- *simd-size-v*<U, UAbi> == size() is false, or
- U does not satisfy *explicitly-convertible-to*<T>.

-?- *Remarks:* The expression inside explicit evaluates to *simd-size-v*<U, UAbi> == size().

In [simd.mask.overview] modify:

[simd.mask.overview]

```
constexpr explicit basic_mask(same_as<value_type> auto) noexcept;
template<size_t UBytes, class UAbi>
  constexpr explicit basic_mask(const basic_mask<UBytes, UAbi>&) noexcept;
template<size_t UBytes, class UAbi>
  constexpr explicit basic_mask(const basic_mask<UBytes, UAbi>&) noexcept = delete;
template<class U, class UAbi>
  constexpr explicit basic_mask(const basic_vec<U, UAbi>&) noexcept = delete;
template<class G>
  constexpr explicit basic_mask(G&& gen);
template<same_as<bitset<size()>> T>
  constexpr basic_mask(const T& b) noexcept;
template<unsigned_integral T>
  requires (!same_as<T, value_type>)
  constexpr explicit basic_mask(T val) noexcept;

// ([simd.mask.subscr]), basic_mask subscript operators
constexpr value_type operator[](simd-size-type) const;
template<simd-integral I>
  constexpr resize_t<I::size(), basic_mask> operator[](const I& indices) const;

// ([simd.mask.unary]), basic_mask unary operators
constexpr basic_mask operator!() const noexcept;
constexpr see below operator+() const noexcept;
constexpr see below operator-() const noexcept;
constexpr see below operator~() const noexcept;

// ([simd.mask.conv]), basic_mask conversions
template<class U, class AUAbi>
  constexpr explicit(sizeof(U) != Bytes) operator basic_vec<U, AUAbi>() const noexcept;
template<class U, class UAbi>
  constexpr operator basic_vec<U, UAbi>() const noexcept = delete;
constexpr bitset<size()> to_bitset() const noexcept;
constexpr unsigned long long to_ullong() const;
```

In [simd.mask.ctor] add:

[simd.mask.ctor]

```
template<size_t UBytes, class UAbi>
  constexpr explicit basic_mask(const basic_mask<UBytes, UAbi>& x) noexcept;
```

2 *Constraints*: `basic_mask<UBytes, UAbi>::size() == size()` is `true`.

3 *Effects*: Initializes the i^{th} element with `x[`i`]` for all i in the range of $[0, \texttt{size}())$.

```
template<size_t UBytes, class UAbi>
  constexpr explicit basic_mask(const basic_mask<UBytes, UAbi>& x) noexcept = delete;
```

-?- *Constraints*: `basic_mask<UBytes, UAbi>::size() == size()` is `false`.

```
template<class G> constexpr explicit basic_mask(G&& gen);
```

In [simd.mask.conv] add:

[simd.mask.conv]

```
template<class U, class AUAbi>
constexpr explicit(sizeof(U) != Bytes) operator basic_vec<U, AUAbi>() const noexcept;
```

1 *Constraints*: *simd-size-v*<U, AUAbi> == ~~*mask-size-v<Bytes, Abi>*~~`size() is true`.

2 *Returns:* A data-parallel object where the i^{th} element is initialized to `static_cast<U>(operator[](`i`))`.

```
template<class U, class UAbi>
  constexpr operator basic_vec<U, UAbi>() const noexcept = delete;
```

-?- *Constraints*: *simd-size-v*<U, UAbi> == size() is false.

```
constexpr bitset<size()> to_bitset() const noexcept;
```
