Ref:	*unspecified*
Date:	2023-09-23
Revises:	*unspecified*
Reply at:	Discussions, issues

mp-units Library Reference Documentations

Note: this is an early draft. It's known to be incomplet and incorrekt, and it has lots of bad formatting.

Contents

1 Scope [scope]

[1] This document describes the contents of the *mp-units library*.

2 References [refs]

1 The following documents are referred to in the text in such a way that some or all of their content constitutes requirements of this document. For dated references, only the edition cited applies. For undated references, the latest edition of the referenced document (including any amendments) applies.

(1.1) — The C++ Standards Committee. N4892: *Working Draft, Standard for Programming Language C++*. Edited by Thomas Köppe. Available from: https://wg21.link/N4892

(1.2) — The C++ Standards Committee. P1841R1: *Wording for Individually Specializable Numeric Traits*. Edited by Walter E. Brown. Available from: https://wg21.link/P1841R1

(1.3) — The C++ Standards Committee. SD-8: *Standard Library Compatibility*. Edited by Bryce Lelbach. Available from: https://wg21.link/SD8

(1.4) — ISO 80000-2:2019, *Quantities and units — Part 2: Mathematics*

(1.5) — IEC 60050 (all parts), *International Electrotechnical Vocabulary (IEV)*

(1.6) — IEC 60050-102:2007/AMD3:2021, *Amendment 3 — International Electrotechnical Vocabulary (IEV) — Part 102: Mathematics — General concepts and linear algebra*

(1.7) — IEC 60050-112:2010/AMD2:2020, *Amendment 2 — International Electrotechnical Vocabulary (IEV) — Part 112: Quantities and units*

2 N4892 is hereinafter called *C++*.

3 IEC 60050 is hereinafter called *IEV*.

3 Terms and definitions [defs]

1 For the purposes of this document, the terms and definitions given in C++, IEC 60050-102:2007/AMD3:2021, IEC 60050-112:2010/AMD2:2020, the terms, definitions, and symbols given in ISO 80000-2:2019, and the following apply.

2 ISO and IEC maintain terminology databases for use in standardization at the following addresses:

(2.1) — ISO Online browsing platform: available at https://www.iso.org/obp

(2.2) — IEC Electropedia: available at http://www.electropedia.org

3 Terms that are used only in a small portion of this document are defined where they are used and italicized where they are defined.

3.1 [def.mod]

modulo

operation performed on a set for which a division (IEV 102-01-21) and an addition (IEV 102-01-11) are defined, the result of which, for elements a and b of the set, is the unique element r, if it exists in the set, such that $a = \lfloor a/b \rfloor b + r$

4 Specification [spec]

4.1 External [spec.ext]

1 The specification of the mp-units library subsumes C++ [description], C++ [requirements], C++ [concepts.equality], and SD-8, all assumingly amended for the context of this library.

[*Note 1*: This means that, non exhaustively,

(1.1) — `::mp_units2` is a reserved namespace, and

(1.2) — `std::vector<mp_units::`*`type`*`>` is a program-defined specialization and a library-defined specialization from the point of view of C++ and this library, respectively.

— *end note*]

2 The mp-units library is not part of the C++ implementation.

4.2 Categories [spec.cats]

1 Detailed specifications for each of the components in the library are in Clause 5–Clause 6, as shown in Table 1.

Table 1: Library categories [tab:lib.cats]

Clause	Category
Clause 5	Numbers library
Clause 6	Quantities library

2 The numbers library (Clause 5) describes components for dealing with numbers.

3 The quantities library (Clause 6) describes components for dealing with quantities.

4.3 Headers [headers]

1 The mp-units library provides the *mp-units library headers*, shown in Table 2.

Table 2: mp-units library headers [tab:headers.mp.units]

`<mp-units/numbers.h>`
`<mp-units/quantity.h>`

4.4 Library-wide requirements [spec.reqs]

4.4.1 Reserved names [spec.res.names]

1 The mp-units library reserves macro names that start with `MP_UNITS`*digit-sequence*$_{opt}$`_`.

5 Numbers library [nums]

5.1 Summary [nums.summary]

1 This Clause describes components for dealing with numbers, as summarized in Table 3.

Table 3: Numbers library summary [tab:nums.summary]

	Subclause	Header
5.3	Traits	`<mp-units/numbers.h>`
5.4	Concepts	

5.2 Header `<mp-units/numbers.h>` synopsis [nums.syn]

```
namespace mp_units {

// 5.3, traits

// 5.3.2, number opt-ins
template<typename T>
struct enable_number;
template<typename T>
struct enable_complex_number;

template<typename T>
constexpr bool enable_number_v = enable_number<T>::value;
template<typename T>
constexpr bool enable_complex_number_v = enable_complex_number<T>::value;

// 5.3.3, identities
template<typename T>
struct number_zero;
template<typename T>
struct number_one;

template<typename T>
constexpr T number_zero_v = number_zero<T>::value;
template<typename T>
constexpr T number_one_v = number_one<T>::value;

// 5.3.4, associated types
template<typename T>
struct vector_scalar;

template<typename T>
using number_difference_t = decltype(std::declval<const T>() - std::declval<const T>());
template<typename T>
using vector_scalar_t = vector_scalar<T>::type;

// 5.4, concepts
template<typename T>
concept number = see below;
template<typename T, typename U>
concept common_number_with = see below;
template<typename T>
concept ordered_number = see below;
template<typename T>
concept number_line = see below;
```

```
template<typename T, typename U>
concept modulo_with = see below;
template<typename T, typename U>
concept compound_modulo_with = see below;
template<typename T, typename U>
concept modulus_for = see below;
template<typename T, typename U>
concept compound_modulus_for = see below;
template<typename T>
concept negative = see below;
template<typename T>
concept set_with_inverse = see below;
template<typename T, typename U>
concept point_space_for = see below;
template<typename T, typename U>
concept compound_point_space_for = see below;
template<typename T>
concept point_space = see below;
template<typename T, typename U>
concept vector_space_for = see below;
template<typename T, typename U>
concept compound_vector_space_for = see below;
template<typename T, typename U>
concept scalar_for = see below;
template<typename T, typename U>
concept field_for = see below;
template<typename T, typename U>
concept compound_scalar_for = see below;
template<typename T, typename U>
concept compound_field_for = see below;
template<typename T>
concept vector_space = see below;
template<typename T>
concept f_vector_space = see below;
template<typename T>
concept field_number = see below;
template<typename T>
concept field_number_line = see below;
template<typename T>
concept scalar_number = see below;

} // namespace mp_units
```

5.3 Traits [num.traits]

5.3.1 Requirements [num.traits.reqs]

1 Subclause 5.3 subsumes C++ [meta.rqmts], assumingly amended for its context. Pursuant to the subsumed C++ [namespace.std] (4.1), each class template specified in 5.3 may be specialized for any numeric type (C++ [numeric.requirements]).

2 Each template *`number-trait`* specified in 5.3 has a partial specialization of the form

```
template<class T>
struct number-trait<const T> : number-trait<T> { };
```

3 Each template *`number-trait`* specified in 5.3.2 is a *Cpp17UnaryTypeTrait* with a base characteristic of `std::bool_constant<B>`. *B* is a value consistent with *`number-trait`*'s specification.

4 Each template specified in 5.3.3 is a numeric distinguished value trait (P1841R3 [num.traits.val]), except that there are no declared specializations for arithmetic or volatile-qualified types.

5 A specialization of any template *`number-trait`* specified in 5.3.4 has no members other than a `type` member that names a type consistent with *`number-trait`*'s specification.

5.3.2 `number` opt-ins [num.opt.ins]

```
template<typename T>
struct enable_number : see below {};
```

1 *Value*: `true` if `T` represents a number, and `false` otherwise.

2 *Default value*: `true` if `vector_scalar_t<T>` is valid, and `false` otherwise.

```
template<class... T>
struct enable_number<std::chrono::time_point<T...>> : std::true_type {};
template<>
struct enable_number<std::chrono::day> : std::true_type {};
template<>
struct enable_number<std::chrono::month> : std::true_type {};
template<>
struct enable_number<std::chrono::year> : std::true_type {};
template<>
struct enable_number<std::chrono::weekday> : std::true_type {};
template<>
struct enable_number<std::chrono::year_month> : std::true_type {};
```

```
template<typename T>
struct enable_complex_number : std::false_type {};
```

3 *Value*: `true` if `T` represents a complex number (IEV 102-02-09), and `false` otherwise.

```
template<class T>
struct enable_complex_number<std::complex<T>> : std::true_type {};
```

5.3.3 Identities [num.ids]

```
template<typename T>
concept inferable-identities =
  common_number_with<T, number_difference_t<T>> && std::constructible_from<T, int>;
```

```
template<typename T>
struct number_zero { see below };
```

1 *Value*: `T`'s neutral element for addition (IEV 102-01-12), if any.

2 *Default value*: If `T` models *`inferable-identities`*, `T(0)`.

```
template<typename T>
struct number_one { see below };
```

3 *Value*: `T`'s neutral element for multiplication (IEV 102-01-19), if any.

4 *Default value*: If `T` models *`inferable-identities`*, `T(1)`.

5.3.4 Associated types [num.assoc.types]

```
template<typename T>
using number_difference_t = decltype(std::declval<const T>() - std::declval<const T>());
```

1 `number_difference_t<T>` represents `T`'s difference's (IEV 102-01-17) type, if any.

```
template<typename T>
struct vector_scalar {};
template<see below>
struct vector_scalar<see below> {
  using type = see below;
};
```

2 *Type*: Scalar (IEV 102-02-18) for the vector space (IEV 102-03-01) `T`.

3 Each row of Table 4 denotes a specialization.

5.4 Concepts [num.concepts]

```
template<typename T>
concept number = enable_number_v<T> && std::regular<T>;
```

Table 4: Specializations of `vector_scalar` [tab:num.scalar]

template-parameter-list	*template-argument-list*	*Type*
`std::integral T`	`T`	`T`
`std::floating_point T`	`T`	`T`
`class T`	`std::complex<T>`	`T`
`class T, class U`	`std::chrono::duration<T, U>`	`T`

```
template<typename T, typename U>
concept common_number_with =
  number<T> && number<U> && std::common_with<T, U> && number<std::common_type_t<T, U>>;

template<typename T>
concept ordered_number = number<T> && std::totally_ordered<T>;

template<class T>
concept number_line =
  ordered_number<T> &&
  requires(T& v) {
    number_one_v<number_difference_t<T>>;
    { ++v } -> std::same_as<T&>;
    { --v } -> std::same_as<T&>;
    { v++ } -> std::same_as<T>;
    { v-- } -> std::same_as<T>;
  };
```

1 Let `v` and `u` be equal objects of type `T`, and `one` be the value `number_one_v<number_difference_t<T>>`. `T` models `number_line` only if

(1.1) — The expressions `++v` and `v++` have the same domain (C++ [concepts.equality]).

(1.2) — The expressions `--v` and `v--` have the same domain.

(1.3) — If `v` is incrementable (C++ [iterator.concept.winc]), then both `++v` and `v++` add `one` to `v`, and the following expressions all equal `true`:

(1.3.1) — `v++ == u`,

(1.3.2) — `((void)v++, v) == ++u`, and

(1.3.3) — `std::addressof(++v) == std::addressof(v)`.

(1.4) — If `v` is decrementable (C++ [range.iota.view]), then both `--v` and `v--` subtract `one` from `v`, and the following expressions all equal `true`:

(1.4.1) — `v-- == u`,

(1.4.2) — `((void)v--, v) == --u`, and

(1.4.3) — `std::addressof(--v) == std::addressof(v)`.

```
template<class T, class U> concept addition-with =
  number<T> &&
  number<U> &&
  requires(const T& c, const U& d) {
    { c + d } -> common_number_with<T>;
    { d + c } -> common_number_with<T>;
  };

template<class T, class U> concept compound-addition-with =
  addition-with<T, U> &&
  requires(T& l, const U& d) {
    { l += d } -> std::same_as<T&>;
  };
```

```
template<class T, class U> concept subtraction-with =
  addition-with<T, U> &&
  requires(const T& c, const U& d) {
    { c - d } -> common_number_with<T>;
  };

template<class T, class U> concept compound-subtraction-with =
  subtraction-with<T, U> &&
  compound-addition-with<T, U> &&
  requires(T& l, const U& d) {
    { l -= d } -> std::same_as<T&>;
  };

template<class T, class U, class V> concept multiplication-with =
  number<T> &&
  number<U> &&
  number<V> &&
  requires(const T& c, const U& u) {
    { c * u } -> common_number_with<V>;
  };

template<class T, class U> concept compound-multiplication-with =
  multiplication-with<T, U, T> &&
  requires(T& l, const U& u) {
    { l *= u } -> std::same_as<T&>;
  };

template<class T, class U> concept division-with =
  multiplication-with<T, U, T> &&
  multiplication-with<U, T, T> &&
  requires(const T& c, const U& u) {
    { c / u } -> common_number_with<T>;
  };

template<class T, class U> concept compound-division-with =
  division-with<T, U> &&
  compound-multiplication-with<T, U> &&
  requires(T& l, const U& u) {
    { l /= u } -> std::same_as<T&>;
  };

template<class T, class U> concept modulo_with =
  number<T> &&
  number<U> &&
  requires(const T& c, const U& u) {
    { c % u } -> common_number_with<T>;
  };

template<class T, class U> concept compound_modulo_with =
  modulo_with<T, U> &&
  requires(T& l, const U& u) {
    { l %= u } -> std::same_as<T&>;
  };
```

2 Let `q` be an object of type `T`, and `r` be an object of type `U`.

3 For *`addition-with`* and *`compound-addition-with`*, let E be the expression `q + r` or `r + q`, and `q += r`, respectively, and let F be the addition (IEV 102-01-11) of the inputs to E.

4 For *`subtraction-with`* and *`compound-subtraction-with`*, let E be the expression `q - r` and `q -= r`, respectively, and let F be the subtraction (IEV 102-01-13) of the inputs to E.

5 For *`multiplication-with`* and *`compound-multiplication-with`*, let E be the expression `q * r` and `q *= r`, respectively, and let F be the multiplication (IEV 102-01-18) of the inputs to E.

6 For *division-with* and *compound-division-with*, let E be the expression `q / r` and `q /= r`, respectively, and let F be the division (IEV 102-01-21) of the inputs to E.

7 For `modulo_with` and `compound_modulo_with`, let E be the expression `q % r` and `q %= r`, respectively, and let F be the modulo (3.1) of the inputs to E.

8 `T` respectively models *addition-with*`<U>`, *compound-addition-with*`<U>`, *subtraction-with*`<U>`, *compound-subtraction-with*`<U>`, *multiplication-with*`<U, V>`, *compound-multiplication-with*`<U>`, *division-with*`<U>`, *compound-division-with*`<U>`, `modulo_with<U>`, and `compound_modulo_with<U>` only if, for each respective E, when the inputs to E are in the domain of E

(8.1) — E performs F in an unspecified set.

(8.2) — If the operator of E is an *assignment-operator*,

(8.2.1) — E maps the value of F to `q`, and the result of E is a reference to `q`, and

(8.2.2) — the result of E is the value of F mapped to the type of E otherwise.

```
template<typename T, typename U>
concept modulus_for = modulo_with<U, T>;
template<typename T, typename U>
concept compound_modulus_for = compound_modulo_with<U, T>;

template<class T> concept negative =
  compound-addition-with<T, T> &&
  requires(const T& c) {
    number_zero_v<T>;
    { -c } -> common_number_with<T>;
  };

template<class T> concept set_with_inverse =
  compound-multiplication-with<T, T> &&
  requires(const T& c) {
    { number_one_v<T> / c } -> std::common_with<T>;
  };
```

9 Let `v` be an object of type `T`.

10 For `negative`, let E be the expression `-v`, and let F be the negative (IEV 102-01-14) of `v`.

11 For `set_with_inverse`, let E be the expression `number_one_v<T> / v`, and let F be the inverse (IEV 102-01-24) of `v`.

12 `T` respectively models `negative<U>` and `set_with_inverse<U>` only if, for the respective E, when `v` is in the domain of E

(12.1) — E performs F in an unspecified set.

(12.2) — The result of E is the value of F mapped to the type of E.

```
template<typename T, typename U>
concept point_space_for =
  subtraction-with<T, U> && negative<U> && common_number_with<number_difference_t<T>, U>;
template<typename T, typename U>
concept compound_point_space_for = point_space_for<T, U> && compound-subtraction-with<T, U>;
template<typename T>
concept point_space = compound_point_space_for<T, number_difference_t<T>>;
```

13 [*Note 1*: The `point_space` concept is modeled by types that behave similarly to `std::chrono::sys_seconds`. — *end note*]

```
template<typename T, typename U>
concept vector_space_for = point_space_for<U, T>;
template<typename T, typename U>
concept compound_vector_space_for = compound_point_space_for<U, T>;

template<typename T>
concept weak-scalar =
  common_number_with<T, number_difference_t<T>> && point_space<T> && negative<T>;
```

```
template<typename T, typename U>
concept scales-with = common_number_with<U, vector_scalar_t<T>> && weak-scalar<U> &&
                      multiplication-with<T, U, T> && set_with_inverse<U>;
template<typename T, typename U>
concept compound-scales-with = scales-with<T, U> && compound-multiplication-with<T, U>;

template<typename T, typename U>
concept scalar_for = scales-with<U, T>;
template<typename T, typename U>
concept field_for = scalar_for<T, U> && division-with<U, T>;
template<typename T, typename U>
concept compound_scalar_for = compound-scales-with<U, T>;
template<typename T, typename U>
concept compound_field_for = compound_scalar_for<T, U> && compound-division-with<U, T>;

template<typename T>
concept vector_space = point_space<T> && compound-scales-with<T, vector_scalar_t<T>>;
template<typename T>
concept f_vector_space = vector_space<T> && compound-division-with<T, vector_scalar_t<T>>;
```

14 [*Note 2*: The `f_vector_space` concept is modeled by types that behave similarly to `std::chrono::seconds`. — *end note*]

```
template<typename T>
concept field_number = f_vector_space<T> && compound-scales-with<T, T>;

template<typename T>
concept field_number_line = field_number<T> && number_line<T>;

template<typename T>
concept scalar_number = field_number<T> && (field_number_line<T> || enable_complex_number_v<T>);
```

15 [*Note 3*: The `field_number` concept is modeled by types that behave similarly to `std::complex<double>`. It represents an approximation of a field, see IEV 102-02-18, Note 2 to entry. — *end note*]

16 [*Note 4*: The `field_number_line` concept is modeled by types that behave similarly to `double`. — *end note*]

17 [*Note 5*: `scalar_number` represents an approximation of a scalar number (IEV 102-02-18). — *end note*]

6 Quantities library [qties]

6.1 Summary [qties.summary]

1 This Clause describes components for dealing with quantities, as summarized in Table 5.

Table 5: Quantities library summary [tab:qties.summary]

	Subclause	Module
6.3	Concepts	`<mp-units/quantity.h>`

6.2 Header `<mp-units/quantity.h>` synopsis [qty.syn]

```
#include <mp-units/numbers.h>  // see 5.2

namespace mp_units {

// 6.3, concepts
template<class>
concept scalar_quantity = see below;

template<Quantity Q>
struct number_scalar<Q> : number_scalar<typename Q::rep> {};

}  // namespace mp_units
```

6.3 Concepts [qty.concepts]

```
template<class T>
concept scalar_quantity = Quantity<T> && scalar_number<typename T::rep>;
```

1 [*Note 1*: The `scalar_quantity` concept represents an approximation of a scalar quantity (IEV 102-02-19). — *end note*]

Cross references

Each clause and subclause label is listed below along with the corresponding clause or subclause number and page number, in alphabetical order by label.

Index

Index of library headers

The bold page number for each entry refers to the page where the synopsis of the header is shown.

Index of library names

Index of library concepts

The bold page number for each entry is the page where the concept is defined. Other page numbers refer to pages where the concept is mentioned in the general text.