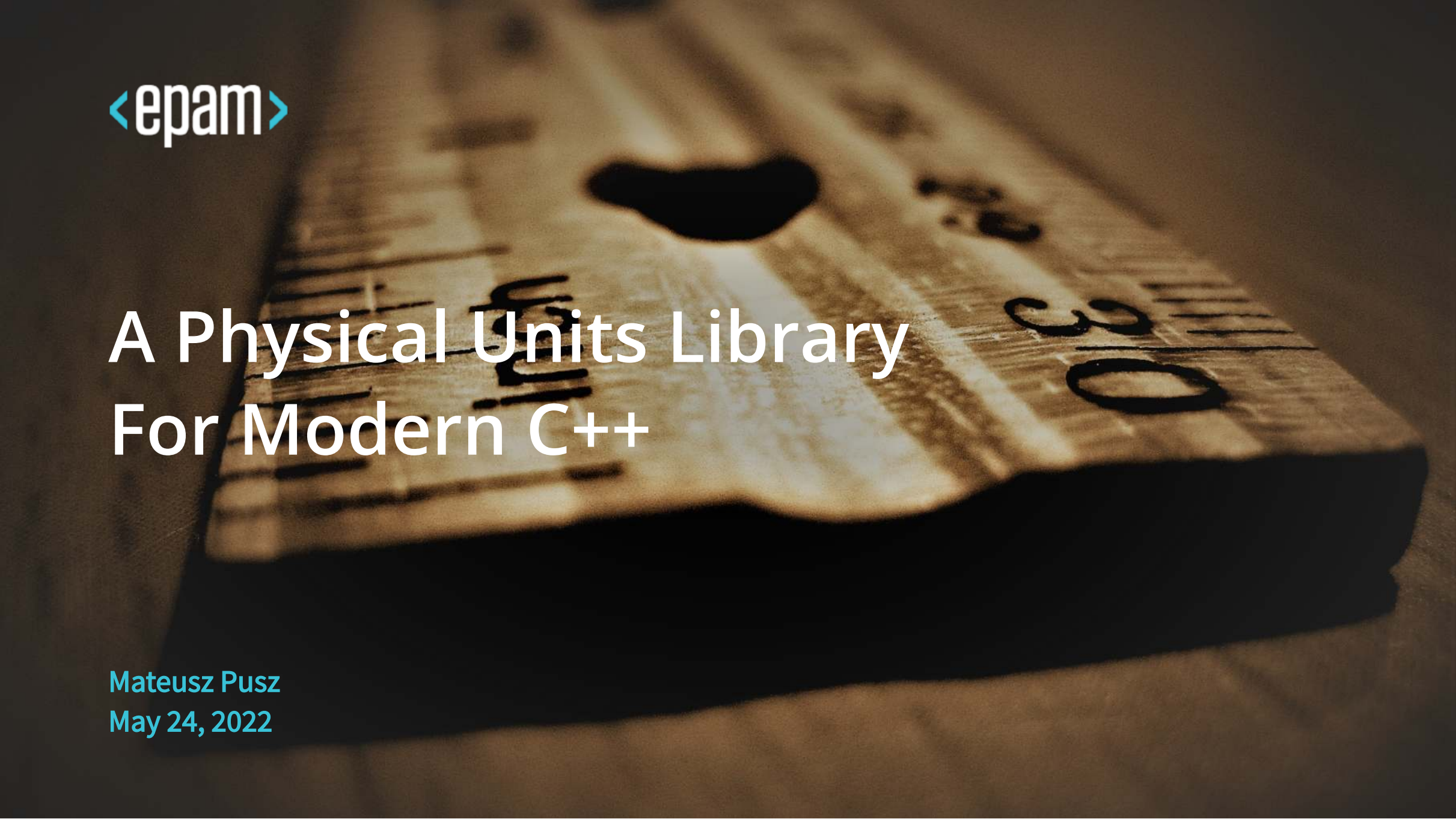


A Physical Units Library For Modern C++

Mateusz Pusz

May 24, 2022

Today I will not talk about

- Rationale
- Toy examples
- Competition
- Performance
- The downcasting facility

Today I will not talk about

- Rationale
- Toy examples
- Competition
- Performance
- The downcasting facility

There are plenty of my talks on the above subject on YouTube already :-)

Agenda

1. Trying `mp-units` library
2. Quick Intro
3. Framework Basics
4. Goodiebox
5. Environment, compatibility, next steps

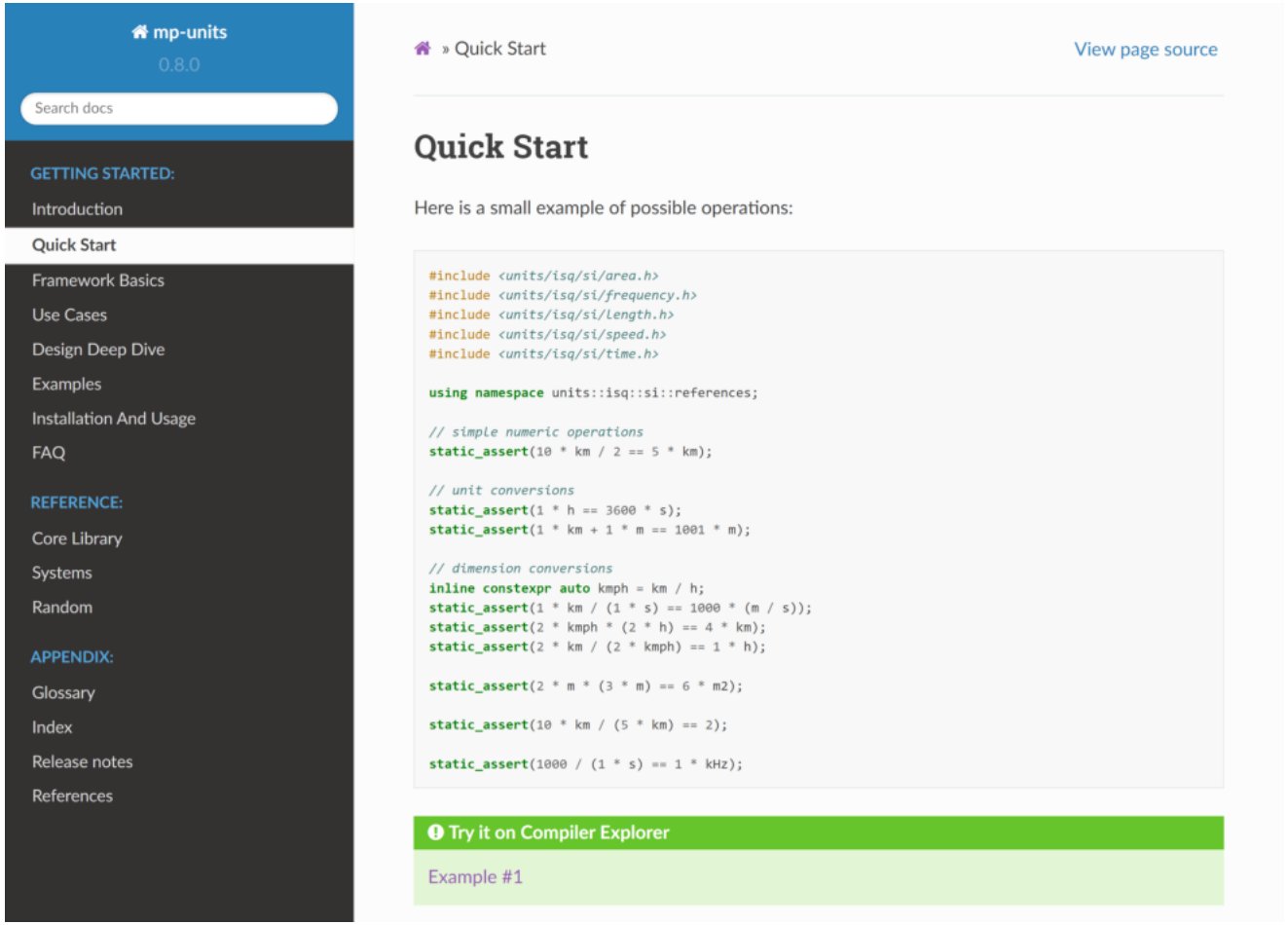
Agenda

1. Trying `mp-units` library
2. Quick Intro
3. Framework Basics
4. Goodiebox
5. Environment, compatibility, next steps

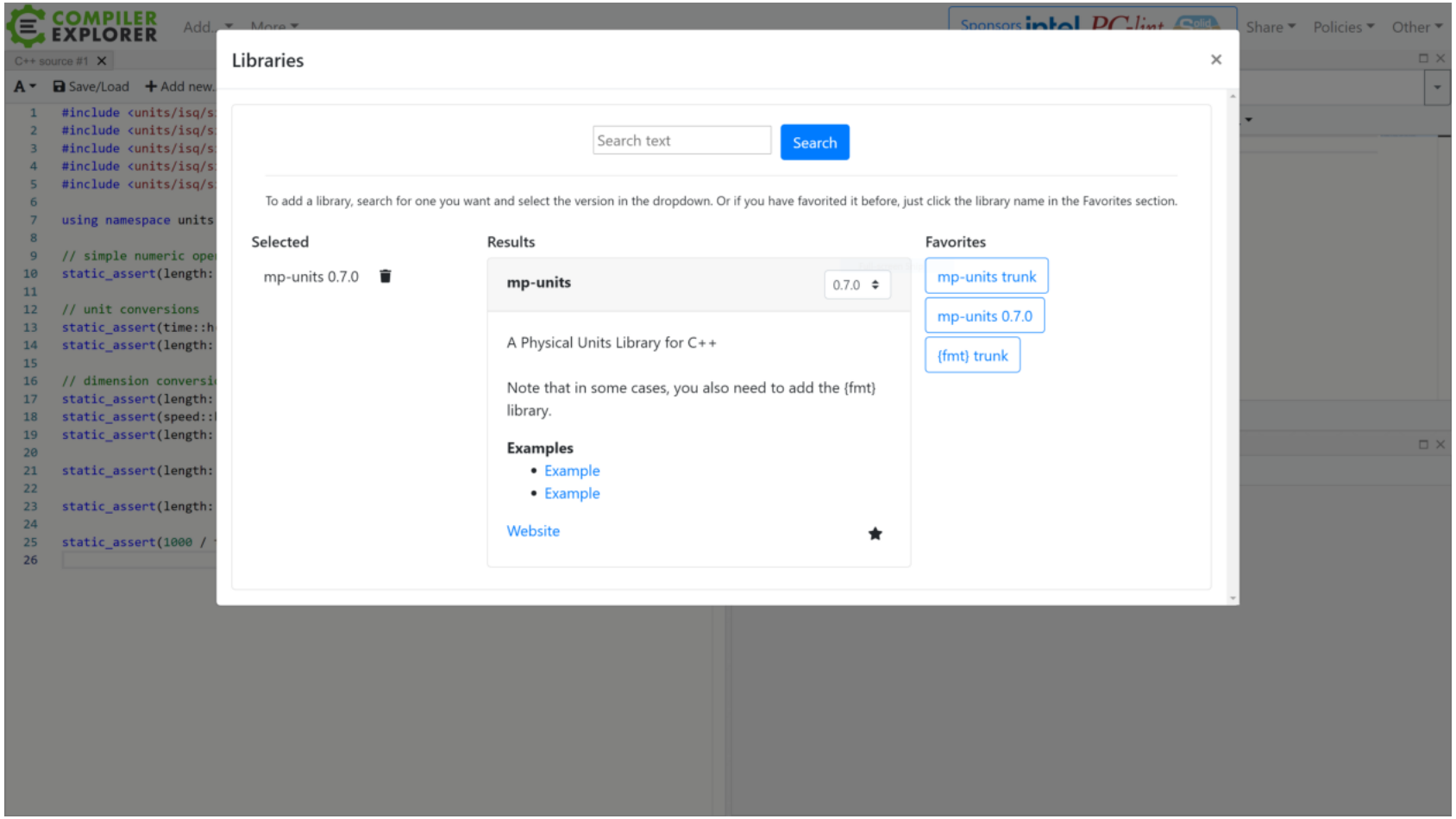
In Q&A please refer to the slide number.

TRYING mp-units LIBRARY

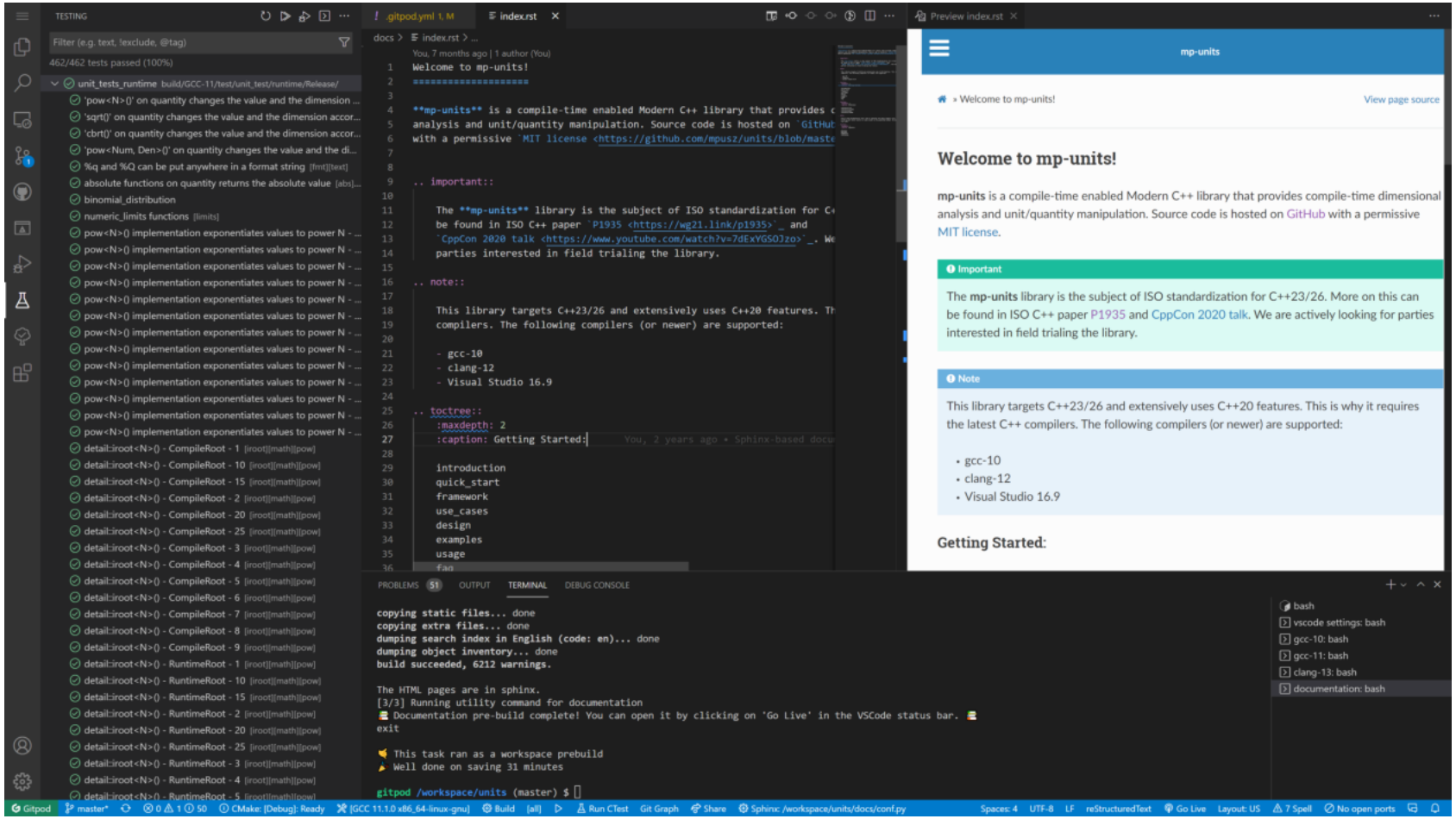
mp-units Documentation (mpusz.github.io/units)



mp-units on the Compiler Explorer



mp-units in GitPod



mp-units in Conan

CONAN CENTER (OFFICIAL RELEASES)

```
[requires]
mp-units/0.7.0

[generators]
CMakeToolchain
CMakeDeps
```

```
conan install .. -pr <your_conan_profile> -s compiler.cppstd=20 -b=missing
cmake .. --toolchain=conan_toolchain.cmake
cmake --build . --config Release
```

mp-units in Conan

LIVE AT HEAD

```
[requires]
mp-units/0.8.0@mpusz/testing

[generators]
CMakeToolchain
CMakeDeps
```

```
conan remote add conan-mpusz https://mpusz.jfrog.io/artifactory/api/conan/conan-oss
```

```
conan install .. -pr <your_conan_profile> -s compiler.cppstd=20 -b=outdated -u
cmake .. --toolchain=conan_toolchain.cmake
cmake --build . --config Release
```

QUICK INTRO

Dimensional Analysis

Power = Energy / Time

Dimensional Analysis

Power = Energy / Time

- `W`

Dimensional Analysis

Power = Energy / Time

- `W`
- `J/s`

Dimensional Analysis

Power = Energy / Time

- W
- J/s
- $N \cdot m/s$
- $kg \cdot m \cdot s^{-2} \cdot m/s$
- $kg \cdot m^2 \cdot s^{-2}/s$
- $kg \cdot m^2 \cdot s^{-3}$

Dimensional Analysis: Unit Aliases (godbolt.org/z/1sdz5Go8P)

```
// simple numeric operations
static_assert(km{10} / 2 == km{5});
```

Dimensional Analysis: Unit Aliases (godbolt.org/z/1sdz5Go8P)

```
// simple numeric operations
static_assert(km{10} / 2 == km{5});
```

```
// unit conversions
static_assert(h{1} == s{3600});
static_assert(km{1} + m{1} == m{1001});
```

Dimensional Analysis: Unit Aliases (godbolt.org/z/1sdz5Go8P)

```
// simple numeric operations
static_assert(km{10} / 2 == km{5});
```

```
// unit conversions
static_assert(h{1} == s{3600});
static_assert(km{1} + m{1} == m{1001});
```

```
// dimension conversions
static_assert(km{1} / s{1} == m_per_s{1000});
static_assert(km_per_h{2} * h{2} == km{4});
static_assert(km{2} / km_per_h{2} == h{1});

static_assert(m{2} * m{3} == m2{6});

static_assert(km{10} / km{5} == 2);

static_assert(1000 / s{1} == kHz{1});
```

Dimensional Analysis: Unit Aliases (godbolt.org/z/nzGd1a1h6)

```
// simple numeric operations
static_assert(length::km{10} / 2 == length::km{5});
```

```
// unit conversions
static_assert(time::h{1} == time::s{3600});
static_assert(length::km{1} + length::m{1} == length::m{1001});
```

```
// dimension conversions
static_assert(length::km{1} / time::s{1} == speed::m_per_s{1000});
static_assert(speed::km_per_h{2} * time::h{2} == length::km{4});
static_assert(length::km{2} / speed::km_per_h{2} == time::h{1});

static_assert(length::m{2} * length::m{3} == area::m2{6});

static_assert(length::km{10} / length::km{5} == 2);

static_assert(1000 / time::s{1} == frequency::kHz{1});
```

Dimensional Analysis: UDLs (godbolt.org/z/G77Eo936r)

```
// simple numeric operations
static_assert(10_q_km / 2 == 5_q_km);
```

```
// unit conversions
static_assert(1_q_h == 3600_q_s);
static_assert(1_q_km + 1_q_m == 1001_q_m);
```

```
// dimension conversions
static_assert(1_q_km / 1_q_s == 1000_q_m_per_s);
static_assert(2_q_km_per_h * 2_q_h == 4_q_km);
static_assert(2_q_km / 2_q_km_per_h == 1_q_h);

static_assert(2_q_m * 3_q_m == 6_q_m2);

static_assert(10_q_km / 5_q_km == 2);

static_assert(1000 / 1_q_s == 1_q_kHz);
```

Dimensional Analysis: References (godbolt.org/z/EPqTc57jx)

```
// simple numeric operations
static_assert(10 * km / 2 == 5 * km);
```

```
// unit conversions
static_assert(1 * h == 3600 * s);
static_assert(1 * km + 1 * m == 1001 * m);
```

```
// dimension conversions
inline constexpr auto kmph = km / h;
static_assert(1 * km / (1 * s) == 1000 * (m / s));
static_assert(2 * kmph * (2 * h) == 4 * km);
static_assert(2 * km / (2 * kmph) == 1 * h);

static_assert(2 * m * (3 * m) == 6 * m2);

static_assert(10 * km / (5 * km) == 2);

static_assert(1000 / (1 * s) == 1 * kHz);
```

The SI Brochure

The SI is a consistent system of units for use in all **aspects of life**, including international trade, manufacturing, security, health and safety, protection of the environment, and in the basic science that underpins all of these.

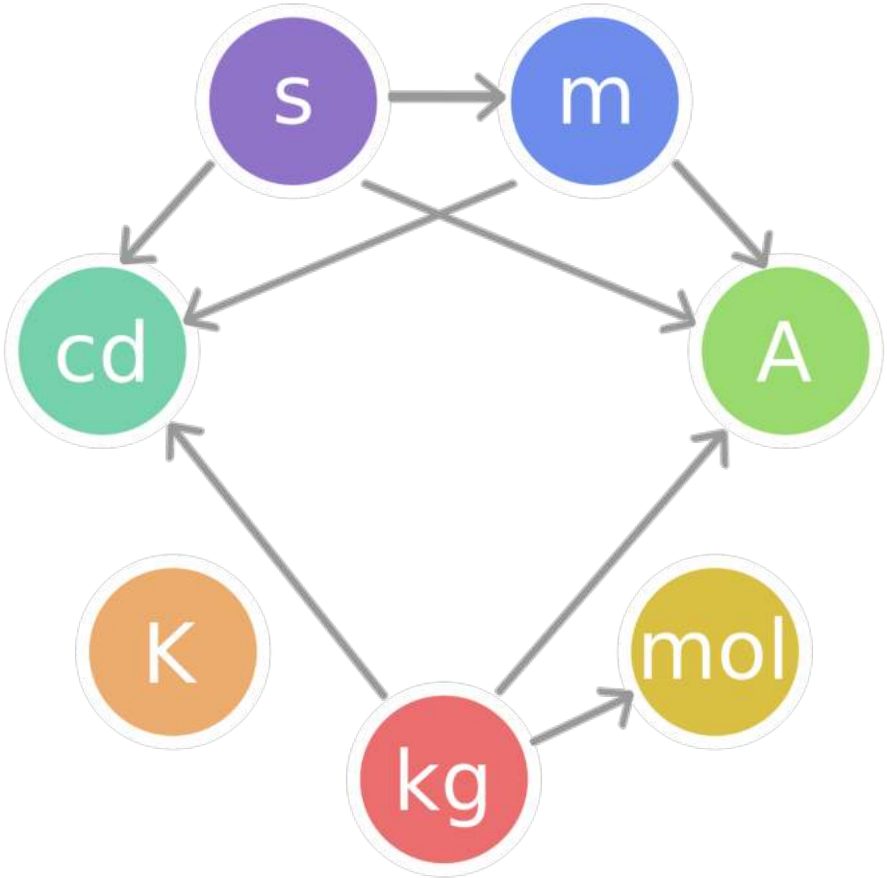
The SI Brochure

The SI is a consistent system of units for use in all **aspects of life**, including international trade, manufacturing, security, health and safety, protection of the environment, and in the basic science that underpins all of these.

The system of quantities underlying the SI and the equations relating them are based on the present description of nature and **are familiar to all scientists, technologists and engineers**.

International System of Units (SI)

- 7 base units
- 22 named units
- 20 prefixes to the unit names and unit symbols



SI Base Units

Historically, SI units have been presented in terms of a set of seven base units.

SI Base Units

Historically, SI units have been presented in terms of a set of seven base units.

All other units, described as derived units, are constructed as products of powers of the base units.

SI Base Units

QUANTITY	DIMENSION SYMBOL	UNIT SYMBOL	UNIT NAME
time	`T`	`s`	second
length	`L`	`m`	metre
mass	`M`	`kg`	kilogram
electric current	`I`	`A`	ampere
thermodynamic temperature	`Θ`	`K`	kelvin
amount of substance	`N`	`mol`	mole
luminous intensity	`J`	`cd`	candela

Examples of SI derived quantities expressed in the SI base units

QUANTITY	DIMENSION SYMBOL	UNIT SYMBOL	UNIT NAME
area	A	m^2	square metre
volume	V	m^3	cubic metre
velocity	v	$\mathrm{m \cdot s^{-1}}$	metre per second
acceleration	a	$\mathrm{m \cdot s^{-2}}$	metre per second squared
density	ρ	$\mathrm{kg \cdot m^{-3}}$	kilogram per cubic metre
magnetic field strength	H	$\mathrm{A \cdot m^{-1}}$	ampere per metre
luminance	Lv	$\mathrm{cd \cdot m^{-2}}$	candela per square metre

Examples of SI derived units with special name

QUANTITY	UNIT SYMBOL	UNIT NAME	IN OTHER SI UNITS	IN SI BASE UNITS
frequency	Hz	hertz	---	s^{-1}
force	N	newton	---	$kg \cdot m \cdot s^{-2}$
pressure	Pa	pascal	N/m^2	$kg \cdot m^{-1} \cdot s^{-2}$
energy	J	joule	$N \cdot m$	$kg \cdot m^2 \cdot s^{-2}$
power	W	watt	J/s	$kg \cdot m^2 \cdot s^{-3}$
electric charge	C	coulomb	---	$s \cdot A$
voltage	V	volt	W/A	$kg \cdot m^2 \cdot s^{-3} \cdot A^{-1}$
capacitance	F	farad	C/V	$kg^{-1} \cdot m^{-2} \cdot s^4 \cdot A^2$
resistance	Ω	ohm	V/A	$kg \cdot m^2 \cdot s^{-3} \cdot A^{-2}$

Quantity (ISO 80000)

Property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed by means of a number and a reference.

Quantity (ISO 80000)

Property of a phenomenon, body, or substance, where the property has a magnitude that can be expressed by means of a number and a reference.

A reference can be a measurement unit, a measurement procedure, a reference material, or a combination of such.

Examples of quantities

- 83 kilograms of mass
- 5 days of time
- 40'000 kilometres of length
- 9.80665 metres per squared second of acceleration
- ...

Units

Real scalar quantity, *defined and adopted by convention*, with which any **other quantity of the same kind can be compared to express the ratio** of the second quantity to the first one as a number.

Units

Real scalar quantity, *defined and adopted by convention*, with which any **other quantity of the same kind can be compared to express the ratio** of the second quantity to the first one as a number.

Measurement units are designated by conventionally assigned names and symbols.

Examples of units and their symbols

BASE UNITS

- metre (**m**)
- second (**s**)

SCALED UNITS

- hour (**h**)

PREFIXED UNITS

- kilometre (**km**)

Examples of units and their symbols

BASE UNITS

- metre (**m**)
- second (**s**)

SCALED UNITS

- hour (**h**)

PREFIXED UNITS

- kilometre (**km**)

COHERENT DERIVED UNITS

- metre per second (**m/s** or **m s⁻¹**)

SCALED DERIVED UNITS

- kilometre per hour (**km/h** or **km h⁻¹**)

DERIVED NAMED UNITS

- watt (**W**)

Examples of units and their symbols

BASE UNITS

- metre (**m**)
- second (**s**)

SCALED UNITS

- hour (**h**)

PREFIXED UNITS

- kilometre (**km**)

COHERENT DERIVED UNITS

- metre per second (**m/s** or **m s⁻¹**)

SCALED DERIVED UNITS

- kilometre per hour (**km/h** or **km h⁻¹**)

DERIVED NAMED UNITS

- watt (**W**)

Base units and coherent derived units form a set of coherent units.

`mp-units` unit types

```
template<typename Child, basic_symbol_text Symbol>
struct named_unit;
```

EXAMPLES

```
struct metre : named_unit<metre, "m"> {};     // base unit
struct second : named_unit<second, "s"> {};   // base unit
struct watt : named_unit<watt, "W"> {};       // derived unit
```

`mp-units` unit types

```
template<typename Child, basic_symbol_text Symbol, ratio R, Unit U>
  requires UnitRatio<R>
struct named_scaled_unit;
```

EXAMPLES

```
struct minute : named_scaled_unit<minute, "min", ratio(60), second> {};
struct hour : named_scaled_unit<hour, "h", ratio(60), minute> {};
struct day : named_scaled_unit<day, "d", ratio(24), hour> {};
```

`mp-units` unit types

```
template<typename Child, Prefix P, NamedUnit U>
  requires detail::can_be_prefixed<U>
struct prefixed_unit;
```

EXAMPLES

```
struct millisecond : prefixed_unit<millisecond, milli, second> {};
struct kilometre : prefixed_unit<kilometre, kilo, metre> {};
```

`mp-units` unit types

```
template<typename Child>
struct derived_unit;
```

EXAMPLES

```
struct metre_per_second : derived_unit<metre_per_second> {};
struct cubic_metre : derived_unit<cubic_metre> {};
```

`mp-units` unit types

```
template<typename Child, DerivedDimension Dim, NamedUnit U, NamedUnit... URest>
  requires detail::compatible_units<typename Dim::recipe, U, URest...>
struct derived_scaled_unit;
```

EXAMPLES

```
struct kilometre_per_hour : derived_scaled_unit<kilometre_per_hour, dim_speed,
                                                kilometre, hour> {};
struct cubic_centimetre : derived_scaled_unit<cubic_centimetre, dim_volume,
                                              centimetre> {};
```

SI prefixes

```
namespace units::isq::si {

struct yocto  : prefix<yocto, "y",  ratio(1, 1, -24)> {};
struct zepto  : prefix<zepto, "z",  ratio(1, 1, -21)> {};
struct atto   : prefix<atto,  "a",  ratio(1, 1, -18)> {};
struct femto  : prefix<femto, "f",  ratio(1, 1, -15)> {};
struct pico   : prefix<pico,  "p",  ratio(1, 1, -12)> {};
struct nano   : prefix<nano,  "n",  ratio(1, 1,  -9)> {};
struct micro  : prefix<micro, basic_symbol_text{"\u00b5", "u"}, ratio(1, 1,  -6)> {};
struct milli  : prefix<milli, "m",  ratio(1, 1,  -3)> {};
struct centi  : prefix<centi, "c",  ratio(1, 1,  -2)> {};
struct deci   : prefix<deci,  "d",  ratio(1, 1,  -1)> {};
struct deca   : prefix<deca,  "da", ratio(1, 1,   1)> {};
struct hecto  : prefix<hecto, "h",  ratio(1, 1,   2)> {};
struct kilo   : prefix<kilo,  "k",  ratio(1, 1,   3)> {};
struct mega   : prefix<mega,  "M",  ratio(1, 1,   6)> {};
struct giga   : prefix<giga,  "G",  ratio(1, 1,   9)> {};
struct tera   : prefix<tera,  "T",  ratio(1, 1,  12)> {};
struct peta   : prefix<peta,  "P",  ratio(1, 1,  15)> {};
struct exa    : prefix<exa,   "E",  ratio(1, 1,  18)> {};
struct zetta  : prefix<zetta, "Z",  ratio(1, 1,  21)> {};
struct yotta  : prefix<yotta, "Y",  ratio(1, 1,  24)> {};

}
```

Units are not enough

Each physical **quantity has only one coherent SI unit**. The converse, however, is not true, because in general **several different quantities may share the same SI unit**.

-- SI Brochure

Units are not enough

Each physical **quantity has only one coherent SI unit**. The converse, however, is not true, because in general **several different quantities may share the same SI unit**.

-- SI Brochure

It is therefore important **not to use the unit alone to specify the quantity**.

quantity class template

```
template<Dimension D, UnitOf<D> U, Representation Rep = double>
class quantity {
public:
  using dimension = D;
  using unit = U;
  using rep = Rep;

  static constexpr units::reference<dimension, unit> reference{};

  [[nodiscard]] constexpr rep& number() & noexcept { return number_; }
  [[nodiscard]] constexpr const rep& number() const& noexcept { return number_; }
  [[nodiscard]] constexpr rep&& number() && noexcept { return std::move(number_); }
  [[nodiscard]] constexpr const rep&& number() const&& noexcept { return std::move(number_); }

  // ...
};
```

`quantity` class template (with deducing `this` in C++23)

```
template<Dimension D, UnitOf<D> U, Representation Rep = double>
class quantity {
public:
  using dimension = D;
  using unit = U;
  using rep = Rep;

  static constexpr units::reference<dimension, unit> reference{};

  template<typename Self>
  [[nodiscard]] constexpr auto&& number(this Self&& self) noexcept
  {
    return std::forward<Self>(self).number_;
  }

  // ...
};
```

Dimension

> Physical quantities can be organized in a system of dimensions, where the system used is decided by convention.
>
> *-- SI Brochure*

Dimension

> Physical quantities can be organized in a system of dimensions, where the system used is decided by convention.
>
> *-- SI Brochure*

BASE DIMENSION

- Each of the **seven base quantities** used in the SI is regarded as **having its own dimension**

Dimension

> Physical quantities can be organized in a system of dimensions, where the system used is decided by convention.
>
> *-- SI Brochure*

BASE DIMENSION

- Each of the **seven base quantities** used in the SI is regarded as **having its own dimension**

DERIVED DIMENSIONS

- The dimensions of the derived quantities are **written as products of powers of the dimensions of the base quantities** using the equations that relate the derived quantities to the base quantities

`mp-units` dimension types

```
template<basic_fixed_string Symbol, NamedUnit U>
struct base_dimension {
  static constexpr auto symbol = Symbol;
  using base_unit = U;
};
```

EXAMPLES

```
struct dim_length : base_dimension<"L", metre> {};
struct dim_time : base_dimension<"T", second> {};
```

`mp-units` dimension types

```
template<typename Child, Unit U, Exponent... Es>
struct derived_dimension {
  using coherent_unit = U;
  using recipe = exponent_list<Es...>;
};
```

EXAMPLES

```
struct dim_speed : derived_dimension<dim_speed, metre_per_second,
                                     exponent<dim_length, 1>, exponent<dim_time, -1>> {};
struct dim_power : derived_dimension<dim_power, watt,
                                     exponent<dim_energy, 1>, exponent<dim_time, -1>> {};
```

International System of Quantities (ISQ)

- System of quantities based on the seven base quantities
  - length
  - mass
  - time
  - electric current
  - thermodynamic temperature
  - amount of substance
  - luminous intensity
- Describes relations between quantities of various dimensions without prescribing specific units

International System of Quantities (ISQ)

- System of quantities based on the seven base quantities
  - length
  - mass
  - time
  - electric current
  - thermodynamic temperature
  - amount of substance
  - luminous intensity
- Describes relations between quantities of various dimensions without prescribing specific units

The International System of Units (SI) is based on the ISQ.

SI 2018

The definition of the SI units is established in terms of a set of seven defining constants. The complete system of units can be derived from the fixed values of these defining constants, expressed in the units of the SI.

The seven defining constants of the SI

DEFINING CONSTANT	SYMBOL	NUMERICAL VALUE	UNIT
hyperfine transition frequency of Cs	$\Delta\nu_{Cs}$	9 192 631 770	Hz
speed of light in vacuum	c	299 792 458	$\mathrm{m\ s^{-1}}$
Planck constant	h	$6.626\,070\,15 \times 10^{-34}$	J s
elementary charge	e	$1.602\,176\,634 \times 10^{-19}$	C
Boltzmann constant	k	$1.380\,649 \times 10^{-23}$	$\mathrm{J\ K^{-1}}$
Avogadro constant	N_A	$6.022\,140\,76 \times 10^{23}$	$\mathrm{mol^{-1}}$
luminous efficacy	K_{cd}	683	$\mathrm{lm\ W^{-1}}$

The seven defining constants of the SI in `mp-units`

```
template<Representation Rep = double>
inline constexpr auto hyperfine_structure_transition_frequency = frequency<hertz, Rep>(9'192'631'770);

template<Representation Rep = double>
inline constexpr auto speed_of_light = speed<metre_per_second, Rep>(299'792'458);

template<Representation Rep = double>
inline constexpr auto planck_constant = energy<joule, Rep>(6.62607015e-34) * time<second, Rep>(1);

template<Representation Rep = double>
inline constexpr auto elementary_charge = electric_charge<coulomb, Rep>(1.602176634e-19);

template<Representation Rep = double>
inline constexpr auto boltzmann_constant = energy<joule, Rep>(1.380649e-23) / thermodynamic_temperature<kelvin, Rep>(1);

template<Representation Rep = double>
inline constexpr auto avogadro_constant = Rep(6.02214076e23) / amount_of_substance<mole, Rep>(1);

template<Representation Rep = double>
inline constexpr auto luminous_efficacy = luminous_flux<lumen, Rep>(683) / power<watt, Rep>(1);
```

Endless possibilities

> Since the number of quantities is without limit, it is not possible to provide a complete list of derived quantities and derived units.
>
> *-- SI Brochure*

Endless possibilities

> Since the number of quantities is without limit, it is not possible to provide a complete list of derived quantities and derived units.
>
> *-- SI Brochure*

Physical units library has to be easy to extend by users!

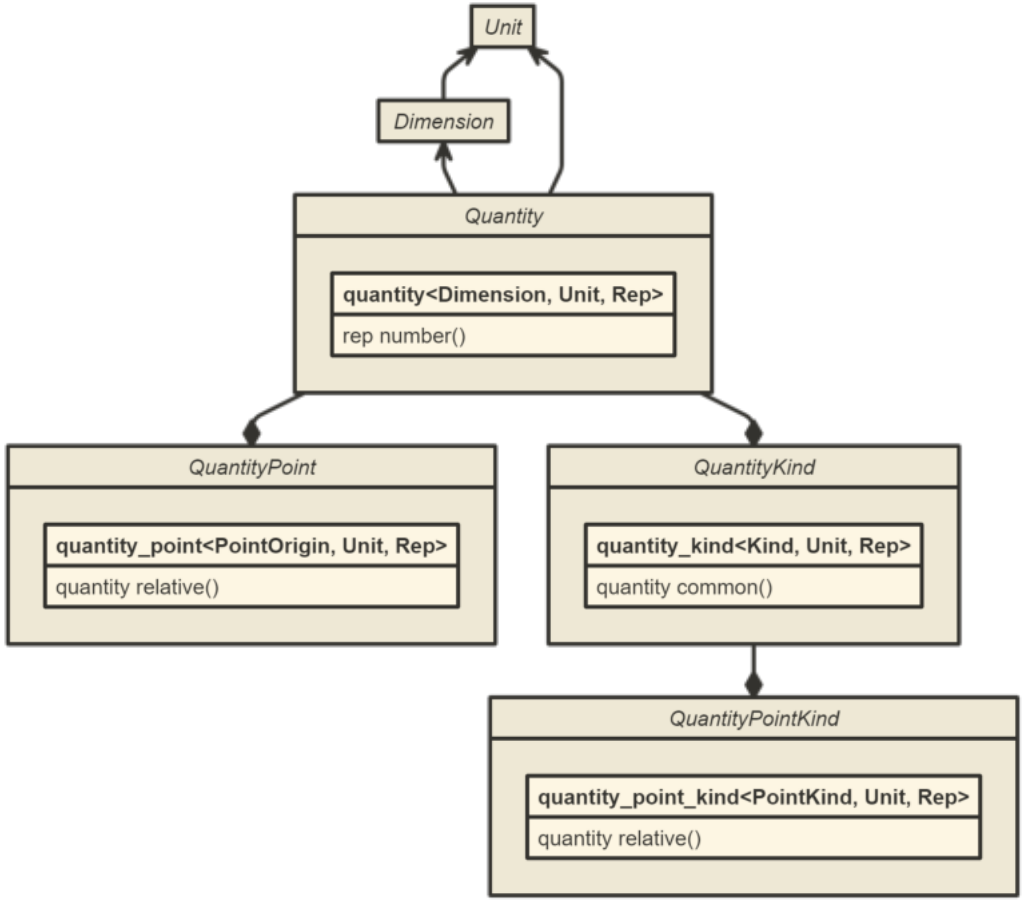
More than one system of measurement

- United States Customary Units
- Imperial units
- CGS
- FPS
- Natural Units
- ...

Unit	Divisions	SI Equivalent
Exact relationships shown in **boldface**		
International		
1 *point* (p)		352.777 778 µm
1 *pica* (P/)	**12 p**	4.233 333 mm
1 *inch* (in or ")	**6 P/**	**25.4 mm**
1 *foot* (ft or ')	**12 in**	**0.304 8 m**[9]
1 *yard* (yd)	**3 ft**	**0.914 4 m**[9]
1 *mile* (mi)	**5 280 ft** or **1 760 yd**	**1.609 344 km**
US Survey		
1 link (li)	$^{33}/_{50}$ **ft** or 7.92 in	0.201 116 8 m
1 (survey) foot (ft)	$^{1200}/_{3937}$ **m**	0.304 800 61 m[9]
1 rod (rd)	**25 li** or **16.5 ft**	5.029 21 m
1 chain (ch)	**4 rd** or **66 ft**	20.116 84 m
1 furlong (fur)	**10 ch**	201.168 4 m
1 survey (or statute) mile (mi)	**8 fur**	1.609 347 km[9]
1 league (lea)	**3 mi**	4.828 042 km
International Nautical[9]		
1 fathom (ftm)	**2 yd**	**1.828 8 m**
1 cable (cb)	**120 ftm** or 1.091 fur	**219.456 m**
1 nautical mile (NM or nmi)	8.439 cb or 1.151 mi	**1.852 km**

FRAMEWORK BASICS

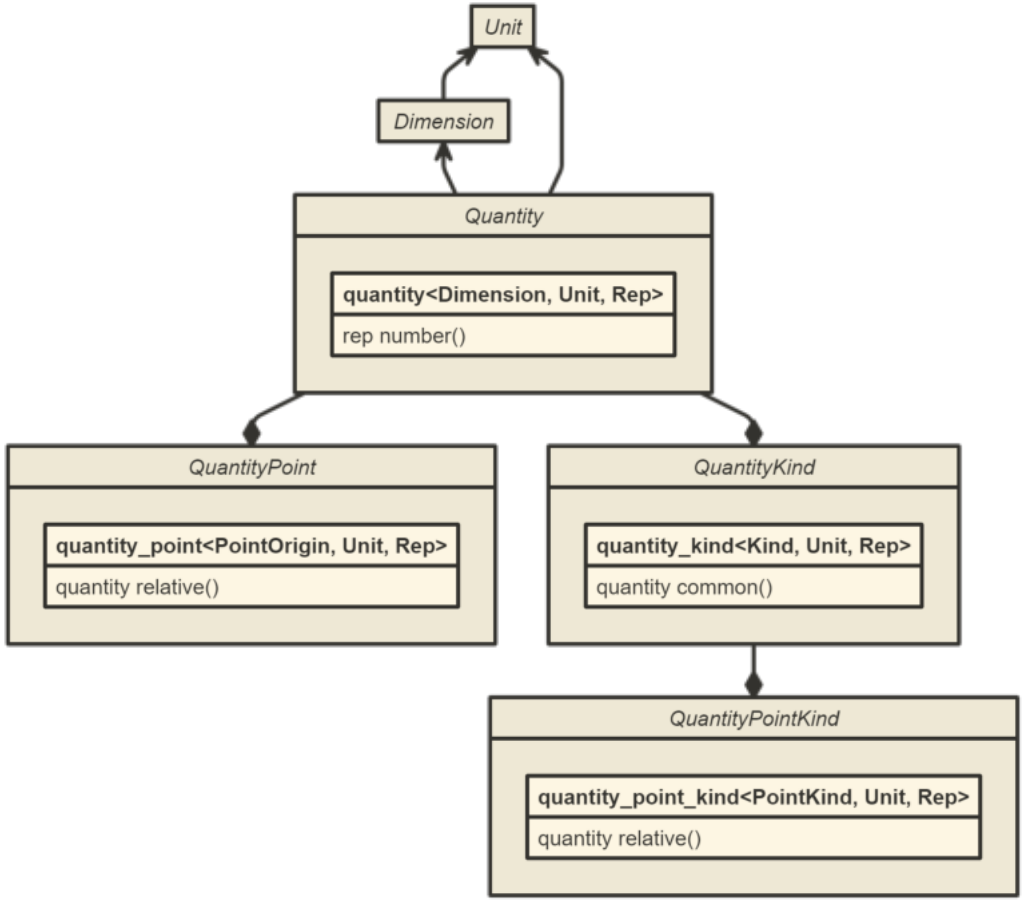
Basic Concepts



Basic Concepts

UNIT

- A basic building block of the library
- Every unit is scaled against its *reference unit*
- Every dimension has a *coherent unit* assigned



Basic Concepts

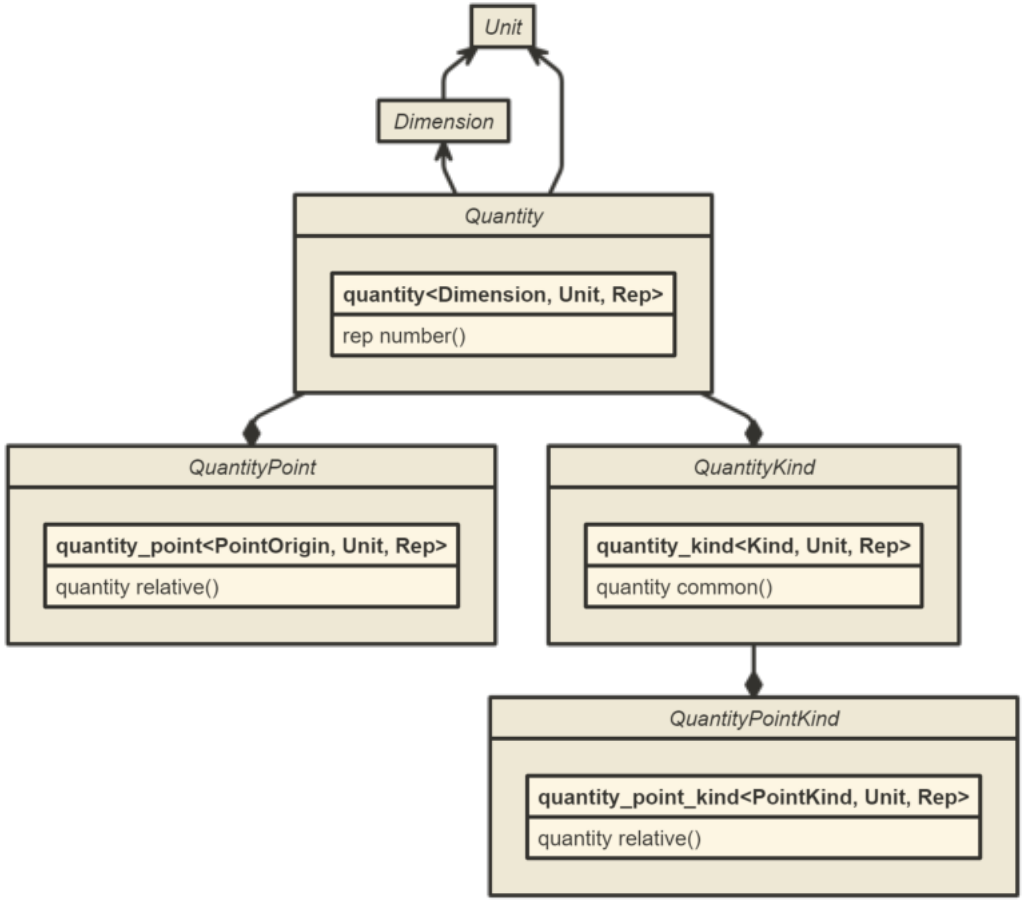
UNIT

- A basic building block of the library
- Every unit is scaled against its *reference unit*
- Every dimension has a *coherent unit* assigned

```
struct metre : named_unit<metre, "m"> {};
struct kilometre : prefixed_unit<kilometre,
                                 kilo, metre> {};
```

```
struct second : named_unit<second, "s"> {};
struct minute : named_scaled_unit<minute, "min",
                    ratio(60), second> {};
struct hour : named_scaled_unit<hour, "h",
                    ratio(60), minute> {};
```

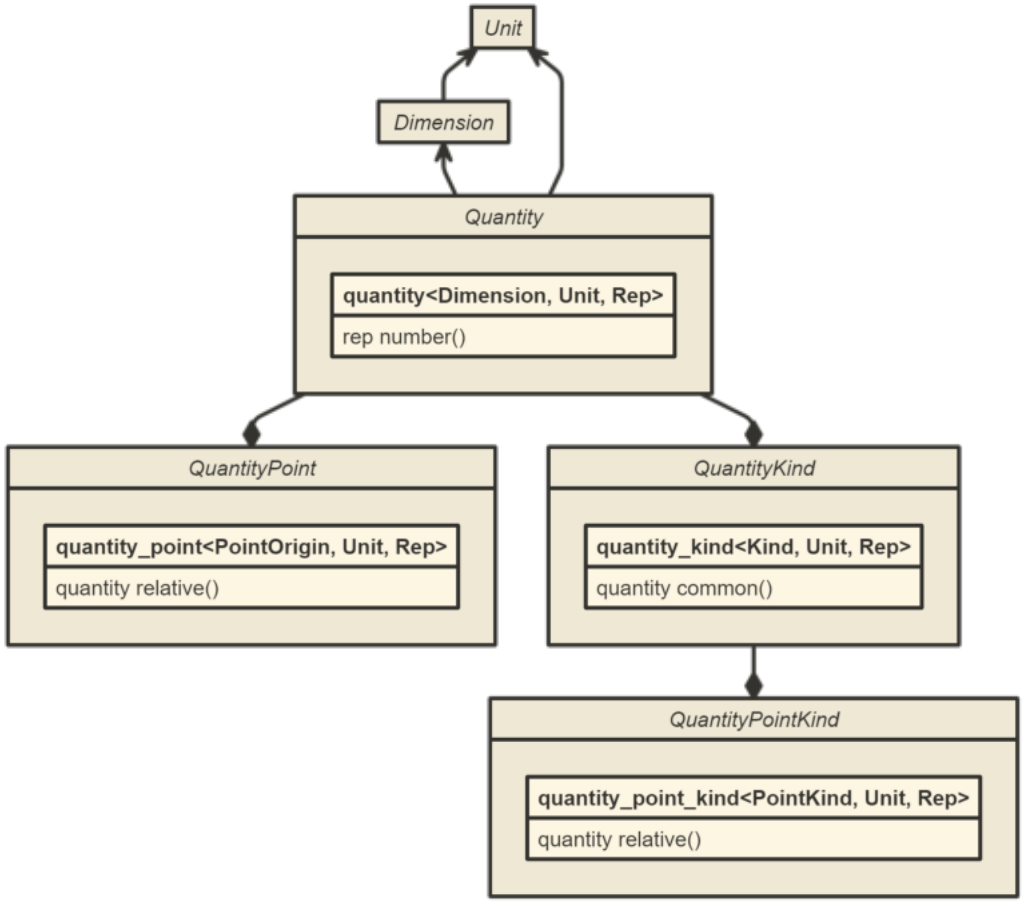
```
struct metre_per_second :
  derived_unit<metre_per_second> {};
struct kilometre_per_hour : derived_scaled_unit<
  kilometre_per_hour, dim_speed, kilometre, hour> {};
```



Basic Concepts

DIMENSION

- Matches a dimension of either a *base* or *derived* quantity

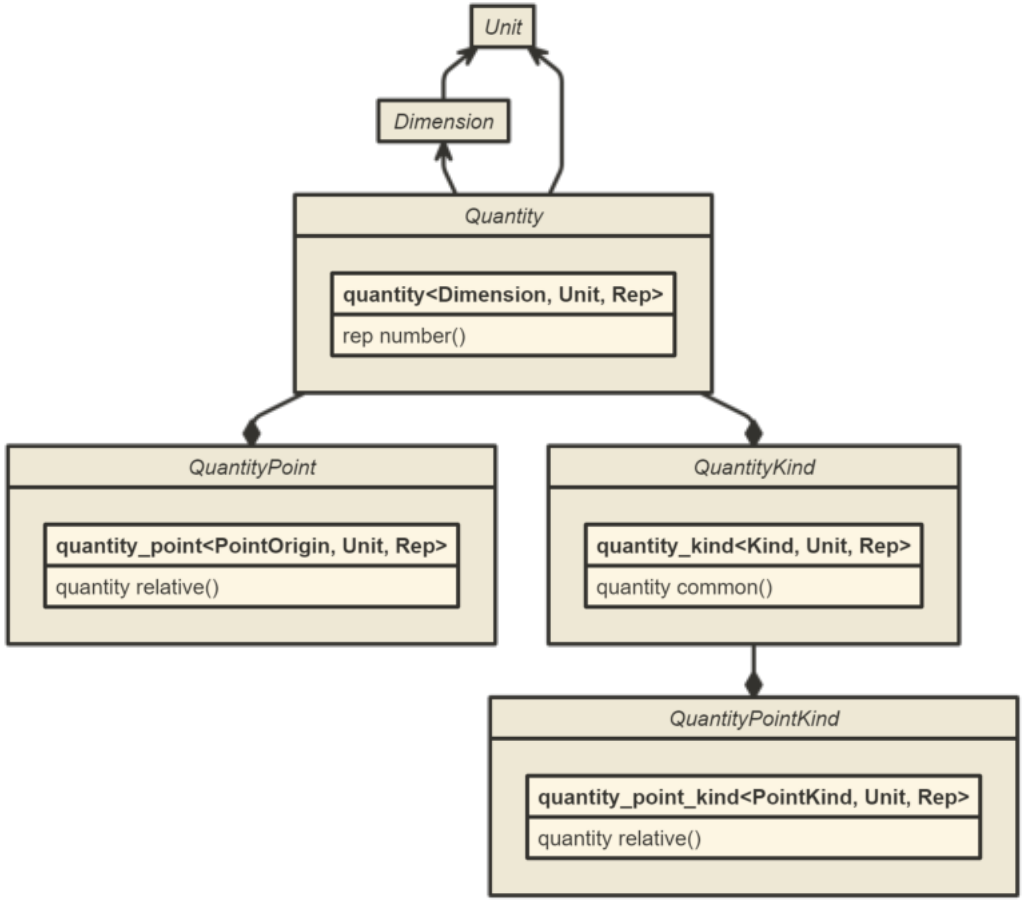


Basic Concepts

DIMENSION

- Matches a dimension of either a *base* or *derived* quantity
- **`base_dimension`** is instantiated with a *unique symbol identifier* and a *base unit*

```
struct dim_length : base_dimension<"L", metre> {};
struct dim_time : base_dimension<"T", second> {};
```



Basic Concepts

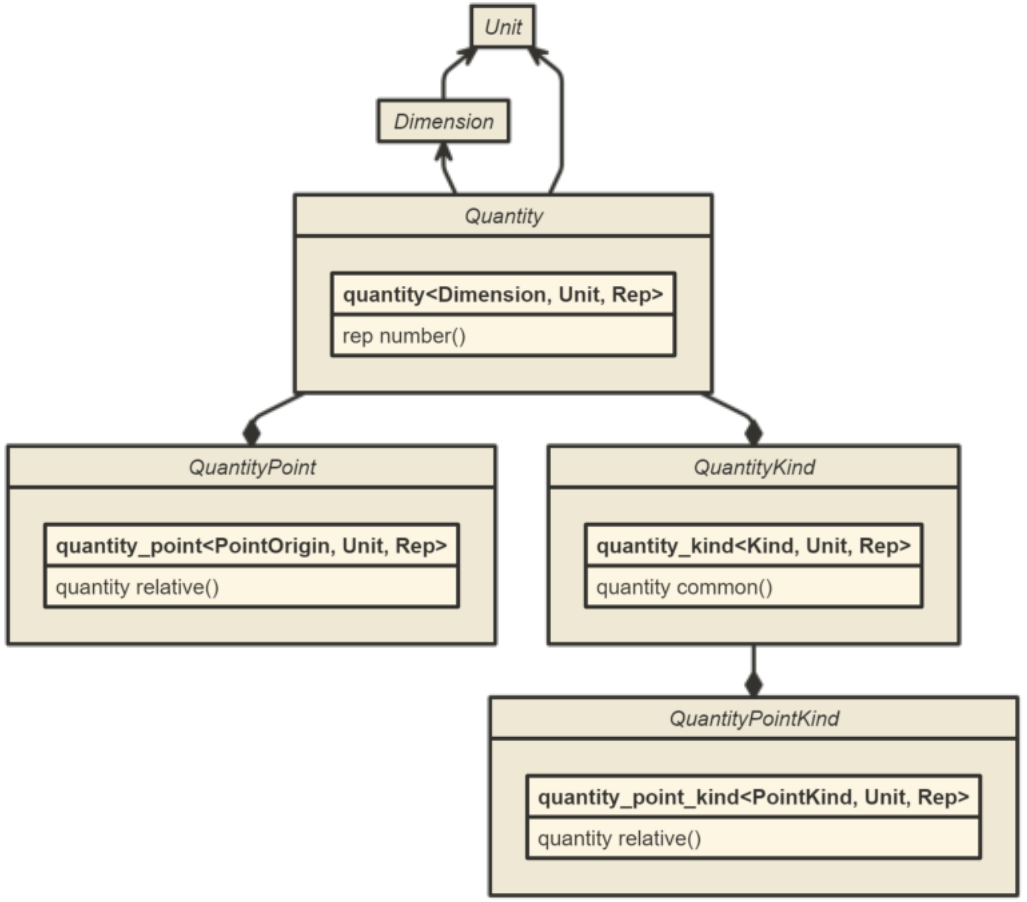
DIMENSION

- Matches a dimension of either a *base* or *derived* quantity
- **`base_dimension`** is instantiated with a *unique symbol identifier* and a *base unit*

```
struct dim_length : base_dimension<"L", metre> {};
struct dim_time : base_dimension<"T", second> {};
```

- **`derived_dimension`** is a *list of exponents* of either base or other derived dimensions

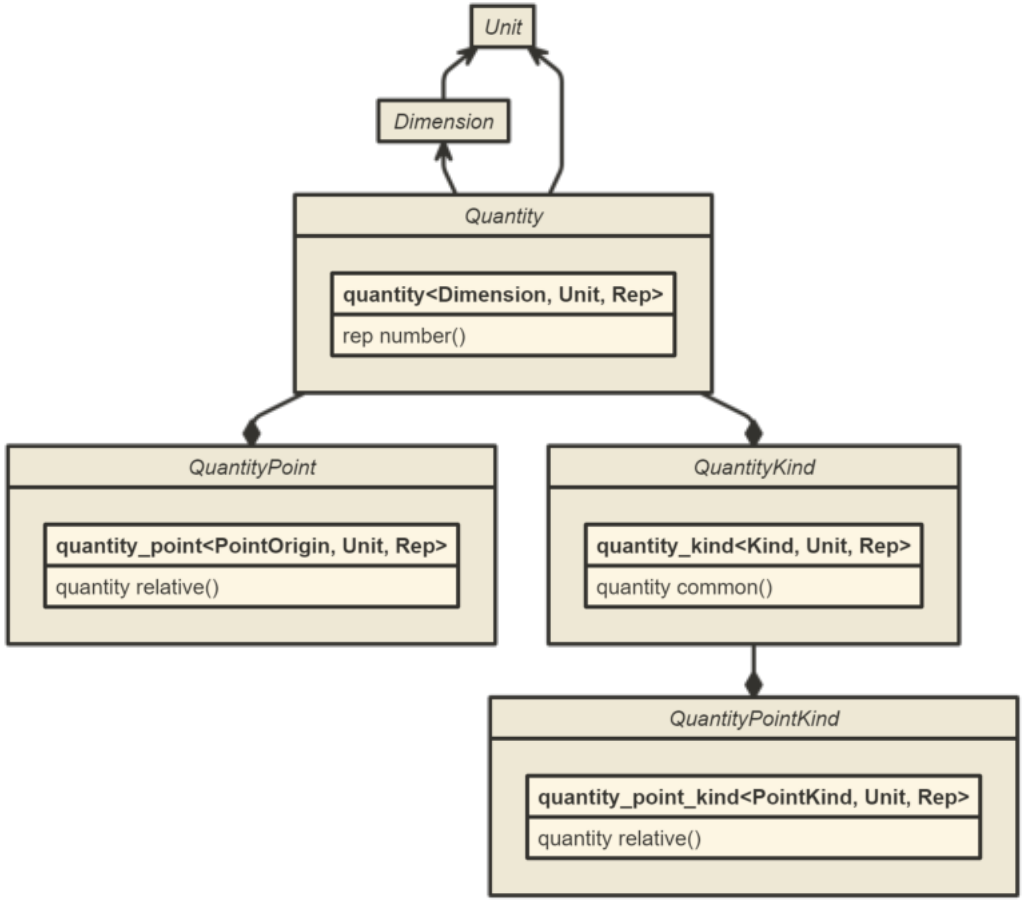
```
struct dim_speed : derived_dimension<dim_speed,
                                     metre_per_second,
                                     exponent<dim_length, 1>,
                                     exponent<dim_time, -1>> {};
```



Basic Concepts

QUANTITY

- A concrete *amount of a unit* for a *specified dimension* with a *specific representation*



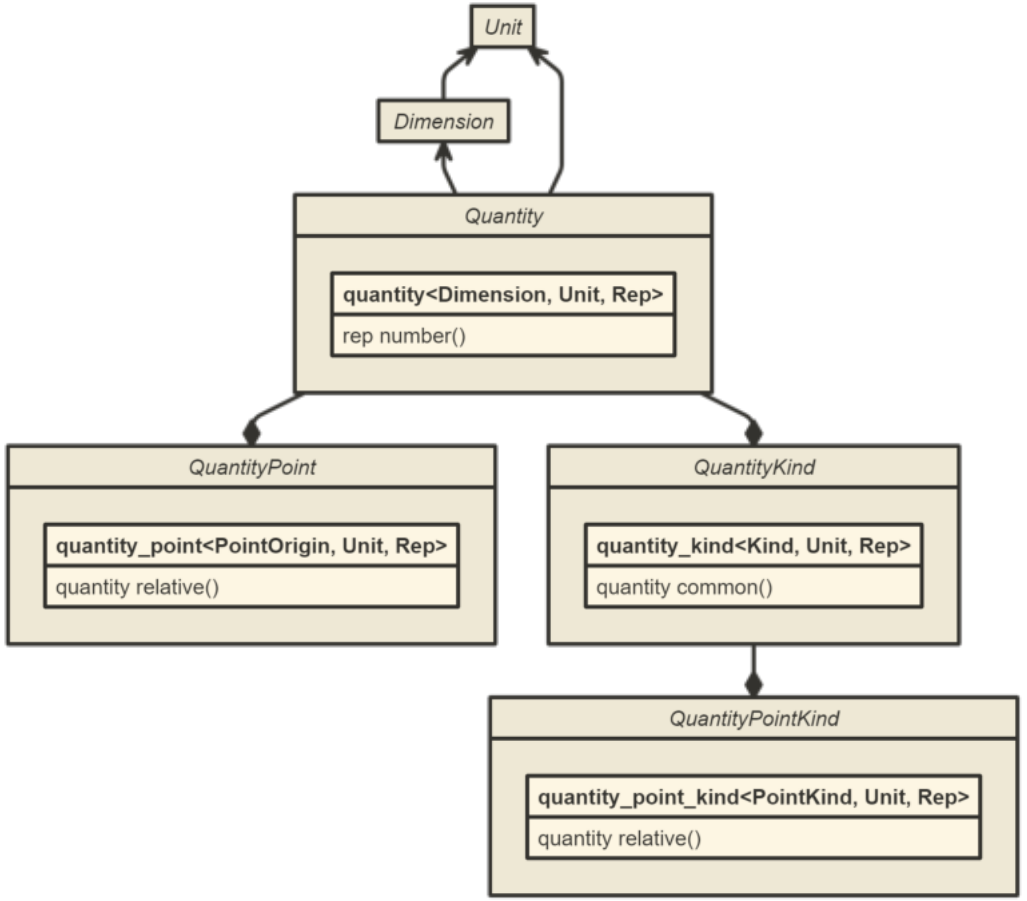
Basic Concepts

QUANTITY

- A concrete *amount of a unit* for a *specified dimension* with a *specific representation*

```
template<Unit U, Representation Rep = double>
using length = quantity<dim_length, U, Rep>;
```

```
si::length<si::kilometre, int> d(3);
```



Basic Concepts

QUANTITY

- A concrete *amount of a unit* for a *specified dimension* with a *specific representation*

```
template<Unit U, Representation Rep = double>
using length = quantity<dim_length, U, Rep>;
```

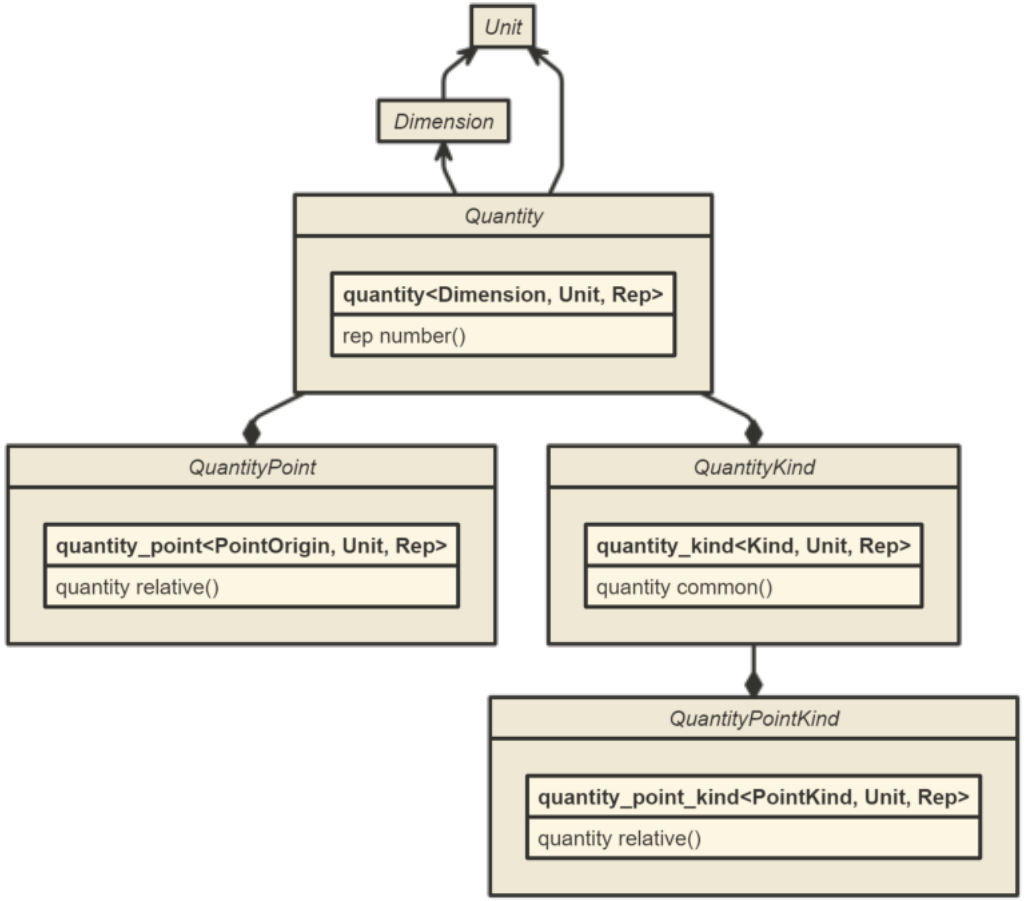
```
si::length<si::kilometre, int> d(3);
```

```
namespace units::aliases::isq::si::inline length {

template<Representation Rep = double>
using km = si::length<si::kilometre, Rep>;

}
```

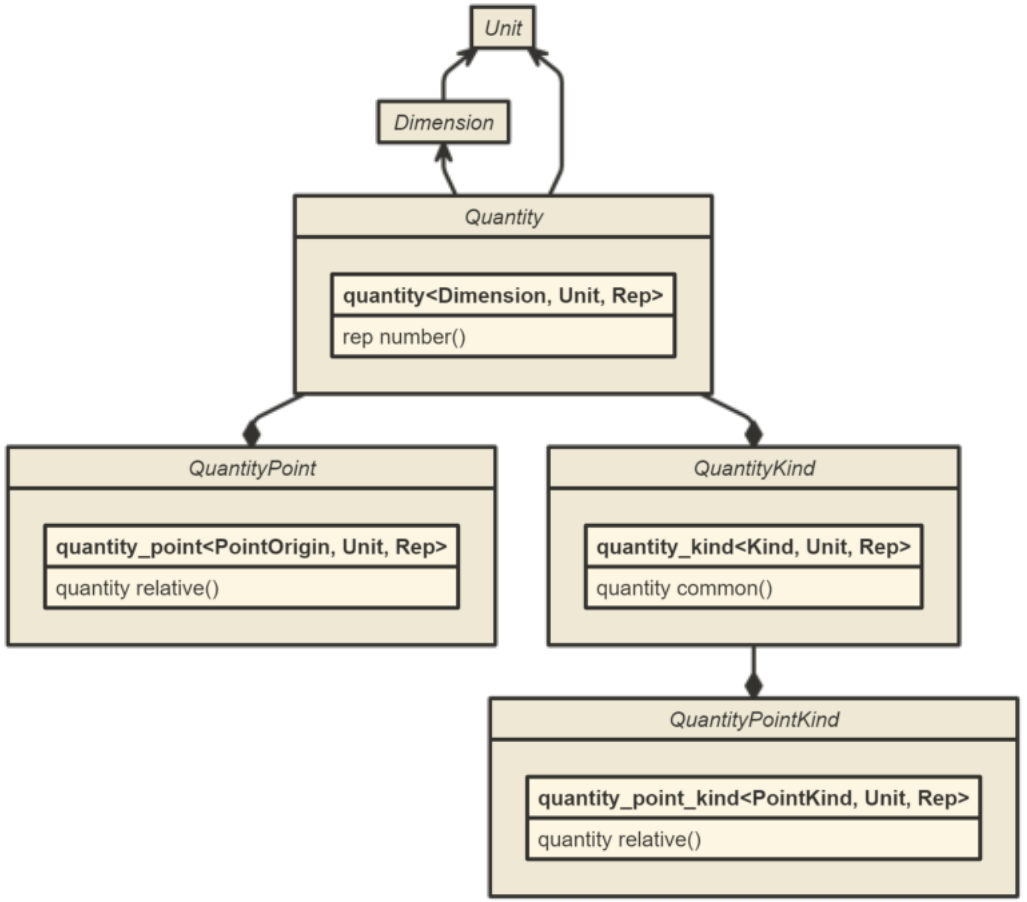
```
si::length::km d(3);
```



Basic Concepts

QUANTITY KIND

- A *quantity of more specific usage*

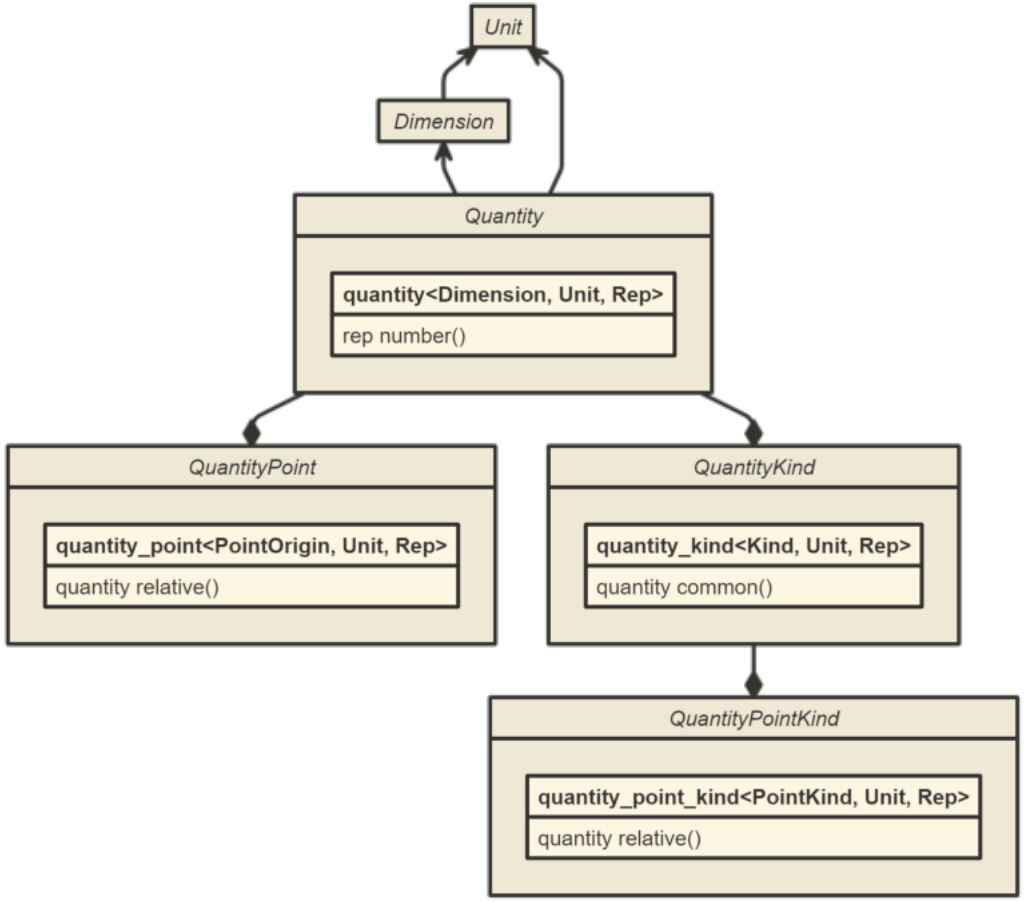


Basic Concepts

QUANTITY KIND

- A *quantity of more specific usage*

```
struct vertical_kind : kind<vertical_kind,
                            si::dim_length> {};
struct horizontal_kind : kind<horizontal_kind,
                              si::dim_length> {};
```



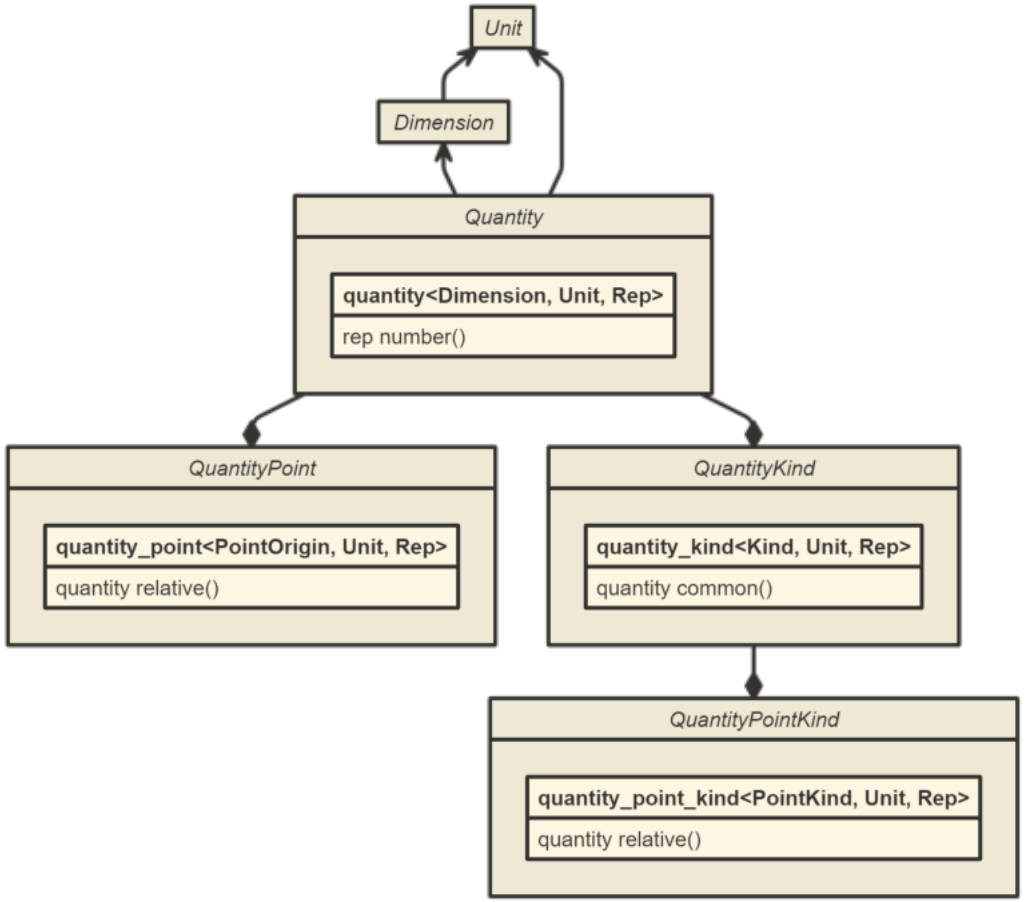
Basic Concepts

QUANTITY KIND

- A *quantity of more specific usage*

```
struct vertical_kind : kind<vertical_kind,
                            si::dim_length> {};
struct horizontal_kind : kind<horizontal_kind,
                              si::dim_length> {};
```

```
using distance = quantity_kind<horizontal_kind,
                               si::kilometre>;
using height = quantity_kind<vertical_kind,
                             si::metre>;
```



Basic Concepts

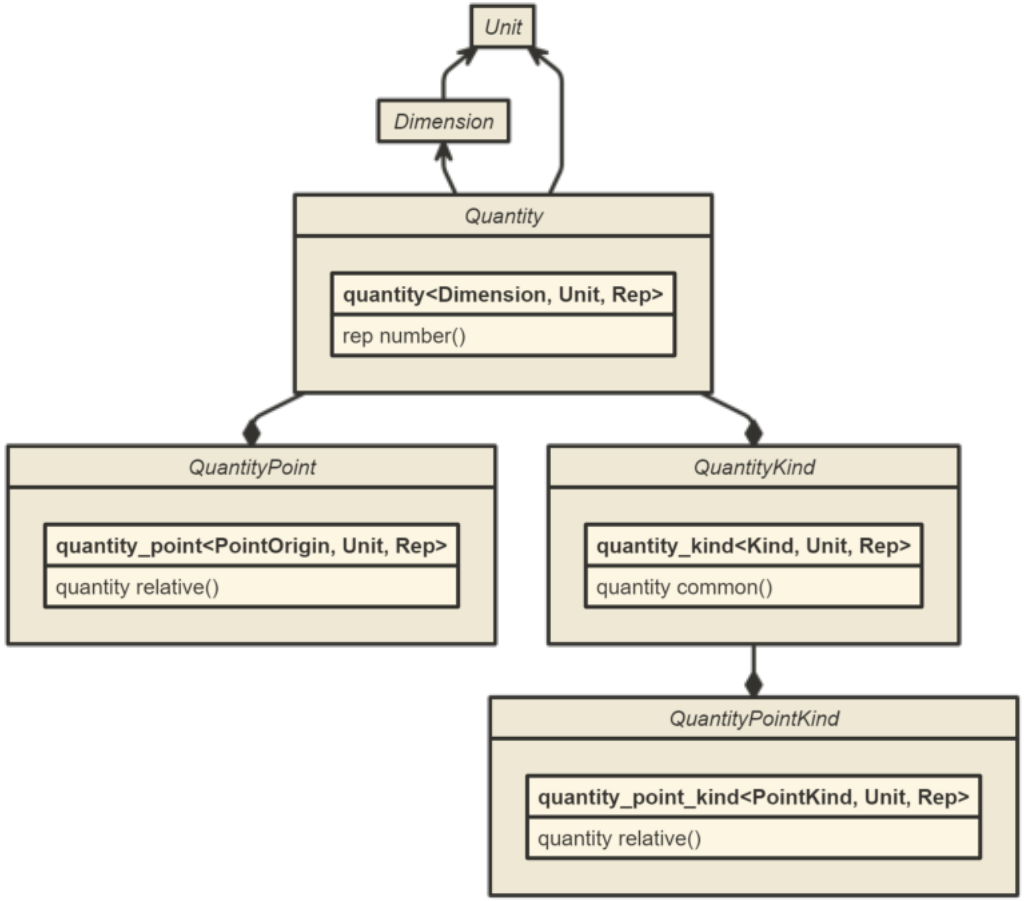
QUANTITY KIND

- A *quantity of more specific usage*

```
struct vertical_kind : kind<vertical_kind,
                            si::dim_length> {};
struct horizontal_kind : kind<horizontal_kind,
                              si::dim_length> {};
```

```
using distance = quantity_kind<horizontal_kind,
                               si::kilometre>;
using height = quantity_kind<vertical_kind,
                             si::metre>;
```

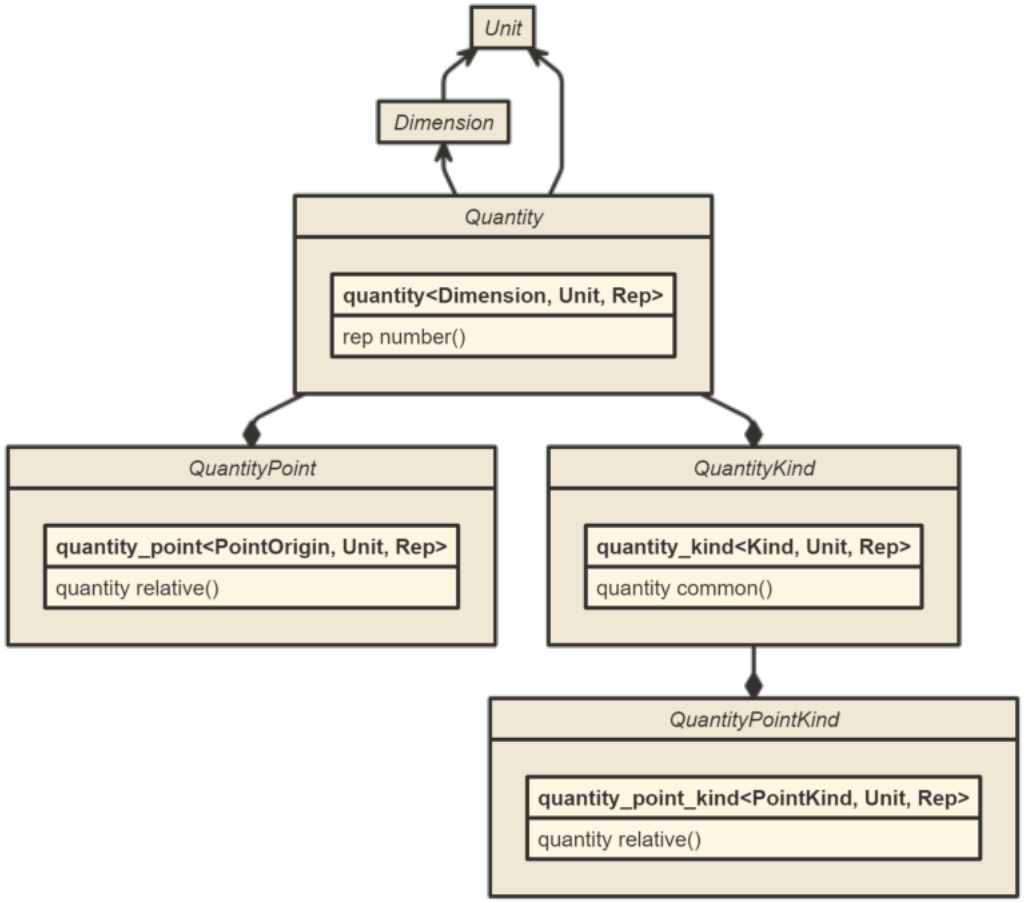
```
struct safety {
  height min_agl_height;
};
```



Basic Concepts

QUANTITY POINT

- An *absolute quantity* with respect to some *origin*

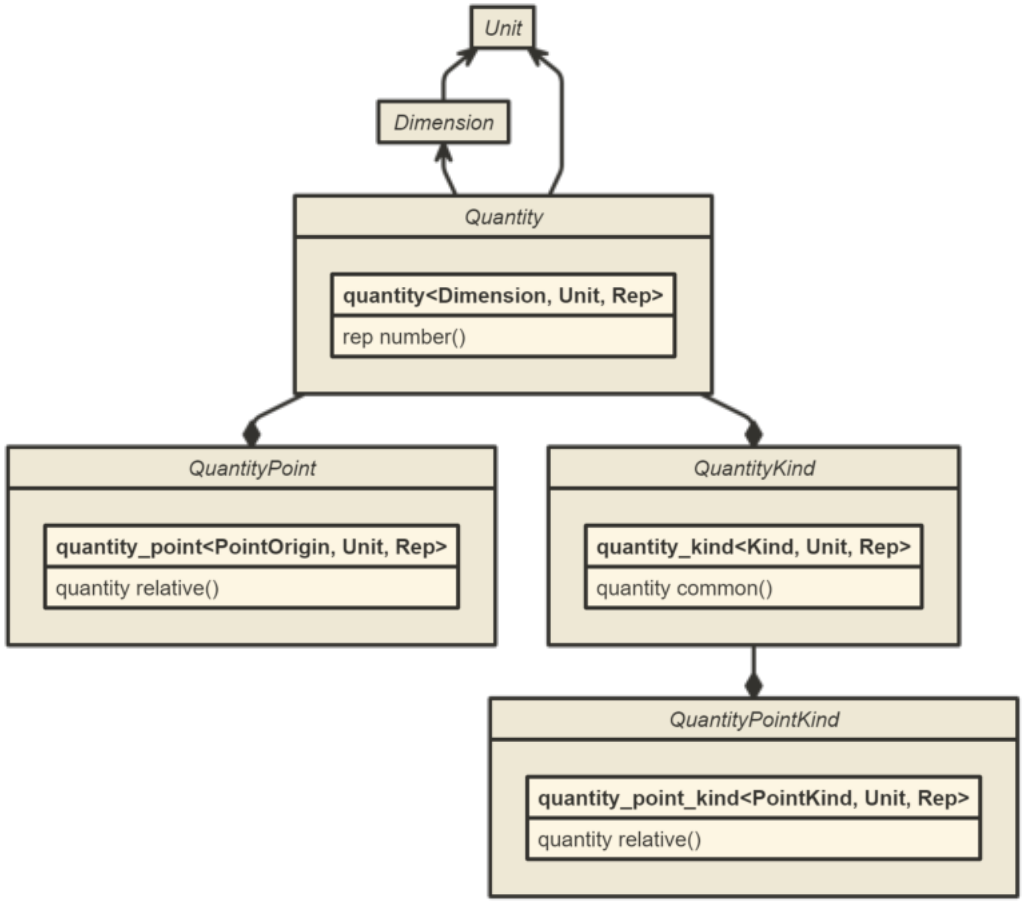


Basic Concepts

QUANTITY POINT

- An *absolute quantity* with respect to some *origin*

```
using timestamp = quantity_point<
          clock_origin<std::chrono::system_clock>,
          si::second>;
```



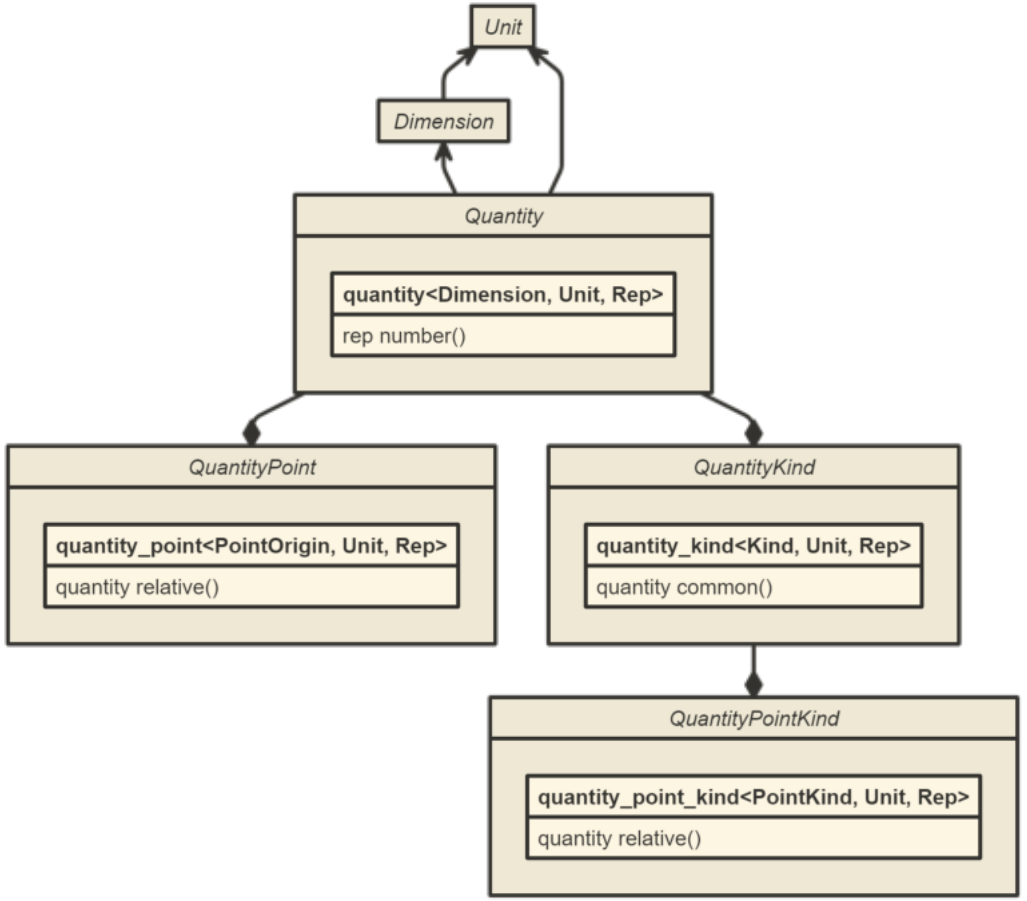
Basic Concepts

QUANTITY POINT

- An *absolute quantity* with respect to some *origin*

```
using timestamp = quantity_point<
          clock_origin<std::chrono::system_clock>,
          si::second>;
```

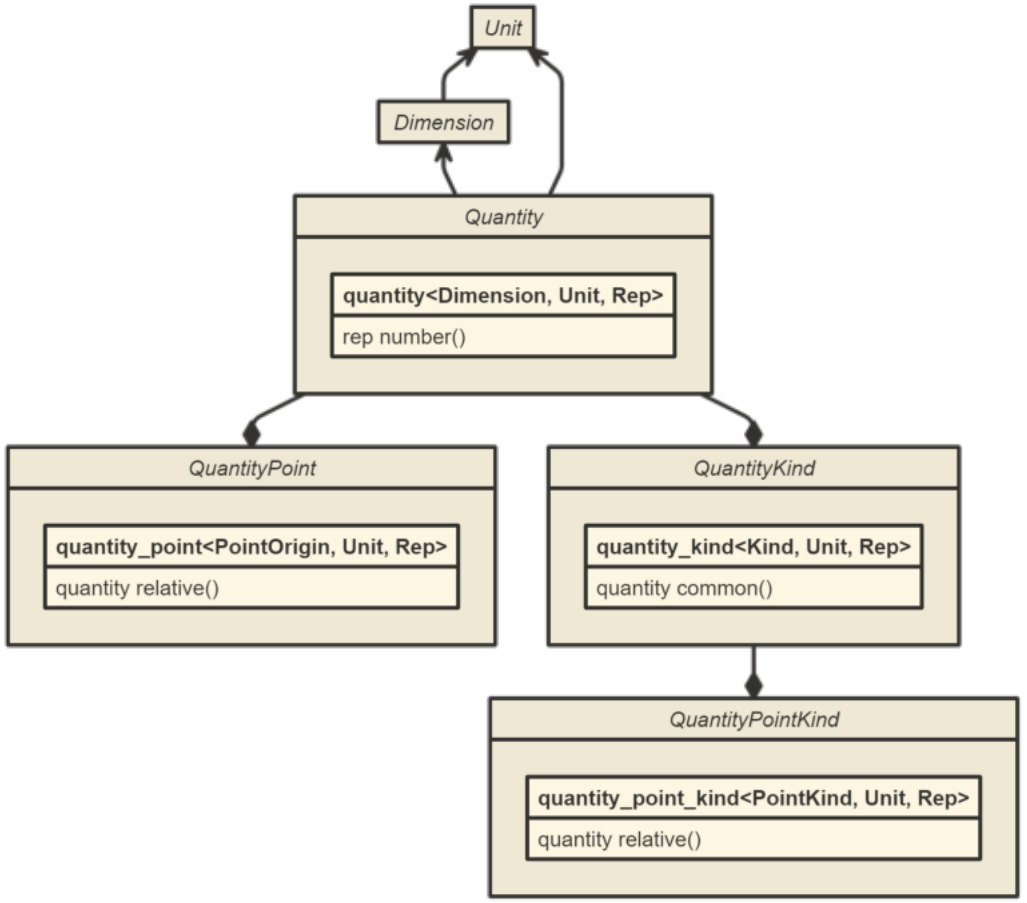
```
using namespace std::chrono;
const timestamp start_time(system_clock::now());
```



Basic Concepts

QUANTITY POINT KIND

- An *absolute quantity kind* with respect to an *origin*

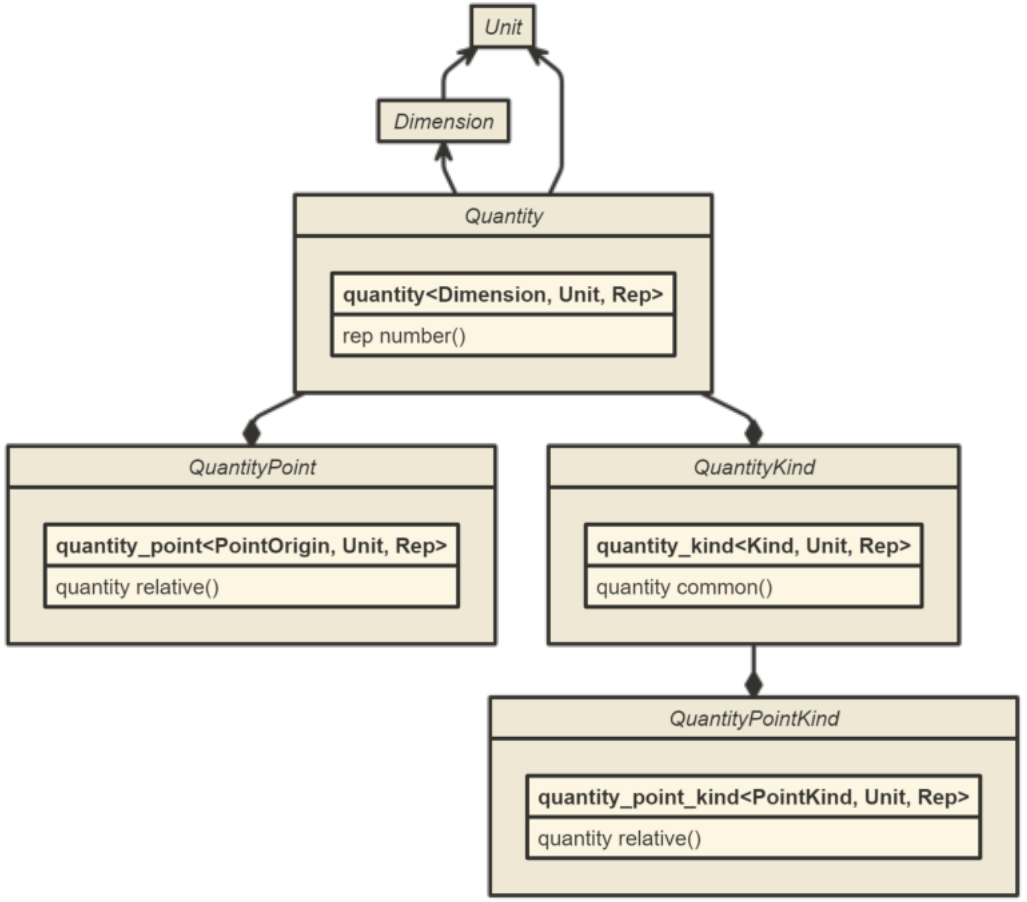


Basic Concepts

QUANTITY POINT KIND

- An *absolute quantity kind* with respect to an *origin*

```
struct vertical_point_kind :
    point_kind<vertical_point_kind, vertical_kind> {};
```



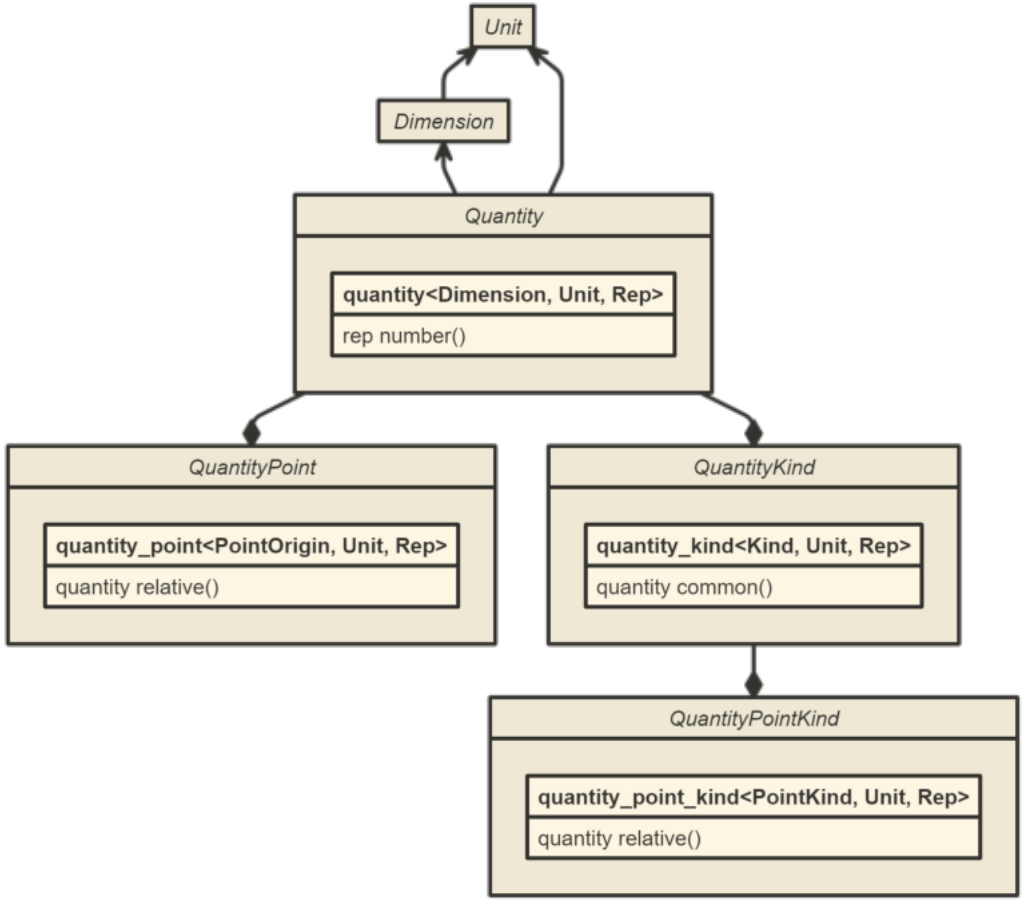
Basic Concepts

QUANTITY POINT KIND

- An *absolute quantity kind* with respect to an *origin*

```
struct vertical_point_kind :
    point_kind<vertical_point_kind, vertical_kind> {};
```

```
using altitude =
    quantity_point_kind<vertical_point_kind,
                        si::international::foot>;
```



Basic Concepts

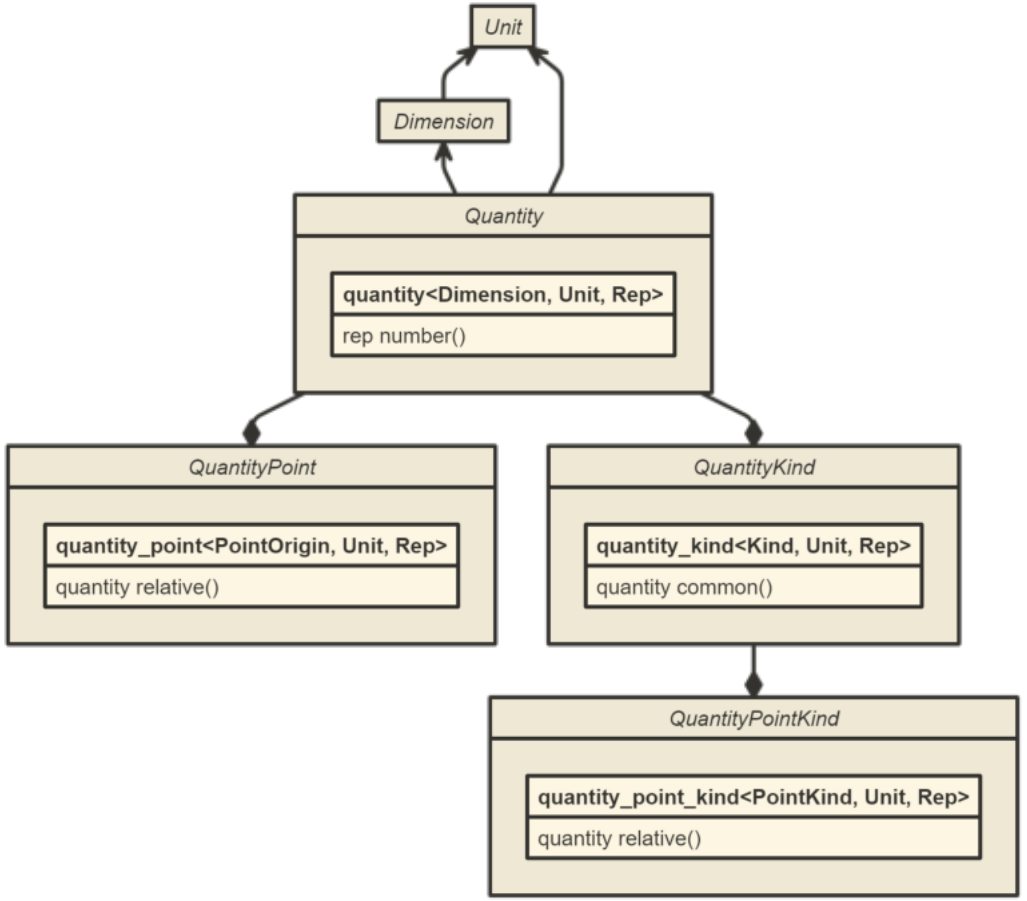
QUANTITY POINT KIND

- An *absolute quantity kind* with respect to an *origin*

```
struct vertical_point_kind :
    point_kind<vertical_point_kind, vertical_kind> {};
```

```
using altitude =
    quantity_point_kind<vertical_point_kind,
                        si::international::foot>;
```

```
struct flight_point {
  timestamp ts;
  altitude alt;
  distance dist;
};
```



Explicit conversions

TO QUANTITY

```
std::cout << "Distance: " << quantity_cast<si::length<si::metre, int>>(d) << '\n';
```

Explicit conversions

TO QUANTITY

```
std::cout << "Distance: " << quantity_cast<si::length<si::metre, int>>(d) << '\n';
```

TO DIMENSION

```
std::cout << "Distance: " << quantity_cast<si::dim_length>(d) << '\n';
```

Explicit conversions

TO QUANTITY

```
std::cout << "Distance: " << quantity_cast<si::length<si::metre, int>>(d) << '\n';
```

TO DIMENSION

```
std::cout << "Distance: " << quantity_cast<si::dim_length>(d) << '\n';
```

TO UNIT

```
std::cout << "Distance: " << quantity_cast<si::metre>(d) << '\n';
```

Explicit conversions

TO QUANTITY

```
std::cout << "Distance: " << quantity_cast<si::length<si::metre, int>>(d) << '\n';
```

TO DIMENSION

```
std::cout << "Distance: " << quantity_cast<si::dim_length>(d) << '\n';
```

TO UNIT

```
std::cout << "Distance: " << quantity_cast<si::metre>(d) << '\n';
```

TO REPRESENTATION TYPE

```
std::cout << "Distance: " << quantity_cast<int>(d) << '\n';
```

Base quantity definition example

```
namespace units::isq {

template<Unit U>
struct dim_time : base_dimension<"T", U> {};

template<typename T>
concept Time = QuantityOfT<T, dim_time>;

}
```

Base quantity definition example

```
namespace units::isq {

template<Unit U>
struct dim_time : base_dimension<"T", U> {};

template<typename T>
concept Time = QuantityOfT<T, dim_time>;

}
```

```
namespace units::isq::si {

struct second : named_unit<second, "s"> {};

struct dim_time : isq::dim_time<second> {};

template<UnitOf<dim_time> U, Representation Rep = double>
using time = quantity<dim_time, U, Rep>;

}
```

Derived quantity definition example

```
namespace units::isq {

template<typename Child, Unit U, DimensionOfT<dim_length> L, DimensionOfT<dim_time> T>
struct dim_speed : derived_dimension<Child, U, exponent<L, 1>, exponent<T, -1>> {};

template<typename T>
concept Speed = QuantityOfT<T, dim_speed>;

}
```

Derived quantity definition example

```
namespace units::isq {

template<typename Child, Unit U, DimensionOfT<dim_length> L, DimensionOfT<dim_time> T>
struct dim_speed : derived_dimension<Child, U, exponent<L, 1>, exponent<T, -1>> {};

template<typename T>
concept Speed = QuantityOfT<T, dim_speed>;

}
```

```
namespace units::isq::si {

struct metre_per_second : derived_unit<metre_per_second> {};

struct dim_speed : isq::dim_speed<dim_speed, metre_per_second, dim_length, dim_time> {};

template<UnitOf<dim_speed> U, Representation Rep = double>
using speed = quantity<dim_speed, U, Rep>;

}
```

Quantity Aliases

```
namespace units::aliases::isq::si::inline length {

template<Representation Rep = double>
using m = units::isq::si::length<units::isq::si::metre, Rep>;

// ...

}
```

User Defined Literals

```
namespace units::isq::si::inline literals {

constexpr auto operator"" _q_m(unsigned long long l)
{
  gsl_ExpectsAudit(std::in_range<std::int64_t>(l));
  return length<metre, std::int64_t>(static_cast<std::int64_t>(l));
}
constexpr auto operator"" _q_m(long double l) { return length<metre, long double>(l); }

// ...

}
```

References

```
namespace units::isq::si {

namespace length_references {

inline constexpr auto m = reference<dim_length, metre>{};
// ...

}

namespace references {

using namespace length_references;

}

}
```

References

```
namespace units::isq::si {

namespace length_references {

inline constexpr auto m = reference<dim_length, metre>{};
// ...

}

namespace references {

using namespace length_references;

}

}
```

Provided only for named units.

Software Engineering Is About Tradeoffs

FEATURE	#1 DIMENSION ALIASES	#2 UNIT ALIASES	#3 REFERENCES	#4 UDLS
Literals and variables support	Yes	Yes	Yes	Literals only
Preserves user provided representation type	No	Yes	Yes	No
Explicit control over the representation type	Yes	Yes	No	No
Readability	Medium	Good	Medium	Good
Possibility to resolve ambiguity	Yes	Yes	Yes	No
Hard to resolve shadowing issues	No	No	Yes	No
Controlled verbosity	No	Yes	No	No
Easy composition for derived units	No	No	Yes	No
Implementation and standardization effort	Lowest	High	Medium	Highest
Compile-time performance	Fastest	Fast	Medium	Slowest

Text Output (godbolt.org/z/snfhK31j5)

OUTPUT STREAMS

```
using namespace units::aliases::isq::si;
constexpr Speed auto v1 = avg_speed(km(220), h(2));
constexpr Speed auto v2 = avg_speed(mi(140), h(2));
std::cout << v1 << '\n';  // 110 km/h
std::cout << v2 << '\n';  // 70 mi/h
```

Text Output (godbolt.org/z/snfhK31j5)

OUTPUT STREAMS

```
using namespace units::aliases::isq::si;
constexpr Speed auto v1 = avg_speed(km(220), h(2));
constexpr Speed auto v2 = avg_speed(mi(140), h(2));
std::cout << v1 << '\n';  // 110 km/h
std::cout << v2 << '\n';  // 70 mi/h
```

std::format

```
std::cout << std::format("{}", km(123)) << '\n';        // 123 km
std::cout << std::format("{:%Q}", km(123)) << '\n';     // 123
std::cout << std::format("{:%q}", km(123)) << '\n';     // km
std::cout << std::format("{:%Q%q}", km(123)) << '\n';   // 123km
```

{fmt} Grammar

```
units-format-spec   ::=  [fill-and-align] [width] [units-specs]
units-specs         ::=  conversion-spec
                         units-specs conversion-spec
                         units-specs literal-char
literal-char        ::=  any character other than '{' or '}'
conversion-spec     ::=  '%' units-type
units-type          ::=  [units-rep-modifier] 'Q'
                         [units-unit-modifier] 'q'
                         one of "nt%"
units-rep-modifier  ::=  [sign] [#] [precision] [L] [units-rep-type]
units-rep-type      ::=  one of "aAbBdeEfFgGoxX"
units-unit-modifier ::=  'A'
```

Unicode and ASCII-only (godbolt.org/z/4sc3PdExK)

```
std::cout << std::format("{}", resistance::R(10)) << '\n';           // 10 Ω
std::cout << std::format("{:%Q %Aq}", R(10)) << '\n';                // 10 ohm
std::cout << std::format("{}", time::us(125)) << '\n';               // 125 µs
std::cout << std::format("{:%Q %Aq}", us(125)) << '\n';              // 125 us
std::cout << std::format("{}", acceleration::m_per_s2(9.8)) << '\n'; // 9.8 m/s²
std::cout << std::format("{:%Q %Aq}", m_per_s2(9.8)) << '\n';        // 9.8 m/s^2
```

- Unicode output by default
- ASCII-only output with **A** modifier

GOODIEBOX

Filling a box (godbolt.org/z/raje5T5hG)

```
using namespace units::aliases::isq::si;

inline constexpr auto g = units::isq::si::si2019::standard_gravity<>;
inline constexpr auto air_density = kg_per_m3{1.225};

class Box {
  area::m2<> base_;
  length::m<> height_;
  density::kg_per_m3<> density_ = air_density;
public:









};
```

Filling a box (godbolt.org/z/raje5T5hG)

```
using namespace units::aliases::isq::si;

inline constexpr auto g = units::isq::si::si2019::standard_gravity<>;
inline constexpr auto air_density = kg_per_m3{1.225};

class Box {
  area::m2<> base_;
  length::m<> height_;
  density::kg_per_m3<> density_ = air_density;
public:
  constexpr Box(const length::m<>& length, const length::m<>& width, length::m<> height) :
    base_(length * width), height_(std::move(height))
  {
  }




};
```

Filling a box (godbolt.org/z/raje5T5hG)

```
using namespace units::aliases::isq::si;

inline constexpr auto g = units::isq::si::si2019::standard_gravity<>;
inline constexpr auto air_density = kg_per_m3{1.225};

class Box {
  area::m2<> base_;
  length::m<> height_;
  density::kg_per_m3<> density_ = air_density;
public:
  constexpr Box(const length::m<>& length, const length::m<>& width, length::m<> height) :
    base_(length * width), height_(std::move(height))
  {
  }

  constexpr void set_contents_density(const density::kg_per_m3<>& density_in);
  [[nodiscard]] constexpr force::N<> filled_weight() const;
  [[nodiscard]] constexpr length::m<> fill_level(const mass::kg<>& measured_mass) const;
  [[nodiscard]] constexpr volume::m3<> spare_capacity(const mass::kg<>& measured_mass) const;
};
```

Filling a box (godbolt.org/z/raje5T5hG)

```
constexpr void Box::set_contents_density(const density::kg_per_m3<>& density_in)
{
  gsl_Expects(density_in > air_density);
  density_ = density_in;
}
```

Filling a box (godbolt.org/z/raje5T5hG)

```
constexpr void Box::set_contents_density(const density::kg_per_m3<>& density_in)
{
  gsl_Expects(density_in > air_density);
  density_ = density_in;
}
```

```
[[nodiscard]] constexpr force::N<> Box::filled_weight() const
{
  const volume::m3 volume = base_ * height_;
  const mass::kg mass = density_ * volume;
  return mass * g;
}
```

Filling a box (godbolt.org/z/raje5T5hG)

```
constexpr void Box::set_contents_density(const density::kg_per_m3<>& density_in)
{
  gsl_Expects(density_in > air_density);
  density_ = density_in;
}
```

```
[[nodiscard]] constexpr force::N<> Box::filled_weight() const
{
  const volume::m3 volume = base_ * height_;
  const mass::kg mass = density_ * volume;
  return mass * g;
}
```

```
[[nodiscard]] constexpr length::m<> Box::fill_level(const mass::kg<>& measured_mass) const
{
  return height_ * measured_mass * g / filled_weight();
}
```

Filling a box (godbolt.org/z/raje5T5hG)

```
constexpr void Box::set_contents_density(const density::kg_per_m3<>& density_in)
{
  gsl_Expects(density_in > air_density);
  density_ = density_in;
}
```

```
[[nodiscard]] constexpr force::N<> Box::filled_weight() const
{
  const volume::m3 volume = base_ * height_;
  const mass::kg mass = density_ * volume;
  return mass * g;
}
```

```
[[nodiscard]] constexpr length::m<> Box::fill_level(const mass::kg<>& measured_mass) const
{
  return height_ * measured_mass * g / filled_weight();
}
```

```
[[nodiscard]] constexpr volume::m3<> Box::spare_capacity(const mass::kg<>& measured_mass) const
{
  return (height_ - fill_level(measured_mass)) * base_;
}
```

Filling a box (godbolt.org/z/raje5T5hG)

```
using namespace units;

const length::m height{mm{200.0}};
auto box = Box(mm{1000.0}, mm{500.0}, height);
box.set_contents_density(kg_per_m3{1000.0});

const auto fill_time = s{200.0};      // time since starting fill
const auto measured_mass = kg{20.0};  // measured mass at fill_time
```

Filling a box (godbolt.org/z/raje5T5hG)

```
using namespace units;

const length::m height{mm{200.0}};
auto box = Box(mm{1000.0}, mm{500.0}, height);
box.set_contents_density(kg_per_m3{1000.0});

const auto fill_time = s{200.0};      // time since starting fill
const auto measured_mass = kg{20.0};  // measured mass at fill_time
```

```
const auto fill_level = box.fill_level(measured_mass);
const Dimensionless auto fill_percent = quantity_cast<percent>(fill_level / height);
const auto spare_capacity = box.spare_capacity(measured_mass);
const auto input_flow_rate = measured_mass / fill_time;  // unknown dimension
const auto float_rise_rate = fill_level / fill_time;
const auto fill_time_left = (height / fill_level - 1) * fill_time;
```

Filling a box (godbolt.org/z/raje5T5hG)

```
std::cout << fmt::format("fill height at {} = {} ({} full)\n", fill_time, fill_level, fill_percent);
std::cout << fmt::format("spare_capacity at {} = {}\n", fill_time, spare_capacity);
std::cout << fmt::format("input flow rate after {} = {}\n", fill_time, input_flow_rate);
std::cout << fmt::format("float rise rate = {}\n", float_rise_rate);
std::cout << fmt::format("box full E.T.A. at current flow rate = {}\n", fill_time_left);
```

Filling a box (godbolt.org/z/raje5T5hG)

```
std::cout << fmt::format("fill height at {} = {} ({} full)\n", fill_time, fill_level, fill_percent);
std::cout << fmt::format("spare_capacity at {} = {}\n", fill_time, spare_capacity);
std::cout << fmt::format("input flow rate after {} = {}\n", fill_time, input_flow_rate);
std::cout << fmt::format("float rise rate = {}\n", float_rise_rate);
std::cout << fmt::format("box full E.T.A. at current flow rate = {}\n", fill_time_left);
```

```
fill height at 200 s = 0.04 m (20 % full)
spare_capacity at 200 s = 0.08 m³
input flow rate after 200 s = 0.1 kg/s
float rise rate = 0.0002 m/s
box full E.T.A. at current flow rate = 800 s
```

Ensuring a Box is actually a box (godbolt.org/z/d7sY59h7z)

```
class Box {
  area::m2<> base_;
  length::m<> height_;
  density::kg_per_m3<> density_ = air_density;
public:
  constexpr Box(const length::m<>& length, const length::m<>& width, length::m<> height) :
    base_(length * width), height_(std::move(height))
  {
  }

  // ...
};
```

```
const length::m height{mm{200.0}};
auto box = Box(mm{1000.0}, mm{500.0}, height);
```

Ensuring a **Box** is actually a box (godbolt.org/z/d7sY59h7z)

```
struct vertical_kind   : units::kind<vertical_kind,     units::isq::si::dim_length> {};
struct horizontal_kind : units::kind<horizontal_kind,   units::isq::si::dim_length> {};
struct area_kind       : units::derived_kind<area_kind, units::isq::si::dim_area, horizontal_kind> {};
struct volume_kind     : units::kind<volume_kind,       units::isq::si::dim_volume> {};
```

Ensuring a **Box** is actually a box (godbolt.org/z/d7sY59h7z)

```
struct vertical_kind   : units::kind<vertical_kind,     units::isq::si::dim_length> {};
struct horizontal_kind : units::kind<horizontal_kind,   units::isq::si::dim_length> {};
struct area_kind       : units::derived_kind<area_kind, units::isq::si::dim_area, horizontal_kind> {};
struct volume_kind     : units::kind<volume_kind,       units::isq::si::dim_volume> {};
```

```
using vertical_length   = units::quantity_kind<vertical_kind,   units::isq::si::metre>;
using horizontal_length = units::quantity_kind<horizontal_kind, units::isq::si::metre>;
using horizontal_area   = units::quantity_kind<area_kind,       units::isq::si::square_metre>;
using volume            = units::quantity_kind<volume_kind,     units::isq::si::cubic_metre>;
```

Ensuring a **Box** is actually a box (godbolt.org/z/d7sY59h7z)

```
struct vertical_kind   : units::kind<vertical_kind,     units::isq::si::dim_length> {};
struct horizontal_kind : units::kind<horizontal_kind,   units::isq::si::dim_length> {};
struct area_kind       : units::derived_kind<area_kind, units::isq::si::dim_area, horizontal_kind> {};
struct volume_kind     : units::kind<volume_kind,       units::isq::si::dim_volume> {};
```

```
using vertical_length   = units::quantity_kind<vertical_kind,   units::isq::si::metre>;
using horizontal_length = units::quantity_kind<horizontal_kind, units::isq::si::metre>;
using horizontal_area   = units::quantity_kind<area_kind,       units::isq::si::square_metre>;
using volume            = units::quantity_kind<volume_kind,     units::isq::si::cubic_metre>;
```

```
template<units::QuantityKindOf<area_kind> QK1, units::QuantityKindOf<vertical_kind> QK2>
constexpr volume operator*(const QK1& lhs, const QK2& rhs)
{
  return volume{lhs.common() * rhs.common()};
}

template<units::QuantityKindOf<vertical_kind> QK1, units::QuantityKindOf<area_kind> QK2>
constexpr volume operator*(const QK1& lhs, const QK2& rhs)
{
  return rhs * lhs;
}
```

Ensuring a Box is actually a box (godbolt.org/z/d7sY59h7z)

```
class Box {
  horizontal_area base_;
  vertical_length height_;
  density::kg_per_m3<> density_ = air_density;
public:
  constexpr Box(const horizontal_length& length, const horizontal_length& width,
                vertical_length height) :
      base_(length * width), height_(std::move(height))
  {
  }

  // ...
};
```

```
const vertical_length height{mm{200.}};
auto box = Box(horizontal_length{mm{1000.}}, horizontal_length{mm{500}}, height);
```

Ensuring a Box is actually a box (godbolt.org/z/d7sY59h7z)

```
[[nodiscard]] constexpr force::N<> Box::filled_weight() const
{
  const volume vol = base_ * height_;
  const mass::kg mass = density_ * vol.common();
  return mass * g;
}

[[nodiscard]] constexpr vertical_length Box::fill_level(const mass::kg<>& measured_mass) const
{
  return height_ * measured_mass * g / filled_weight();
}

[[nodiscard]] constexpr volume Box::spare_capacity(const mass::kg<>& measured_mass) const
{
  return (height_ - fill_level(measured_mass)) * base_;
}
```

ENVIRONMENT, COMPATIBILITY, NEXT STEPS

C++20 in the library

LANGUAGE

- Concept
- Class NTTPs
- Consistent and Defaulted Comparison
- **`explicit(bool)`**
- Down with **`typename`**
- Lambdas in unevaluated contexts
- Immediate functions (**`consteval`**) (TBD)
- Modules (TBD)

LIBRARY

- **`constexpr`** algorithms
- Concepts Library
- **`{fmt}`**

Compilers support

- gcc-10
- clang-12
- Visual Studio 16.9
- Apple clang 13

Plans for C++ Standardization

- ISO C++ Committee dedicated *many hours* for this subject already
- A lot of *interest in the industry*
- Really *positive feedback* so far
- **Hopefully C++26**
  - we need more field experience and feedback
  - C++26 train arrives soon
  - COVID-19 did not help

Plans for C++ Standardization

- ISO C++ Committee dedicated *many hours* for this subject already
- A lot of *interest in the industry*
- Really *positive feedback* so far
- **Hopefully C++26**
  - we need more field experience and feedback
  - C++26 train arrives soon
  - COVID-19 did not help

We want to ensure that the library is ready before we start the process.

Please try and tell us about your experience or requirements

COMPANIES

- Production/POC use
- Requirements

Please try and tell us about your experience or requirements

COMPANIES

- Production/POC use
- Requirements

AUTHORS OF OTHER LIBRARIES

- Implementation experience
- Production feedback

Please try and tell us about your experience or requirements

COMPANIES

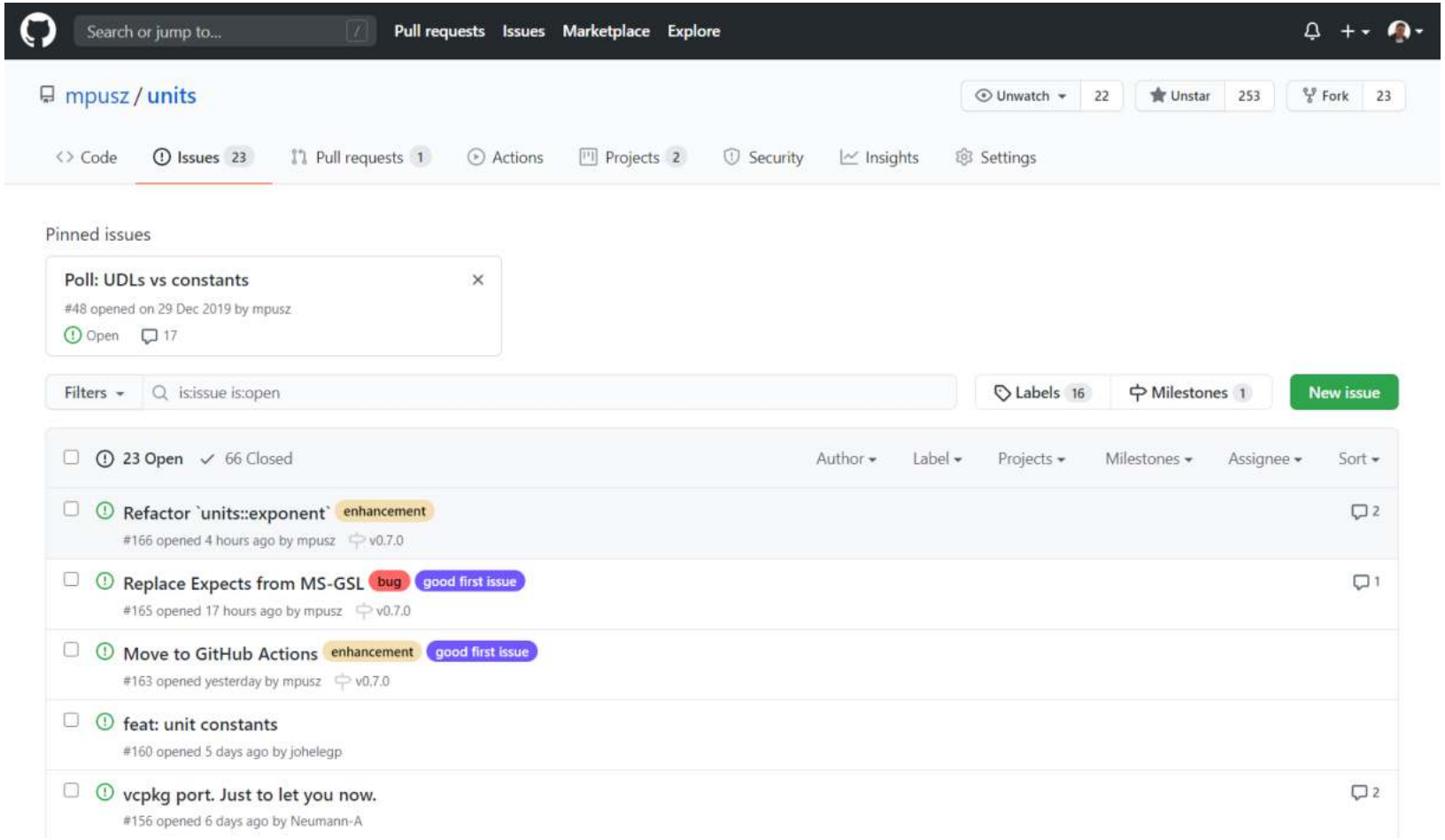
- Production/POC use
- Requirements

AUTHORS OF OTHER LIBRARIES

- Implementation experience
- Production feedback

GitHub Issues are not only to complain about the issues. Please also let us know if you are a happy user :-)

Design Discussions, Issues, Next Steps, Feedback (github.com/mpusz/units/issues)



Thank You! It is not just me...

Thank You! It is not just me...

EXISTING PRACTICE

- Matthias Christian Schabel & Steven Watanabe (Boost.Units)
- Nic Holthaus (github.com/nholthaus/units)
- Martin Moene (github.com/martinmoene/PhysUnits-CT)
- Jan A. Sende (github.com/jansende/benri)
- Others...

Thank You! It is not just me...

EXISTING PRACTICE

- Matthias Christian Schabel & Steven Watanabe (Boost.Units)
- Nic Holthaus (github.com/nholthaus/units)
- Martin Moene (github.com/martinmoene/PhysUnits-CT)
- Jan A. Sende (github.com/jansende/benri)
- Others...

C++ Physical Units library is nothing new. We have a production experience with various implementations for many years now.

Thank You! It is not just me...

CONTRIBUTORS

- Johel Ernesto Guerrero Peña (@JohelEGP)
- Riccardo Brugo (@rbrugo)
- Ramzi Sabra (@yasamoka)
- Andy Little (@kwikius)
- Ralph J. Steinhagen (@RalphSteinhagen)
- Oliver Schönrock (@oschonrock)
- Michael Ford (@mikeford1)
- Jan A. Sende (@jansende)
- @i-ky
- Others...

Thank You! It is not just me...

CONTRIBUTORS

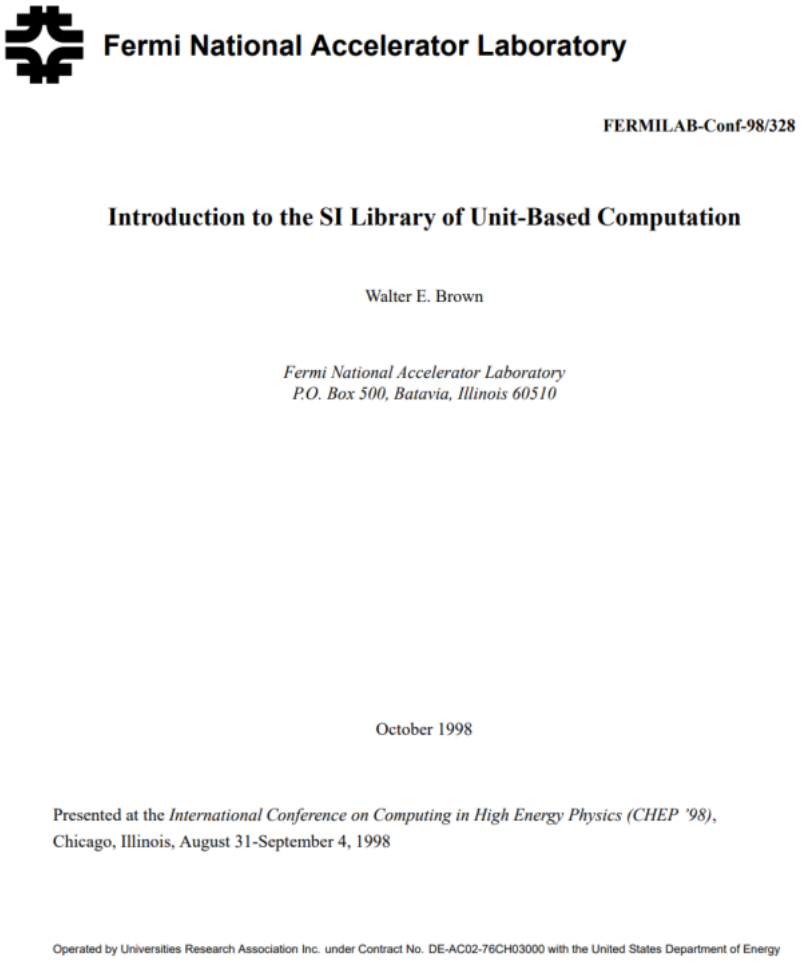
- Johel Ernesto Guerrero Peña (@JohelEGP)
- Riccardo Brugo (@rbrugo)
- Ramzi Sabra (@yasamoka)
- Andy Little (@kwikius)
- Ralph J. Steinhagen (@RalphSteinhagen)
- Oliver Schönrock (@oschonrock)
- Michael Ford (@mikeford1)
- Jan A. Sende (@jansende)
- @i-ky
- Others...

Contribution is not only about co-developing the code but also about taking part in discussions, sharing ideas, or stating requirements.

Thank You! It is not just me...

SPECIAL THANKS

- Walter Brown



Fermi National Accelerator Laboratory

FERMILAB-Conf-98/328

Introduction to the SI Library of Unit-Based Computation

Walter E. Brown

Fermi National Accelerator Laboratory
P.O. Box 500, Batavia, Illinois 60510

October 1998

Presented at the *International Conference on Computing in High Energy Physics (CHEP '98)*, Chicago, Illinois, August 31-September 4, 1998

Operated by Universities Research Association Inc. under Contract No. DE-AC02-76CH03000 with the United States Department of Energy

Thank You! It is not just me...

SPECIAL THANKS

- Howard Hinnant

Thank You! It is not just me...

SPECIAL THANKS

- GCC developers


CAUTION
Programming
is addictive
(and too much fun)