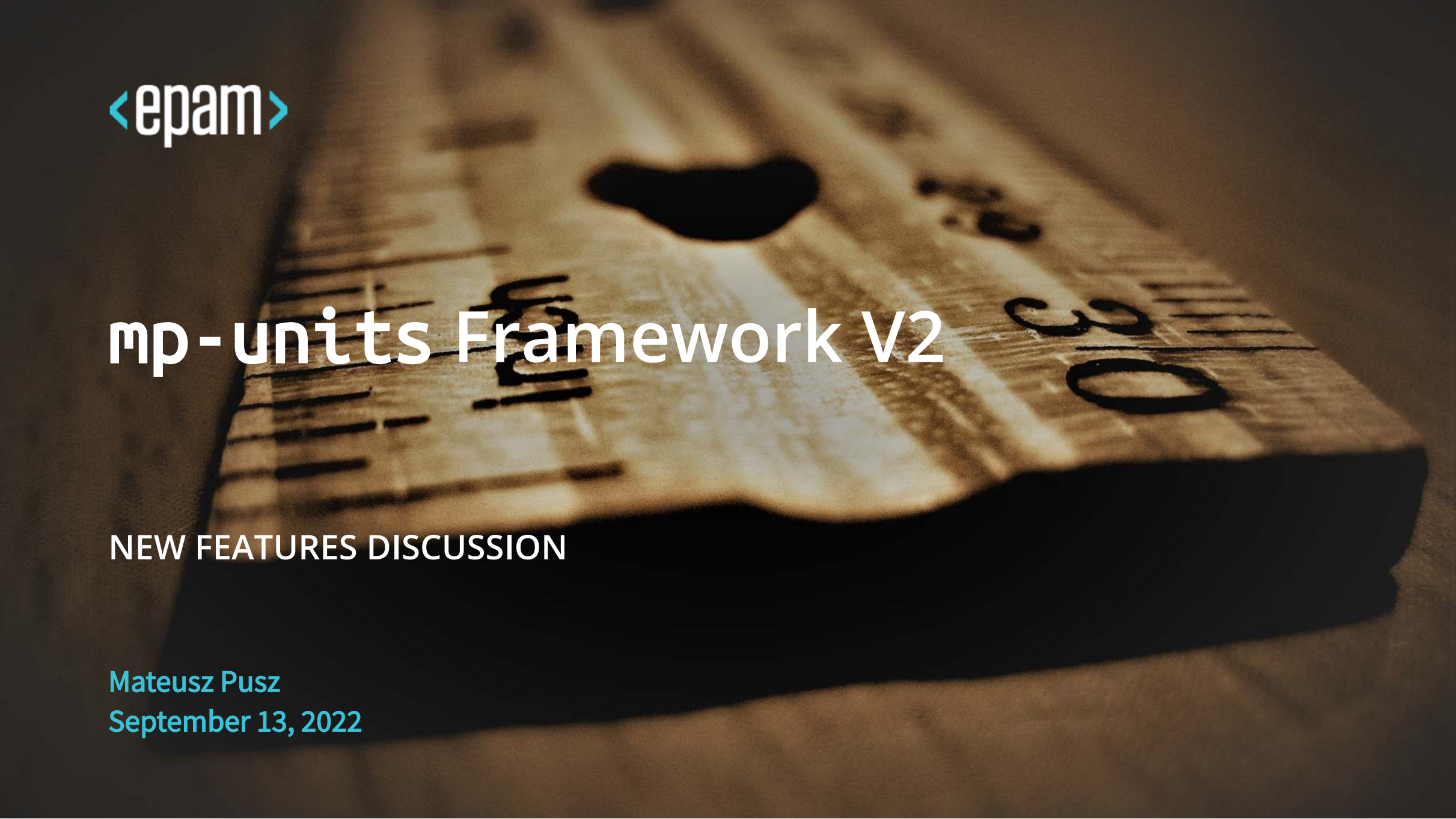

mp-units Framework V2

NEW FEATURES DISCUSSION

Mateusz Pusz

September 13, 2022

Motivation for change/breakage

1. The Downcasting Facility issues
2. Issues not addressable with the current framework
3. Verbose to define (and standardize) systems
4. New features

The Downcasting Facility

An innovative powerful feature that is a cornerstone of the V1 design.

The Downcasting Facility

```cpp
#include <units/isq/si/length.h>
#include <units/isq/si/time.h>
#include <units/isq/si/speed.h>

using namespace units::isq;

constexpr Speed auto avg_speed(Length auto d, Time auto t)
{
  const auto s = d / t;
  std::cout << s << "\n";
  return s;
}
```

The Downcasting Facility

```cpp
#include <units/isq/si/length.h>
#include <units/isq/si/time.h>
#include <units/isq/si/speed.h>

using namespace units::isq;

constexpr Speed auto avg_speed(Length auto d, Time auto t)
{
  const auto s = d / t;
  std::cout << s << "\n";
  return s;
}
```

```cpp
using namespace units::isq::si::references;
auto s = avg_speed(140 * km, 2 * h);
```

The Downcasting Facility

```cpp
#include <units/isq/si/length.h>
#include <units/isq/si/time.h>
#include <units/isq/si/speed.h>

using namespace units::isq;

constexpr Speed auto avg_speed(Length auto d, Time auto t)
{
  const auto s = d / t;
  std::cout << s << "\n";
  return s;
}
```

```cpp
using namespace units::isq::si::references;
auto s = avg_speed(140 * km, 2 * h);
```

```
70 km/h
```

The Downcasting Facility

```cpp
#include <units/isq/si/length.h>
#include <units/isq/si/time.h>
#include <units/isq/si/speed.h>

using namespace units::isq;

constexpr Speed auto avg_speed(Length auto d, Time auto t)
{
  const auto s = d / t;
  std::cout << s << "\n";
  return s;
}
```

```cpp
using namespace units::isq::si::references;
auto s = avg_speed(140 * km, 2 * h);
```

```
(gdb) ptype s
type = class units::quantity<units::isq::si::dim_speed, units::isq::si::kilometre_per_hour, int>
[with D = units::isq::si::dim_speed, U = units::isq::si::kilometre_per_hour, Rep = int] {
...
```

The Downcasting Facility

- Introduces a 1-to-1 relationship between
  - long specialization of a generic class template (**`derived_dimension`** or **`scaled_unit`**)
  - nicely named user type

The Downcasting Facility

- Introduces a 1-to-1 relationship between
  - long specialization of a generic class template (**`derived_dimension`** or **`scaled_unit`**)
  - nicely named user type

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

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

The Downcasting Facility

- Introduces a 1-to-1 relationship between
  - long specialization of a generic class template (**`derived_dimension`** or **`scaled_unit`**)
  - nicely named user type

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

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

CRTP required to provide a nicely named child class type to the downcasting facility framework.

The Downcasting Facility: Issues

The Downcasting Facility: Issues

There are N-to-1 relationships in the physical units domain.

The Downcasting Facility: Issues

There are N-to-1 relationships in the physical units domain.

SAME DIMENSIONS

- Energy = $M\,L^2T^{-2}$; Torque = $M\,L^2T^{-2}$ (according to the SI)
- Frequency = $1 / Time$; Activity of radionuclides = $1 / Time$; Modulation rate = $1 / Time$

The Downcasting Facility: Issues

There are N-to-1 relationships in the physical units domain.

SAME DIMENSIONS

- Energy = $M\,L^2T^{-2}$; Torque = $M\,L^2T^{-2}$ (according to the SI)
- Frequency = $1 / Time$; Activity of radionuclides = $1 / Time$; Modulation rate = $1 / Time$

SAME UNITS

- **Hz** = $1/s$; **Bq** = $1/s$; **baud** = $1/s$; $1/s$ is a derived unit by itself as well

The Downcasting Facility: Issues

There are N-to-1 relationships in the physical units domain.

SAME DIMENSIONS

- Energy = $M\,L^2T^{-2}$; Torque = $M\,L^2T^{-2}$ (according to the SI)
- Frequency = 1 / Time; Activity of radionuclides = 1 / Time; Modulation rate = 1 / Time

SAME UNITS

- **Hz** = 1/s; **Bq** = 1/s; **baud** = 1/s; 1/s is a derived unit by itself as well

Compile-time errors when both definitions found.

The Downcasting Facility: Issues

- Hacks as workarounds

```
struct baud : alias_unit<si::hertz, "Bd"> {};
struct kilobaud : prefixed_alias_unit<si::kilohertz, si::kilo, baud> {};
// ...

using dim_modulation_rate = si::dim_frequency;
```

The Downcasting Facility: Issues

- Hacks as workarounds

```
struct baud : alias_unit<si::hertz, "Bd"> {};
struct kilobaud : prefixed_alias_unit<si::kilohertz, si::kilo, baud> {};
// ...

using dim_modulation_rate = si::dim_frequency;
```

- **`si::dim_frequency`** always in the compilation errors
- Prints **Hz** on temporary results (if not explicitly converted to **baud**)

The Downcasting Facility: Issues

```
#include <units/isq/si/length.h>
#include <units/isq/si/time.h>
#include <units/isq/si/speed.h>

using namespace units::isq;

constexpr Speed auto avg_speed(Length auto d, Time auto t);
```

```
using namespace units::isq::si::references;
auto s = avg_speed(140 * km, 2 * h);
```

The Downcasting Facility: Issues

```
#include <units/isq/si/length.h>
#include <units/isq/si/time.h>
#include <units/isq/dimensions/speed.h>  // include what you use

using namespace units::isq;

constexpr Speed auto avg_speed(Length auto d, Time auto t);
```

```
using namespace units::isq::si::references;
auto s = avg_speed(140 * km, 2 * h);
```

The Downcasting Facility: Issues

```
#include <units/isq/si/length.h>
#include <units/isq/si/time.h>
#include <units/isq/dimensions/speed.h>  // include what you use

using namespace units::isq;

constexpr Speed auto avg_speed(Length auto d, Time auto t);
```

```
using namespace units::isq::si::references;
auto s = avg_speed(140 * km, 2 * h);
```

```
70 [5/18] m/s
```

```
units::quantity<units::unknown_dimension<units::exponent<units::isq::si::dim_length, 1l, 1l>,
units::exponent<units::isq::si::dim_time, -1l, 1l> >, units::scaled_unit<units::magnitude<
units::base_power<long>{2l, units::ratio{-1l, 1l}}, units::base_power<long>{3l,
units::ratio{-2l, 1l}}, units::base_power<long>{5l, units::ratio{1l, 1l}}>{},
units::unknown_coherent_unit<units::exponent<units::isq::si::dim_length, 1l, 1l>,
units::exponent<units::isq::si::dim_time, -1l, 1l> > >, int>
```

The Downcasting Facility: Issues

avg_speed.h

```
#include <units/isq/dimensions/speed.h>
#include <units/isq/si/length.h>
#include <units/isq/si/time.h>

constexpr units::isq::Speed auto avg_speed(units::isq::Length auto d, units::isq::Time auto t);
```

The Downcasting Facility: Issues

avg_speed.h

```
#include <units/isq/dimensions/speed.h>
#include <units/isq/si/length.h>
#include <units/isq/si/time.h>

constexpr units::isq::Speed auto avg_speed(units::isq::Length auto d, units::isq::Time auto t);
```

a.cpp

```
#include <units/isq/si/speed.h>
#include "avg_speed.h"

using namespace units::isq::si::references;
auto s = avg_speed(140 * km, 2 * h);
```

The Downcasting Facility: Issues

avg_speed.h

```
#include <units/isq/dimensions/speed.h>
#include <units/isq/si/length.h>
#include <units/isq/si/time.h>

constexpr units::isq::Speed auto avg_speed(units::isq::Length auto d, units::isq::Time auto t);
```

a.cpp

```
#include <units/isq/si/speed.h>
#include "avg_speed.h"

using namespace units::isq::si::references;
auto s = avg_speed(140 * km, 2 * h);
```

b.cpp

```
#include "avg_speed.h"

using namespace units::isq::si::references;
auto s = avg_speed(140 * km, 2 * h);
```

The Downcasting Facility: Issues

avg_speed.h

```
#include <units/isq/dimensions/speed.h>
#include <units/isq/si/length.h>
#include <units/isq/si/time.h>

constexpr units::isq::Speed auto avg_speed(units::isq::Length auto d, units::isq::Time auto t);
```

a.cpp

```
#include <units/isq/si/speed.h>
#include "avg_speed.h"

using namespace units::isq::si::references;
auto s = avg_speed(140 * km, 2 * h);
```

b.cpp

```
#include "avg_speed.h"

using namespace units::isq::si::references;
auto s = avg_speed(140 * km, 2 * h);
```

ODR violation!

NOTE: This is how some existing customization points behave!

ab.h

```
struct A { int value; };
struct B { int value; A* a; };

std::ostream& operator<<(std::ostream& os, const B& b)
{ return os << "[" << b.value << ", " << b.a->value << "]"; }
```

NOTE: This is how some existing customization points behave!

ab.h

```
struct A { int value; };
struct B { int value; A* a; };

std::ostream& operator<<(std::ostream& os, const B& b)
{ return os << "[" << b.value << ", " << b.a->value << "]"; }
```

file_1.cpp

```
#include "ab.h"

void swap(B& lhs, B& rhs) noexcept
{ std::ranges::swap(lhs.value, rhs.value); }

void foo()
{
  A a1{1}, a2{2};
  B b1{1, &a1}, b2{2, &a2};
  std::ranges::swap(b1, b2);
  std::cout << "b1: " << b1 << ", b2: " << b2 << "\n";
}
```

```
b1: [2, 1], b2: [1, 2]
```

NOTE: This is how some existing customization points behave!

ab.h

```
struct A { int value; };
struct B { int value; A* a; };

std::ostream& operator<<(std::ostream& os, const B& b)
{ return os << "[" << b.value << ", " << b.a->value << "]"; }
```

file_1.cpp

```
#include "ab.h"

void swap(B& lhs, B& rhs) noexcept
{ std::ranges::swap(lhs.value, rhs.value); }

void foo()
{
  A a1{1}, a2{2};
  B b1{1, &a1}, b2{2, &a2};
  std::ranges::swap(b1, b2);
  std::cout << "b1: " << b1 << ", b2: " << b2 << "\n";
}
```

```
b1: [2, 1], b2: [1, 2]
```

file_2.cpp

```
#include "ab.h"



void boo()
{
  A a1{1}, a2{2};
  B b1{1, &a1}, b2{2, &a2};
  std::ranges::swap(b1, b2);
  std::cout << "b1: " << b1 << ", b2: " << b2 << "\n";
}
```

```
b1: [2, 2], b2: [1, 1]
```

The Downcasting Facility: Issues

- Users reported problems with the **inability to define a multiple copies** of custom scaled units

```
namespace device_1_specific {
  struct device_1_specific_unit : units::named_scaled_unit<...> {};
}

namespace device_2_specific {
  struct device_2_specific_unit : units::named_scaled_unit<...> {};
}
```

The Downcasting Facility: Issues

- Users reported problems with the **inability to define a multiple copies** of custom scaled units

```
namespace device_1_specific {
  struct device_1_specific_unit : units::named_scaled_unit<...> {};
}

namespace device_2_specific {
  struct device_2_specific_unit : units::named_scaled_unit<...> {};
}
```

- The facility can recover the type but **cannot recover variable names**, which limits class NTTPs usage

Issues not addressable with the current framework

HARD TO UNDERSTAND UNKNOWN QUANTITIES

```
auto bad = 1 / si::length<si::kilometre>(50);
```

```
units::quantity<units::unknown_dimension<units::exponent<units::isq::si::dim_length, -1, 1> >,
units::scaled_unit<units::magnitude<units::base_power<long int>{2, units::ratio{-3, 1, 0}},
units::base_power<long int>{5, units::ratio{-3, 1, 0}}>(), units::unknown_coherent_unit>, double>
```

- Initially types were shorter and easier to understand
- Library extensions addressing some corner cases like radians and degrees made types much longer

Issues not addressable with the current framework

- Issues with quantity creation helpers
  - regular creation (`quantity<dim_speed, metre_per_second, double>`)
  - dimension aliases (`speed<metre_per_second, double>`)
  - user defined literals (`1._q_m_per_s`)
  - quantity references (`1. * (m / s)`)
  - unit aliases (`m_per_s{1.}`)
  - no single best choice

Issues not addressable with the current framework

- Issues with quantity creation helpers
  - regular creation (**`quantity<dim_speed, metre_per_second, double>`**)
  - dimension aliases (**`speed<metre_per_second, double>`**)
  - user defined literals (**`1._q_m_per_s`**)
  - quantity references (**`1. * (m / s)`**)
  - unit aliases (**`m_per_s{1.}`**)
  - no single best choice
- Dimensions always closely connected to coherent units (even if templated)
  - problems with dimension-specific concepts (**`QuantityOf<isq::si::dim_speed>`** and **`QuantityOfT<isq::dim_speed>`**)
  - standalone dimensional analysis not possible

Verbose to define (and standardize) systems

Even though the library is terse comparing to other products on the market that use macros to provide multiple definitions per one entity, there is still some room for improvement.

Verbose to define (and standardize) systems

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

struct kilogram_metre_sq_per_second : derived_unit<kilogram_metre_sq_per_second> {};

// ...

inline namespace literals {

constexpr auto operator"" _q_kg_m2_per_s(unsigned long long l)
{
  gsl_ExpectsAudit(std::in_range<std::int64_t>(l));
  return angular_momentum<kilogram_metre_sq_per_second, std::int64_t>(static_cast<std::int64_t>(l));
}
constexpr auto operator"" _q_kg_m2_per_s(long double l)
{
  return angular_momentum<kilogram_metre_sq_per_second, long double>(l);
}

}  // namespace literals
}  // namespace units::isq::si

namespace units::aliases::isq::si::inline angular_momentum {

template<Representation Rep = double>
using kg_m2_per_s = units::isq::si::angular_momentum<units::isq::si::kilogram_metre_sq_per_second, Rep>;

}  // namespace units::aliases::isq::si::inline angular_momentum
```

New features

TERSER DEFINITION

- The compiler errors readability constraints requires the **type and a value to have the same name**

```
struct length_dim : base_dimension<"L"> {};
inline constexpr length_dim length_dim;
```

New features

TERSER DEFINITION

- The compiler errors readability constraints requires the **type and a value to have the same name**

```
struct length_dim : base_dimension<"L"> {};
inline constexpr length_dim length_dim;
```

```
inline constexpr struct length_dim : base_dimension<"L"> {} length_dim;
```

New features

TERSER DEFINITION

- The compiler errors readability constraints requires the **type and a value to have the same name**

```
struct length_dim : base_dimension<"L"> {};
inline constexpr length_dim length_dim;
```

```
inline constexpr struct length_dim : base_dimension<"L"> {} length_dim;
```

Besides awkward definition, this does affect the users at all as they always work only with values. The types are only observable in the compilation errors and never used directly in the code.

New features

TERSER DEFINITION

```
namespace mp_units::isq {

inline constexpr struct length_dim : base_dimension<"L"> {} length_dim;
inline constexpr struct time_dim : base_dimension<"T"> {} time_dim;
inline constexpr struct frequency_dim : decltype(1 / time_dim) {} frequency_dim;
inline constexpr struct speed_dim : decltype(length_dim / time_dim) {} speed_dim;

} // namespace mp_units::isq
```

- No more downcasting facility and CRTP idiom
- ISQ dimensions are not templates anymore
- **`units::exponent`** no longer needed
  - derived dimension specification with mathematic operators on values
  - an unconventional usage of variables named as types

New features

TERSER DEFINITION

```
namespace mp_units::isq::si {

template<NamedUnit auto U>
struct kilo : prefixed_unit<"k", mag_power<10, 3>(), U> {};

inline constexpr struct metre : named_unit<"m"> {} metre;
inline constexpr struct second : named_unit<"s"> {} second;

inline constexpr struct kilometre : kilo<metre> {} kilometre;

}  // namespace mp_units::isq::si
```

- **`units::prefix`** abstraction no longer needed

New features

TERSER DEFINITION

```
namespace mp_units::isq::si {

inline constexpr struct hertz : named_unit<"Hz", 1 / second> {} hertz;

inline constexpr struct square_metre : derived_unit<decltype(metre * metre)> {} square_metre;

}  // namespace mp_units::isq::si
```

- Only named units and their prefixed versions need to be specified
  - **`square_metre`**, **`cubic_metre`**, **`second_squared`** provided for convenience
  - **`metre_per_second`** and other derived units not needed anymore
- No CRTP idiom usage anymore

New features

TERSER DEFINITION

```
namespace mp_units::isq::si::unit_symbols {

inline namespace length_units {
inline constexpr auto m = metre;
}

inline namespace time_units {
inline constexpr auto s = second;
}

inline namespace frequency_units {
inline constexpr auto Hz = hertz;
}

}  // namespace mp_units::isq::si::unit_symbols
```

- Dedicated namespace and sub-namespaces to limit shadowing issues
  - short symbols are just an alternative shorter way (contrary to the current references)

New features

TERSER DEFINITION

```
namespace namespace mp_units::isq::si {

inline constexpr struct length : system_reference<length_dim, metre> {} length;
inline constexpr struct time : system_reference<time_dim, second> {} time;
inline constexpr struct frequency : system_reference<frequency_dim, hertz> {} frequency;
inline constexpr struct speed : system_reference<speed_dim, metre / second> {} speed;

}  // namespace mp_units::isq::si
```

- **`system_reference`** binds the quantity dimension to a coherent unit for a specific system
- **`system_reference::operator[Unit]`** serves as a factory for **`reference`** which stores system reference and a specific unit for a quantity

New features

BETTER CONSTRUCTION

```
Frequency auto freq1 = 20 * frequency[hertz];
quantity<frequency[hertz]> freq2(20);
```

```
Speed auto speed1 = 20 * speed[metre / second];
quantity<speed[metre / second]> speed2(20);
```

New features

BETTER CONSTRUCTION

```
Frequency auto freq1 = 20 * frequency[hertz];
quantity<frequency[hertz]> freq2(20);
```

```
Speed auto speed1 = 20 * speed[metre / second];
quantity<speed[metre / second]> speed2(20);
```

```
Frequency auto freq1 = 20 * frequency[Hz];
quantity<frequency[Hz]> freq2(20);
```

```
Speed auto speed1 = 20 * speed[m / s];
quantity<speed[m / s]> speed2(20);
```

New features

BETTER CONSTRUCTION

```
Frequency auto freq1 = 20 * frequency[hertz];
quantity<frequency[hertz]> freq2(20);
```

```
Frequency auto freq1 = 20 * frequency[Hz];
quantity<frequency[Hz]> freq2(20);
```

```
Speed auto speed1 = 20 * speed[metre / second];
quantity<speed[metre / second]> speed2(20);
```

```
Speed auto speed1 = 20 * speed[m / s];
quantity<speed[m / s]> speed2(20);
```

- The same syntax for the type definition and multiplication syntax
- Supports both variables and compile-time literals
- Preserves user provided representation type
- Allows control over the verbosity
- Easy to compose derived units
- Easy to disambiguate
- Removes a lot of definition boilerplate

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·m/s
- kg·m·s^{-2}·m/s
- kg·m^{2}·s^{-2}/s
- kg·m^{2}·s^{-3}

New features

EQUIVALENT UNITS USAGE FOR A QUANTITY

```
auto p1 = quantity<power[W]>(42);
auto p2 = quantity<power[J / s]>(42);
auto p3 = quantity<power[N * m / s]>(42);
auto p4 = quantity<power[kg * m2 / s3]>(42);
```

New features

EQUIVALENT UNITS USAGE FOR A QUANTITY

```
auto p1 = quantity<power[W]>(42);
auto p2 = quantity<power[J / s]>(42);
auto p3 = quantity<power[N * m / s]>(42);
auto p4 = quantity<power[kg * m2 / s3]>(42);
```

PURE DIMENSIONAL ANALYSIS

```
static_assert(length_dim / length_dim == one_dim);
static_assert(1 / time_dim == frequency_dim);
static_assert(length_dim * length_dim == area_dim);
static_assert(area_dim * length_dim == volume_dim);
static_assert(length_dim / time_dim == speed_dim);
static_assert(acceleration_dim / speed_dim == frequency_dim);
static_assert(energy_dim / time_dim == power_dim);
```

New features

THE SAME SYNTAX USED EVERYWHERE

```
auto s1 = 90 * speed[km / h];
auto s2 = quantity<speed[km / h]>{90};
auto s3 = s2[m / h];  // when implicit conversion is allowed
auto s4 = quantity_cast<m / s>(s2);
```

```
static_assert(QuantityOf<km / h>(s1));                  // unit check
static_assert(QuantityOf<speed_dim>(s1));               // dimension check
static_assert(QuantityOf<length_dim / time_dim>(s1));   // equivalence check
static_assert(QuantityOf<speed[km / h]>(s1));           // dimension and unit check
static_assert(QuantityOf<speed_dim, km / h>(s1));       // dimension and unit check
```

New features

EXPRESSION TEMPLATES

```
auto speed = 20 * length[m] / (10 * time[s]);
```

```
mp_units::quantity<mp_units::reference<mp_units::derived_dimension<mp_units::isq::length_dim,
mp_units::per<mp_units::isq::time_dim>>, mp_units::derived_unit<mp_units::isq::si::metre,
mp_units::per<mp_units::isq::si::second>>>{}, int>
```

- Preserve original dimensions and units
- Easier to understand compilation errors
  - a huge improvement over previous quantities of **unknown_dimension** and scaled coherent units
- Much needed with the lack of the downcasting facility

Work In Progress

- The final **design is still not settled**
  - many options left to analyze
  - decide the best possible solutions
- **Reimplement all**
  - systems
  - examples
  - unit tests
- Cross-reference against all known issues
- Implement **`quantity_point`** offsets
- Consider starting the library standardization


CAUTION
Programming
is addictive
(and too much fun)