D.1. The <chrono> header

The `<chrono>` header provides classes for representing points in time and `durations` and clock classes, which act as a source of `time_points`. Each clock has an `is_steady` static data member, which indicates whether it's a *steady* clock that advances at a uniform rate (and can't be adjusted). The `std::chrono::steady_clock` class is the only clock guaranteed to be *steady*.

Header contents

```
namespace std
{
   namespace chrono
   {
       template<typename Rep,typename Period = ratio<1>>
       class duration;
       template<
            typename Clock,
            typename Duration = typename Clock::duration>
       class time_point;
       class system_clock;
       class steady_clock;
       typedef unspecified-clock-type high_resolution_clock;
   }
}
```

D.1.1. std::chrono::duration class template

The `std::chrono::duration` class template provides a facility for representing durations. The template parameters `Rep` and `Period` are the data type to store the duration value and an instantiation of the `std::ratio` class template indicating the length of time (as a fraction of a second) between successive "ticks," respectively. Thus `std::chrono::duration<int, std::milli>` is a count of milliseconds stored in a value of type `int`, whereas `std::chrono::duration<short, std::ratio<1,50>>` is a count of fiftieths of a second stored in a value of type `short`, and `std::chrono::duration <long long, std::ratio<60,1>>` is a count of minutes stored in a value of type `long long`.

Class definition

```
template <class Rep, class Period=ratio<1> >
class duration
{
public:
    typedef Rep rep;
    typedef Period period;

    constexpr duration() = default;
    ~duration() = default;
```

```
    duration(const duration&) = default;
    duration& operator=(const duration&) = default;

    template <class Rep2>
    constexpr explicit duration(const Rep2& r);

    template <class Rep2, class Period2>
    constexpr duration(const duration<Rep2, Period2>& d);

    constexpr rep count() const;
    constexpr duration operator+() const;
    constexpr duration operator-() const;
    duration& operator++();
    duration operator++(int);
    duration& operator--();
    duration operator--(int);
    duration& operator+=(const duration& d);
    duration& operator-=(const duration& d);
    duration& operator*=(const rep& rhs);
    duration& operator/=(const rep& rhs);
    duration& operator%=(const rep& rhs);
    duration& operator%=(const duration& rhs);
    static constexpr duration zero();
    static constexpr duration min();
    static constexpr duration max();
};

template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator==(
     const duration<Rep1, Period1>& lhs,
     const duration<Rep2, Period2>& rhs);

template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator!=(
     const duration<Rep1, Period1>& lhs,
     const duration<Rep2, Period2>& rhs);

template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator<(
     const duration<Rep1, Period1>& lhs,
     const duration<Rep2, Period2>& rhs);

template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator<=(
     const duration<Rep1, Period1>& lhs,
     const duration<Rep2, Period2>& rhs);

template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator>(
     const duration<Rep1, Period1>& lhs,
     const duration<Rep2, Period2>& rhs);

template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator>=(
```

```
      const duration<Rep1, Period1>& lhs,
      const duration<Rep2, Period2>& rhs);

template <class ToDuration, class Rep, class Period>
constexpr ToDuration duration_cast(const duration<Rep, Period>& d);
```

Requirements

`Rep` must be a built-in numeric type, or a number-like user-defined type. `Period` must be an instantiation of `std::ratio<>`.

Std::Chrono::Duration::Rep Typedef

This is a typedef for the type used to hold the number of ticks in a `duration` value.

Declaration

```
typedef Rep rep;
```

`rep` is the type of value used to hold the internal representation of the `duration` object.

Std::Chrono::Duration::Period Typedef

This typedef is for an instantiation of the `std::ratio` class template that specifies the fraction of a second represented by the duration count. For example, if `period` is `std::ratio<1,50>`, a `duration` value with a `count()` of *N* represents *N* fiftieths of a second.

Declaration

```
typedef Period period;
```

Std::Chrono::Duration Default Constructor

Constructs a `std::chrono::duration` instance with a default value.

Declaration

```
constexpr duration() = default;
```

Effects

The internal value of the `duration` (of type `rep`) is default initialized.

Std::Chrono::Duration Converting Constructor From a Count Value

Constructs a `std::chrono::duration` instance with a specified count.

Declaration

```
template <class Rep2>
constexpr explicit duration(const Rep2& r);
```

Effects

The internal value of the `duration` object is initialized with `static_cast<rep>(r)`.

Requirements

This constructor only participates in overload resolution if `Rep2` is implicitly convertible to `Rep` and either `Rep` is a floating point type or `Rep2` is *not* a floating point type.

Postcondition

```
this->count()==static_cast<rep>(r)
```

Std::Chrono::Duration Converting Constructor from Another Std::Chrono::Duration Value

Constructs a `std::chrono::duration` instance by scaling the count value of another `std::chrono::duration` object.

Declaration

```
template <class Rep2, class Period2>
constexpr duration(const duration<Rep2,Period2>& d);
```

Effects

The internal value of the `duration` object is initialized with `duration_cast<duration<Rep,Period>>(d).count()`.

Requirements

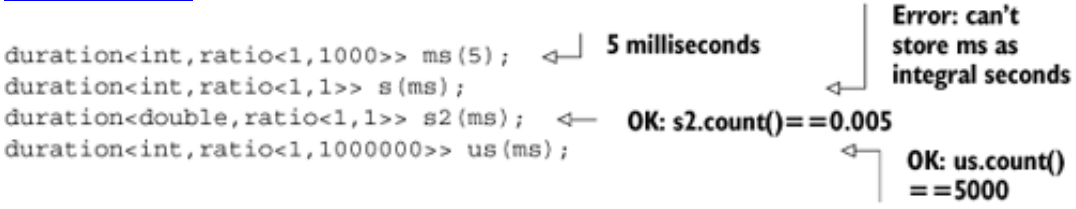
This constructor only participates in overload resolution if `Rep` is a floating point type or `Rep2` is *not* a floating point type and `Period2` is a whole number multiple of `Period` (that is, `ratio_divide<Period2,Period>::den==1`). This avoids accidental truncation (and corresponding loss of precision) from storing a duration with small periods in a variable representing a duration with a longer period.

Postcondition

```
this->count()==duration_cast<duration<Rep,Period>>(d).count()
```

Examples

[View full size image]



Std::Chrono::Duration::Count Member Function

Retrieves the value of the duration.

Declaration

```
constexpr rep count() const;
```

Returns

The internal value of the `duration` object, as a value of type `rep`.

Std::Chrono::Duration::Operator+ Unary Plus Operator

This is a no-op: it just returns a copy of `*this`.

Declaration

```
constexpr duration operator+() const;
```

Returns

```
*this
```

Std::Chrono::Duration::Operator-Unary Minus Operator

Returns a duration such that the `count()` value is the negative value of `this->count()`.

Declaration

```
constexpr duration operator-() const;
```

Returns

```
duration(-this->count());
```

Std::Chrono::Duration::Operator++ Pre-Increment Operator

Increments the internal count.

Declaration

```
duration& operator++();
```

Effects

```
++this->internal_count;
```

Returns

```
*this
```

Std::Chrono::Duration::Operator++ Post-Increment Operator

Increments the internal count and return the value of `*this` prior to the increment.

Declaration

```
duration operator++(int);
```

Effects

```
duration temp(*this);
++(*this);
return temp;
```

Std::Chrono::Duration::Operator-- Pre-Decrement Operator

Decrements the internal count.

Declaration

```
duration& operator--();
```

Effects

```
--this->internal_count;
```

Returns

```
*this
```

Std::Chrono::Duration::Operator-- Post-Decrement Operator

Decrements the internal count and return the value of `*this` prior to the decrement.

Declaration

```
duration operator--(int);
```

Effects

```
duration temp(*this);
--(*this);
return temp;
```

Std::Chrono::Duration::Operator+= Compound Assignment Operator

Adds the count for another `duration` object to the internal count for `*this`.

Declaration

```
duration& operator+=(duration const& other);
```

Effects

```
internal_count+=other.count();
```

Returns

```
*this
```

Std::Chrono::Duration::Operator-= Compound Assignment Operator

Subtracts the count for another `duration` object from the internal count for `*this`.

Declaration

```
duration& operator-=(duration const& other);
```

Effects

```
internal_count-=other.count();
```

Returns

```
*this
```

Std::Chrono::Duration::Operator= Compound Assignment Operator*

Multiplies the internal count for `*this` by the specified value.

Declaration

```
duration& operator*=(rep const& rhs);
```

Effects

```
internal_count*=rhs;
```

Returns

```
*this
```

Std::Chrono::Duration::Operator/= Compound Assignment Operator

Divides the internal count for `*this` by the specified value.

Declaration

```
duration& operator/=(rep const& rhs);
```

Effects

```
internal_count/=rhs;
```

Returns

```
*this
```

Std::Chrono::Duration::Operator%= Compound Assignment Operator

Adjusts the internal count for `*this` to be the remainder when divided by the specified value.

Declaration

```
duration& operator%=(rep const& rhs);
```

Effects

```
internal_count%=rhs;
```

Returns

```
*this
```

Std::Chrono::Duration::Operator%= Compound Assignment Operator

Adjusts the internal count for `*this` to be the remainder when divided by the count of the other `duration` object.

Declaration

```
duration& operator%=(duration const& rhs);
```

Effects

```
internal_count%=rhs.count();
```

Returns

```
*this
```

Std::Chrono::Duration::Zero Static Member Function

Returns a `duration` object representing a value of zero.

Declaration

```
constexpr duration zero();
```

Returns

```
duration(duration_values<rep>::zero());
```

Std::Chrono::Duration::Min Static Member Function

Returns a `duration` object holding the minimum possible value for the specified instantiation.

Declaration

```
constexpr duration min();
```

Returns

```
duration(duration_values<rep>::min());
```

Std::Chrono::Duration::Max Static Member Function

Returns a `duration` object holding the maximum possible value for the specified instantiation.

Declaration

```
constexpr duration max();
```

Returns

```
duration(duration_values<rep>::max());
```

Std::Chrono::Duration Equality Comparison Operator

Compares two `duration` objects for equality, even if they have distinct representations and/or periods.

Declaration

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator==(
     const duration<Rep1, Period1>& lhs,
     const duration<Rep2, Period2>& rhs);
```

Requirements

Either `lhs` must be implicitly convertible to `rhs`, or vice versa. If neither can be implicitly converted to the other, or they are distinct instantiations of `duration` but each can implicitly convert to the other, the expression is ill formed.

Effects

If `CommonDuration` is a synonym for `std::common_type< duration< Rep1, Period1>, duration< Rep2, Period2>>::type`, then `lhs==rhs` returns `CommonDuration(lhs).count()==CommonDuration(rhs).count()`.

Std::Chrono::Duration Inequality Comparison Operator

Compares two `duration` objects for inequality, even if they have distinct representations and/or periods.

Declaration

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator!=(
     const duration<Rep1, Period1>& lhs,
     const duration<Rep2, Period2>& rhs);
```

Requirements

Either `lhs` must be implicitly convertible to `rhs`, or vice versa. If neither can be implicitly converted to the other, or they are distinct instantiations of `duration` but each can implicitly convert to the other, the expression is ill formed.

Returns

```
!(lhs==rhs)
```

Std::Chrono::Duration Less-Than Comparison Operator

Compares two `duration` objects to see if one is less than the other, even if they have distinct representations and/or periods.

Declaration

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator<(
     const duration<Rep1, Period1>& lhs,
     const duration<Rep2, Period2>& rhs);
```

Requirements

Either `lhs` must be implicitly convertible to `rhs`, or vice versa. If neither can be implicitly converted to the other, or they are distinct instantiations of `duration` but each can implicitly convert to the other, the expression is ill formed.

Effects

If `CommonDuration` is a synonym for `std::common_type< duration< Rep1, Period1>, duration< Rep2, Period2>>::type`, then `lhs<rhs` returns `CommonDuration(lhs).count() <CommonDuration(rhs).count()`.

Std::Chrono::Duration Greater-Than Comparison Operator

Compares two `duration` objects to see if one is greater than the other, even if they have distinct representations and/or periods.

Declaration

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator>(
     const duration<Rep1, Period1>& lhs,
     const duration<Rep2, Period2>& rhs);
```

Requirements

Either `lhs` must be implicitly convertible to `rhs`, or vice versa. If neither can be implicitly converted to the other, or they are distinct instantiations of `duration` but each can implicitly convert to the other, the expression is ill formed.

Returns

```
rhs<lhs
```

Std::Chrono::Duration Less-Than-or-Equals Comparison Operator

Compares two `duration` objects to see if one is less than or equal to the other, even if they have distinct representations and/or periods.

Declaration

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator<=(
     const duration<Rep1, Period1>& lhs,
     const duration<Rep2, Period2>& rhs);
```

Requirements

Either `lhs` must be implicitly convertible to `rhs`, or vice versa. If neither can be implicitly converted to the other, or they are distinct instantiations of `duration` but each can implicitly convert to the other, the expression is ill formed.

Returns

```
!(rhs<lhs)
```

Std::Chrono::Duration Greater-Than-or-Equals Comparison Operator

Compares two `duration` objects to see if one is greater than or equal to the other, even if they have distinct representations and/or periods.

Declaration

```
template <class Rep1, class Period1, class Rep2, class Period2>
constexpr bool operator>=(
     const duration<Rep1, Period1>& lhs,
     const duration<Rep2, Period2>& rhs);
```

Requirements

Either `lhs` must be implicitly convertible to `rhs`, or vice versa. If neither can be implicitly converted to the other, or they are distinct instantiations of `duration` but each can implicitly convert to the other, the expression is ill formed.

Returns

```
!(lhs<rhs)
```

Std::Chrono::Duration_Cast Nonmember Function

Explicitly converts a `std::chrono::duration` object to a specific `std::chrono::duration` instantiation.

Declaration

```
template <class ToDuration, class Rep, class Period>
```

```
constexpr ToDuration duration_cast(const duration<Rep, Period>& d);
```

Requirements

`ToDuration` must be an instantiation of `std::chrono::duration`.

Returns

The duration `d` converted to the duration type specified by `ToDuration`. This is done in such a way as to minimize any loss of precision resulting from conversions between different scales and representation types.

D.1.2. std::chrono::time_point class template

The `std::chrono::time_point` class template represents a point in time, as measured by a particular clock. It's specified as a duration since the *epoch* of that particular clock. The template parameter `Clock` identifies the clock (each distinct clock must have a unique type), whereas the `Duration` template parameter is the type to use for measuring the duration since the epoch and must be an instantiation of the `std::chrono::duration` class template. The `Duration` defaults to the default duration type of the `Clock`.

Class definition

```
template <class Clock,class Duration = typename Clock::duration>
class time_point
{
public:
    typedef Clock clock;
    typedef Duration duration;
    typedef typename duration::rep rep;
    typedef typename duration::period period;

    time_point();
    explicit time_point(const duration& d);

    template <class Duration2>
    time_point(const time_point<clock, Duration2>& t);

    duration time_since_epoch() const;

    time_point& operator+=(const duration& d);
    time_point& operator-=(const duration& d);

    static constexpr time_point min();
    static constexpr time_point max();
};
```

Std::Chrono::Time_Point Default Constructor

Constructs a `time_point` representing the epoch of the associated `Clock`; the internal duration is initialized with `Duration::zero()`.

Declaration

```
time_point();
```

Postcondition

For a newly default-constructed `time_point` object `tp`, `tp.time_since_epoch() ==tp::duration::zero()`.

Std::Chrono::Time_Point Duration Constructor

Constructs a `time_point` representing the specified duration since the epoch of the associated `Clock`.

Declaration

```
explicit time_point(const duration& d);
```

Postcondition

For a `time_point` object `tp`, constucted with `tp(d)` for some duration `d`, `tp.time_since_epoch()==d`.

Std::Chrono::Time_Point Conversion Constructor

Constructs a `time_point` object from another `time_point` object with the same `Clock` but a distinct `Duration`.

Declaration

```
template <class Duration2>
time_point(const time_point<clock, Duration2>& t);
```

Requirements

`Duration2` shall be implicitly convertible to `Duration`.

Effects

As-if `time_point(t.time_since_epoch())`

The value returned from `t.time_since_epoch()` is implicitly converted to an object of type `Duration`, and that value is stored in the newly constructed `time_point` object.

Std::Chrono::Time_Point::Time_Since_Epoch Member Function

Retrieves the duration since the clock epoch for a particular `time_point` object.

Declaration

```
duration time_since_epoch() const;
```

Returns

The `duration` value stored in `*this`.

Std::Chrono::Time_Point::Operator+= Compound Assignment Operator

Adds the specified `duration` to the value stored in the specified `time_point` object.

Declaration

```
time_point& operator+=(const duration& d);
```

Effects

Adds `d` to the internal `duration` object of `*this`, as-if

```
this->internal_duration += d;
```

Returns

```
*this
```

Std::Chrono::Time_Point::Operator-= Compound Assignment Operator

Subtracts the specified `duration` from the value stored in the specified `time_point` object.

Declaration

```
time_point& operator-=(const duration& d);
```

Effects

Subtracts `d` from the internal `duration` object of `*this`, as-if

```
this->internal_duration -= d;
```

Returns

```
*this
```

Std::Chrono::Time_Point::Min Static Member Function

Obtains a `time_point` object representing the minimum possible value for its type.

Declaration

```
static constexpr time_point min();
```

Returns

```
time_point(time_point::duration::min()) (see 11.1.1.15)
```

Std::Chrono::Time_Point::Max Static Member Function

Obtains a `time_point` object representing the maximum possible value for its type.

Declaration

```
static constexpr time_point max();
```

Returns

`time_point(time_point::duration::max()) (see 11.1.1.16)`

D.1.3. std::chrono::system_clock class

The `std::chrono::system_clock` class provides a means of obtaining the current wall-clock time from the systemwide real-time clock. The current time can be obtained by calling `std::chrono::system_clock::now()`. Instances of `std::chrono::system_clock::time_point` can be converted to and from `time_t` with the `std::chrono::system_clock::to_time_t()` and `std::chrono::system_clock::to_time_point()` functions. The system clock isn't *steady*, so a subsequent call to `std::chrono::system_clock::now()` may return an earlier time than a previous call (for example, if the operating system clock is manually adjusted or synchronized with an external clock).

Class definition

```
class system_clock
{
public:
    typedef unspecified-integral-type rep;
    typedef std::ratio<unspecified,unspecified> period;
    typedef std::chrono::duration<rep, period> duration;
    typedef std::chrono::time_point<system_clock> time_point;
    static const bool is_steady=unspecified;

    static time_point now() noexcept;

    static time_t to_time_t(const time_point& t) noexcept;
    static time_point from_time_t(time_t t) noexcept;

};
```

Std::Chrono::System_Clock::Rep Typedef

A typedef for an integral type used to hold the number of ticks in a `duration` value.

Declaration

```
typedef unspecified-integral-type rep;
```

Std::Chrono::System_Clock::Period Typedef

A typedef for an instantiation of the `std::ratio` class template that specifies the smallest number of seconds (or fractions of a second) between distinct values of `duration` or `time_point`. The `period` specifies the *precision* of the clock, not the tick frequency.

Declaration

```
typedef std::ratio<unspecified, unspecified> period;
```

Std::Chrono::System_Clock::Duration Typedef

An instantiation of the `std::chrono::duration` class template that can hold the difference between any two time points returned by the systemwide real-time clock.

Declaration

```
typedef std::chrono::duration<
    std::chrono::system_clock::rep,
    std::chrono::system_clock::period> duration;
```

Std::Chrono::System_Clock::Time_Point Typedef

An instantiation of the `std::chrono::time_point` class template that can hold time points returned by the systemwide real-time clock.

Declaration

```
typedef std::chrono::time_point<std::chrono::system_clock> time_point;
```

Std::Chrono::System_Clock::Now Static Member Function

Obtains the current wall-clock time from the systemwide real-time clock.

Declaration

```
time_point now() noexcept;
```

Returns

A `time_point` representing the current time of the systemwide real-time clock.

Throws

An exception of type `std::system_error` if an error occurs.

Std::Chrono::System_Clock::To_Time_T Static Member Function

Converts an instance of `time_point` to `time_t`.

Declaration

```
time_t to_time_t(time_point const& t) noexcept;
```

Returns

A `time_t` value that represents the same point in time as `t` rounded or truncated to seconds precision.

Throws

An exception of type `std::system_error` if an error occurs.

Std::Chrono::System_Clock::From_Time_T Static Member Function

Converts an instance of `time_t` to `time_point`.

Declaration

```
time_point from_time_t(time_t const& t) noexcept;
```

Returns

A `time_point` value that represents the same point in time as `t`.

Throws

An exception of type `std::system_error` if an error occurs.

D.1.4. std::chrono::steady_clock class

The `std::chrono::steady_clock` class provides access to the systemwide steady clock. The current time can be obtained by calling `std::chrono::steady_clock::now()`. There is no fixed relationship between values returned by `std::chrono::steady_clock::now()` and wall-clock time. A steady clock can't go backwards, so if one call to `std::chrono::steady_clock::now()` happens-before another call to `std::chrono::steady_clock::now()`, the second call must return a time point equal to or later than the first. The clock advances at a uniform rate as far as possible.

Class definition

```
class steady_clock
{
public:
    typedef unspecified-integral-type rep;
    typedef std::ratio<
        unspecified,unspecified> period;
    typedef std::chrono::duration<rep, period> duration;
    typedef std::chrono::time_point<steady_clock>
        time_point;
    static const bool is_steady=true;

    static time_point now() noexcept;
};
```

Std::Chrono::Steady_Clock::Rep Typedef

This typedef is for an integral type used to hold the number of ticks in a `duration` value.

Declaration

```
typedef unspecified-integral-type rep;
```

Std::Chrono::Steady_Clock::Period Typedef

This is a typedef for an instantiation of the `std::ratio` class template that specifies the smallest number of seconds (or fractions of a second) between distinct values of `duration` or `time_point`. The `period` specifies the *precision* of the clock, not the tick frequency.

Declaration

```
typedef std::ratio<unspecified, unspecified> period;
```

Std::Chrono::Steady_Clock::Duration Typedef

This is an instantiation of the `std::chrono::duration` class template that can hold the difference between any two time points returned by the systemwide steady clock.

Declaration

```
typedef std::chrono::duration<
    std::chrono::steady_clock::rep,
    std::chrono::steady_clock::period> duration;
```

Std::Chrono::Steady_Clock::Time_Point Typedef

This instantiation of the `std::chrono::time_point` class template can hold time points returned by the systemwide steady clock.

Declaration

```
typedef std::chrono::time_point<std::chrono::steady_clock> time_point;
```

Std::Chrono::Steady_Clock::Now Static Member Function

Obtains the current time from the systemwide steady clock.

Declaration

```
time_point now() noexcept;
```

Returns

A `time_point` representing the current time of the systemwide steady clock.

Throws

An exception of type `std::system_error` if an error occurs.

Synchronization

If one call to `std::chrono::steady_clock::now()` happens-before another, the `time_point` returned by the first call shall compare less-than or equal-to the `time_point` returned by the second call.

D.1.5. std::chrono::high_resolution_clock typedef

The `std::chrono::high_resolution_clock` class provides access to the systemwide clock with the highest resolution. As for all clocks, the current time can be obtained by calling `std::chrono::high_resolution_clock::now()`. `std::chrono::high_resolution_clock` may be a typedef for the `std::chrono::system_clock` class or `std::chrono::steady_clock` class, or it may be a separate type.

Although `std::chrono::high_resolution_clock` has the highest resolution of all the library-supplied clocks, `std::chrono::high_resolution_clock::now()` still takes a finite amount of time.

You must take care to account for the overhead of calling `std::chrono::high_resolution_clock::now()` when timing short operations.

Class definition

```
class high_resolution_clock
{
public:
    typedef unspecified-integral-type rep;
    typedef std::ratio<
        unspecified,unspecified> period;
    typedef std::chrono::duration<rep, period> duration;
    typedef std::chrono::time_point<
        unspecified> time_point;
    static const bool is_steady=unspecified;

    static time_point now() noexcept;
};
```