D.2. <condition_variable> header

The `<condition_variable>` header provides condition variables. These are basic-level synchronization mechanisms that allow a thread to block until notified that some condition is true or a timeout period has elapsed.

Header contents

```
namespace std
{
    enum class cv_status { timeout, no_timeout };

    class condition_variable;
    class condition_variable_any;
}
```

D.2.1. std::condition_variable class

The `std::condition_variable` class allows a thread to wait for a condition to become `true`.

Instances of `std::condition_variable` aren't `CopyAssignable`, `CopyConstructible`, `MoveAssignable`, or `MoveConstructible`.

Class definition

```
class condition_variable
{
public:
    condition_variable();
    ~condition_variable();

    condition_variable(condition_variable const& ) = delete;
    condition_variable& operator=(condition_variable const& ) = delete;

    void notify_one() noexcept;
    void notify_all() noexcept;

    void wait(std::unique_lock<std::mutex>& lock);

    template <typename Predicate>
    void wait(std::unique_lock<std::mutex>& lock,Predicate pred);

    template <typename Clock, typename Duration>
    cv_status wait_until(
        std::unique_lock<std::mutex>& lock,
        const std::chrono::time_point<Clock, Duration>& absolute_time);

    template <typename Clock, typename Duration, typename Predicate>
    bool wait_until(
```

```
            std::unique_lock<std::mutex>& lock,
            const std::chrono::time_point<Clock, Duration>& absolute_time,
            Predicate pred);
        template <typename Rep, typename Period>
        cv_status wait_for(
            std::unique_lock<std::mutex>& lock,
            const std::chrono::duration<Rep, Period>& relative_time);

        template <typename Rep, typename Period, typename Predicate>
        bool wait_for(
            std::unique_lock<std::mutex>& lock,
            const std::chrono::duration<Rep, Period>& relative_time,
            Predicate pred);
    };

    void notify_all_at_thread_exit(condition_variable&,unique_lock<mutex>);
```

Std::Condition_Variable Default Constructor

Constructs a `std::condition_variable` object.

Declaration

```
condition_variable();
```

Effects

Constructs a new `std::condition_variable` instance.

Throws

An exception of type `std::system_error` if the condition variable could not be constructed.

Std::Condition_Variable Destructor

Destroys a `std::condition_variable` object.

Declaration

```
~condition_variable();
```

Preconditions

There are no threads blocked on `*this` in a call to `wait()`, `wait_for()`, or `wait_until()`.

Effects

Destroys `*this`.

Throws

Nothing.

Std::Condition_Variable::Notify_One Member Function

Wakes one of the threads currently waiting on a `std::condition_variable`.

Declaration

```
void notify_one() noexcept;
```

Effects

Wakes one of the threads waiting on `*this` at the point of the call. If there are no threads waiting, the call has no effect.

Throws

`std::system_error` if the effects can't be achieved.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()`, and `wait_until()` on a single `std::condition_variable` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable::Notify_All Member Function

Wake all of the threads currently waiting on a `std::condition_variable`.

Declaration

```
void notify_all() noexcept;
```

Effects

Wakes all of the threads waiting on `*this` at the point of the call. If there are no threads waiting, the call has no effect.

Throws

`std::system_error` if the effects can't be achieved.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()`, and `wait_until()` on a single `std::condition_variable` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable::Wait Member Function

Waits until the `std::condition_variable` is woken by a call to `notify_one()` or `notify_all()` or a spurious wakeup.

Declaration

```
void wait(std::unique_lock<std::mutex>& lock);
```

Preconditions

`lock.owns_lock()` is `true`, and the lock is owned by the calling thread.

Effects

Atomically unlocks the supplied `lock` object and block until the thread is woken by a call to `notify_one()` or `notify_all()` by another thread, or the thread is woken spuriously. The `lock` object is locked again before the call to `wait()` returns.

Throws

`std::system_error` if the effects can't be achieved. If the `lock` object is unlocked during the call to `wait()`, it's locked again on exit, even if the function exits via an exception.

Note

The spurious wakeups mean that a thread calling `wait()` may wake even though no thread has called `notify_one()` or `notify_all()`. It's therefore recommended that the overload of `wait()` that takes a predicate is used in preference where possible. Otherwise, it's recommended that `wait()` be called in a loop that tests the predicate associated with the condition variable.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()`, and `wait_until()` on a single `std::condition_variable` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable::Wait Member Function Overload That Takes a Predicate

Waits until the `std::condition_variable` is woken by a call to `notify_one()` or `notify_all()`, and the predicate is `true`.

Declaration

```
template<typename Predicate>
void wait(std::unique_lock<std::mutex>& lock,Predicate pred);
```

Preconditions

The expression `pred()` shall be valid and shall return a value that is convertible to `bool`. `lock.owns_lock()` shall be `true`, and the lock shall be owned by the thread calling `wait()`.

Effects

As-if

```
while(!pred())
{
    wait(lock);
}
```

Throws

Any exception thrown by a call to `pred`, or `std::system_error` if the effects couldn't be achieved.

Note

The potential for spurious wakeups means that it's unspecified how many times `pred` will be called. `pred` will always be invoked with the mutex referenced by `lock` locked, and the function shall return if (and only if) an evaluation of `(bool)pred()` returns `true`.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()` and `wait_until()` on a single `std::condition_variable` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable::Wait_For Member Function

Waits until the `std::condition_variable` is notified by a call to `notify_one()` or `notify_all()`, or until a specified time period has elapsed or the thread is woken spuriously.

Declaration

```
template<typename Rep,typename Period>
cv_status wait_for(
    std::unique_lock<std::mutex>& lock,
    std::chrono::duration<Rep,Period> const& relative_time);
```

Preconditions

`lock.owns_lock()` is `true`, and the lock is owned by the calling thread.

Effects

Atomically unlocks the supplied `lock` object and block until the thread is woken by a call to `notify_one()` or `notify_all()` by another thread, or the time period specified by `relative_time` has elapsed or the thread is woken spuriously. The `lock` object is locked again before the call to `wait_for()` returns.

Returns

`std::cv_status::no_timeout` if the thread was woken by a call to `notify_one()` or `notify_all()` or a spurious wakeup, `std::cv_status::timeout` otherwise.

Throws

`std::system_error` if the effects can't be achieved. If the `lock` object is unlocked during the call to `wait_for()`, it's locked again on exit, even if the function exits via an exception.

Note

The spurious wakeups mean that a thread calling `wait_for()` may wake even though no thread has called `notify_one()` or `notify_all()`. It's therefore recommended that the overload of `wait_for()` that takes a predicate is used in preference where possible. Otherwise, it's recommended that `wait_for()` be called in a loop that tests the predicate associated with the condition variable. Care must be taken when doing this to ensure that the timeout is still valid; `wait_until()` may be more appropriate in many circumstances. The thread may be blocked for longer than the specified duration. Where possible, the elapsed time is determined by a steady clock.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()`, and `wait_until()` on a single

`std::condition_variable` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable::Wait_For Member Function Overload That Takes a Predicate

Wait until the `std::condition_variable` is woken by a call to `notify_one()` or `notify_all()` and the predicate is `true`, or until the specified time period has elapsed.

Declaration

```
template<typename Rep,typename Period,typename Predicate>
bool wait_for(
    std::unique_lock<std::mutex>& lock,
    std::chrono::duration<Rep,Period> const& relative_time,
    Predicate pred);
```

Preconditions

The expression `pred()` shall be valid and shall return a value that's convertible to `bool`. `lock.owns_lock()` shall be `true`, and the lock shall be owned by the thread calling `wait()`.

Effects

As-if

```
internal_clock::time_point end=internal_clock::now()+relative_time;
while(!pred())
{
    std::chrono::duration<Rep,Period> remaining_time=
        end-internal_clock::now();
    if(wait_for(lock,remaining_time)==std::cv_status::timeout)
        return pred();
}
return true;
```

Returns

`true` if the most recent call to `pred()` returned `true`, `false` if the time period specified by `relative_time` has elapsed and `pred()` returned `false`.

Note

The potential for spurious wakeups means that it's unspecified how many times `pred` will be called. `pred` will always be invoked with the mutex referenced by `lock` locked, and the function shall return if (and only if) an evaluation of `(bool)pred()` returns `true` or the time period specified by `relative_time` has elapsed. The thread may be blocked for longer than the specified duration. Where possible, the elapsed time is determined by a steady clock.

Throws

Any exception thrown by a call to `pred`, or `std::system_error` if the effects couldn't be achieved.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()`, and `wait_until()` on a single `std::condition_variable` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable::Wait_Until Member Function

Waits until the `std::condition_variable` is notified by a call to `notify_one()` or `notify_all()` or until a specified time has been reached or the thread is woken spuriously.

Declaration

```
template<typename Clock,typename Duration>
cv_status wait_until(
    std::unique_lock<std::mutex>& lock,
    std::chrono::time_point<Clock,Duration> const& absolute_time);
```

Preconditions

`lock.owns_lock()` is `true`, and the lock is owned by the calling thread.

Effects

Atomically unlocks the supplied `lock` object and block until the thread is woken by a call to `notify_one()` or `notify_all()` by another thread, or `Clock::now()` returns a time equal to or later than `absolute_time` or the thread is woken spuriously. The `lock` object is locked again before the call to `wait_until()` returns.

Returns

`std::cv_status::no_timeout` if the thread was woken by a call to `notify_one()` or `notify_all()` or a spurious wakeup, `std::cv_status::timeout` otherwise.

Throws

`std::system_error` if the effects can't be achieved. If the `lock` object is unlocked during the call to `wait_until()`, it's locked again on exit, even if the function exits via an exception.

Note

The spurious wakeups mean that a thread calling `wait_until()` may wake even though no thread has called `notify_one()` or `notify_all()`. It's therefore recommended that the overload of `wait_until()` that takes a predicate is used in preference where possible. Otherwise, it's recommended that `wait_until()` be called in a loop that tests the predicate associated with the condition variable. There's no guarantee as to how long the calling thread will be blocked, only that if the function returns `false`, then `Clock::now()` returns a time equal to or later than `absolute_time` at the point at which the thread became unblocked.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()`, and `wait_until()` on a single `std::condition_variable` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable::Wait_Until Member Function Overload That Takes a

Predicate

Wait until the `std::condition_variable` is woken by a call to `notify_one()` or `notify_all()` and the predicate is `true`, or until the specified time has been reached.

Declaration

```
template<typename Clock,typename Duration,typename Predicate>
bool wait_until(
    std::unique_lock<std::mutex>& lock,
    std::chrono::time_point<Clock,Duration> const& absolute_time,
    Predicate pred);
```

Preconditions

The expression `pred()` shall be valid and shall return a value that is convertible to `bool`. `lock.owns_lock()` shall be `true`, and the lock shall be owned by the thread calling `wait()`.

Effects

As-if

```
while(!pred())
{
    if(wait_until(lock,absolute_time)==std::cv_status::timeout)
        return pred();
}
return true;
```

Returns

`true` if the most recent call to `pred()` returned `true`, `false` if a call to `Clock::now()` returned a time equal to or later than the time specified by `absolute_time` and `pred()` returned `false`.

Note

The potential for spurious wakeups means that it's unspecified how many times `pred` will be called. `pred` will always be invoked with the mutex referenced by `lock` locked, and the function shall return if (and only if) an evaluation of `(bool)pred()` returns `true` or `Clock::now()` returns a time equal to or later than `absolute_time`. There's no guarantee as to how long the calling thread will be blocked, only that if the function returns `false`, then `Clock::now()` returns a time equal to or later than `absolute_time` at the point at which the thread became unblocked.

Throws

Any exception thrown by a call to `pred`, or `std::system_error` if the effects couldn't be achieved.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_until()`, and `wait_until()` on a single `std::condition_variable` instance are serialized. A call to `notify_one()` or `notify_all()` will wake only threads that started waiting *prior* to that call.

Std::Notify_All_At_Thread_Exit Nonmember Function

Wake all of the threads waiting on a `std::condition_variable` when the current thread exits.

Declaration

```
void notify_all_at_thread_exit(
    condition_variable& cv,unique_lock<mutex> lk);
```

Preconditions

`lk.owns_lock()` is `true`, and the lock is owned by the calling thread. `lk.mutex()` shall return the same value as for any of the lock objects supplied to `wait(), wait_for()`, or `wait_until()` on `cv` from concurrently waiting threads.

Effects

Transfers ownership of the lock held by `lk` into internal storage and schedules `cv` to be notified when the calling thread exits. This notification shall be as-if

```
lk.unlock();
cv.notify_all();
```

Throws

`std::system_error` if the effects can't be achieved.

Note

The lock is held until the thread exits, so care must be taken to avoid deadlock. It's recommended that the calling thread should exit as soon as possible and that no blocking operations be performed on this thread.

The user should ensure that waiting threads don't erroneously assume that the thread has exited when they are woken, particularly with the potential for spurious wakeups. This can be achieved by testing a predicate on the waiting thread that's only made `true` by the notifying thread under the protection of the mutex and without releasing the lock on the mutex prior to the call of `notify_all_at_thread_exit`.std::condition_variable_any class.

D.2.2. std::condition_variable_any class

The `std::condition_variable_any` class allows a thread to wait for a condition to become `true`. Whereas `std::condition_variable` can be used only with `std::unique_lock<std::mutex>`, `std::condition_variable_any` can be used with *any* type that meets the `Lockable` requirements.

Instances of `std::condition_variable_any` aren't `CopyAssignable, Copy-Constructible, MoveAssignable`, or `MoveConstructible`.

Class definition

```
class condition_variable_any
{
public:
    condition_variable_any();
```

```
    ~condition_variable_any();

    condition_variable_any(
        condition_variable_any const& ) = delete;
    condition_variable_any& operator=(
        condition_variable_any const& ) = delete;

    void notify_one() noexcept;
    void notify_all() noexcept;

    template<typename Lockable>
    void wait(Lockable& lock);

    template <typename Lockable, typename Predicate>
    void wait(Lockable& lock, Predicate pred);

    template <typename Lockable, typename Clock,typename Duration>
    std::cv_status wait_until(
        Lockable& lock,
        const std::chrono::time_point<Clock, Duration>& absolute_time);

    template <
        typename Lockable, typename Clock,
        typename Duration, typename Predicate>
    bool wait_until(
        Lockable& lock,
        const std::chrono::time_point<Clock, Duration>& absolute_time,
        Predicate pred);

    template <typename Lockable, typename Rep, typename Period>
    std::cv_status wait_for(
        Lockable& lock,
        const std::chrono::duration<Rep, Period>& relative_time);
    template <
        typename Lockable, typename Rep,
        typename Period, typename Predicate>
    bool wait_for(
        Lockable& lock,
        const std::chrono::duration<Rep, Period>& relative_time,
        Predicate pred);
};
```

Std::Condition_Variable_Any Default Constructor

Constructs a `std::condition_variable_any` object.

Declaration

```
condition_variable_any();
```

Effects

Constructs a new `std::condition_variable_any` instance.

Throws

An exception of type `std::system_error` if the condition variable couldn't be constructed.

Std::Condition_Variable_Any Destructor

Destroys a `std::condition_variable_any` object.

Declaration

```
~condition_variable_any();
```

Preconditions

There are no threads blocked on `*this` in a call to `wait(), wait_for()`, or `wait_until()`.

Effects

Destroys `*this`.

Throws

Nothing.

Std::Condition_Variable_Any::Notify_One Member Function

Wakes one of the threads currently waiting on a `std::condition_variable_any`.

Declaration

```
void notify_one() noexcept;
```

Effects

Wakes one of the threads waiting on `*this` at the point of the call. If there are no threads waiting, the call has no effect.

Throws

`std::system_error` if the effects can't be achieved.

Synchronization

Calls to `notify_one(), notify_all(), wait(), wait_for()`, and `wait_until()` on a single `std::condition_variable_any` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable_Any::Notify_All Member Function

Wakes all of the threads currently waiting on a `std::condition_variable_any`.

Declaration

```
void notify_all() noexcept;
```

Effects

Wakes all of the threads waiting on `*this` at the point of the call. If there are no threads waiting, the

call has no effect.

Throws

`std::system_error` if the effects can't be achieved.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()`, and `wait_until()` on a single `std::condition_variable_any` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable_Any::Wait Member Function

Waits until the `std::condition_variable_any` is woken by a call to `notify_one()` or `notify_all()` or a spurious wakeup.

Declaration

```
template<typename Lockable>
void wait(Lockable& lock);
```

Preconditions

`Lockable` meets the `Lockable` requirements, and `lock` owns a lock.

Effects

Atomically unlocks the supplied `lock` object and block until the thread is woken by a call to `notify_one()` or `notify_all()` by another thread, or the thread is woken spuriously. The `lock` object is locked again before the call to `wait()` returns.

Throws

`std::system_error` if the effects can't be achieved. If the `lock` object is unlocked during the call to `wait()`, it's locked again on exit, even if the function exits via an exception.

Note

The spurious wakeups mean that a thread calling `wait()` may wake even though no thread has called `notify_one()` or `notify_all()`. It's therefore recommended that the overload of `wait()` that takes a predicate is used in preference where possible. Otherwise, it's recommended that `wait()` be called in a loop that tests the predicate associated with the condition variable.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()`, and `wait_until()` on a single `std::condition_variable_any` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable_Any::Wait Member Function Overload That Takes a Predicate

Waits until the `std::condition_variable_any` is woken by a call to `notify_one()` or `notify_all()` and the predicate is `true`.

Declaration

```
template<typename Lockable,typename Predicate>
void wait(Lockable& lock,Predicate pred);
```

Preconditions

The expression `pred()` shall be valid and shall return a value that's convertible to `bool`. `Lockable` meets the `Lockable` requirements, and `lock` owns a lock.

Effects

As-if

```
while(!pred())
{
    wait(lock);
}
```

Throws

Any exception thrown by a call to `pred`, or `std::system_error` if the effects could not be achieved.

Note

The potential for spurious wakeups means that it's unspecified how many times `pred` will be called. `pred` will always be invoked with the mutex referenced by `lock` locked, and the function shall return if (and only if) an evaluation of `(bool)pred()` returns `true`.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()`, and `wait_until()` on a single `std::condition_variable_any` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable_Any::Wait_For Member Function

Waits until the `std::condition_variable_any` is notified by a call to `notify_one()` or `notify_all()` or until a specified time period has elapsed or the thread is woken spuriously.

Declaration

```
template<typename Lockable,typename Rep,typename Period>
std::cv_status wait_for(
    Lockable& lock,
    std::chrono::duration<Rep,Period> const& relative_time);
```

Preconditions

`Lockable` meets the `Lockable` requirements, and `lock` owns a lock.

Effects

Atomically unlocks the supplied `lock` object and block until the thread is woken by a call to

notify_one() or notify_all() by another thread or the time period specified by relative_time has elapsed or the thread is woken spuriously. The lock object is locked again before the call to wait_for() returns.

Returns

std::cv_status::no_timeout if the thread was woken by a call to notify_one() or notify_all() or a spurious wakeup, std::cv_status::timeout otherwise.

Throws

std::system_error if the effects can't be achieved. If the lock object is unlocked during the call to wait_for(), it's locked again on exit, even if the function exits via an exception.

Note

The spurious wakeups mean that a thread calling wait_for() may wake even though no thread has called notify_one() or notify_all(). It's therefore recommended that the overload of wait_for() that takes a predicate is used in preference where possible. Otherwise, it's recommended that wait_for() be called in a loop that tests the predicate associated with the condition variable. Care must be taken when doing this to ensure that the timeout is still valid; wait_until() may be more appropriate in many circumstances. The thread may be blocked for longer than the specified duration. Where possible, the elapsed time is determined by a steady clock.

Synchronization

Calls to notify_one(), notify_all(), wait(), wait_for(), and wait_until() on a single std::condition_variable_any instance are serialized. A call to notify_one() or notify_all() will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable_Any::Wait_For Member Function Overload That Takes a Predicate

Waits until the std::condition_variable_any is woken by a call to notify_one() or notify_all() and the predicate is true, or until the specified time period has elapsed.

Declaration

```
template<typename Lockable,typename Rep,
    typename Period, typename Predicate>
bool wait_for(
    Lockable& lock,
    std::chrono::duration<Rep,Period> const& relative_time,
    Predicate pred);
```

Preconditions

The expression pred() shall be valid and shall return a value that's convertible to bool. Lockable meets the Lockable requirements, and lock owns a lock.

Effects

As-if

```
internal_clock::time_point end=internal_clock::now()+relative_time;
while(!pred())
{
    std::chrono::duration<Rep,Period> remaining_time=
        end-internal_clock::now();
    if(wait_for(lock,remaining_time)==std::cv_status::timeout)
        return pred();
}
return true;
```

Returns

`true` if the most recent call to `pred()` returned `true`, `false` if the time period specified by `relative_time` has elapsed and `pred()` returned `false`.

Note

The potential for spurious wakeups means that it's unspecified how many times `pred` will be called. `pred` will always be invoked with the mutex referenced by `lock` locked, and the function shall return if (and only if) an evaluation of `(bool)pred()` returns `true` or the time period specified by `relative_time` has elapsed. The thread may be blocked for longer than the specified duration. Where possible, the elapsed time is determined by a steady clock.

Throws

Any exception thrown by a call to `pred`, or `std::system_error` if the effects couldn't be achieved.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()`, and `wait_until()` on a single `std::condition_variable_any` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable_Any::Wait_Until Member Function

Waits until the `std::condition_variable_any` is notified by a call to `notify_one()` or `notify_all()` or until a specified time has been reached or the thread is woken spuriously.

Declaration

```
template<typename Lockable,typename Clock,typename Duration>
std::cv_status wait_until(
    Lockable& lock,
    std::chrono::time_point<Clock,Duration> const& absolute_time);
```

Preconditions

`Lockable` meets the `Lockable` requirements, and `lock` owns a lock.

Effects

Atomically unlocks the supplied `lock` object and block until the thread is woken by a call to `notify_one()` or `notify_all()` by another thread or `Clock::now()` returns a time equal to or later than `absolute_time` or the thread is woken spuriously. The `lock` object is locked again before the call

to `wait_until()` returns.

Returns

`std::cv_status::no_timeout` if the thread was woken by a call to `notify_one()` or `notify_all()` or a spurious wakeup, `std::cv_status::timeout` otherwise.

Throws

`std::system_error` if the effects can't be achieved. If the `lock` object is unlocked during the call to `wait_until()`, it's locked again on exit, even if the function exits via an exception.

Note

The spurious wakeups mean that a thread calling `wait_until()` may wake even though no thread has called `notify_one()` or `notify_all()`. It's therefore recommended that the overload of `wait_until()` that takes a predicate is used in preference where possible. Otherwise, it's recommended that `wait_until()` be called in a loop that tests the predicate associated with the condition variable. There's no guarantee as to how long the calling thread will be blocked, only that if the function returns `false`, then `Clock::now()` returns a time equal to or later than `absolute_time` at the point at which the thread became unblocked.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_for()`, and `wait_until()` on a single `std::condition_variable_any` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.

Std::Condition_Variable_Any::Wait_Until Member Function Overload That Takes a Predicate

Waits until the `std::condition_variable_any` is woken by a call to `notify_one()` or `notify_all()` and the predicate is `true`, or until the specified time has been reached.

Declaration

```
template<typename Lockable,typename Clock,
    typename Duration, typename Predicate>
bool wait_until(
    Lockable& lock,
    std::chrono::time_point<Clock,Duration> const& absolute_time,
    Predicate pred);
```

Preconditions

The expression `pred()` shall be valid, and shall return a value that's convertible to `bool`. `Lockable` meets the `Lockable` requirements, and `lock` owns a lock.

Effects

As-if

```
while(!pred())
{
    if(wait_until(lock,absolute_time)==std::cv_status::timeout)
```

```
        return pred();
}
return true;
```

Returns

`true` if the most recent call to `pred()` returned `true`, `false` if a call to `Clock::now()` returned a time equal to or later than the time specified by `absolute_time`, and `pred()` returned `false`.

Note

The potential for spurious wakeups means that it's unspecified how many times `pred` will be called. `pred` will always be invoked with the mutex referenced by `lock` locked, and the function shall return if (and only if) an evaluation of `(bool)pred()` returns `true` or `Clock::now()` returns a time equal to or later than `absolute_time`. There's no guarantee as to how long the calling thread will be blocked, only that if the function returns `false`, then `Clock::now()` returns a time equal to or later than `absolute_time` at the point at which the thread became unblocked.

Throws

Any exception thrown by a call to `pred`, or `std::system_error` if the effects couldn't be achieved.

Synchronization

Calls to `notify_one()`, `notify_all()`, `wait()`, `wait_until()`, and `wait_until()` on a single `std::condition_variable_any` instance are serialized. A call to `notify_one()` or `notify_all()` will only wake threads that started waiting *prior* to that call.