Not logged in
register log in

Information
Tutorials
Reference
Articles
Forum

Reference

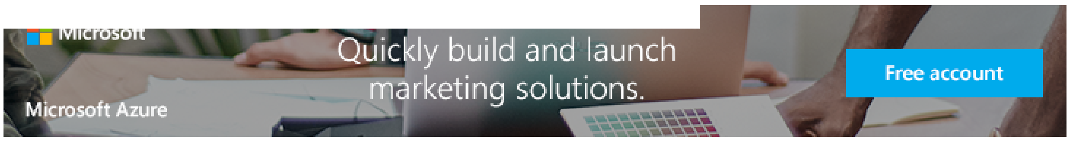
C library:
Containers:
Input/Output:
Multi-threading:
<atomic>
<condition_variable>
<future>
<mutex>
<thread>
Other:

<condition_variable>

classes:
condition_variable
condition_variable_any
enum classes:
cv_status
functions:
notify_all_at_thread_exit

condition_variable

condition_variable::condition_variable
condition_variable::~condition_variable
member functions:
condition_variable::notify_all
condition_variable::notify_one
condition_variable::wait
condition_variable::wait_for
condition_variable::wait_until



class
# std::condition_variable

<condition_variable>

`class condition_variable;`

Condition variable

A *condition variable* is an object able to block the calling thread until *notified* to resume.

It uses a `unique_lock` (over a `mutex`) to lock the thread when one of its *wait functions* is called. The thread remains blocked until woken up by another thread that calls a *notification function* on the same `condition_variable` object.

Objects of type `condition_variable` always use `unique_lock<mutex>` to wait: for an alternative that works with any kind of *lockable type*, see `condition_variable_any`

Member functions

(constructor)	Construct condition_variable (public member function)
(destructor)	Destroy condition_variable (public member function)

Wait functions

wait	Wait until notified (public member function)
wait_for	Wait for timeout or until notified (public member function)
wait_until	Wait until notified or time point (public member function)

Notify functions

notify_one	Notify one (public member function)
notify_all	Notify all (public member function)

Example

```cpp
// condition_variable example
#include <iostream>           // std::cout
#include <thread>             // std::thread
#include <mutex>              // std::mutex, std::unique_lock
#include <condition_variable> // std::condition_variable

std::mutex mtx;
std::condition_variable cv;
bool ready = false;

void print_id (int id) {
  std::unique_lock<std::mutex> lck(mtx);
  while (!ready) cv.wait(lck);
  // ...
  std::cout << "thread " << id << '\n';
}

void go() {
  std::unique_lock<std::mutex> lck(mtx);
  ready = true;
  cv.notify_all();
}

int main ()
{
  std::thread threads[10];
  // spawn 10 threads:
  for (int i=0; i<10; ++i)
    threads[i] = std::thread(print_id,i);

  std::cout << "10 threads ready to race...\n";
  go();                       // go!

  for (auto& th : threads) th.join();

  return 0;
}
```

Possible output (thread order may vary):

```
10 threads ready to race...
thread 2
thread 0
thread 9
thread 4
thread 6
thread 8
thread 7
thread 5
thread 3
thread 1
```