C++
Information
Tutorials
Reference
Articles
Forum

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

<thread>
classes:
thread
namespaces:
this_thread

thread
thread::thread
thread::~thread
member functions:
thread::detach
thread::get_id
thread::join
thread::joinable
thread::native_handle
thread::operator=
thread::swap
member types:
thread::id
thread::native_handle_type
static member functions:
thread::hardware_concurrency
non-member overloads:
swap (thread)

class
std::thread

<thread>

```
class thread;
```

Thread

Class to represent individual *threads of execution*.

A *thread of execution* is a sequence of instructions that can be executed concurrently with other such sequences in *multithreading* environments, while sharing a same address space.

An initialized `thread` object represents an active thread of execution; Such a `thread` object is *joinable*, and has a unique *thread id*.

A default-constructed (non-initialized) `thread` object is *not joinable*, and its *thread id* is common for all *non-joinable* threads.

A *joinable* thread becomes *not joinable* if *moved from*, or if either `join` or `detach` are called on them.

Member types

id	Thread id (public member type)
native_handle_type	Native handle type (public member type)

Member functions

(constructor)	Construct thread (public member function)
(destructor)	Thread destructor (public member function)
operator=	Move-assign thread (public member function)
get_id	Get thread id (public member function)
joinable	Check if joinable (public member function)
join	Join thread (public member function)
detach	Detach thread (public member function)
swap	Swap threads (public member function)
native_handle	Get native handle (public member function)
hardware_concurrency [static]	Detect hardware concurrency (public static member function)

Non-member overloads

swap (thread)	Swap threads (function)

Example

```
1  // thread example
2  #include <iostream>       // std::cout
3  #include <thread>         // std::thread
4
5  void foo()
6  {
7    // do stuff...
8  }
9
10 void bar(int x)
11 {
12   // do stuff...
13 }
14
15 int main()
16 {
17   std::thread first (foo);     // spawn new thread that calls foo()
18   std::thread second (bar,0);  // spawn new thread that calls bar(0)
19
20   std::cout << "main, foo and bar now execute concurrently...\n";
21
22   // synchronize threads:
23   first.join();                // pauses until first finishes
24   second.join();               // pauses until second finishes
25
26   std::cout << "foo and bar completed.\n";
27
28   return 0;
29 }
```

Output:

```
main, foo and bar now execute concurrently...
foo and bar completed.
```