

# std::future

Defined in header `<future>`

```
template< class T > class future;
```

 (1) (since C++11)

```
template< class T > class future<T&>;
```

 (2) (since C++11)

```
template<> class future<void>;
```

 (3) (since C++11)

The class template `std::future` provides a mechanism to access the result of asynchronous operations:

- An asynchronous operation (performed via `std::async`, `std::packaged_task`, or `std::promise`) can provide a `std::future` object to the creator of that asynchronous operation.
- The creator of the asynchronous operation can then use a variety of methods to query, wait for, or extract a value from the `std::future`. These methods may block if the asynchronous operation has not yet provided a value.
- When the asynchronous operation is ready to send a result to the creator, it can do so by modifying *shared state* (e.g. `std::promise::set_value`) that is linked to the creator's `std::future`.

Note that a `std::future` is a unique reference to the result in a shared state, i.e. the result is not shared with any other asynchronous return objects. Use `std::shared_future` for non-unique access to the result.

## Member functions

|               |                                                                                                                               |
|---------------|-------------------------------------------------------------------------------------------------------------------------------|
| (constructor) | constructs the future object<br>(public member function)                                                                      |
| (destructor)  | destructs the future object<br>(public member function)                                                                       |
| operator=     | moves the future object<br>(public member function)                                                                           |
| share         | transfers the shared state from <code>*this</code> to a <code>shared_future</code> and returns it<br>(public member function) |

## Getting the result

|     |                                                |
|-----|------------------------------------------------|
| get | returns the result<br>(public member function) |
|-----|------------------------------------------------|

## State

|            |                                                                                                                              |
|------------|------------------------------------------------------------------------------------------------------------------------------|
| valid      | checks if the future has a shared state<br>(public member function)                                                          |
| wait       | waits for the result to become available<br>(public member function)                                                         |
| wait_for   | waits for the result, returns if it is not available for the specified timeout duration<br>(public member function)          |
| wait_until | waits for the result, returns if it is not available until specified time point has been reached<br>(public member function) |

## Examples

Run this code

```
#include <future>
#include <iostream>
#include <thread>
```

```

int main()
{
    // future from a packaged_task
    std::packaged_task<int> task([]{ return 7; }); // wrap the function
    std::future<int> f1 = task.get_future(); // get a future
    std::thread t(std::move(task)); // launch on a thread

    // future from an async()
    std::future<int> f2 = std::async(std::launch::async, []{ return 8; });

    // future from a promise
    std::promise<int> p;
    std::future<int> f3 = p.get_future();
    std::thread([&p]{ p.set_value_at_thread_exit(9); }).detach();

    std::cout << "Waiting..." << std::flush;
    f1.wait();
    f2.wait();
    f3.wait();
    std::cout << "Done!\nResults are: "
              << f1.get() << ' ' << f2.get() << ' ' << f3.get() << '\n';
    t.join();
}

```

Output:

```

Waiting...Done!
Results are: 7 8 9

```

## Example with exceptions

Run this code

```

#include <future>
#include <iostream>
#include <thread>

int main()
{
    std::promise<int> p;
    std::future<int> f = p.get_future();

    std::thread t([&p]
    {
        try
        {
            // code that may throw
            throw std::runtime_error("Example");
        }
        catch (...)
        {
            try
            {
                // store anything thrown in the promise
                p.set_exception(std::current_exception());
            }
            catch (...) {} // set_exception() may throw too
        }
    });

    try

```

```
{
    std::cout << f.get();
}
catch (const std::exception& e)
{
    std::cout << "Exception from the thread: " << e.what() << '\n';
}
t.join();
}
```

Output:

Exception from the thread: Example

## See also

|                              |                                                                                                                                                |
|------------------------------|------------------------------------------------------------------------------------------------------------------------------------------------|
| <b>async</b> (C++11)         | runs a function asynchronously (potentially in a new thread) and returns a <b>std::future</b> that will hold the result<br>(function template) |
| <b>shared_future</b> (C++11) | waits for a value (possibly referenced by other futures) that is set asynchronously<br>(class template)                                        |

[Support us](#)   [Recent changes](#)   [FAQ](#)   [Offline version](#)

[What links here](#)   [Related changes](#)   [Upload file](#)   [Special pages](#)   [Printable version](#)   [Permanent link](#)   [Page information](#)

In other languages   [Deutsch](#)   [Español](#)   [Français](#)   [Italiano](#)   [日本語](#)   [Português](#)   [Русский](#)   [中文](#)