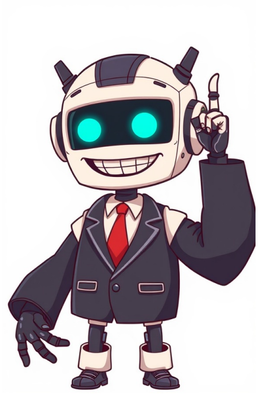
Click to prove you're human



From cppreference.com Main template T get(); (1) (since C++11) std::future specializations T& get(); (2) (since C++11) std::future specialization void get(); (3) (since C++11) The get member function waits (by calling wait()) until the shared state is ready, then retrieves the value stored in the shared state (if any). Right after calling this function, valid() is false. If valid() is false before the call to this function, the behavior is undefined. [edit] Return value 1) The value v stored in the shared state, as std::move(v). 2) The reference stored as value in the shared state. 3) (none) If an exception was stored in the shared state referenced by the future (e.g. via a call to std::promise::set_exception()) then that exception will be thrown. [edit] Notes The C++ standard recommends the implementations to detect the case when valid() is false before the call and throw a std::future_error with an error condition of std::future_errc::no_state. [edit] Example Possible output: [0.000004s] launching thread [0.000461s] waiting for the future, f.valid() = 1 [1.001156s] f.get() returned with 7, f.valid() = 0 [1.001192s] launching thread [1.001275s] waiting for the future, f.valid() = 1 [2.002356s] caught exception 7, f.valid() = 0 [edit] Defect reports The following behavior-changing defect reports were applied retroactively to previously published C++ standards. DR Applied to Behavior as published Correct behavior LWG 2096 C++11 overload (1) needed to check whether T is MoveAssignable not required [edit] See also checks if the future has a shared state (public member function) [edit] From cppreference.com < cpp | thread template< class T > class future; (1) (since C++11) template< class T > class future; (2) (since C++11) template class future; (3) (since C++11) The class template std::future provides a mechanism to access the result of asynchronous operations: 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 std::future references shared state that is not shared with any other asynchronous return objects (as opposed to std::shared_future). [edit] Member functions constructs the future object (public member function) [edit] destructs the future object (public member function) [edit] moves the future object (public member function) [edit] transfers the shared state from *this to a shared_future and returns it (public member function) [edit] returns the result (public member function) [edit] checks if the future has a shared state (public member function) [edit] waits for the result to become available (public member function) [edit] waits for the result, returns if it is not available for the specified timeout duration (public member function) [edit] waits for the result, returns if it is not available until specified time point has been reached (public member function) [edit] [edit] Examples #include #include #include int main() { // future from a packaged_task std::packaged_task task([]{ return 7; }); // wrap the function std::future f1 = task.get_future(); // get a future std::thread t(std::move(task)); // launch on a thread // future from an async() std::future f2 = std::async(std::launch::async, []{ return 8; }); // future from a promise std::promise p; std::future f3 = p.get_future(); std::thread([&p]{ p.set_value_at_thread_exit(9); }).detach(); std::cout