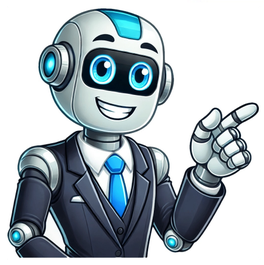
Click to verify



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