This is an exercise C++ project for writing a timer-service thread-class.

The solution comprises a single project "TaskWakeTimer.vcxproj" with the following main classes:  
- CTaskTimerService … implements a service-class/thread to be used by client threads to sleep N   
 seconds. This class is a singleton and its main interface is the function Sleep().  
- CTaskTimerClient (implemented in "main.cpp") … simulates a client thread. Such threads call   
 Sleep() with a time argument. This function will block them on semaphore for the given time and   
 release them after time expires.

In addition, there are also 2 auxillary singleton classes:  
- CTimeFromStart … enables to get time from application launch.  
- CThreadSafePrintf … enables to print to stdio in thread-safe manner.

The gross class picture is the following:

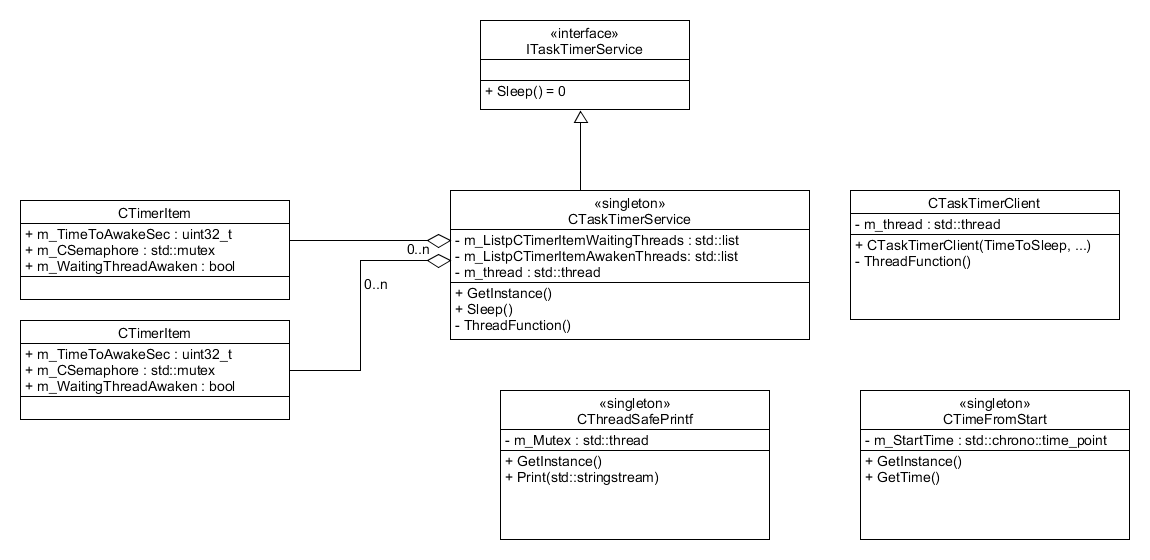


Figure 1: class diagram

The basic-flow diagram looks like this:

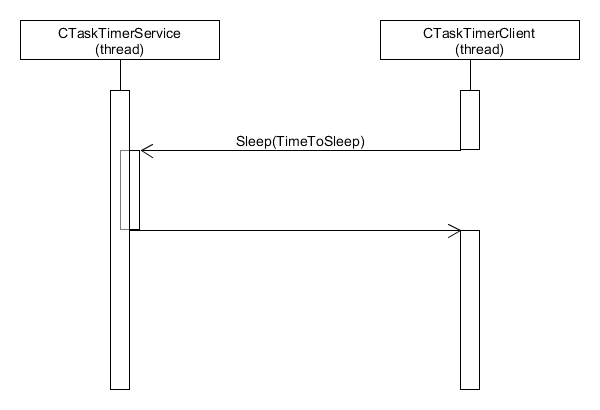


Figure : basic interaction

Client-thread call CTaskTimerService::Sleep with the requested sleep-time. Within CTaskTimerService::Sleep a CTimerItem is created and added to the waiting-threads list CTaskTimerService::m\_ListpCTimerItemWaitingThreads and the calling thread is blocked on semaphore.

CTaskTimerService's thread scan the list every 1sec and compares system-time with the requested time-to-wake-on CTimerItem::m\_TimeToAwakeSec in each list element. When system-time is equal or greater than the time-to-wake, the semaphore is signaled and the waiting thread is released to resume its run.

Released CTimerItem objects are saved in a released-threads list named CTaskTimerService::m\_ m\_ListpCTimerItemAwakenThreads . It's removed from the list only after the thread marks it resumed its run.  
Currently, the released-threads is not monitored and erroneous situations where threads do no signal they are awaken is not handled. This is a TBD issue.

Notice: implementations lacks adequate destructors. .