Class **Mutex** < Object

1.9

A mutex is a semaphore object that can be used to synchronize access to resources shared across threads. We discuss mutexes (and other synchronization mechanisms) starting on page 191. Because the code examples tend to be long, I haven't duplicated them in this library description.

Instance methods

lock *mutex*.lock → *mutex*

Takes a lock on *mutex*. Suspends if *mutex* is already locked by another thread and raises a ThreadError if the mutex is already locked by the calling thread.

locked? *mutex*.locked? → true or false

Returns the current locked state of *mutex*.

sleep *mutex*.sleep(*time* | nil) → *seconds_slept*

Releases the current thread's lock on *mutex*, sleeps for *time* seconds (or forever if nil is passed), and then regains the lock. Returns the number of seconds actually slept.

synchronize *mutex*.synchronize { *block* } → *obj*

Locks *mutex*, executes the block, and then unlocks *mutex*. Returns the value returned by the block.

try_lock *mutex*.try_lock → true or false

If *mutex* is not currently locked, locks it and returns true. Otherwise, returns false. (That is, try_lock is like lock, but it will never wait for a mutex to become available.)

unlock *mutex*.unlock → *mutex*

Unlock *mutex*, which must be locked by the current thread.