NSConditionLock

Inherits From:	NSObject
Conforms To:	NSLocking NSObject (NSObject)
Declared In:	Foundation/NSLock.h

Class Description

The NSConditionLock class defines objects whose locks can be associated with specific, user-defined conditions. Using an NSConditionLock object, you can ensure that a thread can acquire a lock only if a certain condition is met. Once it has acquired the lock and executed the critical section of code, the thread can relinquish the lock and set the associated condition to something new. The conditions themselves are arbitrary: You define them as needed for your application.

Typically, you use an NSConditionLock object when threads in your application need to execute in a particular order, such as when one thread produces data that another consumes. While the producer is executing, the consumer sleeps waiting to acquire a lock that's conditional upon the producer's completion of its operation. An application can have multiple NSConditionLock objects, each protecting different sections of code. However, these objects must be created before the application becomes multithreaded.

The locking and unlocking methods that NSConditionLock objects respond to can be used in any combination. For example, you can pair a **lock** message with **unlockWithCondition:**, or a **lockWhenCondition:** message with **unlock**.

The following example shows how the producer-consumer problem might be handled using condition locks. Imagine that an application contains a queue of data. A producer thread adds data to the queue, and consumer threads extract data from the queue.

The producer need not wait for a condition, but must wait for the lock to be made available so it can safely add data to the queue. For example, a producer could use a lock this way:

```
id condLock = [[NSConditionLock alloc] initWithCondition:NO_DATA];

[condLock lock];
/* Add data to the queue. */
[condLock unlockWithCondition:HAS_DATA];
```

Note that in acquiring the lock, the producer sets its condition to the user-defined value NO_DATA. After adding data to the queue, the producer relinquishes the lock, setting its condition to HAS_DATA.

A consumer thread waits until there's data available and all other threads are out of locked critical sections. In the following code, the consumer sleeps until there is data in the queue and a lock can be acquired:

```
[condLock lockWhenCondition:HAS_DATA];
/* Remove data from the queue. */
[condLock unlockWithCondition:(isEmpty ? NO_DATA : HAS_DATA)];
```

The consumer removes some data from the queue and then relinquishes the lock, setting its value to NO_DATA or HAS_DATA, depending on whether the queue is now empty.

The NSConditionLock, NSLock, and NSRecursiveLock classes all adopt the NSLocking protocol with various features and performance characteristics; see the other class descriptions for more information.

dopted Protocols

NSLocking	– lock – unlock

ethod Types

Initializing an NSConditionLock	– initWithCondition:
Returning the condition	– condition
Acquiring and releasing a lock	– lockBeforeDate: – lockWhenCondition: – lockWhenCondition:beforeDate: – tryLock – tryLockWhenCondition: – unlockWithCondition:

nstance Methods

condition

– (int)**condition**

Returns the condition that's associated with the receiver. If no condition has been set, returns 0.

initWithCondition:

– (id)**initWithCondition:**(int)*condition*

Initializes a newly allocated NSConditionLock and sets its condition to *condition*. The value of the *condition* argument is user-defined; see the class description for more information. Returns **self**.

lockBeforeDate:

– (BOOL)**lockBeforeDate:**(NSDate *)*limit*

Attempts to acquire a lock before the date represented by *limit*. The thread is blocked until the receiver acquires the lock or *limit* is reached. Returns YES if the lock is acquired within the time limit. Returns NO if the time limit expires before a lock can be acquired.

The condition associated with the receiver isn't taken into account in this operation.

See also: – **lockWhenCondition:beforeDate:**

lockWhenCondition:

– (void)**lockWhenCondition:**(int)*condition*

Attempts to acquire a lock. The receiver's condition must be equal to *condition* before the locking operation will succeed. This method blocks the thread's execution until the lock can be acquired.

See also: – **lockWhenCondition:beforeDate:**, – **unlockWithCondition:**

lockWhenCondition:beforeDate:

– (BOOL)**lockWhenCondition:**(int)*condition* **beforeDate:**(NSDate *)*limit*

Attempts to acquire a lock before the date represented by *limit*. The receiver's condition must be equal to *condition* before the locking operation will succeed. Returns YES if the lock is acquired within this time limit. Returns NO if the time limit expires before a lock can be acquired. This method blocks the thread's execution until the lock can be acquired or *limit* is reached.

See also: – **lockBeforeDate:**, – **lockWhenCondition:**

tryLock

– (BOOL)**tryLock**

Attempts to acquire a lock without regard to the receiver's condition. Returns immediately with a value of YES if successful and NO otherwise.

See also: – **tryLockWhenCondition:**

tryLockWhenCondition:

– (BOOL)**tryLockWhenCondition:**(int)*condition*

Attempts to acquire a lock if *condition* is true. As part of its implementation, this method invokes *lockWhenCondition:beforeDate:*. Returns immediately, with a value of YES if successful and NO otherwise.

See also: – **tryLock**

unlockWithCondition:

– (void)**unlockWithCondition:**(int)*condition*

Relinquishes the lock and sets the receiver's condition to *condition*.

See also: – **lockWhenCondition:**