
NSCondition Class Reference

[Cocoa > Process Management](#)



2008-09-09



Apple Inc.
© 2008 Apple Inc.
All rights reserved.

No part of this publication may be reproduced, stored in a retrieval system, or transmitted, in any form or by any means, mechanical, electronic, photocopying, recording, or otherwise, without prior written permission of Apple Inc., with the following exceptions: Any person is hereby authorized to store documentation on a single computer for personal use only and to print copies of documentation for personal use provided that the documentation contains Apple's copyright notice.

The Apple logo is a trademark of Apple Inc.

Use of the "keyboard" Apple logo (Option-Shift-K) for commercial purposes without the prior written consent of Apple may constitute trademark infringement and unfair competition in violation of federal and state laws.

No licenses, express or implied, are granted with respect to any of the technology described in this document. Apple retains all intellectual property rights associated with the technology described in this document. This document is intended to assist application developers to develop applications only for Apple-labeled computers.

Every effort has been made to ensure that the information in this document is accurate. Apple is not responsible for typographical errors.

Apple Inc.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010

Apple, the Apple logo, Cocoa, Mac, and Mac OS are trademarks of Apple Inc., registered in the United States and other countries.

Simultaneously published in the United States and Canada.

Even though Apple has reviewed this document, APPLE MAKES NO WARRANTY OR REPRESENTATION, EITHER EXPRESS OR IMPLIED, WITH RESPECT TO THIS DOCUMENT, ITS QUALITY, ACCURACY, MERCHANTABILITY, OR FITNESS FOR A PARTICULAR PURPOSE. AS A RESULT, THIS DOCUMENT IS PROVIDED "AS IS," AND YOU, THE READER, ARE ASSUMING THE ENTIRE RISK AS TO ITS QUALITY AND ACCURACY.

IN NO EVENT WILL APPLE BE LIABLE FOR DIRECT, INDIRECT, SPECIAL, INCIDENTAL, OR CONSEQUENTIAL DAMAGES RESULTING FROM ANY

DEFECT OR INACCURACY IN THIS DOCUMENT, even if advised of the possibility of such damages.

THE WARRANTY AND REMEDIES SET FORTH ABOVE ARE EXCLUSIVE AND IN LIEU OF ALL OTHERS, ORAL OR WRITTEN, EXPRESS OR IMPLIED. No Apple dealer, agent, or employee is authorized to make any modification, extension, or addition to this warranty.

Some states do not allow the exclusion or limitation of implied warranties or liability for incidental or consequential damages, so the above limitation or exclusion may not apply to you. This warranty gives you specific legal rights, and you may also have other rights which vary from state to state.

Contents

NSCondition Class Reference 5

- Overview 5
- Tasks 6
 - Waiting for the Lock 6
 - Signaling Waiting Threads 6
 - Accessor Methods 6
- Instance Methods 7
 - broadcast 7
 - name 7
 - setName: 7
 - signal 8
 - wait 8
 - waitUntilDate: 9

Document Revision History 11

Index 13

NSCondition Class Reference

Inherits from	NSObject
Conforms to	NSLocking NSObject (NSObject)
Framework	/System/Library/Frameworks/Foundation.framework
Availability	Available in Mac OS X v10.5 and later.
Companion guide	Threading Programming Guide
Declared in	NSLock.h

Overview

The `NSCondition` class implements a condition variable whose semantics follow those used for POSIX-style conditions. A condition object acts as both a lock and a checkpoint in a given thread. The lock protects your code while it tests the condition and performs the task triggered by the condition. The checkpoint behavior requires that the condition be true before the thread proceeds with its task. While the condition is not true, the thread blocks. It remains blocked until another thread signals the condition object.

The semantics for using an `NSCondition` object are as follows:

1. Lock the condition object.
2. Test a boolean predicate. (This predicate is a boolean flag or other variable in your code that indicates whether it is safe to perform the task protected by the condition.)
3. If the boolean predicate is false, call the condition object's `wait` or `waitUntilDate:` method to block the thread. Upon returning from these methods, go to step 2 to retest your boolean predicate. (Continue waiting and retesting the predicate until it is true.)
4. If the boolean predicate is true, perform the task.
5. Optionally update any predicates (or signal any conditions) affected by your task.
6. When your task is done, unlock the condition object.

The pseudocode for performing the preceding steps would therefore look something like the following:

```
lock the condition
while (!(boolean_predicate)) {
    wait on condition
```

```

}
do protected work
(optionally, signal or broadcast the condition again or change a predicate value)
unlock the condition

```

Whenever you use a condition object, the first step is to lock the condition. Locking the condition ensures that your predicate and task code are protected from interference by other threads using the same condition. Once you have completed your task, you can set other predicates or signal other conditions based on the needs of your code. You should always set predicates and signal conditions while holding the condition object's lock.

When a thread waits on a condition, the condition object unlocks its lock and blocks the thread. When the condition is signaled, the system wakes up the thread. The condition object then reacquires its lock before returning from the `wait` or `waitUntilDate:` method. Thus, from the point of view of the thread, it is as if it always held the lock.

A boolean predicate is an important part of the semantics of using conditions because of the way signaling works. Signaling a condition does not guarantee that the condition itself is true. There are timing issues involved in signaling that may cause false signals to appear. Using a predicate ensures that these spurious signals do not cause you to perform work before it is safe to do so. The predicate itself is simply a flag or other variable in your code that you test in order to acquire a Boolean result.

For more information on how to use conditions, see Using POSIX Thread Locks in *Threading Programming Guide*.

Tasks

Waiting for the Lock

- [wait](#) (page 8)
Blocks the current thread until the condition is signaled.
- [waitUntilDate:](#) (page 9)
Blocks the current thread until the condition is signaled or the specified time limit is reached.

Signaling Waiting Threads

- [signal](#) (page 8)
Signals the condition, waking up one thread waiting on it.
- [broadcast](#) (page 7)
Signals the condition, waking up all threads waiting on it.

Accessor Methods

- [setName:](#) (page 7)
Assigns a name to the receiver.

- [name](#) (page 7)
Returns the name associated with the receiver.

Instance Methods

broadcast

Signals the condition, waking up all threads waiting on it.

- (void)broadcast

Discussion

If no threads are waiting on the condition, this method does nothing.

To avoid race conditions, you should invoke this method only while the receiver is locked.

Availability

Available in Mac OS X v10.5 and later.

Declared In

NSLock.h

name

Returns the name associated with the receiver.

- (NSString *)name

Return Value

The name of the receiver.

Availability

Available in Mac OS X v10.5 and later.

See Also

- [setName:](#) (page 7)

Declared In

NSLock.h

setName:

Assigns a name to the receiver.

- (void)setName:(NSString *)*newName*

Parameters

newName

The new name for the receiver. This method makes a copy of the specified string.

Discussion

You can use a name string to identify a condition object within your code. Cocoa also uses this name as part of any error descriptions involving the receiver.

Availability

Available in Mac OS X v10.5 and later.

See Also

- [name](#) (page 7)

Declared In

NSLock.h

signal

Signals the condition, waking up one thread waiting on it.

```
- (void)signal
```

Discussion

You use this method to wake up one thread that is waiting on the condition. You may call this method multiple times to wake up multiple threads. If no threads are waiting on the condition, this method does nothing.

To avoid race conditions, you should invoke this method only while the receiver is locked.

Availability

Available in Mac OS X v10.5 and later.

Declared In

NSLock.h

wait

Blocks the current thread until the condition is signaled.

```
- (void)wait
```

Discussion

You must lock the receiver prior to calling this method.

Availability

Available in Mac OS X v10.5 and later.

See Also

- [lock](#) (NSLocking)

Declared In

NSLock.h

waitUntilDate:

Blocks the current thread until the condition is signaled or the specified time limit is reached.

```
- (BOOL)waitUntilDate:(NSDate *)limit
```

Parameters

limit

The time at which to wake up the thread if the condition has not been signaled.

Return Value

YES if the condition was signaled; otherwise, NO if the time limit was reached.

Discussion

You must lock the receiver prior to calling this method.

Availability

Available in Mac OS X v10.5 and later.

See Also

- lock (NSLocking)

Declared In

NSLock.h

Document Revision History

This table describes the changes to *NSCondition Class Reference*.

Date	Notes
2008-09-09	Corrected availability information.
2007-04-30	New document describing methods for implementing a POSIX-style condition-based lock.

REVISION HISTORY

Document Revision History

Index

B

broadcast [instance method 7](#)

N

name [instance method 7](#)

S

setName: [instance method 7](#)

signal [instance method 8](#)

W

wait [instance method 8](#)

waitUntilDate: [instance method 9](#)