

---

# CATransaction Class Reference

Graphics & Animation: Animation



2010-01-14



Apple Inc.  
© 2010 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, iPhone, and Quartz are trademarks of Apple Inc., registered in the United States and other countries.

iOS is a trademark or registered trademark of Cisco in the U.S. and other countries and is used under license.

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

## **CATransaction Class Reference 5**

---

Overview	5
Tasks	5
Creating and Committing Transactions	5
Overriding Animation Duration and Timing	6
Temporarily Disabling Property Animations	6
Getting and Setting Completion Block Objects	6
Managing Concurrency	6
Getting and Setting Transaction Properties	6
Class Methods	7
animationDuration	7
animationTimingFunction	7
begin	8
commit	8
completionBlock	8
disableActions	9
flush	9
lock	10
setAnimationDuration:	10
setAnimationTimingFunction:	10
setCompletionBlock:	11
setDisableActions:	11
setValue:forKey:	12
unlock	12
valueForKey:	12
Constants	13
Transaction properties	13

## **Document Revision History 15**

---



# CATransaction Class Reference

---

<b>Inherits from</b>	NSObject
<b>Conforms to</b>	NSObject (NSObject)
<b>Framework</b>	/System/Library/Frameworks/QuartzCore.framework
<b>Availability</b>	Available in iOS 2.0 and later.
<b>Declared in</b>	CATransaction.h
<b>Companion guides</b>	Core Animation Programming Guide Core Animation Cookbook

## Overview

`CATransaction` is the Core Animation mechanism for batching multiple layer-tree operations into atomic updates to the render tree. Every modification to a layer tree must be part of a transaction. Nested transactions are supported.

Core Animation supports two types of transactions: *implicit* transactions and *explicit* transactions. Implicit transactions are created automatically when the layer tree is modified by a thread without an active transaction and are committed automatically when the thread's run-loop next iterates. Explicit transactions occur when the application sends the `CATransaction` class a `begin` (page 8) message before modifying the layer tree, and a `commit` (page 8) message afterwards.

`CATransaction` allows you to override default animation properties that are set for animatable properties. You can customize duration, timing function, whether changes to properties trigger animations, and provide a handler that informs you when all animations from the transaction group are completed.

During a transaction you can temporarily acquire a recursive spin-lock for managing property atomicity.

## Tasks

### Creating and Committing Transactions

- + `begin` (page 8)  
Begin a new transaction for the current thread.
- + `commit` (page 8)  
Commit all changes made during the current transaction.

- + [flush](#) (page 9)  
Flushes any extant implicit transaction.

## Overriding Animation Duration and Timing

- + [animationDuration](#) (page 7)  
Returns the animation duration used by all animations within this transaction group.
- + [setAnimationDuration:](#) (page 10)  
Sets the animation duration used by all animations within this transaction group.
- + [animationTimingFunction](#) (page 7)  
Returns the timing function used for all animations within this transaction group.
- + [setAnimationTimingFunction:](#) (page 10)  
Sets the timing function used for all animations within this transaction group.

## Temporarily Disabling Property Animations

- + [disableActions](#) (page 9)  
Returns whether actions triggered as a result of property changes made within this transaction group are suppressed.
- + [setDisableActions:](#) (page 11)  
Sets whether actions triggered as a result of property changes made within this transaction group are suppressed.

## Getting and Setting Completion Block Objects

- + [completionBlock](#) (page 8)  
Returns the completion block object.
- + [setCompletionBlock:](#) (page 11)  
Sets the completion block object.

## Managing Concurrency

- + [lock](#) (page 10)  
Attempts to acquire a recursive spin-lock lock, ensuring that returned layer values are valid until unlocked.
- + [unlock](#) (page 12)  
Relinquishes a previously acquired transaction lock.

## Getting and Setting Transaction Properties

- + [setValue:forKey:](#) (page 12)  
Sets the arbitrary keyed-data for the specified key.

+ [valueForKey:](#) (page 12)

Returns the arbitrary keyed-data specified by the given key.

## Class Methods

### animationDuration

Returns the animation duration used by all animations within this transaction group.

+ (CTimeInterval)animationDuration

#### Return Value

An interval of time used as the duration.

#### Discussion

This is a convenience method that returns an `NSNumber` containing the seconds for the [valueForKey:](#) (page 12) value returned by the [kCATransactionAnimationDuration](#) (page 13) key.

#### Availability

Available in iOS 3.0 and later.

#### See Also

+ [setAnimationDuration:](#) (page 10)

#### Declared In

CATransaction.h

### animationTimingFunction

Returns the timing function used for all animations within this transaction group.

+ (CAMediaTimingFunction \*)animationTimingFunction

#### Return Value

An instance of `CAMediaTimingFunction`.

#### Discussion

This is a convenience method that returns the `CAMediaTimingFunction` for the [valueForKey:](#) (page 12) value returned by the [kCATransactionAnimationTimingFunction](#) (page 13) key.

#### Availability

Available in iOS 3.0 and later.

#### See Also

+ [setAnimationTimingFunction:](#) (page 10)

#### Declared In

CATransaction.h

## begin

Begin a new transaction for the current thread.

```
+ (void)begin
```

### Discussion

The transaction is nested within the thread's current transaction, if there is one.

### Availability

Available in iOS 2.0 and later.

### See Also

+ [commit](#) (page 8)

+ [flush](#) (page 9)

### Declared In

CATransaction.h

## commit

Commit all changes made during the current transaction.

```
+ (void)commit
```

### Special Considerations

Raises an exception if no current transaction exists.

### Availability

Available in iOS 2.0 and later.

### See Also

+ [begin](#) (page 8)

+ [flush](#) (page 9)

### Declared In

CATransaction.h

## completionBlock

Returns the completion block object.

```
+ (void)completionBlock
```

### Discussion

See [setCompletionBlock:](#) (page 11) for a description of the role of the completion block object.

### Availability

Available in iOS 4.0 and later.

### See Also

+ [completionBlock](#) (page 8)



**Declared In**

CATransaction.h

**disableActions**

Returns whether actions triggered as a result of property changes made within this transaction group are suppressed.

```
+ (BOOL)disableActions
```

**Return Value**

YES if actions are disabled.

**Discussion**

This is a convenience method that returns the `boolValue` for the `valueForKey:` (page 12) value returned by the `kCATransactionDisableActions` (page 13) key.

**Availability**

Available in iOS 3.0 and later.

**See Also**[+ setDisableActions:](#) (page 11)**Declared In**

CATransaction.h

**flush**

Flushes any extant implicit transaction.

```
+ (void)flush
```

**Discussion**

Delays the commit until any nested explicit transactions have completed.

Flush is typically called automatically at the end of the current runloop, regardless of the runloop mode. If your application does not have a runloop, you must call this method explicitly.

However, you should attempt to avoid calling `flush` explicitly. By allowing `flush` to execute during the runloop your application will achieve better performance, atomic screen updates will be preserved, and transactions and animations that work from transaction to transaction will continue to function.

**Availability**

Available in iOS 2.0 and later.

**See Also**[+ begin](#) (page 8)[+ commit](#) (page 8)**Declared In**

CATransaction.h

## lock

Attempts to acquire a recursive spin-lock lock, ensuring that returned layer values are valid until unlocked.

```
+ (void)lock
```

### Discussion

Core Animation uses a data model that promises not to corrupt the internal data structures when called from multiple threads concurrently, but not that data returned is still valid if the property was valid on another thread. By locking during a transaction you can ensure that data the is read, modified, and set is correctly managed.

### Availability

Available in iOS 3.0 and later.

### See Also

+ [unlock](#) (page 12)

### Declared In

CATransaction.h

## setAnimationDuration:

Sets the animation duration used by all animations within this transaction group.

```
+ (void)setAnimationDuration:(CFTimeInterval)duration
```

### Parameters

*duration*

An interval of time used as the duration.

### Discussion

This is a convenience method that sets an `NSNumber` containing the seconds for the `valueForKey:` (page 12) value of the `kCATransactionAnimationDuration` (page 13) key.

### Availability

Available in iOS 3.0 and later.

### See Also

+ [animationDuration](#) (page 7)

### Declared In

CATransaction.h

## setAnimationTimingFunction:

Sets the timing function used for all animations within this transaction group.

```
+ (void)setAnimationTimingFunction:(CAMediaTimingFunction *)function
```

### Parameters

*function*

An instance of `CAMediaTimingFunction`.

**Discussion**

This is a convenience method that sets the `CAMediaTimingFunction` for the `valueForKey:` (page 12) value of the `kCATransactionAnimationTimingFunction` (page 13) key.

**Availability**

Available in iOS 3.0 and later.

**See Also**

+ [animationTimingFunction](#) (page 7)

**Declared In**

`CATransaction.h`

**setCompletionBlock:**

Sets the completion block object.

```
+ (void)setCompletionBlock:(void (^)(void))block
```

**Parameters**

*block*

A block object called when animations for this transaction group are completed.

The block object takes no parameters and returns no value.

**Discussion**

The completion block object that is guaranteed to be called (on the main thread) as soon as all animations subsequently added by this transaction group have completed (or have been removed.) If no animations are added before the current transaction group is committed (or the completion block is set to a different value,) the block will be invoked immediately.

**Availability**

Available in iOS 4.0 and later.

**See Also**

+ [completionBlock](#) (page 8)

**Declared In**

`CATransaction.h`

**setDisableActions:**

Sets whether actions triggered as a result of property changes made within this transaction group are suppressed.

```
+ (void)setDisableActions:(BOOL)flag
```

**Parameters**

*flag*

YES, if actions should be disabled.

**Discussion**

This is a convenience method that invokes `setValue:forKey:` (page 12) with an `NSNumber` containing a YES for the `kCATransactionDisableActions` (page 13) key.

**Availability**

Available in iOS 3.0 and later.

**See Also**

+ [disableActions](#) (page 9)

**Declared In**

CATransaction.h

**setValue:forKey:**

Sets the arbitrary keyed-data for the specified key.

```
+ (void)setValue:(id)anObject forKey:(NSString *)key
```

**Parameters**

*anObject*

The value for the key identified by *key*.

*key*

The name of one of the receiver's properties.

**Discussion**

Nested transactions have nested data scope; setting a key always sets it in the innermost scope.

**Availability**

Available in iOS 2.0 and later.

**Declared In**

CATransaction.h

**unlock**

Relinquishes a previously acquired transaction lock.

```
+ (void)unlock
```

**Availability**

Available in iOS 3.0 and later.

**See Also**

+ [lock](#) (page 10)

**Declared In**

CATransaction.h

**valueForKey:**

Returns the arbitrary keyed-data specified by the given key.

```
+ (id)valueForKey:(NSString *)key
```

**Parameters***key*

The name of one of the receiver's properties.

**Return Value**

The value for the data specified by the key.

**Discussion**

Nested transactions have nested data scope. Requesting a value for a key first searches the innermost scope, then the enclosing transactions.

**Availability**

Available in iOS 2.0 and later.

**Declared In**

`CATransaction.h`

## Constants

### Transaction properties

These constants define the property keys used by `valueForKey:` (page 12) and `setValue:forKey:` (page 12).

```
NSString * const kCATransactionAnimationDuration;
NSString * const kCATransactionDisableActions;
NSString * const kCATransactionAnimationTimingFunction;
NSString * const kCATransactionCompletionBlock;
```

**Constants**

`kCATransactionAnimationDuration`

Duration, in seconds, for animations triggered within the transaction group. The value for this key must be an instance of `NSNumber`.

Available in iOS 2.0 and later.

Declared in `CATransaction.h`.

`kCATransactionDisableActions`

If YES, implicit actions for property changes made within the transaction group are suppressed. The value for this key must be an instance of `NSNumber`.

Available in iOS 2.0 and later.

Declared in `CATransaction.h`.

`kCATransactionAnimationTimingFunction`

An instance of `CAMediaTimingFunction` that overrides the timing function for all animations triggered within the transaction group.

Available in iOS 3.0 and later.

Declared in `CATransaction.h`.

`kCATransactionCompletionBlock`

A completion block object that is guaranteed to be called (on the main thread) as soon as all animations subsequently added by this transaction group have completed (or have been removed.) If no animations are added before the current transaction group is committed (or the completion block is set to a different value,) the block will be invoked immediately.

Available in iOS 4.0 and later.

Declared in `CATransaction.h`.

**Declared In**

`CATransaction.h`

# Document Revision History

---

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

Date	Notes
2010-01-14	Added discussion of runloops to +flush method.
2009-06-01	Updated for iOS 3.0. Added new convenience methods, methods for locking, overriding timing functions, and completion blocks.
2007-07-24	New document that describes the class that provides nested transaction support for Core Animation.

**REVISION HISTORY**

Document Revision History