

---

# CATransaction Class Reference

[Graphics & Imaging](#) > Quartz



2007-07-24



Apple Inc.  
© 2007 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, and Quartz are trademarks of Apple Inc., registered in the United States and other countries.

iPhone is a trademark of Apple Inc.

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
  - Getting and Setting Transaction Properties 6
- Class Methods 6
  - begin 6
  - commit 6
  - flush 6
  - setValue:forKey: 7
  - valueForKey: 7
- Constants 8
  - Transaction properties 8

---

## **Document Revision History 9**

---

---

## **Index 11**

---



# 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 Mac OS X v10.5 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 6) message before modifying the layer tree, and a `commit` (page 6) message afterwards.

In some circumstances (for example, if there is no run-loop, or the run-loop is blocked) it may be necessary to use explicit transactions to get timely render tree updates.

## Tasks

### Creating and Committing Transactions

- + `begin` (page 6)  
Begin a new transaction for the current thread.
- + `commit` (page 6)  
Commit all changes made during the current transaction.
- + `flush` (page 6)  
Flushes any extant implicit transaction.

## Getting and Setting Transaction Properties

- + [valueForKey:](#) (page 7)  
Returns the arbitrary keyed-data specified by the given key.
- + [setValue:forKey:](#) (page 7)  
Sets the arbitrary keyed-data for the specified key.

## Class Methods

### 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 Mac OS X v10.5 and later.

#### 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 Mac OS X v10.5 and later.

#### Declared In

CATransaction.h

### flush

Flushes any extant implicit transaction.

```
+ (void)flush
```

#### Discussion

Delays the commit until any nested explicit transactions have completed.

**Availability**

Available in Mac OS X v10.5 and later.

**Declared In**

CATransaction.h

**setValueForKey:**

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 Mac OS X v10.5 and later.

**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 Mac OS X v10.5 and later.

**Declared In**

CATransaction.h

## Constants

### Transaction properties

These constants define the property keys used by `valueForKey:` (page 7) and `setValue:forKey:` (page 7).

```
NSString * const kCATransactionAnimationDuration;  
NSString * const kCATransactionDisableActions;
```

#### Constants

`kCATransactionAnimationDuration`

Default duration, in seconds, for animations added to layers. The value for this key must be an instance of `NSNumber`.

Available in Mac OS X v10.5 and later.

Declared in `CATransaction.h`.

`kCATransactionDisableActions`

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

Available in Mac OS X v10.5 and later.

Declared in `CATransaction.h`.

#### Declared In

`CATransaction.h`



# Document Revision History

---

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

Date	Notes
2007-07-24	New document that describes the class that provides nested transaction support for Core Animation.

## REVISION HISTORY

### Document Revision History

# Index

---

## B

---

`begin` class method [6](#)

## C

---

`commit` class method [6](#)

## F

---

`flush` class method [6](#)

## K

---

`kCATransactionAnimationDuration` constant [8](#)

`kCATransactionDisableActions` constant [8](#)

## S

---

`setValue:forKey:` class method [7](#)

## T

---

Transaction properties [8](#)

## V

---

`valueForKey:` class method [7](#)