

Core Motion Framework Reference

Contents

Core Motion Framework Reference 4

Classes 5

CMAccelerometerData Class Reference 6

Overview 6

Tasks 6

Properties 7

Constants 7

CMAccelerometerData Class Reference 9

Overview 9

Tasks 10

Properties 11

Instance Methods 13

Constants 14

CMDeviceMotion Class Reference 17

Overview 17

Tasks 18

Properties 18

Constants 21

CMGyroData Class Reference 23

Overview 23

Tasks 23

Properties 24

Constants 24

CMLogItem Class Reference 26

Overview 26

Tasks 26

Properties 27

CMagnetometerData Class Reference 28

Overview 28

Tasks 28

Properties 29

Constants 29

CMotionActivity Class Reference 31

Overview 31

Tasks 32

Properties 32

Constants 34

CMotionActivityManager Class Reference 36

Overview 36

Tasks 36

Class Methods 37

Instance Methods 37

Constants 40

CMotionManager Class Reference 42

Overview 42

Tasks 44

Properties 48

Class Methods 56

Instance Methods 57

Constants 64

CMStepCounter Class Reference 70

Overview 70

Tasks 70

Class Methods 71

Instance Methods 71

Constants 74

Document Revision History 76

Core Motion Framework Reference

Framework	/System/Library/Frameworks/CoreMotion.framework
Header file directories	/System/Library/Frameworks/CoreMotion.framework/Headers
Companion guide	Event Handling Guide for iOS
Declared in	CMAccelerometer.h CMAttitude.h CMDeviceMotion.h CMError.h CMErrorDomain.h CMGyro.h CMLogItem.h CMMagnetometer.h CMMotionActivity.h CMMotionActivityManager.h CMMotionManager.h CMStepCounter.h

The Core Motion framework lets your application receive motion data from device hardware and process that data. The framework supports accessing both raw and processed accelerometer data using block-based interfaces. For devices with a built-in gyroscope, you can retrieve the raw gyro data as well as processed data reflecting the attitude and rotation rates of the device. You can use both the accelerometer and gyro-based data for games or other apps that use motion as input or as a way to enhance the overall user experience.

Classes

CMAccelerometerData Class Reference

Inherits from	CMLogItem : NSObject
Conforms to	NSCopying (CMLogItem) NSSecureCoding (CMLogItem) NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreMotion.framework
Availability	Available in iOS 4.0 and later.
Declared in	CMAccelerometer.h

Overview

An instance of the `CMAccelerometerData` class represents an accelerometer event. It is a measurement of acceleration along the three spatial axes at a moment of time.

An application accesses `CMAccelerometerData` objects through the block handler specified as the last parameter of the [startAccelerometerUpdatesToQueue:withHandler:](#) (page 57) method and through the `accelerometerData` property, both declared by the `CMMotionManager` class. The superclass of `CMAccelerometerData`, `CMLogItem`, defines a `timestamp` property that records when the acceleration measurement was taken.

Tasks

Accessing Accelerometer Data

[acceleration](#) (page 7) *property*

The acceleration measured by the accelerometer. (read-only)

Properties

acceleration

The acceleration measured by the accelerometer. (read-only)

```
@property(readonly, nonatomic) CMAcceleration acceleration
```

Discussion

The description of the [CMAcceleration](#) (page 7) structure type describes the fields used for measuring acceleration.

Availability

Available in iOS 4.0 and later.

Declared in

CMAccelerometer.h

Constants

CMAcceleration

The type of a structure containing 3-axis acceleration values.

```
typedef struct {  
    double x;  
    double y;  
    double z;  
} CMAcceleration;
```

Fields

x

X-axis acceleration in G's (gravitational force).

y

Y-axis acceleration in G's (gravitational force).

z

Z-axis acceleration in G's (gravitational force).

Discussion

A G is a unit of gravitation force equal to that exerted by the earth's gravitational field (9.81 m s^{-2}).

Availability

Available in iOS 4.0 and later.

Declared in

CMAccelerometer.h

CMAttitude Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSSecureCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreMotion.framework
Availability	Available in iOS 4.0 and later.
Declared in	CMAttitude.h
Companion guide	Event Handling Guide for iOS

Overview

An instance of the `CMAttitude` class represents a measurement of the device's attitude at a point in time. "Attitude" refers to the orientation of a body relative to a given frame of reference.

The `CMAttitude` class offers three different mathematical representations of attitude: a rotation matrix, a quaternion, and Euler angles (roll, pitch, and yaw values). You access `CMAttitude` objects through the `attitude` property of each `CMDeviceMotion` objects passed to an application. An application starts receiving these device-motion objects as a result of calling the [startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:](#) (page 60) method, the [startDeviceMotionUpdatesToQueue:withHandler:](#) (page 58) method, the [startDeviceMotionUpdatesUsingReferenceFrame:](#) (page 59) method or the [startDeviceMotionUpdates](#) (page 58) method of the `CMMotionManager` class.

Note: Core Motion outputs a direction cosine matrix (DCM)—basically a rotation from the last "old" orientation to the new orientation of the device.

Tasks

Getting a Mathematical Representation of Attitude as Euler Angles

`roll` (page 11) *property*

The roll of the device, in radians. (read-only)

`pitch` (page 11) *property*

The pitch of the device, in radians. (read-only)

`yaw` (page 12) *property*

The yaw of the device, in radians. (read-only)

Getting a Mathematical Representation of Attitude as a Rotation Matrix

`rotationMatrix` (page 12) *property*

Returns a rotation matrix representing the device's attitude. (read-only)

Getting a Mathematical Representation of Attitude as a Quaternion

`quaternion` (page 11) *property*

Returns a quaternion representing the device's attitude. (read-only)

Obtaining the Change in Attitude

– `multiplyByInverseOfAttitude:` (page 13)

Yields the change in attitude given a specific attitude.

Properties

pitch

The pitch of the device, in radians. (read-only)

```
@property(readonly, nonatomic) double pitch
```

Discussion

A pitch is a rotation around a lateral axis that passes through the device from side to side.

Availability

Available in iOS 4.0 and later.

See Also

[@property roll](#) (page 11)

[@property yaw](#) (page 12)

Declared in

CMAttitude.h

quaternion

Returns a quaternion representing the device's attitude. (read-only)

```
@property(readonly, nonatomic) CMQuaternion quaternion
```

Discussion

See the discussion of the [CMQuaternion](#) (page 14) type in “Constants” for further information.

Availability

Available in iOS 4.0 and later.

See Also

[@property rotationMatrix](#) (page 12)

Declared in

CMAttitude.h

roll

The roll of the device, in radians. (read-only)

```
@property(readonly, nonatomic) double roll
```

Discussion

A roll is a rotation around a longitudinal axis that passes through the device from its top to bottom.

Availability

Available in iOS 4.0 and later.

See Also

[@property pitch](#) (page 11)

[@property yaw](#) (page 12)

Declared in

CMAAttitude.h

rotationMatrix

Returns a rotation matrix representing the device's attitude. (read-only)

```
@property(readonly, nonatomic) CMRotationMatrix rotationMatrix
```

Discussion

A rotation matrix in linear algebra describes the rotation of a body in three-dimensional Euclidean space.

Availability

Available in iOS 4.0 and later.

See Also

[@property quaternion](#) (page 11)

Related Sample Code

pARk

Declared in

CMAAttitude.h

yaw

The yaw of the device, in radians. (read-only)

```
@property(readonly, nonatomic) double yaw
```

Discussion

A yaw is a rotation around an axis that runs vertically through the device. It is perpendicular to the body of the device, with its origin at the center of gravity and directed toward the bottom of the device.

Availability

Available in iOS 4.0 and later.

See Also

[@property roll](#) (page 11)

[@property pitch](#) (page 11)

Declared in

CMAAttitude.h

Instance Methods

multiplyByInverseOfAttitude:

Yields the change in attitude given a specific attitude.

– (void)multiplyByInverseOfAttitude:(CMAAttitude *)attitude

Parameters

attitude

An object representing the device's attitude at a given moment of measurement.

Discussion

This method multiplies the inverse of the specified CMAAttitude object by the attitude represented by the receiving object. It replaces the receiving instance with the attitude *change* relative to the object passed in attitude. You should cache the CMAAttitude instance you want to use as a reference and pass that object as the argument to subsequent calls of this method.

Availability

Available in iOS 4.0 and later.

Declared in

CMAAttitude.h

Constants

CMRotationMatrix

The type of a structure representing a rotation matrix.

```
typedef struct
{
    double m11, m12, m13;
    double m21, m22, m23;
    double m31, m32, m33;
} CMRotationMatrix;
```

Fields

m11–m33

Each field in this structure defines an element of the rotation matrix by its position. For example, m11 is the element in row 1, column 1; m31 is the element in row 3, column 1; m13 is the element in row 1, column 3.

Availability

Available in iOS 4.0 and later.

Declared in

CMAAttitude.h

CMQuaternion

The type for a quaternion representing a measurement of attitude.

```
typedef struct {
    double x, y, z, w;
} CMQuaternion
```

Constants

x

A value for the X-axis.

y

A value for the Y-axis.

z

A value for the Z-axis.

w

A value for the W-axis.

Discussion

A quaternion offers a way to parameterize attitude. If `q` is an instance of `CMQuaternion`, mathematically it represents the following unit quaternion: $q.x*i + q.y*j + q.z*k + q.w$. A unit quaternion represents a rotation of θ radians about the unit vector $\{x, y, z\}$, and $\{q.x, q.y, q.z, q.w\}$ satisfies the following:

```
q.x = x * sin(theta / 2)
q.y = y * sin(theta / 2)
q.z = z * sin(theta / 2)
q.w = cos(theta / 2)
```

Availability

Available in iOS 4.0 and later.

Declared in

`CMAttitude.h`

CMAttitudeReferenceFrame

Enum constants for indicating the reference frames from which all attitude samples are referenced.

```
typedef enum {
    CMAttitudeReferenceFrameXArbitraryZVertical = 1 << 0,
    CMAttitudeReferenceFrameXArbitraryCorrectedZVertical = 1 << 1,
    CMAttitudeReferenceFrameXMagneticNorthZVertical = 1 << 2,
    CMAttitudeReferenceFrameXTrueNorthZVertical = 1 << 3
} CMAttitudeReferenceFrame;
```

Constants

`CMAttitudeReferenceFrameXArbitraryZVertical`

Describes a reference frame in which the Z axis is vertical and the X axis points in an arbitrary direction in the horizontal plane.

Available in iOS 5.0 and later.

Declared in `CMAttitude.h`.

`CMAAttitudeReferenceFrameXArbitraryCorrectedZVertical`

Describes the same reference frame as `CMAAttitudeReferenceFrameXArbitraryZVertical` except that the magnetometer, when available and calibrated, is used to improve long-term yaw accuracy. Using this constant instead of `CMAAttitudeReferenceFrameXArbitraryZVertical` results in increased CPU usage.

Available in iOS 5.0 and later.

Declared in `CMAAttitude.h`.

`CMAAttitudeReferenceFrameXMagneticNorthZVertical`

Describes a reference frame in which the Z axis is vertical and the X axis points toward magnetic north. Note that using this reference frame may require device movement to calibrate the magnetometer.

Available in iOS 5.0 and later.

Declared in `CMAAttitude.h`.

`CMAAttitudeReferenceFrameXTrueNorthZVertical`

Describes a reference frame in which the Z axis is vertical and the X axis points toward true north. Note that using this reference frame may require device movement to calibrate the magnetometer. It also requires the location to be available in order to calculate the difference between magnetic and true north.

Available in iOS 5.0 and later.

Declared in `CMAAttitude.h`.

Discussion

Constants of this data type are returned or referenced by the `attitudeReferenceFrame` property and the [startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:](#) (page 60) and [startDeviceMotionUpdatesUsingReferenceFrame:](#) (page 59) methods of the `CMMotionManager` class. A bitmask of these constants is returned by the [availableAttitudeReferenceFrames](#) (page 56) class method.

Declared in

`CMAAttitude.h`

CMDeviceMotion Class Reference

Inherits from	CMLogItem : NSObject
Conforms to	NSCopying (CMLogItem) NSSecureCoding (CMLogItem) NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreMotion.framework
Availability	Available in iOS 4.0 and later.
Declared in	CMDeviceMotion.h
Companion guide	Event Handling Guide for iOS
Related sample code	pARk

Overview

An instance of `CMDeviceMotion` encapsulates measurements of the attitude, rotation rate, and acceleration of a device.

An application receives or samples `CMDeviceMotion` objects at regular intervals after calling the [startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:](#) (page 60) method, the [startDeviceMotionUpdatesToQueue:withHandler:](#) (page 58) method, the [startDeviceMotionUpdatesUsingReferenceFrame:](#) (page 59) method, or the [startDeviceMotionUpdates](#) (page 58) method of the `CMMotionManager` class.

The accelerometer measures the sum of two acceleration vectors: gravity and user acceleration. User acceleration is the acceleration that the user imparts to the device. Because Core Motion is able to track a device's attitude using both the gyroscope and the accelerometer, it can differentiate between gravity and user acceleration. A `CMDeviceMotion` object provides both measurements in the [gravity](#) (page 19) and [userAcceleration](#) (page 20) properties.

Tasks

Getting Attitude and Rotation Rate

[attitude](#) (page 18) *property*

The attitude of the device. (read-only)

[rotationRate](#) (page 20) *property*

The rotation rate of the device. (read-only)

Getting Acceleration Data

[gravity](#) (page 19) *property*

The gravity acceleration vector expressed in the device's reference frame. (read-only)

[userAcceleration](#) (page 20) *property*

The acceleration that the user is giving to the device. (read-only)

Getting the Calibrated Magnetic Field

[magneticField](#) (page 19) *property*

Returns the magnetic field vector with respect to the device. (read-only)

Properties

attitude

The attitude of the device. (read-only)

```
@property(readonly, nonatomic) CMAcceleration *attitude
```

Discussion

A `CMAcceleration` object represents a measurement of attitude—that is, the orientation of a body relative to a given frame of reference.

Availability

Available in iOS 4.0 and later.

See Also

[@property rotationRate](#) (page 20)

Related Sample Code

pARk

Declared in

CMDeviceMotion.h

gravity

The gravity acceleration vector expressed in the device's reference frame. (read-only)

```
@property(readonly, nonatomic) CMAcceleration gravity
```

Discussion

The total acceleration of the device is equal to gravity plus the acceleration the user imparts to the device ([userAcceleration](#) (page 20)).

Availability

Available in iOS 4.0 and later.

Declared in

CMDeviceMotion.h

magneticField

Returns the magnetic field vector with respect to the device. (read-only)

```
@property(readonly, nonatomic) CMCalibratedMagneticField magneticField
```

Discussion

The [CMCalibratedMagneticField](#) (page 21) returned by this property gives you the total magnetic field in the device's vicinity without device bias. Unlike the `magneticField` property of the `CMMagnetometer` class, these values reflect the earth's magnetic field plus surrounding fields, minus device bias.

If the device does not have a magnetometer, the `accuracy` field of the property's value (a [CMCalibratedMagneticField](#) (page 21) structure) is [CMMagneticFieldCalibrationAccuracyUncalibrated](#) (page 21).

Availability

Available in iOS 5.0 and later.

Declared in
CMDeviceMotion.h

rotationRate

The rotation rate of the device. (read-only)

```
@property(readonly, nonatomic) CMRotationRate rotationRate
```

Discussion

A [CMRotationRate](#) (page 24) structure contains data specifying the device's rate of rotation around three axes. The value of this property contains a measurement of gyroscope data whose bias has been removed by Core Motion algorithms. The identically name property of [CMGyroData](#), on the other hand, gives the raw data from the gyroscope. The structure type is declared in [CMGyroData.h](#).

Availability

Available in iOS 4.0 and later.

See Also

[@property attitude](#) (page 18)

Declared in
CMDeviceMotion.h

userAcceleration

The acceleration that the user is giving to the device. (read-only)

```
@property(readonly, nonatomic) CMAcceleration userAcceleration
```

Discussion

The total acceleration of the device is equal to [gravity](#) (page 19) plus the acceleration the user imparts to the device.

Availability

Available in iOS 4.0 and later.

Declared in
CMDeviceMotion.h

Constants

CMCalibratedMagneticField

Calibrated magnetic field data and an estimate of the accuracy of the calibration.

```
typedef struct {  
    CMMagneticField field;  
    CMMagneticFieldCalibrationAccuracy accuracy;  
} CMCalibratedMagneticField;
```

Fields

field

A structure containing 3-axis calibrated magnetic field data. See the description of the [CMMagneticField](#) (page 29) structure.

accuracy

An enum-constant value that indicates the accuracy of the magnetic field estimate. See [“CMMagneticFieldCalibrationAccuracy”](#) (page 21).

Availability

Available in iOS 5.0 and later.

Declared in

CMDeviceMotion.h

CMMagneticFieldCalibrationAccuracy

Indicates the calibration accuracy of a magnetic field estimate

```
typedef enum {    CMMagneticFieldCalibrationAccuracyUncalibrated = -1,  
    CMMagneticFieldCalibrationAccuracyLow,  
    CMMagneticFieldCalibrationAccuracyMedium,  
    CMMagneticFieldCalibrationAccuracyHigh } CMMagneticFieldCalibrationAccuracy;
```

Constants

CMMagneticFieldCalibrationAccuracyUncalibrated

The magnetic field estimate is not calibrated.

Available in iOS 5.0 and later.

Declared in CMDeviceMotion.h.

`CMMagneticFieldCalibrationAccuracyLow`

The accuracy of the magnetic field calibration is low.

Available in iOS 5.0 and later.

Declared in `CMDeviceMotion.h`.

`CMMagneticFieldCalibrationAccuracyMedium`

The accuracy of the magnetic field calibration is medium.

Available in iOS 5.0 and later.

Declared in `CMDeviceMotion.h`.

`CMMagneticFieldCalibrationAccuracyHigh`

The accuracy of the magnetic field calibration is high.

Available in iOS 5.0 and later.

Declared in `CMDeviceMotion.h`.

Discussion

One of the enum constants of the `CMMagneticFieldCalibrationAccuracy` type is the value of the accuracy field of the [CMCalibratedMagneticField](#) (page 21) structure returned from the [magneticField](#) (page 19) property.

Declared in

`CMDeviceMotion.h`

CMGyroData Class Reference

Inherits from	CMLogItem : NSObject
Conforms to	NSCopying (CMLogItem) NSSecureCoding (CMLogItem) NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreMotion.framework
Availability	Available in iOS 4.0 and later.
Declared in	CMGyro.h
Companion guide	Event Handling Guide for iOS

Overview

An instance of the `CMGyroData` class contains a single measurement of the device's rotation rate.

An application receives or samples `CMGyroData` objects at regular intervals after calling the [startGyroUpdatesToQueue:withHandler:](#) (page 61) method or the [startGyroUpdates](#) (page 60) method of the `CMMotionManager` class.

Tasks

Getting the Rotation Rate

[rotationRate](#) (page 24) *property*

The rotation rate as measured by the device's gyroscope. (read-only)

Properties

rotationRate

The rotation rate as measured by the device's gyroscope. (read-only)

```
@property(readonly, nonatomic) CMRotationRate rotationRate
```

Discussion

This property yields a measurement of the device's rate of rotation around three axes. Whereas this property gives the raw data from the gyroscope, the identically named property of `CMDeviceMotion` gives a [CMRotationRate](#) (page 24) structure measuring gyroscope data whose bias has been removed by Core Motion algorithms.

Availability

Available in iOS 4.0 and later.

Declared in

CMGyro.h

Constants

CMRotationRate

The type of structures representing a measurement of rotation rate.

```
typedef struct {  
    double x;  
    double y;  
    double z;  
} CMRotationRate
```

Constants

x

The X-axis rotation rate in radians per second. The sign follows the right hand rule: If the right hand is wrapped around the X axis such that the tip of the thumb points toward positive X, a positive rotation is one toward the tips of the other four fingers.

y

The Y-axis rotation rate in radians per second. The sign follows the right hand rule: If the right hand is wrapped around the Y axis such that the tip of the thumb points toward positive Y, a positive rotation is one toward the tips of the other four fingers.

z

The Z-axis rotation rate in radians per second. The sign follows the right hand rule: If the right hand is wrapped around the Z axis such that the tip of the thumb points toward positive Z, a positive rotation is one toward the tips of the other four fingers.

Availability

Available in iOS 4.0 and later.

Declared in

CMGyro.h

CMLogItem Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSSecureCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreMotion.framework
Availability	Available in iOS 4.0 and later.
Declared in	CMLogItem.h

Overview

The `CMLogItem` class is a base class for Core Motion classes that handle specific types of motion events. Objects of this class represent a piece of time-tagged data that can be logged to a file.

`CMLogItem` defines a read-only [timestamp](#) (page 27) property that records the time a motion-event measurement was taken.

Tasks

Getting the Time of the Event

[timestamp](#) (page 27) *property*

The time when the logged item is valid. (read-only)

Properties

timestamp

The time when the logged item is valid. (read-only)

```
@property(readonly, nonatomic) NSTimeInterval timestamp
```

Discussion

The time stamp is the amount of time in seconds since the phone booted.

Availability

Available in iOS 4.0 and later.

Declared in

CMLogItem.h

CMMagnetometerData Class Reference

Inherits from	CMLogItem : NSObject
Conforms to	NSCopying (CMLogItem) NSSecureCoding (CMLogItem) NSObject (NSObject)
Framework	/System/Library/Framework/CoreMotion.framework
Availability	Available in iOS 5.0 and later.
Declared in	CMMagnetometer.h

Overview

Instances of the `CMMagnetometerData` class encapsulated measurements of the magnetic field made by the device's magnetometer.

Your application can obtain samples of magnetometer measurements, as represented by instances of this class, from the block handler of the `startMagnetometerUpdatesToQueue:withHandler:` (page 62) method or from the `magnetometerData` (page 55) property of the `CMMotionManager` class.

Note: The `magnetometerData` (page 55) property of `CMMotionManager` provides a non-nil value only if you have called the `startMagnetometerUpdates` (page 61) method or the `startMagnetometerUpdatesToQueue:withHandler:` method to start magnetometer updates.

Tasks

MethodGroup

`magneticField` (page 29) *property*

Returns the magnetic field measured by the magnetometer. (read-only)

Properties

magneticField

Returns the magnetic field measured by the magnetometer. (read-only)

```
@property(readonly, nonatomic) CMMagneticField magneticField
```

Discussion

The value of this property is the total magnetic field observed by the device which is equal to the Earth's geomagnetic field plus bias introduced from the device itself and its surroundings.

This is the “raw” magnetic-field value, unlike the calibrated value of the [magneticField](#) (page 19) property of `CMDeviceMotion` which filters out the bias introduced by the device and, in some cases, its surrounding fields.

Availability

Available in iOS 5.0 and later.

Declared in

`CMMagnetometer.h`

Constants

CMMagneticField

A structure containing 3-axis magnetometer data

```
typedef struct {  
    double x;  
    double y;  
    double z;  
} CMMagneticField;
```

Fields

x

X-axis magnetic field in microteslas.

y

Y-axis magnetic field in microteslas.

z

Z-axis magnetic field in microteslas.

Availability

Available in iOS 5.0 and later.

Declared in

CMMagnetometer.h

CMMotionActivity Class Reference

Inherits from	CMLogItem : NSObject
Conforms to	NSCopying (CMLogItem) NSSecureCoding (CMLogItem) NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreMotion.framework
Availability	Available in iOS 7.0 and later.
Declared in	CMMotionActivity.h

Overview

The `CMMotionActivity` class contains the data for a single motion update event. On devices that support motion, you can use a `CMMotionActivityManager` object to request updates when the current type of motion changes. When a change occurs, the update information is packaged into a `CMMotionActivity` object and sent to your app.

The motion-related properties of this class are not mutually exclusive. In other words, it is possible for more than one of the motion-related properties to contain the value `YES`. For example, if the user was driving in a car and the car stopped at a red light, the update event associated with that change in motion would have both the [automotive](#) (page 32) and [stationary](#) (page 33) properties set to `YES`. It is also possible for all of the properties to be set to `NO` when the device is in motion but the movement does not correlate to walking, running, or automotive travel.

You do not create instances of this class yourself. The `CMMotionActivityManager` object creates them and sends them to the handler block you registered. For more information about how to initiate the delivery of motion activity updates to your app, see *CMMotionActivityManager Class Reference*.

Tasks

Getting the Type of Motion

[stationary](#) (page 33) *property*

A Boolean indicating whether the device is stationary. (read-only)

[walking](#) (page 34) *property*

A Boolean indicating whether the device is on a walking person. (read-only)

[running](#) (page 33) *property*

A Boolean indicating whether the device is on a running person. (read-only)

[automotive](#) (page 32) *property*

A Boolean indicating whether the device is in an automobile. (read-only)

[unknown](#) (page 34) *property*

A Boolean indicating whether the type of motion is unknown. (read-only)

Getting Metadata for the Motion

[startDate](#) (page 33) *property*

The time at which the change in motion occurred. (read-only)

[confidence](#) (page 33) *property*

The confidence in the assessment of the motion type. (read-only)

Properties

automotive

A Boolean indicating whether the device is in an automobile. (read-only)

```
@property(readonly, nonatomic) BOOL automotive;
```

Availability

Available in iOS 7.0 and later.

Declared in

CMMotionActivity.h

confidence

The confidence in the assessment of the motion type. (read-only)

```
@property(readonly, nonatomic) CMMotionActivityConfidence confidence;
```

Availability

Available in iOS 7.0 and later.

Declared in

CMMotionActivity.h

running

A Boolean indicating whether the device is on a running person. (read-only)

```
@property(readonly, nonatomic) BOOL running;
```

Availability

Available in iOS 7.0 and later.

Declared in

CMMotionActivity.h

startDate

The time at which the change in motion occurred. (read-only)

```
@property(readonly, nonatomic) NSDate *startDate;
```

Availability

Available in iOS 7.0 and later.

Declared in

CMMotionActivity.h

stationary

A Boolean indicating whether the device is stationary. (read-only)

```
@property(readonly, nonatomic) BOOL stationary;
```

Availability

Available in iOS 7.0 and later.

Declared in

CMMotionActivity.h

unknown

A Boolean indicating whether the type of motion is unknown. (read-only)

```
@property(readonly, nonatomic) BOOL unknown;
```

Discussion

This property is set to YES when there is no way to estimate the current type of motion. For example, this property might be YES if the device was turned on recently and not enough motion data had been gathered to determine the type of motion.

Availability

Available in iOS 7.0 and later.

Declared in

CMMotionActivity.h

walking

A Boolean indicating whether the device is on a walking person. (read-only)

```
@property(readonly, nonatomic) BOOL walking;
```

Availability

Available in iOS 7.0 and later.

Declared in

CMMotionActivity.h

Constants

CMMotionActivityConfidence

The confidence that the motion data is accurate.

```
typedef enum : NSInteger {  
    CMMotionActivityConfidenceLow = 0,
```

```
CMMotionActivityConfidenceMedium,  
CMMotionActivityConfidenceHigh  
}CMMotionActivityConfidence;
```

Constants

CMMotionActivityConfidenceLow

Confidence is low.

Available in iOS 7.0 and later.

Declared in CMMotionActivity.h.

CMMotionActivityConfidenceMedium

Confidence is good.

Available in iOS 7.0 and later.

Declared in CMMotionActivity.h.

CMMotionActivityConfidenceHigh

Confidence is high.

Available in iOS 7.0 and later.

Declared in CMMotionActivity.h.

CMMotionActivityManager Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreMotion.framework
Availability	Available in iOS 7.0 and later.
Declared in	CMMotionActivityManager.h

Overview

The `CMMotionActivityManager` class provides access to the motion data stored by a device. Motion data reflects whether the user is walking, running, in a vehicle, or stationary for periods of time. A navigation app might look for changes in the current type of motion and offer different directions for each. Using this class, you can ask for notifications when the current type of motion changes or you can gather past motion change data.

Tasks

Determining Activity Availability

+ [isActivityAvailable](#) (page 37)

Returns a Boolean indicating whether motion data is available on the current device.

Starting and Stopping Activity Updates

– [startActivityUpdatesToQueue:withHandler:](#) (page 38)

Starts the delivery of current motion data updates to your app.

– [stopActivityUpdates](#) (page 39)

Stops the delivery of motion updates to your app

Getting Historical Activity Data

– [queryActivityStartingFromDate:toDate:toQueue:withHandler:](#) (page 37)

Gathers and returns historical motion data for the specified time period

Class Methods

isActivityAvailable

Returns a Boolean indicating whether motion data is available on the current device.

+ (BOOL)isActivityAvailable

Return Value

YES if motion data is available or NO if it is not.

Discussion

Motion data is not available on all iOS devices. Use this method to determine if support is available on the current device.

Availability

Available in iOS 7.0 and later.

Declared in

CMMotionActivityManager.h

Instance Methods

queryActivityStartingFromDate:toDate:toQueue:withHandler:

Gathers and returns historical motion data for the specified time period

– (void)queryActivityStartingFromDate:(NSDate *)start toDate:(NSDate *)end
toQueue:(NSOperationQueue *)queue withHandler:(CMMotionActivityQueryHandler)handler

Parameters

start

The start time to use when gathering motion data. This parameter must not be nil.

end

The end time to use when gathering motion data. This parameter must not be nil.

queue

The operation queue on which to execute the specified `handler` block. You can specify a custom queue or use the operation queue associated with your app's main thread. This parameter must not be `nil`.

handler

The block to execute with the results. For information about the parameters of this block, see [CMMotionActivityQueryHandler](#) (page 40). This parameter must not be `nil`.

Discussion

This method runs asynchronously, returning immediately and delivering the results to the specified `handler` block. The system stores only the last seven days worth of activity data at most. If there are no samples for the specified range of time, an error object with the code [CMErrorUnknown](#) (page 68) is passed to the `handler` block.

Availability

Available in iOS 7.0 and later.

Declared in

`CMMotionActivityManager.h`

startActivityUpdatesToQueue:withHandler:

Starts the delivery of current motion data updates to your app.

```
– (void)startActivityUpdatesToQueue:(NSOperationQueue *)queue  
withHandler:(CMMotionActivityHandler)handler
```

Parameters

queue

The operation queue on which to execute the specified `handler` block. You can specify a custom queue or use the operation queue associated with your app's main thread. This parameter must not be `nil`.

handler

The block to execute when a change in the current type of motion is detected. For information about the parameters of this block, see [CMMotionActivityHandler](#) (page 40). This property must not be `nil`.

Discussion

This method initiates the tracking of motion data asynchronously. Upon calling this method, the motion activity manager executes the `handler` block on the specified queue, reporting the current motion in effect for the device. After that, the motion activity manager executes the `handler` block only when the motion data changes.

The `handler` block is executed on a best effort basis and updates are not delivered while your app is suspended. If updates arrived while your app was suspended, the last update is delivered to your app when it resumes execution. To get all of the updates that occurred while your app was suspended, use the [queryActivityStartingFromDate:toDate:toQueue:withHandler:](#) (page 37) method.

Upon starting activity updates, the motion activity manager reports changes to its block until you call the `stopActivityUpdates` method. If you call this method again with a new block, the activity manager stops delivering updates to the old block and delivers them to the new block instead. Updates stop altogether when the motion activity manager object itself is deallocated.

Availability

Available in iOS 7.0 and later.

See Also

– [stopActivityUpdates](#) (page 39)

Declared in

`CMMotionActivityManager.h`

stopActivityUpdates

Stops the delivery of motion updates to your app

– `(void)stopActivityUpdates`

Discussion

Call this method to stop the delivery of updates that you started by calling the [startActivityUpdatesToQueue:withHandler:](#) (page 38) method. This method does not stop queries started using the [queryActivityStartingFromDate:toDate:toQueue:withHandler:](#) method.

Availability

Available in iOS 7.0 and later.

See Also

– [startActivityUpdatesToQueue:withHandler:](#) (page 38)

Declared in

`CMMotionActivityManager.h`

Constants

CMMotionActivityHandler

A block that reports the current motion associated with the device.

```
typedef void (^CMMotionActivityHandler)(CMMotionActivity *activity)
```

Discussion

This block takes the following parameter:

activity

The motion activity object that defines the current type of motion for the device.

Availability

Available in iOS 7.0 and later.

Declared in

CMMotionActivityManager.h

CMMotionActivityQueryHandler

A block that reports the motion updates that occurred between the specified query interval.

```
typedef void (^CMMotionActivityQueryHandler)(NSArray *activities, NSError *error)
```

Discussion

This block takes the following parameters:

activities

An array of CMMotionActivity objects indicating the updates that occurred. The objects in the array are ordered by the time at which they occurred in the specified time interval. Use the [startDate](#) (page 33) property in each motion object to determine when the update occurred.

error

An error object indicating that there was a problem gathering the data or `nil` if the motion data was determined correctly.

Availability

Available in iOS 7.0 and later.

Declared in

CMMotionActivityManager.h

CMMotionManager Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreMotion.framework
Availability	Available in iOS 4.0 and later.
Declared in	CMMotionManager.h CMErrorDomain.h CMError.h
Companion guide	Event Handling Guide for iOS
Related sample code	MotionGraphs pARk

Overview

A `CMMotionManager` object is the gateway to the motion services provided by iOS. These services provide an app with accelerometer data, rotation-rate data, magnetometer data, and other device-motion data such as attitude. These types of data originate with a device's accelerometers and (on some models) its magnetometer and gyroscope.

Note: Methods, properties, and data types for processing magnetometer data were introduced in iOS 5.0.

After creating an instance of `CMMotionManager`, an app can use it to receive four types of motion: raw accelerometer data, raw gyroscope data, raw magnetometer data, and processed device-motion data (which includes accelerometer, rotation-rate, and attitude measurements). The processed device-motion data provided by Core Motion's sensor fusion algorithms gives the device's attitude, rotation rate, calibrated magnetic fields, the direction of gravity, and the acceleration the user is imparting to the device.

Important: An app should create only a single instance of the `CMMotionManager` class. Multiple instances of this class can affect the rate at which data is received from the accelerometer and gyroscope.

An app can take one of two approaches when receiving motion data, by handling it at specified update intervals or periodically sampling the motion data. With both of these approaches, the app should call the appropriate stop method (`stopAccelerometerUpdates` (page 63), `stopGyroUpdates` (page 63), `stopMagnetometerUpdates` (page 64), and `stopDeviceMotionUpdates` (page 63)) when it has finished processing accelerometer, rotation-rate, magnetometer, or device-motion data.

Handling Motion Updates at Specified Intervals

To receive motion data at specific intervals, the app calls a “start” method that takes an operation queue (instance of `NSOperationQueue`) and a block handler of a specific type for processing those updates. The motion data is passed into the block handler. The frequency of updates is determined by the value of an “interval” property.

- **Accelerometer.** Set the `accelerometerUpdateInterval` (page 49) property to specify an update interval. Call the `startAccelerometerUpdatesToQueue:withHandler:` (page 57) method, passing in a block of type `CMAccelerometerHandler` (page 64). Accelerometer data is passed into the block as `CMAccelerometerData` objects.
- **Gyroscope.** Set the `gyroUpdateInterval` (page 53) property to specify an update interval. Call the `startGyroUpdatesToQueue:withHandler:` (page 61) method, passing in a block of type `CMGyroHandler` (page 65). Rotation-rate data is passed into the block as `CMGyroData` objects.
- **Magnetometer.** Set the `magnetometerUpdateInterval` (page 55) property to specify an update interval. Call the `startMagnetometerUpdatesToQueue:withHandler:` (page 62) method, passing a block of type `CMMagnetometerHandler` (page 66). Magnetic-field data is passed into the block as `CMMagnetometerData` objects.
- **Device motion.** Set the `deviceMotionUpdateInterval` (page 51) property to specify an update interval. Call the `startDeviceMotionUpdatesUsingReferenceFrame:` (page 59) or `startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:` (page 60) or `startDeviceMotionUpdatesToQueue:withHandler:` (page 58) method, passing in a block of type `CMDeviceMotionHandler` (page 66). With the former method (new in iOS 5.0), you can specify a reference frame to be used for the attitude estimates. Rotation-rate data is passed into the block as `CMDeviceMotion` objects.

Periodic Sampling of Motion Data

To handle motion data by periodic sampling, the app calls a “start” method taking no arguments and periodically accesses the motion data held by a property for a given type of motion data. This approach is the recommended approach for apps such as games. Handling accelerometer data in a block introduces additional overhead, and most game apps are interested only the latest sample of motion data when they render a frame.

- **Accelerometer.** Call [startAccelerometerUpdates](#) (page 57) to begin updates and periodically access `CMAccelerometerData` objects by reading the [accelerometerData](#) (page 48) property.
- **Gyroscope.** Call [startGyroUpdates](#) (page 60) to begin updates and periodically access `CMGyroData` objects by reading the [gyroData](#) (page 53) property.
- **Magnetometer.** Call [startMagnetometerUpdates](#) (page 61) to begin updates and periodically access `CMMagnetometerData` objects by reading the [magnetometerData](#) (page 55) property.
- **Device motion.** Call the [startDeviceMotionUpdatesUsingReferenceFrame:](#) (page 59) or [startDeviceMotionUpdates](#) (page 58) method to begin updates and periodically access `CMDeviceMotion` objects by reading the [deviceMotion](#) (page 50) property. The `startDeviceMotionUpdatesUsingReferenceFrame:` method (new in iOS 5.0) lets you specify a reference frame to be used for the attitude estimates.

Hardware Availability and State

If a hardware feature (for example, a gyroscope) is not available on a device, calling a start method related to that feature has no effect. You can find out whether a hardware feature is available or active by checking the appropriate property; for example, for gyroscope data, you can check the value of the [gyroAvailable](#) (page 52) or [gyroActive](#) (page 52) properties.

Tasks

Managing Accelerometer Updates

[accelerometerUpdateInterval](#) (page 49) *property*

The interval, in seconds, for providing accelerometer updates to the block handler.

– [startAccelerometerUpdatesToQueue:withHandler:](#) (page 57)

Starts accelerometer updates on an operation queue and with a specified handler.

– [startAccelerometerUpdates](#) (page 57)

Starts accelerometer updates without a handler.

- [stopAccelerometerUpdates](#) (page 63)
Stops accelerometer updates.

Determining Whether the Accelerometer Is Active and Available

[accelerometerActive](#) (page 48) *property*

A Boolean value that indicates whether accelerometer updates are currently happening. (read-only)

[accelerometerAvailable](#) (page 48) *property*

A Boolean value that indicates whether an accelerometer is available on the device. (read-only)

Accessing Accelerometer Data

[accelerometerData](#) (page 48) *property*

The latest sample of accelerometer data. (read-only)

Managing Gyroscope Updates

[gyroUpdateInterval](#) (page 53) *property*

The interval, in seconds, for providing gyroscope updates to the block handler.

- [startGyroUpdatesToQueue:withHandler:](#) (page 61)
Starts gyroscope updates on an operation queue and with a specified handler.
- [startGyroUpdates](#) (page 60)
Starts gyroscope updates without a handler.
- [stopGyroUpdates](#) (page 63)
Stops gyroscope updates.

Determining Whether the Gyroscope Is Active and Available

[gyroActive](#) (page 52) *property*

A Boolean value that determines whether gyroscope updates are currently happening. (read-only)

[gyroAvailable](#) (page 52) *property*

A Boolean value that indicates whether a gyroscope is available on the device. (read-only)

Accessing Gyroscope Data

[gyroData](#) (page 53) *property*

The latest sample of gyroscope data. (read-only)

Managing Magnetometer Updates

[magnetometerUpdateInterval](#) (page 55) *property*

The interval, in seconds, at which the system delivers magnetometer data to the block handler.

– [startMagnetometerUpdatesToQueue:withHandler:](#) (page 62)

Starts magnetometer updates on an operation queue and with a specified handler.

– [startMagnetometerUpdates](#) (page 61)

Starts magnetometer updates without a block handler.

– [stopMagnetometerUpdates](#) (page 64)

Stops magnetometer updates.

Determining Whether the Magnetometer Is Active and Available

[magnetometerActive](#) (page 54) *property*

A Boolean value that determines whether magnetometer updates are currently happening. (read-only)

[magnetometerAvailable](#) (page 54) *property*

A Boolean value that indicates whether a magnetometer is available on the device. (read-only)

Accessing Magnetometer Data

[magnetometerData](#) (page 55) *property*

The latest sample of magnetometer data. (read-only)

Managing the Device Movement Display

[showsDeviceMovementDisplay](#) (page 55) *property*

Controls whether the device-movement display is shown.

Managing Device Motion Updates

[deviceMotionUpdateInterval](#) (page 51) *property*

The interval, in seconds, for providing device-motion updates to the block handler.

- [startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:](#) (page 60)
Starts device-motion updates on an operation queue and using a specified reference frame and block handler.
- [startDeviceMotionUpdatesToQueue:withHandler:](#) (page 58)
Starts device-motion updates on an operation queue and using a specified block handler.
- [startDeviceMotionUpdatesUsingReferenceFrame:](#) (page 59)
Starts device-motion updates using a reference frame but without a block handler.
- [startDeviceMotionUpdates](#) (page 58)
Starts device-motion updates without a block handler.
- [stopDeviceMotionUpdates](#) (page 63)
Stops device-motion updates.

Accessing Attitude Reference Frames

[attitudeReferenceFrame](#) (page 49) *property*

Returns either the reference frame currently being used or the default attitude reference frame (read-only)

+ [availableAttitudeReferenceFrames](#) (page 56)

Returns a bitmask specifying the available attitude reference frames on the device.

Determining Whether the Device Motion Hardware Is Active and Available

[deviceMotionActive](#) (page 50) *property*

A Boolean value that determines whether the app is receiving updates from the device-motion service. (read-only)

[deviceMotionAvailable](#) (page 51) *property*

A Boolean value that indicates whether the device-motion service is available on the device. (read-only)

Accessing Device Motion Data

[deviceMotion](#) (page 50) *property*

The latest sample of device-motion data. (read-only)

Properties

accelerometerActive

A Boolean value that indicates whether accelerometer updates are currently happening. (read-only)

```
@property(readonly, nonatomic, getter=isAccelerometerActive) BOOL accelerometerActive
```

Discussion

This property indicates whether [startAccelerometerUpdatesToQueue:withHandler:](#) (page 57) or [startAccelerometerUpdates](#) (page 57) has been called since the last time [stopAccelerometerUpdates](#) (page 63) was called. (If the start methods hadn't been called, the app could be getting updates from the accelerometer after calling, for example, [startDeviceMotionUpdates](#) (page 58), but this property would return NO.)

Availability

Available in iOS 4.0 and later.

See Also

[@property accelerometerAvailable](#) (page 48)

Declared in

CMMotionManager.h

accelerometerAvailable

A Boolean value that indicates whether an accelerometer is available on the device. (read-only)

```
@property(readonly, nonatomic, getter=isAccelerometerAvailable) BOOL accelerometerAvailable
```

Availability

Available in iOS 4.0 and later.

See Also

[@property accelerometerActive](#) (page 48)

Declared in

CMMotionManager.h

accelerometerData

The latest sample of accelerometer data. (read-only)


```
@property(readonly) CMAccelerometerData *accelerometerData
```

Discussion

If no accelerometer data is available, the value of this property is `nil`. An app that is receiving accelerometer data after calling [startAccelerometerUpdates](#) (page 57) periodically checks the value of this property and processes the acceleration data.

Availability

Available in iOS 4.0 and later.

Declared in

`CMMotionManager.h`

accelerometerUpdateInterval

The interval, in seconds, for providing accelerometer updates to the block handler.

```
@property(assign, nonatomic) NSTimeInterval accelerometerUpdateInterval
```

Discussion

The system supplies accelerometer updates to the block handler specified in [startAccelerometerUpdatesToQueue:withHandler:](#) (page 57) at regular intervals determined by the value of this property. The interval units are in seconds. The value of this property is capped to minimum and maximum values; the maximum value is determined by the maximum frequency supported by the hardware. If your app is sensitive to the intervals of acceleration data, it should always check the timestamps of the delivered `CMAccelerometerData` instances to determine the true update interval.

Availability

Available in iOS 4.0 and later.

Declared in

`CMMotionManager.h`

attitudeReferenceFrame

Returns either the reference frame currently being used or the default attitude reference frame (read-only)

@property(readonly, nonatomic) CMAccelerationReferenceFrame attitudeReferenceFrame

Discussion

If device motion is active, this property returns the reference frame currently in use. If device motion is not active but has been active since the app was last launched, this property returns the last frame used. If device motion has not been active since the app was last launched, this property returns the default attitude reference frame for the device. If device motion is not available on the device, the value is undefined.

Availability

Available in iOS 5.0 and later.

See Also

[@property deviceMotionAvailable](#) (page 51)
+ [availableAttitudeReferenceFrames](#) (page 56)

Declared in

CMMotionManager.h

deviceMotion

The latest sample of device-motion data. (read-only)

@property(readonly) CMDeviceMotion *deviceMotion

Discussion

If no device-motion data is available, the value of this property is `nil`. An app that is receiving device-motion data after calling [startDeviceMotionUpdates](#) (page 58) periodically checks the value of this property and processes the device-motion data.

Availability

Available in iOS 4.0 and later.

Declared in

CMMotionManager.h

deviceMotionActive

A Boolean value that determines whether the app is receiving updates from the device-motion service. (read-only)

@property(readonly, nonatomic, getter=isDeviceMotionActive) BOOL deviceMotionActive

Discussion

This property indicates whether [startDeviceMotionUpdatesToQueue:withHandler:](#) (page 58) or [startDeviceMotionUpdates](#) (page 58) has been called since the last time [stopDeviceMotionUpdates](#) (page 63) was called.

Availability

Available in iOS 4.0 and later.

See Also

[@property deviceMotionAvailable](#) (page 51)

Declared in

CMMotionManager.h

deviceMotionAvailable

A Boolean value that indicates whether the device-motion service is available on the device. (read-only)

@property(readonly, nonatomic, getter=isDeviceMotionAvailable) BOOL deviceMotionAvailable

Discussion

The device-motion service is available if a device has both an accelerometer and a gyroscope. Because all devices have accelerometers, this property is functionally equivalent to [gyroAvailable](#) (page 52).

Availability

Available in iOS 4.0 and later.

See Also

[@property deviceMotionActive](#) (page 50)

Declared in

CMMotionManager.h

deviceMotionUpdateInterval

The interval, in seconds, for providing device-motion updates to the block handler.

```
@property(assign, nonatomic) NSTimeInterval deviceMotionUpdateInterval
```

Discussion

The system supplies device-motion updates to the block handler specified in [startDeviceMotionUpdatesToQueue:withHandler:](#) (page 58) at regular intervals determined by the value of this property. The interval units are in seconds. The value of this property is capped to minimum and maximum values; the maximum value is determined by the maximum frequency supported by the hardware. If your app is sensitive to the intervals of device-motion data, it should always check the timestamps of the delivered `CMDeviceMotion` instances to determine the true update interval.

Availability

Available in iOS 4.0 and later.

Declared in

`CMMotionManager.h`

gyroActive

A Boolean value that determines whether gyroscope updates are currently happening. (read-only)

```
@property(readonly, nonatomic, getter=isGyroActive) BOOL gyroActive
```

Discussion

This property indicates whether [startGyroUpdatesToQueue:withHandler:](#) (page 61) or [startGyroUpdates](#) (page 60) has been called since the last time [stopGyroUpdates](#) (page 63) was called. (If the start methods hadn't been called, the app could be getting updates from the gyroscope after calling, for example, [startDeviceMotionUpdates](#) (page 58), but this property would return NO.)

Availability

Available in iOS 4.0 and later.

See Also

[@property gyroAvailable](#) (page 52)

Declared in

`CMMotionManager.h`

gyroAvailable

A Boolean value that indicates whether a gyroscope is available on the device. (read-only)

`@property(readonly, nonatomic, getter=isGyroAvailable) BOOL gyroAvailable`

Availability

Available in iOS 4.0 and later.

See Also

[@property gyroActive](#) (page 52)

Declared in

`CMMotionManager.h`

gyroData

The latest sample of gyroscope data. (read-only)

`@property(readonly) CMGyroData *gyroData`

Discussion

If no gyroscope data is available, the value of this property is `nil`. An app that is receiving gyroscope data after calling [startGyroUpdates](#) (page 60) periodically checks the value of this property and processes the gyroscope data.

Availability

Available in iOS 4.0 and later.

Declared in

`CMMotionManager.h`

gyroUpdateInterval

The interval, in seconds, for providing gyroscope updates to the block handler.

`@property(assign, nonatomic) NSTimeInterval gyroUpdateInterval`

Discussion

The system supplies gyroscope (that is, rotation rate) updates to the block handler specified in [startGyroUpdatesToQueue:withHandler:](#) (page 61) at regular intervals determined by the value of this property. The interval units are in seconds. The value of this property is capped to minimum and maximum values; the maximum value is determined by the maximum frequency supported by the hardware. If your app is sensitive to the intervals of gyroscope data, it should always check the timestamps of the delivered `CMGyroData` instances to determine the true update interval.

Availability

Available in iOS 4.0 and later.

Declared in

CMMotionManager.h

magnetometerActive

A Boolean value that determines whether magnetometer updates are currently happening. (read-only)

```
@property(readonly, nonatomic, getter=isMagnetometerActive) BOOL magnetometerActive
```

Discussion

This property indicates whether the [startMagnetometerUpdatesToQueue:withHandler:](#) (page 62) or [startMagnetometerUpdates](#) (page 61) method has been called since the last time the [stopMagnetometerUpdates](#) (page 64) method was called. (If the start methods hadn't been called, the app could be getting updates from the magnetometer after calling, for example, [startDeviceMotionUpdates](#) (page 58), but this property would return NO.)

Availability

Available in iOS 5.0 and later.

See Also

[@property magnetometerAvailable](#) (page 54)

Declared in

CMMotionManager.h

magnetometerAvailable

A Boolean value that indicates whether a magnetometer is available on the device. (read-only)

```
@property(readonly, nonatomic, getter=isMagnetometerAvailable) BOOL magnetometerAvailable
```

Availability

Available in iOS 5.0 and later.

See Also

[@property magnetometerActive](#) (page 54)

Declared in

CMMotionManager.h

magnetometerData

The latest sample of magnetometer data. (read-only)

```
@property(readonly) CMMagnetometerData *magnetometerData
```

Discussion

If no magnetometer data is available, the value of this property is `nil`. An app that is receiving magnetometer data after calling [startMagnetometerUpdates](#) (page 61) periodically checks the value of this property and processes the gyroscope data.

Availability

Available in iOS 5.0 and later.

See Also

[@property magnetometerUpdateInterval](#) (page 55)

Declared in

`CMMotionManager.h`

magnetometerUpdateInterval

The interval, in seconds, at which the system delivers magnetometer data to the block handler.

```
@property(assign, nonatomic) NSTimeInterval magnetometerUpdateInterval
```

Discussion

The supplies magnetometer data to the block handler specified in [startMagnetometerUpdatesToQueue:withHandler:](#) (page 62) at regular intervals determined by the value of this property. The interval unit are in seconds. The value of this property is capped to minimum and maximum values; the maximum value is determined by the maximum frequency supported by the hardware. If your app is sensitive to the intervals of magnetometer data, it should always check the timestamps of the delivered `CMMagnetometerData` instances to determine the true update interval.

Availability

Available in iOS 5.0 and later.

Declared in

`CMMotionManager.h`

showsDeviceMovementDisplay

Controls whether the device-movement display is shown.

`@property(assign, nonatomic) BOOL showsDeviceMovementDisplay`

Discussion

When a device requires movement (for example, to calibrate the compass), the value of this property indicates if the system's device-movement display should be shown. When a device requires movement, the block handler of type [CMDeviceMotionHandler](#) (page 66) reports the [CMErrorDeviceRequiresMovement](#) (page 68) error once. By default, this property is NO.

Availability

Available in iOS 5.0 and later.

Declared in

`CMMotionManager.h`

Class Methods

[availableAttitudeReferenceFrames](#)

Returns a bitmask specifying the available attitude reference frames on the device.

+ (NSUInteger)availableAttitudeReferenceFrames

Return Value

A bitmask that you can bitwise-AND with the enum constants of the [CMAttitudeReferenceFrame](#) (page 15) type.

Discussion

For example, to determine whether [CMAttitudeReferenceFrameXMagneticNorthZVertical](#) (page 16) is available on the device, you would perform the following test:

```
if ([CMMotionManager availableAttitudeReferenceFrames] &
    CMAttitudeReferenceFrameXMagneticNorthZVertical) {
    // do something appropriate here
}
```

Availability

Available in iOS 5.0 and later.

See Also

[@property attitudeReferenceFrame](#) (page 49)

Declared in
CMMotionManager.h

Instance Methods

startAccelerometerUpdates

Starts accelerometer updates without a handler.

– (void)startAccelerometerUpdates

Discussion

You can get the latest accelerometer data through the [accelerometerData](#) (page 48) property. You must call [stopAccelerometerUpdates](#) (page 63) when you no longer want your app to process accelerometer updates.

Availability

Available in iOS 4.0 and later.

See Also

– [startAccelerometerUpdatesToQueue:withHandler:](#) (page 57)

Declared in
CMMotionManager.h

startAccelerometerUpdatesToQueue:withHandler:

Starts accelerometer updates on an operation queue and with a specified handler.

– (void)startAccelerometerUpdatesToQueue:(NSOperationQueue *)queue
withHandler:(CMAccelerometerHandler)handler

Parameters

queue

An operation queue provided by the caller. Because the processed events might arrive at a high rate, using the main operation queue is not recommended.

handler

A block that is invoked with each update to handle new accelerometer data. The block must conform to the [CMAccelerometerHandler](#) (page 64) type.

Discussion

You must call [stopAccelerometerUpdates](#) (page 63) when you no longer want your app to process accelerometer updates.

Availability

Available in iOS 4.0 and later.

See Also

– [startAccelerometerUpdates](#) (page 57)

Related Sample Code

MotionGraphs

Declared in

CMMotionManager.h

startDeviceMotionUpdates

Starts device-motion updates without a block handler.

– (void)startDeviceMotionUpdates

Discussion

You can get the latest device-motion data through the [deviceMotion](#) (page 50) property. You must call [stopDeviceMotionUpdates](#) (page 63) when you no longer want your app to process device-motion updates. This method uses the reference frame returned by [attitudeReferenceFrame](#) (page 49) for device-motion updates.

Availability

Available in iOS 4.0 and later.

See Also

– [startDeviceMotionUpdatesToQueue:withHandler:](#) (page 58)

Declared in

CMMotionManager.h

startDeviceMotionUpdatesToQueue:withHandler:

Starts device-motion updates on an operation queue and using a specified block handler.

– (void)startDeviceMotionUpdatesToQueue:(NSOperationQueue *)queue
withHandler:(CMDeviceMotionHandler)handler

Parameters

queue

An operation queue provided by the caller. Because the processed events might arrive at a high rate, using the main operation queue is not recommended.

handler

A block that is invoked with each update to handle new device-motion data. The block must conform to the [CMDeviceMotionHandler](#) (page 66) type.

Discussion

This method uses the reference frame returned by [attitudeReferenceFrame](#) (page 49) for device-motion updates. You must call [stopDeviceMotionUpdates](#) (page 63) when you no longer want your app to process device-motion updates.

Availability

Available in iOS 4.0 and later.

See Also

– [startDeviceMotionUpdates](#) (page 58)

Declared in

`CMMotionManager.h`

startDeviceMotionUpdatesUsingReferenceFrame:

Starts device-motion updates using a reference frame but without a block handler.

–

```
(void)startDeviceMotionUpdatesUsingReferenceFrame:(CMAccelerationReferenceFrame)referenceFrame
```

Parameters

referenceFrame

A constant identifying the reference frame to use for device-motion updates.

Discussion

You can get the latest device-motion data through the [deviceMotion](#) (page 50) property. You must call [stopDeviceMotionUpdates](#) (page 63) when you no longer want your app to process device-motion updates.

Availability

Available in iOS 5.0 and later.

Declared in

`CMMotionManager.h`

startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:

Starts device-motion updates on an operation queue and using a specified reference frame and block handler.

–
(void)startDeviceMotionUpdatesUsingReferenceFrame:(CMAAttitudeReferenceFrame)referenceFrame
toQueue:(NSOperationQueue *)queue withHandler:(CMDeviceMotionHandler)handler

Parameters

referenceFrame

A constant identifying the reference frame to use for device-motion updates.

queue

An operation queue provided by the caller. Because the processed events might arrive at a high rate, using the main operation queue is not recommended.

handler

A block that is invoked with each update to handle new device-motion data. The block must conform to the [CMDeviceMotionHandler](#) (page 66) type.

Discussion

You must call [stopDeviceMotionUpdates](#) (page 63) when you no longer want your app to process device-motion updates.

Availability

Available in iOS 5.0 and later.

Declared in

CMMotionManager.h

startGyroUpdates

Starts gyroscope updates without a handler.

– (void)startGyroUpdates

Discussion

You can get the latest gyroscope data through the [gyroData](#) (page 53) property. You must call [stopGyroUpdates](#) (page 63) when you no longer want your app to process gyroscope updates.

Availability

Available in iOS 4.0 and later.

See Also

– [startGyroUpdatesToQueue:withHandler:](#) (page 61)

Declared in

CMMotionManager.h

startGyroUpdatesToQueue:withHandler:

Starts gyroscope updates on an operation queue and with a specified handler.

– (void)startGyroUpdatesToQueue:(NSOperationQueue *)queue
withHandler:(CMGyroHandler)handler

Parameters

queue

An operation queue provided by the caller. Because the processed events might arrive at a high rate, using the main operation queue is not recommended.

handler

A block that is invoked with each update to handle new gyroscope data. The block must conform to the [CMGyroHandler](#) (page 65) type.

Discussion

You must call [stopGyroUpdates](#) (page 63) when you no longer want your app to process gyroscope updates.

Availability

Available in iOS 4.0 and later.

See Also

– [startGyroUpdates](#) (page 60)

Related Sample Code
MotionGraphs

Declared in

CMMotionManager.h

startMagnetometerUpdates

Starts magnetometer updates without a block handler.

– (void)startMagnetometerUpdates

Discussion

You can get the latest magnetometer data through the [magnetometerData](#) (page 55) property. You must call [stopMagnetometerUpdates](#) (page 64) when you no longer want your app to process magnetometer updates.

Availability

Available in iOS 5.0 and later.

See Also

– [startMagnetometerUpdatesToQueue:withHandler:](#) (page 62)

Declared in

`CMMotionManager.h`

[startMagnetometerUpdatesToQueue:withHandler:](#)

Starts magnetometer updates on an operation queue and with a specified handler.

```
– (void)startMagnetometerUpdatesToQueue:(NSOperationQueue *)queue  
withHandler:(CMMagnetometerHandler)handler
```

Parameters

`queue`

An operation queue provided by the caller. Because the processed events might arrive at a high rate, using the main operation queue is not recommended.

`handler`

A block that is invoked with each update to handle new magnetometer data. The block must conform to the [CMMagnetometerHandler](#) (page 66) type.

Discussion

You must call [stopMagnetometerUpdates](#) (page 64) when you no longer want your app to process magnetometer updates.

Availability

Available in iOS 5.0 and later.

See Also

– [startMagnetometerUpdates](#) (page 61)

Declared in

`CMMotionManager.h`

stopAccelerometerUpdates

Stops accelerometer updates.

– (void)stopAccelerometerUpdates

Availability

Available in iOS 4.0 and later.

See Also

- [startAccelerometerUpdatesToQueue:withHandler:](#) (page 57)
- [startAccelerometerUpdates](#) (page 57)

Related Sample Code
MotionGraphs

Declared in

CMMotionManager.h

stopDeviceMotionUpdates

Stops device-motion updates.

– (void)stopDeviceMotionUpdates

Availability

Available in iOS 4.0 and later.

See Also

- [startDeviceMotionUpdatesToQueue:withHandler:](#) (page 58)
- [startDeviceMotionUpdates](#) (page 58)

Related Sample Code
MotionGraphs

Declared in

CMMotionManager.h

stopGyroUpdates

Stops gyroscope updates.

– (void)stopGyroUpdates

Availability

Available in iOS 4.0 and later.

See Also

- [startGyroUpdatesToQueue:withHandler:](#) (page 61)
- [startGyroUpdates](#) (page 60)

Related Sample Code
[MotionGraphs](#)

Declared in

CMMotionManager.h

stopMagnetometerUpdates

Stops magnetometer updates.

- (void)stopMagnetometerUpdates

Availability

Available in iOS 5.0 and later.

See Also

- [startMagnetometerUpdatesToQueue:withHandler:](#) (page 62)
- [startMagnetometerUpdates](#) (page 61)

Declared in

CMMotionManager.h

Constants

CMAccelerometerHandler

The type of block callback for handling accelerometer data.

```
typedef void (^CMAccelerometerHandler)(CMAccelerometerData *accelerometerData, NSError *error);
```


Discussion

Blocks of type `CMAccelerometerHandler` are called when there is accelerometer data to process. You pass the block into [startAccelerometerUpdatesToQueue:withHandler:](#) (page 57) as the second argument. Blocks of this type return no value but take two arguments:

`accelerometerData`

An object that encapsulates a [CMAcceleration](#) (page 7) structure with fields holding acceleration values for the three axes of movement.

`error`

An error object representing an error encountered in providing accelerometer updates. If an error occurs, you should stop accelerometer updates and inform the user of the problem. If there is no error, this argument is `nil`. Core Motion errors are of the [CMErrorDomain](#) (page 67) domain and the [CMError](#) (page 67) type.

Availability

Available in iOS 4.0 and later.

Declared in

`CMMotionManager.h`

CMGyroHandler

The type of block callback for handling gyroscope data.

```
typedef void (^CMGyroHandler)(CMGyroData *gyroData, NSError *error);
```

Discussion

Blocks of type `CMGyroHandler` are called when there is gyroscope data to process. You pass the block into [startGyroUpdatesToQueue:withHandler:](#) (page 61) as the second argument. Blocks of this type return no value but take two arguments:

`gyroData`

An object that encapsulates a [CMRotationRate](#) (page 24) structure with fields holding rotation-rate values for the three axes of movement.

`error`

An error object representing an error encountered in providing gyroscope data. If an error occurs, you should stop gyroscope updates and inform the user of the problem. If there is no error, this argument is `nil`. Core Motion errors are of the [CMErrorDomain](#) (page 67) domain and the [CMError](#) (page 67) type.

Availability

Available in iOS 4.0 and later.

Declared in

CMMotionManager.h

CMMagnetometerHandler

The type of block callback for handling magnetometer data.

```
typedef void (^CMMagnetometerHandler)(CMMagnetometerData *magnetometerData, NSError *error);
```

Discussion

Blocks of type `CMMagnetometerHandler` are called when there is magnetometer data to process. You pass the block into the [startMagnetometerUpdatesToQueue:withHandler:](#) (page 62) method as the second argument. Blocks of this type return no value but take two arguments:

`magnetometerData`

An object that encapsulates a [CMagneticField](#) (page 29) structure with fields holding magnetic-field values for the three axes of movement.

`error`

An error object representing an error encountered in providing magnetometer data. If an error occurs, you should stop magnetometer updates and inform the user of the problem. If there is no error, this argument is `nil`. Core Motion errors are of the [CMErrorDomain](#) (page 67) domain and the [CMError](#) (page 67) type.

Availability

Available in iOS 5.0 and later.

Declared in

CMMotionManager.h

CMDeviceMotionHandler

The type of block callback for handling device-motion data.

```
typedef void (^CMDeviceMotionHandler)(CMDeviceMotion *motion, NSError *error);
```

Discussion

Blocks of type `CMDeviceMotionHandler` are called when there is device-motion data to process. You pass the block into [startDeviceMotionUpdatesToQueue:withHandler:](#) (page 58) as the second argument. Blocks of this type return no value but take two arguments:

`motion`

A `CMDeviceMotion` object, which encapsulates other objects and a structure representing attitude, rotation rate, gravity, and user acceleration.

`error`

An error object representing an error encountered in providing gyroscope data. If an error occurs, you should stop gyroscope updates and inform the user of the problem. If there is no error, this argument is `nil`. Core Motion errors are of the [CMErrorDomain](#) (page 67) domain and the [CMError](#) (page 67) type.

Availability

Available in iOS 4.0 and later.

Declared in

`CMMotionManager.h`

Core Motion Error Domain

The error domain for Core Motion.

```
extern NSString *const CMErrorDomain;
```

Constants

`CMErrorDomain`

Identifies the domain of `NSError` objects returned from Core Motion.

Available in iOS 4.0 and later.

Declared in `CMErrorDomain.h`.

Declared in

`CMErrorDomain.h`

CMError

The type for Core Motion errors.

```
typedef enum {  
    CMErrorNULL = 100,
```

```
CMErrorDeviceRequiresMovement,  
CMErrorTrueNorthNotAvailable,  
CMErrorUnknown,  
CMErrorMotionActivityNotAvailable,  
CMErrorMotionActivityNotAuthorized,  
CMErrorMotionActivityNotEntitled,  
CMErrorInvalidParameter  
} CMError;
```

Constants

CMErrorNULL

No error.

Available in iOS 4.0 and later.

Declared in `CMError.h`.

CMErrorDeviceRequiresMovement

The device must move for a sampling of motion data to occur.

Available in iOS 5.0 and later.

Declared in `CMError.h`.

CMErrorTrueNorthNotAvailable

True north is not available on this device. This usually indicates that the device's location is not yet available.

Available in iOS 5.0 and later.

Declared in `CMError.h`.

CMErrorUnknown

An unknown error occurred.

Available in iOS 7.0 and later.

Declared in `CMError.h`.

CMErrorMotionActivityNotAvailable

Motion activity support is not available on the current device.

Available in iOS 7.0 and later.

Declared in `CMError.h`.

CMErrorMotionActivityNotAuthorized

The app is not currently authorized to use motion activity support.

Available in iOS 7.0 and later.

Declared in `CMError.h`.

`CLErrorMotionActivityNotEntitled`

The app is missing a required entitlement.

Available in iOS 7.0 and later.

Declared in `CLError.h`.

`CLErrorInvalidParameter`

An invalid parameter was specified.

Available in iOS 7.0 and later.

Declared in `CLError.h`.

Declared in

`CLError.h`

CMStepCounter Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreMotion.framework
Availability	Available in iOS 7.0 and later.
Declared in	CMStepCounter.h

Overview

The `CMStepCounter` class provides access to the number of steps the user has taken with the device. Step information is gathered on devices with the appropriate built-in hardware and stored so that you can run queries to determine the user's recent physical activity. You use this class to gather both current step data and any historical data.

Tasks

Determining Step Counting Availability

+ `isStepCountingAvailable` (page 71)

Returns a Boolean indicating whether step-counting support is available on the current device.

Starting and Stopping Step Counting Updates

– `startStepCountingUpdatesToQueue:updateOn:withHandler:` (page 72)

Starts the delivery of current step-counting data to your app.

– `stopStepCountingUpdates` (page 73)

Stops the delivery of step-counting updates to your app.

Getting Historical Step Counting Data

- [queryStepCountStartingFrom:to:toQueue:withHandler:](#) (page 71)
Gathers and returns historical step count data for the specified time period.

Class Methods

isStepCountingAvailable

Returns a Boolean indicating whether step-counting support is available on the current device.

+ (BOOL)isStepCountingAvailable

Return Value

YES if step-counting support is available or NO if it is not.

Discussion

Step-counting support is not available on all iOS devices. Use this method to determine if support is available on the current device.

Availability

Available in iOS 7.0 and later.

Declared in

CMStepCounter.h

Instance Methods

queryStepCountStartingFrom:to:toQueue:withHandler:

Gathers and returns historical step count data for the specified time period.

– (void)queryStepCountStartingFrom:(NSDate *)start to:(NSDate *)end
toQueue:(NSOperationQueue *)queue withHandler:(CMStepQueryHandler)handler

Parameters

start

The start time to use when gathering step count data. This parameter must not be `nil`.

end

The end time to use when gathering step count data. This parameter must not be `nil`.

queue

The operation queue on which to execute the specified `handler` block. You can specify a custom queue or use the operation queue associated with your app's main thread. This parameter must not be `nil`.

handler

The block to execute with the results. For information about the parameters of this block, see [CMStepQueryHandler](#) (page 74). This parameter must not be `nil`.

Discussion

This method runs asynchronously, returning immediately and delivering the results to the specified `handler` block. The system stores only the last seven days worth of step data at most. If there are no samples for the specified range of time, a value of 0 is passed to the `handler` block.

Availability

Available in iOS 7.0 and later.

Declared in

`CMStepCounter.h`

startStepCountingUpdatesToQueue:updateOn:withHandler:

Starts the delivery of current step-counting data to your app.

```
– (void)startStepCountingUpdatesToQueue:(NSOperationQueue *)queue  
updateOn:(NSInteger)stepCounts withHandler:(CMStepUpdateHandler)handler
```

Parameters

queue

The operation queue on which to execute the specified `handler` block. You can specify a custom queue or use the operation queue associated with your app's main thread. This parameter must not be `nil`.

stepCounts

The number of steps to record before executing the `handler` block. The number of steps must be greater than 0.

handler

The block to execute when the number of steps has been reached or exceeded. For information about the parameters of this block, see [CMStepUpdateHandler](#) (page 74). This property must not be `nil`.

Discussion

This method initiates tracking of the user's steps and calls the provided block periodically to deliver the results. When you call this method, the step counter resets the current step count value to 0 and begins counting. Each time the step counter records the number of steps in the `stepCounts` parameter, it executes the specified

handler block. For example, if `stepCounts` is 100, it would send updates at 100 steps, 200 steps, 300 steps, and so on. The number of steps reported to your handler is always the total number of steps since you called this method.

The handler block is executed on a best effort basis each time the step count threshold is exceeded. If your app is suspended when the threshold is exceeded, the block is not executed. When your app resumes, the block is not executed until the threshold is exceeded again.

To stop the delivery of step-counting updates, call the `stopStepCountingUpdates` method. Updates also stop when the step counter object itself is deallocated.

Availability

Available in iOS 7.0 and later.

See Also

– [stopStepCountingUpdates](#) (page 73)

Declared in

`CMStepCounter.h`

stopStepCountingUpdates

Stops the delivery of step-counting updates to your app.

– (void)stopStepCountingUpdates

Discussion

Call this method to stop the delivery of updates that you started by calling the `startStepCountingUpdatesToQueue:updateOn:withHandler:` method. This method does not stop queries started using the [queryStepCountStartingFrom:to:toQueue:withHandler:](#) (page 71) method.

Availability

Available in iOS 7.0 and later.

See Also

– [startStepCountingUpdatesToQueue:updateOn:withHandler:](#) (page 72)

Declared in

`CMStepCounter.h`

Constants

CMStepQueryHandler

A block that reports the number of steps for a query operation.

```
typedef void (^CMStepQueryHandler)(NSInteger numberOfSteps, NSError *error);
```

Discussion

This block takes two parameters:

numberOfSteps

The number of steps that occurred between the start and end times specified by the query.

error

An error object indicating that there was a problem gathering the data or `nil` if the number of steps was determined correctly.

Availability

Available in iOS 7.0 and later.

Declared in

`CMStepCounter.h`

CMStepUpdateHandler

A block that reports the number of steps recorded since updates began.

```
typedef void (^CMStepUpdateHandler)(NSInteger numberOfSteps, NSDate *timestamp, NSError *error);
```

Discussion

This block takes the following parameters:

numberOfSteps

The total number of steps since the [startStepCountingUpdatesToQueue:updateOn:withHandler:](#) (page 72) method was called.

timestamp

The time at which the current step count was reported.

error

An error object indicating that there was a problem gathering the data or `nil` if the number of steps was determined correctly.

Availability

Available in iOS 7.0 and later.

Declared in

CMStepCounter.h

Document Revision History

This table describes the changes to *Core Motion Framework Reference*.

Date	Notes
2013-09-18	Added new classes for step counting and motion activity.
2011-10-12	Added CMMagnetometerData class, new in iOS 5.0.
2010-04-27	First version of the reference describing the API for handling accelerometer data and other kinds of motion events.



Apple Inc.
Copyright © 2013 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.

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.

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

Apple and the Apple logo are trademarks of Apple Inc., registered in the U.S. and other countries.

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

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.