Core Motion Framework Reference

Data Management: Event Handling

2010-04-27

Contents

Introduction **Core Motion Framework Reference 5**

Part I **Classes 7**

Chapter 1 **CMAccelerometerData Class Reference 9**

Overview 9
Tasks 9
Properties 9
Constants 10

Chapter 2 **CMAttitude Class Reference 11**

Overview 11
Tasks 11
Properties 12
Instance Methods 14
Constants 14

Chapter 3 **CMDeviceMotion Class Reference 17**

Overview 17
Tasks 17
Properties 18

Chapter 4 **CMGyroData Class Reference 21**

Overview 21
Tasks 21
Properties 21
Constants 22

Chapter 5 **CMLogItem Class Reference 23**

Overview 23
Tasks 23
Properties 23

Chapter 6 **CMMotionManager Class Reference 25**

Overview 25

Tasks 27
Properties 28
Instance Methods 33
Constants 37

Document Revision History 41

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 CMMotionManager.h

The Core Motion framework lets your application receive motion data from device hardware and process that data. This hardware includes an accelerometer and, on some device models, a gyroscope. Through the `CMMotionManager` class you can start receiving accelerometer, gyroscope, and combined device-motion events at a regular interval or you can poll for them periodically.

Classes

PART I

Classes

CMAccelerometerData Class Reference

Inherits from	CMLogItem : NSObject
Conforms to	NSCoding (CMLogItem) NSCopying (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 33) 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 10) *property*
The acceleration measured by the accelerometer. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

acceleration

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

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

Discussion
The description of the `CMAcceleration` (page 10) 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	NSCoding NSCopying 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 either the `startDeviceMotionUpdatesToQueue:withHandler:` (page 34) method or the `startDeviceMotionUpdates` (page 34) method of the `CMMotionManager` class.

Tasks

Getting a Mathematical Representation of Attitude as Euler Angles

`roll` (page 13) *property*
    The roll of the device, in radians. (read-only)

`pitch` (page 12) *property*
    The pitch of the device, in radians. (read-only)

`yaw` (page 13) *property*
    The yaw of the device, in radians. (read-only)

Getting a Mathematical Representation of Attitude as a Rotation Matrix

`rotationMatrix` (page 13) *property*
Returns a rotation matrix representing the device's attitude. (read-only)

Getting a Mathematical Representation of Attitude as a Quaternion

`quaternion` (page 12) *property*
Returns a quaternion representing the device's attitude. (read-only)

Obtaining the Change in Attitude

`– multiplyByInverseOfAttitude:` (page 14)
Yields the change in attitude given a specific attitude.

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

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 13)
`@property yaw` (page 13)

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 15) type in "Constants" for further information.

Availability
Available in iOS 4.0 and later.

See Also
  `@property rotationMatrix` (page 13)

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 12)
  `@property yaw` (page 13)

Declared In
`CMAttitude.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 12)

Declared In
`CMAttitude.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 13)
- `@property pitch` (page 12)

Declared In
`CMAttitude.h`

Instance Methods

multiplyByInverseOfAttitude:

Yields the change in attitude given a specific attitude.

```
- (void)multiplyByInverseOfAttitude:(CMAttitude *)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 `CMAttitude` 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 `CMAttitude` 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
`CMAttitude.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

`CMAttitude.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 theta 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`

CMDeviceMotion Class Reference

Inherits from	CMLogItem : NSObject
Conforms to	NSCoding (CMLogItem) NSCopying (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

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 `startDeviceMotionUpdatesToQueue:withHandler:` (page 34) method or the `startDeviceMotionUpdates` (page 34) 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 18) and `userAcceleration` (page 19) properties.

Tasks

Getting Attitude and Rotation Rate

`attitude` (page 18) *property*
The attitude of the device. (read-only)

`rotationRate` (page 19) *property*
The rotation rate of the device. (read-only)

Getting Acceleration Data

`gravity` (page 18) *property*
The gravity acceleration vector expressed in the device's reference frame. (read-only)

`userAcceleration` (page 19) *property*
The acceleration that the user is giving to the device. (read-only)

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

attitude

The attitude of the device. (read-only)

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

Discussion
A `CMAttitude` 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 19)

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 19)).

Availability
Available in iOS 4.0 and later.

Declared In
`CMDeviceMotion.h`

rotationRate

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

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

Discussion

A `CMRotationRate` (page 22) 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 18) plus the acceleration the user imparts to the device.

Availability

Available in iOS 4.0 and later.

Declared In

`CMDeviceMotion.h`

CMGyroData Class Reference

Inherits from	CMLogItem : NSObject
Conforms to	NSCoding (CMLogItem) NSCopying (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 35) method or the `startGyroUpdates` (page 35) method of the `CMMotionManager` class.

Tasks

Getting the Rotation Rate

`rotationRate` (page 22) *property*
The rotation rate as measured by the device's gyroscope. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

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 22) 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	NSCoding NSCopying 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 23) property that records the time a motion-event measurement was taken.

Tasks

Getting the Time of the Event

`timestamp` (page 23) *property*
The time when the logged item is valid. (read-only)

Properties

For more about Objective-C properties, see "Properties" in *The Objective-C Programming Language*.

timestamp

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

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

Availability
Available in iOS 4.0 and later.

Declared In
`CMLogItem.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

Overview

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

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

Important: An application 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 application 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 application should call the appropriate stop method (`stopAccelerometerUpdates` (page 36), `stopGyroUpdates` (page 36), and `stopDeviceMotionUpdates` (page 36)) when it has finished processing accelerometer, rotation-rate, or device-motion data.

Handing Motion Updates at Specified Intervals

To receive motion data at specific intervals, he application 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 30) property to specify an update interval. Call the `startAccelerometerUpdatesToQueue:withHandler:` (page 33) method, passing in a block of type `CMAccelerometerHandler` (page 37). Accelerometer data is passed into the block as `CMAccelerometerData` objects.
- **Gyroscope.** Set the `gyroUpdateInterval` (page 32) property to specify an update interval. Call the `startGyroUpdatesToQueue:withHandler:` (page 35) method, passing in a block of type`CMGyroHandler` (page 37). Rotation-rate data is passed into the block as `CMGyroData` objects.
- **Device motion.** Set the `deviceMotionUpdateInterval` (page 31) property to specify an update interval. Call the `startDeviceMotionUpdatesToQueue:withHandler:` (page 34) method, passing in a block of type `CMDeviceMotionHandler` (page 38). Rotation-rate data is passed into the block as `CMDeviceMotion` objects.

Periodic Sampling of Motion Data

To handle motion data by periodic sampling, the application 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 applications such as games. Handling accelerometer data in a block introduces additional latency, and most game applications are interested only the latest sample of accelerometer data when they render a frame.

- **Accelerometer.** Call `startAccelerometerUpdates` (page 33) to begin updates and periodically access `CMAccelerometerData` objects by reading the `accelerometerData` (page 29) property.
- **Gyroscope.** Call `startGyroUpdates` (page 35) to begin updates and periodically access `CMGyroData` objects by reading the `gyroData` (page 32) property.
- **Device motion.** Call `startDeviceMotionUpdates` (page 34) to begin updates and periodically access `CMDeviceMotion` objects by reading the `deviceMotion` (page 30) property.

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 32) or `gyroActive` (page 31) properties.

Tasks

Managing Accelerometer Updates

`accelerometerUpdateInterval` (page 30) *property*
The interval, in seconds, for providing accelerometer updates to the block handler.

– `startAccelerometerUpdatesToQueue:withHandler:` (page 33)
Starts accelerometer updates on an operation queue and with a specified handler.

– `startAccelerometerUpdates` (page 33)
Starts accelerometer updates without a handler.

– `stopAccelerometerUpdates` (page 36)
Stops accelerometer updates.

Determining Whether the Accelerometer Is Active and Available

`accelerometerActive` (page 28) *property*
A Boolean value that indicates whether accelerometer updates are currently happening. (read-only)

`accelerometerAvailable` (page 29) *property*
A Boolean value that indicates whether an accelerometer is available on the device. (read-only)

Accessing Accelerometer Data

`accelerometerData` (page 29) *property*
The latest sample of accelerometer data. (read-only)

Managing Gyroscope Updates

`gyroUpdateInterval` (page 32) *property*
The interval, in seconds, for providing gyroscope updates to the block handler.

– `startGyroUpdatesToQueue:withHandler:` (page 35)
Starts gyroscope updates on an operation queue and with a specified handler.

– `startGyroUpdates` (page 35)
Starts gyroscope updates without a handler.

– `stopGyroUpdates` (page 36)
Stops gyroscope updates.

Determining Whether the Gyroscope Is Active and Available

`gyroActive` (page 31) *property*
A Boolean value that determines whether whether gyroscope updates are currently happening.. (read-only)

`gyroAvailable` (page 32) *property*
    A Boolean value that indicates whether a gyroscope is available on the device. (read-only)

Accessing Gyroscope Data

`gyroData` (page 32) *property*
    The latest sample of gyroscope data. (read-only)

Managing Device Motion Updates

`deviceMotionUpdateInterval` (page 31) *property*
    The interval, in seconds, for providing device-motion updates to the block handler.

– `startDeviceMotionUpdatesToQueue:withHandler:` (page 34)
    Starts device-motion updates on an operation queue and using a specified block handler.

– `startDeviceMotionUpdates` (page 34)
    Starts device-motion updates without a block handler.

– `stopDeviceMotionUpdates` (page 36)
    Stops device-motion updates.

Determining Whether the Device Motion Hardware Is Active and Available

`deviceMotionActive` (page 30) *property*
    A Boolean value that determines whether the application is receiving updates from the device-motion service. (read-only)

`deviceMotionAvailable` (page 31) *property*
    A Boolean value that indicates whether the device-motion service is available on the device. (read-only)

Accessing Device Motion Data

`deviceMotion` (page 30) *property*
    The latest sample of device-motion data. (read-only)

Properties

For more about Objective-C properties, see “Properties” in *The Objective-C Programming Language*.

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 33) or `startAccelerometerUpdates` (page 33) has been called since the last time `stopAccelerometerUpdates` (page 36) was called. (If the start methods hadn't been called, the application could be getting updates from the accelerometer after calling, for example, `startDeviceMotionUpdates` (page 34), but this property would return `NO`.)

Availability
Available in iOS 4.0 and later.

See Also
`@property accelerometerAvailable` (page 29)

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 28)

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 application that is receiving accelerometer data after calling `startAccelerometerUpdates` (page 33) 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 33) 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 you application 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`

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 application that is receiving device-motion data after calling `startDeviceMotionUpdates` (page 34) 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 application 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 34) or `startDeviceMotionUpdates` (page 34) has been called since the last time `stopDeviceMotionUpdates` (page 36) was called.

Availability
Available in iOS 4.0 and later.

See Also
`@property deviceMotionAvailable` (page 31)

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 32).

Availability
Available in iOS 4.0 and later.

See Also
`@property deviceMotionActive` (page 30)

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 34) 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 you application 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 whether gyroscope updates are currently happening.. (read-only)

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

Discussion
This property indicates whether `startGyroUpdatesToQueue:withHandler:` (page 35) or `startGyroUpdates` (page 35) has been called since the last time `stopGyroUpdates` (page 36) was called. (If the start methods hadn't been called, the application could be getting updates from the gyroscope after calling, for example, `startDeviceMotionUpdates` (page 34), but this property would return `NO`.)

Availability
Available in iOS 4.0 and later.

See Also
`@property gyroAvailable` (page 32)

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 31)

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 application that is receiving gyroscope data after calling `startGyroUpdates` (page 35) 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 35) 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 you application 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`

Instance Methods

startAccelerometerUpdates

Starts accelerometer updates without a handler.

```
- (void)startAccelerometerUpdates
```

Discussion

You can get the latest accelerometer data through the `accelerometerData` (page 29) property. You must call `stopAccelerometerUpdates` (page 36) when you no longer want your application to process accelerometer updates.

Availability

Available in iOS 4.0 and later.

See Also

`– startAccelerometerUpdatesToQueue:withHandler:` (page 33)

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 37) type.

Discussion

You must call `stopAccelerometerUpdates` (page 36) when you no longer want your application to process accelerometer updates.

Availability

Available in iOS 4.0 and later.

See Also

`– startAccelerometerUpdates` (page 33)

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 30) property. You must call `stopDeviceMotionUpdates` (page 36) when you no longer want your application to process device-motion updates.

Availability

Available in iOS 4.0 and later.

See Also

`– startDeviceMotionUpdatesToQueue:withHandler:` (page 34)

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 38) type.

Discussion
You must call `stopDeviceMotionUpdates` (page 36) when you no longer want your application to process device-motion updates.

Availability
Available in iOS 4.0 and later.

See Also
`- startDeviceMotionUpdates` (page 34)

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 32) property. You must call `stopGyroUpdates` (page 36) when you no longer want your application to process gyroscope updates.

Availability
Available in iOS 4.0 and later.

See Also
`- startGyroUpdatesToQueue:withHandler:` (page 35)

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 37) type.

Discussion
You must call `stopGyroUpdates` (page 36) when you no longer want your application to process gyroscope updates.

Availability
Available in iOS 4.0 and later.

See Also
- `startGyroUpdates` (page 35)

Declared In
`CMMotionManager.h`

stopAccelerometerUpdates

Stops accelerometer updates.

```
- (void)stopAccelerometerUpdates
```

Availability
Available in iOS 4.0 and later.

See Also
- `startAccelerometerUpdatesToQueue:withHandler:` (page 33)
- `startAccelerometerUpdates` (page 33)

Declared In
`CMMotionManager.h`

stopDeviceMotionUpdates

Stops device-motion updates.

```
- (void)stopDeviceMotionUpdates
```

Availability
Available in iOS 4.0 and later.

See Also
- `startDeviceMotionUpdatesToQueue:withHandler:` (page 34)
- `startDeviceMotionUpdates` (page 34)

Declared In
`CMMotionManager.h`

stopGyroUpdates

Stops gyroscope updates.

```
- (void)stopGyroUpdates
```

Availability
Available in iOS 4.0 and later.

See Also
- `startGyroUpdatesToQueue:withHandler:` (page 35)
- `startGyroUpdates` (page 35)

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 33) as the second argument. Blocks of this type return no value but take two arguments:

`accelerometerData`
An object that encapsulates a `CMAcceleration` (page 10) 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 38) domain and the `CMError` (page 39) 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 35) as the second argument. Blocks of this type return no value but take two arguments:

`gyroData`
An object that encapsulates a `CMRotationRate` (page 22) 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 38) domain and the `CMError` (page 39) type.

Availability
Available in iOS 4.0 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 34) 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 38) domain and the `CMError` (page 39) 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
} CMError;
```

Constants

`CMErrorNULL`

Description forthcoming.

Available in iOS 4.0 and later.

Declared in `CMError.h`.

Availability

Available in iOS 4.0 and later.

Declared In

`CMError.h`

Document Revision History

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

Date	Notes
2010-04-27	First version of the reference describing the API for handling accelerometer data and other kinds of motion events.