Core Motion Framework Reference

Developer

Contents

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 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 magnetometer and a gyroscope. Through the `CMMotionManager` class you can start receiving accelerometer, gyroscope, magnetometer, and combined device-motion events at regular intervals or you can poll for them periodically.

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 46) 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

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 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	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 the `startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:` (page 49) method, the`startDeviceMotionUpdatesToQueue:withHandler:` (page 47) method, the `startDeviceMotionUpdatesUsingReferenceFrame:` (page 48) method or the `startDeviceMotionUpdates` (page 47) 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

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 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
`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 11)

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 11)

`@property pitch` (page 11)

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`

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`.

`CMAttitudeReferenceFrameXArbitraryCorrectedZVertical`

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

Available in iOS 5.0 and later.

Declared in `CMAttitude.h`.

`CMAttitudeReferenceFrameXMagneticNorthZVertical`

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 `CMAttitude.h`.

`CMAttitudeReferenceFrameXTrueNorthZVertical`

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 `CMAttitude.h`.

Discussion

Constants of this data type are returned or referenced by the `attitudeReferenceFrame` property and the `startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:` (page 49) and `startDeviceMotionUpdatesUsingReferenceFrame:` (page 48) methods of the `CMMotionManager` class. A bitmask of these constants is returned by the `availableAttitudeReferenceFrames` (page 45) class method.

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 `startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:` (page 49) method, the `startDeviceMotionUpdatesToQueue:withHandler:` (page 47) method, the `startDeviceMotionUpdatesUsingReferenceFrame:` (page 48) method, or the `startDeviceMotionUpdates` (page 47) 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

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 20)

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	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 50) method or the `startGyroUpdates` (page 49) 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

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 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	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 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

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
```

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	NSCoding (CMLogItem) NSCopying (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 51) method or from the `magnetometerData` (page 44) property of the `CMMotionManager` class.

> **Note** The `magnetometerData` (page 44) property of `CMMotionManager` provides a non-`nil` value only if you have called the `startMagnetometerUpdates` (page 51) 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

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

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`

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, 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 application 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 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 52), `stopGyroUpdates` (page 52), `stopMagnetometerUpdates` (page 53), and `stopDeviceMotionUpdates` (page 52)) when it has finished processing accelerometer, rotation-rate, magnetometer, or device-motion data.

Handing Motion Updates at Specified Intervals

To receive motion data at specific intervals, the 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 38) property to specify an update interval. Call the `startAccelerometerUpdatesToQueue:withHandler:` (page 46) method, passing in a block of type `CMAccelerometerHandler` (page 53). Accelerometer data is passed into the block as `CMAccelerometerData` objects.
- **Gyroscope.** Set the `gyroUpdateInterval` (page 42) property to specify an update interval. Call the `startGyroUpdatesToQueue:withHandler:` (page 50) method, passing in a block of type`CMGyroHandler` (page 54). Rotation-rate data is passed into the block as `CMGyroData` objects.
- **Magnetometer.** Set the `magnetometerUpdateInterval` (page 44) property to specify an update interval. Call the `startMagnetometerUpdatesToQueue:withHandler:` (page 51) method, passing a block of type `CMMagnetometerHandler` (page 55). Magnetic-field data is passed into the block as `CMMagnetometerData` objects.
- **Device motion.** Set the `deviceMotionUpdateInterval` (page 40) property to specify an update interval. Call the or `startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:` (page 49) or `startDeviceMotionUpdatesToQueue:withHandler:` (page 47) method, passing in a block of type `CMDeviceMotionHandler` (page 55). 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 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 overhead, and most game applications are interested only the latest sample of motion data when they render a frame.

- **Accelerometer.** Call `startAccelerometerUpdates` (page 46) to begin updates and periodically access `CMAccelerometerData` objects by reading the `accelerometerData` (page 38) property.
- **Gyroscope.** Call `startGyroUpdates` (page 49) to begin updates and periodically access `CMGyroData` objects by reading the `gyroData` (page 42) property.
- **Magnetometer.** Call `startMagnetometerUpdates` (page 51) to begin updates and periodically access `CMMagnetometerData` objects by reading the `magnetometerData` (page 44) property.
- **Device motion.** Call the `startDeviceMotionUpdatesUsingReferenceFrame:` (page 48) or `startDeviceMotionUpdates` (page 47) method to begin updates and periodically access `CMDeviceMotion` objects by reading the `deviceMotion` (page 39) 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 41) or `gyroActive` (page 41) properties.

Tasks

Managing Accelerometer Updates

`accelerometerUpdateInterval` (page 38) *property*

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

– `startAccelerometerUpdatesToQueue:withHandler:` (page 46)

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

– `startAccelerometerUpdates` (page 46)

Starts accelerometer updates without a handler.

– `stopAccelerometerUpdates` (page 52)
Stops accelerometer updates.

Determining Whether the Accelerometer Is Active and Available

`accelerometerActive` (page 37) *property*
A Boolean value that indicates whether accelerometer updates are currently happening. (read-only)

`accelerometerAvailable` (page 37) *property*
A Boolean value that indicates whether an accelerometer is available on the device. (read-only)

Accessing Accelerometer Data

`accelerometerData` (page 38) *property*
The latest sample of accelerometer data. (read-only)

Managing Gyroscope Updates

`gyroUpdateInterval` (page 42) *property*
The interval, in seconds, for providing gyroscope updates to the block handler.

– `startGyroUpdatesToQueue:withHandler:` (page 50)
Starts gyroscope updates on an operation queue and with a specified handler.

– `startGyroUpdates` (page 49)
Starts gyroscope updates without a handler.

– `stopGyroUpdates` (page 52)
Stops gyroscope updates.

Determining Whether the Gyroscope Is Active and Available

`gyroActive` (page 41) *property*
A Boolean value that determines whether gyroscope updates are currently happening. (read-only)

`gyroAvailable` (page 41) *property*
A Boolean value that indicates whether a gyroscope is available on the device. (read-only)

Accessing Gyroscope Data

`gyroData` (page 42) *property*
The latest sample of gyroscope data. (read-only)

Managing Magnetometer Updates

`magnetometerUpdateInterval` (page 44) *property*
The interval, in seconds, at which the system delivers magnetometer data to the block handler.

– `startMagnetometerUpdatesToQueue:withHandler:` (page 51)
Starts magnetometer updates on an operation queue and with a specified handler.

– `startMagnetometerUpdates` (page 51)
Starts magnetometer updates without a block handler.

– `stopMagnetometerUpdates` (page 53)
Stops magnetometer updates.

Determining Whether the Magnetometer Is Active and Available

`magnetometerActive` (page 43) *property*
A Boolean value that determines whether magnetometer updates are currently happening. (read-only)

`magnetometerAvailable` (page 43) *property*
A Boolean value that indicates whether a magnetometer is available on the device. (read-only)

Accessing Magnetometer Data

`magnetometerData` (page 44) *property*
The latest sample of magnetometer data. (read-only)

Managing the Device Movement Display

`showsDeviceMovementDisplay` (page 45) *property*
Controls whether the device-movement display is shown.

Managing Device Motion Updates

`deviceMotionUpdateInterval` (page 40) *property*
The interval, in seconds, for providing device-motion updates to the block handler.

– `startDeviceMotionUpdatesUsingReferenceFrame:toQueue:withHandler:` (page 49)
Starts device-motion updates on an operation queue and using a specified reference frame and block handler.

– `startDeviceMotionUpdatesToQueue:withHandler:` (page 47)
Starts device-motion updates on an operation queue and using a specified block handler.

– `startDeviceMotionUpdatesUsingReferenceFrame:` (page 48)
Starts device-motion updates using a reference frame but without a block handler.

– `startDeviceMotionUpdates` (page 47)
Starts device-motion updates without a block handler.

– `stopDeviceMotionUpdates` (page 52)
Stops device-motion updates.

Accessing Attitude Reference Frames

`attitudeReferenceFrame` (page 39) *property*
Returns either the reference frame currently being used or the default attitude reference frame (read-only)

+ `availableAttitudeReferenceFrames` (page 45)
Returns a bitmask specifying the available attitude reference frames on the device.

Determining Whether the Device Motion Hardware Is Active and Available

`deviceMotionActive` (page 39) *property*
A Boolean value that determines whether the application is receiving updates from the device-motion service. (read-only)

`deviceMotionAvailable` (page 40) *property*
A Boolean value that indicates whether the device-motion service is available on the device. (read-only)

Accessing Device Motion Data

`deviceMotion` (page 39) *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 46) or `startAccelerometerUpdates` (page 46) has been called since the last time `stopAccelerometerUpdates` (page 52) 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 47), but this property would return NO.)

Availability

Available in iOS 4.0 and later.

See Also

`@property accelerometerAvailable` (page 37)

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 37)

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 46) 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 46) 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`

attitudeReferenceFrame

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

```
@property(readonly, nonatomic) CMAttitudeReferenceFrame 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 application was last launched, this property returns the last frame used. If device motion has not been active since the application 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 40)
`+ availableAttitudeReferenceFrames` (page 45)

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 47) 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 47) or `startDeviceMotionUpdates` (page 47) has been called since the last time `stopDeviceMotionUpdates` (page 52) was called.

Availability

Available in iOS 4.0 and later.

See Also

`@property deviceMotionAvailable` (page 40)

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 41).

Availability

Available in iOS 4.0 and later.

See Also

`@property deviceMotionActive` (page 39)

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 47) 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 gyroscope updates are currently happening. (read-only)

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

Discussion

This property indicates whether `startGyroUpdatesToQueue:withHandler:` (page 50) or `startGyroUpdates` (page 49) has been called since the last time `stopGyroUpdates` (page 52) 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 47), but this property would return `NO`.)

Availability

Available in iOS 4.0 and later.

See Also

`@property gyroAvailable` (page 41)

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 41)

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 49) 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 50) 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 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

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 51) or `startMagnetometerUpdates` (page 51) method has been called since the last time the `stopMagnetometerUpdates` (page 53) method was called. (If the start methods hadn't been called, the application could be getting updates from the magnetometer after calling, for example, `startDeviceMotionUpdates` (page 47), but this property would return `NO`.)

Availability
Available in iOS 5.0 and later.

See Also
`@property magnetometerAvailable` (page 43)

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 43)

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 application that is receiving magnetometer data after calling `startMagnetometerUpdates` (page 51) 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 44)

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 51) 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 application 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 55) reports the `CMErrorDeviceRequiresMovement` (page 57) 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 15) 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 39)

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 38) property. You must call `stopAccelerometerUpdates` (page 52) when you no longer want your application to process accelerometer updates.

Availability
Available in iOS 4.0 and later.

See Also
`– startAccelerometerUpdatesToQueue:withHandler:` (page 46)

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 53) type.

Discussion

You must call `stopAccelerometerUpdates` (page 52) when you no longer want your application to process accelerometer updates.

Availability

Available in iOS 4.0 and later.

See Also

– `startAccelerometerUpdates` (page 46)

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 39) property. You must call `stopDeviceMotionUpdates` (page 52) when you no longer want your application to process device-motion updates. This method uses the reference frame returned by `attitudeReferenceFrame` (page 39) for device-motion updates.

Availability

Available in iOS 4.0 and later.

See Also

– `startDeviceMotionUpdatesToQueue:withHandler:` (page 47)

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 55) type.

Discussion

This method uses the reference frame returned by `attitudeReferenceFrame` (page 39) for device-motion updates. You must call `stopDeviceMotionUpdates` (page 52) when you no longer want your application to process device-motion updates.

Availability

Available in iOS 4.0 and later.

See Also

– `startDeviceMotionUpdates` (page 47)

Declared in

`CMMotionManager.h`

startDeviceMotionUpdatesUsingReferenceFrame:

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

```
-
(void)startDeviceMotionUpdatesUsingReferenceFrame:(CMAttitudeReferenceFrame)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 39) property. You must call `stopDeviceMotionUpdates` (page 52) when you no longer want your application 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:(CMAttitudeReferenceFrame)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 55) type.

Discussion
You must call `stopDeviceMotionUpdates` (page 52) when you no longer want your application 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 42) property. You must call `stopGyroUpdates` (page 52) when you no longer want your application to process gyroscope updates.

Availability

Available in iOS 4.0 and later.

See Also

– `startGyroUpdatesToQueue:withHandler:` (page 50)

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 54) type.

Discussion

You must call `stopGyroUpdates` (page 52) when you no longer want your application to process gyroscope updates.

Availability

Available in iOS 4.0 and later.

See Also

– `startGyroUpdates` (page 49)

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 44) property. You must call `stopMagnetometerUpdates` (page 53) when you no longer want your application to process magnetometer updates.

Availability

Available in iOS 5.0 and later.

See Also

– `startMagnetometerUpdatesToQueue:withHandler:` (page 51)

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 55) type.

Discussion

You must call `stopMagnetometerUpdates` (page 53) when you no longer want your application to process magnetometer updates.

Availability

Available in iOS 5.0 and later.

See Also
– `startMagnetometerUpdates` (page 51)

Declared in
`CMMotionManager.h`

stopAccelerometerUpdates

Stops accelerometer updates.

```
- (void)stopAccelerometerUpdates
```

Availability
Available in iOS 4.0 and later.

See Also
– `startAccelerometerUpdatesToQueue:withHandler:` (page 46)
– `startAccelerometerUpdates` (page 46)

Declared in
`CMMotionManager.h`

stopDeviceMotionUpdates

Stops device-motion updates.

```
- (void)stopDeviceMotionUpdates
```

Availability
Available in iOS 4.0 and later.

See Also
– `startDeviceMotionUpdatesToQueue:withHandler:` (page 47)
– `startDeviceMotionUpdates` (page 47)

Declared in
`CMMotionManager.h`

stopGyroUpdates

Stops gyroscope updates.

```
- (void)stopGyroUpdates
```

Availability
Available in iOS 4.0 and later.

See Also
- `startGyroUpdatesToQueue:withHandler:` (page 50)
- `startGyroUpdates` (page 49)

Declared in
`CMMotionManager.h`

stopMagnetometerUpdates

Stops magnetometer updates.

```
- (void)stopMagnetometerUpdates
```

Availability
Available in iOS 5.0 and later.

See Also
- `startMagnetometerUpdatesToQueue:withHandler:` (page 51)
- `startMagnetometerUpdates` (page 51)

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 46) 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 56) domain and the `CMError` (page 56) 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 50) 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 56) domain and the `CMError` (page 56) 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 51) method as the second argument. Blocks of this type return no value but take two arguments:

`magnetometerData`
: An object that encapsulates a `CMMagneticField` (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 56) domain and the `CMError` (page 56) 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 47) 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 56) domain and the `CMError` (page 56) 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
} 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`.

Declared in

`CMError.h`

Document Revision History

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

Date	Notes
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.
1 Infinite Loop
Cupertino, CA 95014
408-996-1010