Core Location Framework Reference

Developer

Contents

Core Location Framework Reference

Framework	/System/Library/Frameworks/CoreLocation.framework
Header file directories	/System/Library/Frameworks/CoreLocation.framework/Headers
Companion guide	Location and Maps Programming Guide
Declared in	CLBeaconRegion.h CLCircularRegion.h CLError.h CLErrorDomain.h CLGeocoder.h CLHeading.h CLLocation.h CLLocationManager.h CLLocationManagerDelegate.h CLPlacemark.h CLRegion.h

The Core Location framework lets you determine the current location or heading associated with a device. The framework uses the available hardware to determine the user's position and heading. You use the classes and protocols in this framework to configure and schedule the delivery of location and heading events. You can also use it to define geographic regions and monitor when the user crosses the boundaries of those regions. In iOS, you can also define a region around a Bluetooth beacon.

Classes

CLBeacon Class Reference

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

Overview

The `CLBeacon` class represents a beacon that was encountered during region monitoring. You do not create instances of this class directly. The location manager object reports encountered beacons to its associated delegate object. You can use the information in a beacon object to identify which beacon was encountered.

The identity of a beacon is defined by its `proximityUUID` (page 9), `major` (page 8), and `minor` (page 9) properties. These values are coded into the beacon itself. For a more thorough description of the meaning of those values, see *CLBeaconRegion Class Reference* .

Tasks

Identifying the Beacon

`proximityUUID` (page 9) *property*
  The proximity ID of the beacon. (read-only)

`major` (page 8) *property*
  The most significant value in the beacon. (read-only)

`minor` (page 9) *property*
    The least significant value in the beacon. (read-only)

Determining the Beacon Distance

`proximity` (page 9) *property*
    The relative distance to the beacon. (read-only)

`accuracy` (page 8) *property*
    The accuracy of the proximity value, measured in meters from the beacon. (read-only)

`rssi` (page 10) *property*
    The received signal strength of the beacon, measured in decibels. (read-only)

Properties

accuracy

The accuracy of the proximity value, measured in meters from the beacon. (read-only)

```
@property (readonly, nonatomic) CLLocationAccuracy accuracy;
```

Discussion
Indicates the one sigma horizontal accuracy in meters. Use this property to differentiate between beacons with the same proximity value. Do not use it to identify a precise location for the beacon. Accuracy values may fluctuate due to RF interference.

A negative value in this property signifies that the actual accuracy could not be determined.

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

major

The most significant value in the beacon. (read-only)

```
@property (readonly, nonatomic) NSNumber *major;
```

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

minor

The least significant value in the beacon. (read-only)

```
@property (readonly, nonatomic) NSNumber *minor;
```

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

proximity

The relative distance to the beacon. (read-only)

```
@property (readonly, nonatomic) CLProximity proximity;
```

Discussion
The value in this property gives a general sense of the relative distance to the beacon. Use it to quickly identify beacons that are nearer to the user rather than farther away.

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

proximityUUID

The proximity ID of the beacon. (read-only)

```
@property (readonly, nonatomic) NSUUID *proximityUUID;
```

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

rssi

The received signal strength of the beacon, measured in decibels. (read-only)

```
@property (readonly, nonatomic) NSInteger rssi;
```

Discussion
This value is the average RSSI value of the samples received since the range of the beacon was last reported to your app.

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

Constants

CLProximity

Constants that reflect the relative distance to a beacon.

```
typedef {
   CLProximityUnknown,
   CLProximityImmediate,
   CLProximityNear,
   CLProximityFar
} CLProximity;
```

Constants
`CLProximityUnknown`
> The proximity of the beacon could not be determined.
>
> Available in iOS 7.0 and later.
>
> Declared in `CLRegion.h`.

`CLProximityImmediate`
: The beacon is in the user's immediate vicinity.

  Available in iOS 7.0 and later.

  Declared in `CLRegion.h`.

`CLProximityNear`
: The beacon is relatively close to the user.

  Available in iOS 7.0 and later.

  Declared in `CLRegion.h`.

`CLProximityFar`
: The beacon is far away.

  Available in iOS 7.0 and later.

  Declared in `CLRegion.h`.

CLBeaconRegion Class Reference

Inherits from	CLRegion : NSObject
Conforms to	NSCopying (CLRegion) NSSecureCoding (CLRegion) NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreLocation.framework
Availability	Available in iOS 7.0 and later.
Declared in	CLBeaconRegion.h
Companion guide	Location and Maps Programming Guide

Overview

A `CLBeaconRegion` object defines a type of region that is based on the device's proximity to a Bluetooth beacon, as opposed to a geographic location. A beacon region looks for devices whose identifying information matches the information you provide. When that device comes in range, the region triggers the delivery of an appropriate notification.

You can monitor beacon regions in two ways. To receive notifications when a device enters or exits the vicinity of a beacon, use the `startMonitoringForRegion:` (page 74) method of your location manager object. While a beacon is in range, you can also call the `startRangingBeaconsInRegion:` (page 77) method to begin receiving notifications when the relative distance to the beacon changes.

If you want to configure the current iOS device as a Bluetooth beacon, create a beacon region with the appropriate identifying information. You can then call the `peripheralDataWithMeasuredPower:` (page 18) method of the region to get a dictionary that you can use to advertise the device with the Core Bluetooth framework. For more information about using that framework to advertise the device as a beacon, see *Location and Maps Programming Guide* .

Specifying a Beacon's Identity

You identify beacons using a combination of three values:

- The `proximityUUID` (page 15) property contains the identifier that you use to identify your company's beacons. You typically generate only one UUID for your company's beacons but can generate more as needed. You generate this value using the `uuidgen` command-line tool.
- The `major` (page 14) property contains a value that can be used to group related sets of beacons. For example, a department store might assign the same major value for all of the beacons on the same floor.
- The `minor` (page 14) property specifies the individual beacon within a group. For example, for a group of beacons on the same floor of a department store, this value might be assigned to a beacon in a particular section.

You program the identity values into the beacon hardware itself using the tools provided by the beacon manufacturer. In your app, you then use those values to identify which beacon was found and respond appropriately.

Tasks

Initializing the Beacon Region

– `initWithProximityUUID:identifier:` (page 16)
Initializes and returns a region object that targets a beacon with the specified proximity ID.

– `initWithProximityUUID:major:identifier:` (page 16)
Initializes and returns a region object that targets a beacon with the specified proximity ID and major value.

– `initWithProximityUUID:major:minor:identifier:` (page 17)
Initializes and returns a region object that targets a beacon with the specified proximity ID, major value, and minor value.

Accessing the Beacon Attributes

`proximityUUID` (page 15) *property*
The unique ID of the beacons being targeted. (read-only)

`major` (page 14) *property*
The value identifying a group of beacons. (read-only)

`minor` (page 14) *property*
The value identifying a specific beacon within a group. (read-only)

Delivering Beacon Notifications

`notifyEntryStateOnDisplay` (page 15) *property*
A Boolean indicating whether beacon notifications are sent when the device's display is on.

Getting Beacon Advertisement Data

– `peripheralDataWithMeasuredPower:` (page 18)
Retrieves data that can be used to advertise the current device as a beacon.

Properties

major

The value identifying a group of beacons. (read-only)

```
@property (readonly, nonatomic) NSNumber *major
```

Discussion
If you do not specify a major value for the beacon, the value in this property is `nil`. When this property is `nil`, the major value of the beacon is ignored when determining if it is a match.

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

minor

The value identifying a specific beacon within a group. (read-only)

```
@property (readonly, nonatomic) NSNumber *minor
```

Discussion
If you do not specify a minor value for the beacon, the value in this property is `nil`. When this property is `nil`, the minor value of the beacon is ignored when determining if it is a match.

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

notifyEntryStateOnDisplay

A Boolean indicating whether beacon notifications are sent when the device's display is on.

```
@property (nonatomic, assign) BOOL notifyEntryStateOnDisplay;
```

Discussion
When set to `YES`, the location manager sends beacon notifications when the user turns on the display and the device is already inside the region. These notifications are sent even if your app is not running. In that situation, the system launches your app into the background so that it can handle the notifications. In both situations, the location manager calls the `locationManager:didDetermineState:forRegion:` (page 106) method of its delegate object.

The default value for this property is `NO`.

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

proximityUUID

The unique ID of the beacons being targeted. (read-only)

```
@property (readonly, nonatomic) NSUUID *proximityUUID
```

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

Instance Methods

initWithProximityUUID:identifier:

Initializes and returns a region object that targets a beacon with the specified proximity ID.

```
- (id)initWithProximityUUID:(NSUUID *)proximityUUID identifier:(NSString *)identifier;
```

Parameters

`proximityUUID`
The unique ID of the beacons being targeted. This value must not be `nil`.

`identifier`
A unique identifier to associate with the returned region object. You use this identifier to differentiate regions within your application. This value must not be `nil`.

Return Value
An initialized beacon region object.

Discussion
This method creates a region that results in the reporting of all beacons with the specified `proximityUUID` value. The `major` (page 14) and `minor` (page 14) values of the beacons are ignored.

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

initWithProximityUUID:major:identifier:

Initializes and returns a region object that targets a beacon with the specified proximity ID and major value.

```
- (id)initWithProximityUUID:(NSUUID *)proximityUUID major:(CLBeaconMajorValue)major
 identifier:(NSString *)identifier;
```

Parameters

`proximityUUID`
The unique ID of the beacons being targeted. This value must not be `nil`.

`major`
The major value that you use to identify one or more beacons.

`identifier`

A unique identifier to associate with the returned region object. You use this identifier to differentiate regions within your application. This value must not be `nil`.

Return Value

An initialized beacon region object.

Discussion

This method creates a region that reports all beacons with the specified `proximityUUID` and `major` values. The beacon's `minor` (page 14) value is ignored.

Availability

Available in iOS 7.0 and later.

Declared in

`CLBeaconRegion.h`

initWithProximityUUID:major:minor:identifier:

Initializes and returns a region object that targets a beacon with the specified proximity ID, major value, and minor value.

```
- (id)initWithProximityUUID:(NSUUID *)proximityUUID major:(CLBeaconMajorValue)major
 minor:(CLBeaconMinorValue)minor identifier:(NSString *)identifier;
```

Parameters

`proximityUUID`

The proximity ID of the beacon being targeted. This value must not be `nil`.

`major`

The major value that you use to identify one or more beacons.

`minor`

The minor value that you use to identify a specific beacon.

`identifier`

A unique identifier to associate with the returned region object. You use this identifier to differentiate regions within your application. This value must not be `nil`.

Return Value

An initialized beacon region object.

Discussion

This method creates a region that reports the beacon with the specified `proximityUUID`, `major`, and `minor` values.

Availability

Available in iOS 7.0 and later.

Declared in

`CLBeaconRegion.h`

peripheralDataWithMeasuredPower:

Retrieves data that can be used to advertise the current device as a beacon.

```
- (NSMutableDictionary *)peripheralDataWithMeasuredPower:(NSNumber *)measuredPower
```

Parameters

`measuredPower`
: The received signal strength indicator (RSSI) value (measured in decibels) for the device. This value represents the measured strength of the beacon from one meter away and is used during ranging. Specify `nil` to use the default value for the device.

Return Value

A dictionary of data that you can use in conjunction with a `CBPeripheralManager` to advertise the current device as a beacon.

Discussion

The returned dictionary encodes the beacon's identifying information along with other information needed to advertise the beacon. You should not need to access the dictionary contents directly. Pass the dictionary to the `startAdvertising:` method of a `CBPeripheralManager` to begin advertising the beacon.

Availability

Available in iOS 7.0 and later.

Declared in

`CLBeaconRegion.h`

Constants

CLBeaconMajorValue

The most significant value in a beacon.

```
typedef uint16_t CLBeaconMajorValue;
```

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

CLBeaconMinorValue

The least significant value in a beacon.

```
typedef uint16_t CLBeaconMinorValue;
```

Availability
Available in iOS 7.0 and later.

Declared in
`CLBeaconRegion.h`

CLCircularRegion Class Reference

Inherits from	CLRegion : NSObject
Conforms to	NSCopying (CLRegion) NSSecureCoding (CLRegion) NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreLocation.framework
Availability	Available in iOS 7.0 and later.
Declared in	CLCircularRegion.h
Companion guide	Location and Maps Programming Guide

Overview

The `CLCircularRegion` class defines the location and boundaries for a circular geographic region. You can use instances of this class to define geo fences for a specific location. The crossing of a geo fence's boundary causes the location manager to notify its delegate.

Tasks

Initializing a Circular Region

– `initWithCenter:radius:identifier:` (page 22)

Initializes and returns a region object defining a circular geographic area.

Accessing a Region's Attributes

`center` (page 21) *property*

The center point of the geographic area. (read-only)

`radius` (page 21) *property*
The radius (measured in meters) that defines the geographic area's outer boundary. (read-only)

Hit Testing in a Region

– `containsCoordinate:` (page 22)
Returns a Boolean value indicating whether the geographic area contains the specified coordinate.

Properties

center

The center point of the geographic area. (read-only)

```
@property (readonly, nonatomic) CLLocationCoordinate2D center
```

Availability
Available in iOS 7.0 and later.

Declared in
`CLCircularRegion.h`

radius

The radius (measured in meters) that defines the geographic area's outer boundary. (read-only)

```
@property (readonly, nonatomic) CLLocationDistance radius
```

Availability
Available in iOS 7.0 and later.

Declared in
`CLCircularRegion.h`

Instance Methods

containsCoordinate:

Returns a Boolean value indicating whether the geographic area contains the specified coordinate.

```
- (BOOL)containsCoordinate:(CLLocationCoordinate2D)coordinate;
```

Parameters

`coordinate`

The coordinate to test against the region.

Return Value

`YES` if the coordinate lies within the region's boundaries or `NO` if it does not.

Availability

Available in iOS 7.0 and later.

Declared in

`CLCircularRegion.h`

initWithCenter:radius:identifier:

Initializes and returns a region object defining a circular geographic area.

```
- (id)initWithCenter:(CLLocationCoordinate2D)center radius:(CLLocationDistance)radius
 identifier:(NSString *)identifier;
```

Parameters

`center`

The center point of the geographic region to monitor.

`radius`

The distance (measured in meters) from the center point of the geographic region to the edge of the circular boundary.

`identifier`

A unique identifier to associate with the region object. You use this identifier to differentiate regions within your application. This value must not be `nil`.

Return Value

An initialized region object.

Discussion
When defining a geographic region, remember that the location manager does not generate notifications immediately upon crossing a region boundary. Instead, it applies time and distance criteria to ensure that the crossing was intended and should genuinely trigger a notification. So choose a center point and radius that are appropriate and give you enough time to alert the user. For more information, see the information about region monitoring in *Location and Maps Programming Guide* .

Availability
Available in iOS 7.0 and later.

Declared in
`CLCircularRegion.h`

CLGeocoder Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreLocation.framework
Availability	Available in iOS 5.0 and later.
Declared in	CoreLocation/CLGeocoder.h
Companion guide	Location and Maps Programming Guide
Related sample code	CurrentAddress GeocoderDemo

Overview

The `CLGeocoder` class provides services for converting between a coordinate (specified as a latitude and longitude) and the user-friendly representation of that coordinate. A user-friendly representation of the coordinate typically consists of the street, city, state, and country information corresponding to the given location, but it may also contain a relevant point of interest, landmarks, or other identifying information. A geocoder object is a single-shot object that works with a network-based service to look up placemark information for its specified coordinate value.

To use a geocoder object, create it and call one of its forward- or reverse-geocoding methods to begin the request. **Reverse-geocoding** requests take a latitude and longitude value and find a user-readable address. **Forward-geocoding** requests take a user-readable address and find the corresponding latitude and longitude value. Forward-geocoding requests may also return additional information about the specified location, such as a point of interest or building at that location. For both types of request, the results are returned using a `CLPlacemark` object. In the case of forward-geocoding requests, multiple placemark objects may be returned if the provided information yielded multiple possible locations.

To make smart decisions about what types of information to return, the geocoder server uses all the information provided to it when processing the request. For example, if the user is moving quickly along a highway, it might return the name of the overall region, and not the name of a small park that the user is passing through.

Applications should be conscious of how they use geocoding. Geocoding requests are rate-limited for each app, so making too many requests in a short period of time may cause some of the requests to fail. (When the maximum rate is exceeded, the geocoder returns an error object with the value `kCLErrorNetwork` (page 127) to the associated completion handler.) Here are some rules of thumb for using this class effectively:

- Send at most one geocoding request for any one user action.
- If the user performs multiple actions that involve geocoding the same location, reuse the results from the initial geocoding request instead of starting individual requests for each action.
- When you want to update the user's current location automatically (such as when the user is moving), issue new geocoding requests only when the user has moved a significant distance and after a reasonable amount of time has passed. For example, in a typical situation, you should not send more than one geocoding request per minute.
- Do not start a geocoding request at a time when the user will not see the results immediately. For example, do not start a request if your application is inactive or in the background.

The computer or device must have access to the network in order for the geocoder object to return detailed placemark information. Although, the geocoder stores enough information locally to report the localized country name and ISO country code for many locations. If country information is not available for a specific location, the geocoder may still report an error to your completion block.

You can use geocoder objects either in conjunction with, or independent of, the classes of the Map Kit framework.

Tasks

Reverse Geocoding a Location

– `reverseGeocodeLocation:completionHandler:` (page 29)
 Submits a reverse-geocoding request for the specified location.

Geocoding an Address

– `geocodeAddressDictionary:completionHandler:` (page 27)
 Submits a forward-geocoding request using the specified address dictionary.

– `geocodeAddressString:completionHandler:` (page 28)
 Submits a forward-geocoding request using the specified string.

– `geocodeAddressString:inRegion:completionHandler:` (page 28)
    Submits a forward-geocoding request using the specified string and region information.

Managing Geocoding Requests

– `cancelGeocode` (page 26)
    Cancels a pending geocoding request.

`geocoding` (page 26) *property*
    A Boolean value indicating whether the receiver is in the middle of geocoding its value. (read-only)

Properties

geocoding

A Boolean value indicating whether the receiver is in the middle of geocoding its value. (read-only)

```
@property (nonatomic, readonly, getter=isGeocoding) BOOL geocoding
```

Discussion
This property contains the value `YES` if the process is ongoing or `NO` if the process is done or has not yet been initiated.

Availability
Available in iOS 5.0 and later.

Declared in
`CLGeocoder.h`

Instance Methods

cancelGeocode

Cancels a pending geocoding request.

```
- (void)cancelGeocode
```

Discussion

You can use this method to cancel a pending request and free up the resources associated with that request. Canceling a pending request causes the completion handler block to be called.

If the request is not pending, because it has already returned or has not yet begun, this method does nothing.

Availability

Available in iOS 5.0 and later.

Declared in

`CLGeocoder.h`

geocodeAddressDictionary:completionHandler:

Submits a forward-geocoding request using the specified address dictionary.

```
- (void)geocodeAddressDictionary:(NSDictionary *)addressDictionary
completionHandler:(CLGeocodeCompletionHandler)completionHandler
```

Parameters

`addressDictionary`

An Address Book dictionary containing information about the address to look up.

`completionHandler`

A block object containing the code to execute at the end of the request. This code is called whether the request is successful or unsuccessful.

Discussion

This method submits the specified location data to the geocoding server asynchronously and returns. Your completion handler block will be executed on the main thread. After initiating a forward-geocoding request, do not attempt to initiate another forward- or reverse-geocoding request.

Geocoding requests are rate-limited for each app, so making too many requests in a short period of time may cause some of the requests to fail. When the maximum rate is exceeded, the geocoder passes an error object with the value `kCLErrorNetwork` (page 127) to your completion handler.

Availability

Available in iOS 5.0 and later.

Related Sample Code

GeocoderDemo

Declared in

`CLGeocoder.h`

geocodeAddressString:completionHandler:

Submits a forward-geocoding request using the specified string.

```
- (void)geocodeAddressString:(NSString *)addressString
completionHandler:(CLGeocodeCompletionHandler)completionHandler
```

Parameters

`addressString`

A string describing the location you want to look up. For example, you could specify the string "1 Infinite Loop, Cupertino, CA" to locate Apple headquarters.

`completionHandler`

A block object containing the code to execute at the end of the request. This code is called whether the request is successful or unsuccessful.

Discussion

This method submits the specified location data to the geocoding server asynchronously and returns. Your completion handler block will be executed on the main thread. After initiating a forward-geocoding request, do not attempt to initiate another forward- or reverse-geocoding request.

Geocoding requests are rate-limited for each app, so making too many requests in a short period of time may cause some of the requests to fail. When the maximum rate is exceeded, the geocoder passes an error object with the value `kCLErrorNetwork` (page 127) to your completion handler.

Availability

Available in iOS 5.0 and later.

Related Sample Code
GeocoderDemo

Declared in

`CLGeocoder.h`

geocodeAddressString:inRegion:completionHandler:

Submits a forward-geocoding request using the specified string and region information.

```
- (void)geocodeAddressString:(NSString *)addressString inRegion:(CLRegion *)region
completionHandler:(CLGeocodeCompletionHandler)completionHandler
```

Parameters

`addressString`

A string describing the location you want to look up. For example, you could specify the string “1 Infinite Loop, Cupertino, CA” to locate Apple headquarters.

`region`

A geographical region to use as a hint when looking up the specified address. Specifying a region lets you prioritize the returned set of results to locations that are close to some specific geographical area, which is typically the user’s current location. If `nil` and the application is authorized for location services, the set of results is prioritized based on the user’s approximate location. Invoking this method does not trigger a location services authorization request.

`completionHandler`

A block object containing the code to execute at the end of the request. This code is called whether the request is successful or unsuccessful.

Discussion

This method submits the specified location data to the geocoding server asynchronously and returns. Your completion handler block will be executed on the main thread. After initiating a forward-geocoding request, do not attempt to initiate another forward- or reverse-geocoding request.

Geocoding requests are rate-limited for each app, so making too many requests in a short period of time may cause some of the requests to fail. When the maximum rate is exceeded, the geocoder passes an error object with the value `kCLErrorNetwork` (page 127) to your completion handler.

Availability

Available in iOS 5.0 and later.

Related Sample Code
GeocoderDemo

Declared in

`CLGeocoder.h`

reverseGeocodeLocation:completionHandler:

Submits a reverse-geocoding request for the specified location.

```
- (void)reverseGeocodeLocation:(CLLocation *)location
completionHandler:(CLGeocodeCompletionHandler)completionHandler
```

Parameters

`location`

The location object containing the coordinate data to look up.

`completionHandler`

A block object containing the code to execute at the end of the request. This code is called whether the request is successful or unsuccessful.

Discussion

This method submits the specified location data to the geocoding server asynchronously and returns. Your completion handler block will be executed on the main thread. After initiating a reverse-geocoding request, do not attempt to initiate another reverse- or forward-geocoding request.

Geocoding requests are rate-limited for each app, so making too many requests in a short period of time may cause some of the requests to fail. When the maximum rate is exceeded, the geocoder passes an error object with the value `kCLErrorNetwork` (page 127) to your completion handler.

Availability

Available in iOS 5.0 and later.

Related Sample Code
GeocoderDemo

Declared in
`CLGeocoder.h`

Constants

CLGeocodeCompletionHandler

A block to be called when a geocoding request is complete.

```
typedef void (^CLGeocodeCompletionHandler)(NSArray *placemark, NSError *error);
```

Discussion

Upon completion of a geocoding request, a block of this form is called to give you a chance to process the results. The parameters of this block are as follows:

`placemark`

Contains an array of `CLPlacemark` objects. For most geocoding requests, this array should contain only one entry. However, forward-geocoding requests may return multiple placemark objects in situations where the specified address could not be resolved to a single location.

If the request was canceled or there was an error in obtaining the placemark information, this parameter is `nil`.

`error`
: Contains a pointer to an error object (if any) indicating why the placemark data was not returned. For a list of possible error codes, see *Core Location Constants Reference* .

Availability
Available in iOS 5.0 and later.

Declared in
`CLGeocoder.h`

CLHeading Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSSecureCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreLocation.framework
Availability	Available in iOS 3.0 and later.
Declared in	CLHeading.h
Companion guide	Location and Maps Programming Guide
Related sample code	Teslameter

Overview

A `CLHeading` object contains heading data generated by a `CLLocationManager` object. The heading data consists of computed values for true and magnetic north. It also includes the raw data for the three-dimensional vector used to compute those values.

Typically, you do not create instances of this class yourself, nor do you subclass it. Instead, you receive instances of this class through the delegate assigned to the `CLLocationManager` object whose `startUpdatingHeading` (page 78) method you called.

Note: If you want heading objects to contain valid data for the `trueHeading` (page 35) property, your location manager object should also be configured to deliver location updates. You can start the delivery of these updates by calling the location manager object's `startUpdatingLocation` (page 79) method.

Tasks

Accessing the Heading Attributes

`magneticHeading` (page 34) *property*
: The heading (measured in degrees) relative to magnetic north. (read-only)

`trueHeading` (page 35) *property*
: The heading (measured in degrees) relative to true north. (read-only)

`headingAccuracy` (page 34) *property*
: The maximum deviation (measured in degrees) between the reported heading and the true geomagnetic heading. (read-only)

`timestamp` (page 35) *property*
: The time at which this heading was determined. (read-only)

– `description` (page 37)
: Returns the heading data in a formatted text string.

Accessing the Raw Heading Data

`x` (page 36) *property*
: The geomagnetic data (measured in microteslas) for the x-axis. (read-only)

`y` (page 36) *property*
: The geomagnetic data (measured in microteslas) for the y-axis. (read-only)

`z` (page 37) *property*
: The geomagnetic data (measured in microteslas) for the z-axis. (read-only)

Properties

headingAccuracy

The maximum deviation (measured in degrees) between the reported heading and the true geomagnetic heading. (read-only)

```
@property(readonly, nonatomic) CLLocationDirection headingAccuracy
```

Discussion
A positive value in this property represents the potential error between the value reported by the `magneticHeading` (page 34) property and the actual direction of magnetic north. Thus, the lower the value of this property, the more accurate the heading. A negative value means that the reported heading is invalid, which can occur when the device is uncalibrated or there is strong interference from local magnetic fields.

Availability
Available in iOS 3.0 and later.

Declared in
`CLHeading.h`

magneticHeading

The heading (measured in degrees) relative to magnetic north. (read-only)

```
@property(readonly, nonatomic) CLLocationDirection magneticHeading
```

Discussion
The value in this property represents the heading relative to the magnetic North Pole, which is different from the geographic North Pole. The value `0` means the device is pointed toward magnetic north, `90` means it is pointed east, `180` means it is pointed south, and so on. The value in this property should always be valid.

In iOS 3.x and earlier, the value in this property is always measured relative to the top of the device in a portrait orientation, regardless of the device's actual physical or interface orientation. In iOS 4.0 and later, the value is measured relative to the heading orientation specified by the location manager. For more information, see the `headingOrientation` (page 61) property in *CLLocationManager Class Reference* .

If the `headingAccuracy` property contains a negative value, the value in this property should be considered unreliable.

Availability
Available in iOS 3.0 and later.

See Also
`@property headingAccuracy` (page 34)
`@property trueHeading` (page 35)

Declared in
`CLHeading.h`

timestamp

The time at which this heading was determined. (read-only)

```
@property(readonly, nonatomic) NSDate *timestamp
```

Availability
Available in iOS 3.0 and later.

Declared in
`CLHeading.h`

trueHeading

The heading (measured in degrees) relative to true north. (read-only)

```
@property(readonly, nonatomic) CLLocationDirection trueHeading
```

Discussion
The value in this property represents the heading relative to the geographic North Pole. The value `0` means the device is pointed toward true north, `90` means it is pointed due east, `180` means it is pointed due south, and so on. A negative value indicates that the heading could not be determined.

In iOS 3.x and earlier, the value in this property is always measured relative to the top of the device in a portrait orientation, regardless of the device's actual physical or interface orientation. In iOS 4.0 and later, the value is measured relative to the heading orientation specified by the location manager. For more information, see the `headingOrientation` (page 61) property in *CLLocationManager Class Reference*.

Important: This property contains a valid value only if location updates are also enabled for the corresponding location manager object. Because the position of true north is different from the position of magnetic north on the Earth's surface, Core Location needs the current location of the device to compute the value of this property.

Availability
Available in iOS 3.0 and later.

See Also

`@property magneticHeading` (page 34)

Declared in

`CLHeading.h`

x

The geomagnetic data (measured in microteslas) for the x-axis. (read-only)

```
@property(readonly, nonatomic) CLHeadingComponentValue x
```

Discussion

This value represents the x-axis deviation from the magnetic field lines being tracked by the device.

Availability

Available in iOS 3.0 and later.

Related Sample Code

Teslameter

Declared in

`CLHeading.h`

y

The geomagnetic data (measured in microteslas) for the y-axis. (read-only)

```
@property(readonly, nonatomic) CLHeadingComponentValue y
```

Discussion

This value represents the y-axis deviation from the magnetic field lines being tracked by the device.

Availability

Available in iOS 3.0 and later.

Related Sample Code

Teslameter

Declared in

`CLHeading.h`

z

The geomagnetic data (measured in microteslas) for the z-axis. (read-only)

```
@property(readonly, nonatomic) CLHeadingComponentValue z
```

Discussion
This value represents the z-axis deviation from the magnetic field lines being tracked by the device.

Availability
Available in iOS 3.0 and later.

Related Sample Code
Teslameter

Declared in
`CLHeading.h`

Instance Methods

description

Returns the heading data in a formatted text string.

```
- (NSString *)description
```

Return Value
A string of the form "magneticHeading <*magnetic*> trueHeading <*heading*> accuracy <*accuracy*> x <*x*> y <*y*> z <*z*> @ <*date-time*>" where <*magnetic*>, <*heading*>, <*accuracy*>, <*x*>, <*y*>, and <*z*> are formatted floating-point numbers and <*date-time*> is a formatted date string that includes date, time, and time zone information.

Availability
Available in iOS 3.0 and later.

Declared in
`CLHeading.h`

Constants

CLHeadingComponentValue

A type used to report magnetic differences reported by the onboard hardware.

```
typedef double CLHeadingComponentValue;
```

Availability
Available in iOS 3.0 and later.

Declared in
`CLHeading.h`

CLLocation Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSSecureCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreLocation.framework
Availability	Available in iOS 2.0 and later.
Declared in	CLLocation.h
Companion guide	Location and Maps Programming Guide
Related sample code	AVMovieExporter GeocoderDemo LocateMe pARk PhotosByLocation

Overview

A `CLLocation` object represents the location data generated by a `CLLocationManager` object. This object incorporates the geographical coordinates and altitude of the device's location along with values indicating the accuracy of the measurements and when those measurements were made. In iOS, this class also reports information about the speed and heading in which the device is moving.

Typically, you use a `CLLocationManager` object to create instances of this class based on the last known location of the user's device. You can create instances yourself, however, if you want to cache custom location data or get the distance between two points.

This class is designed to be used as is and should not be subclassed.

Tasks

Initializing a Location Object

– `initWithLatitude:longitude:` (page 48)

Initializes and returns a location object with the specified latitude and longitude.

– `initWithCoordinate:altitude:horizontalAccuracy:verticalAccuracy:timestamp:` (page 47)

Initializes and returns a location object with the specified coordinate information.

– `initWithCoordinate:altitude:horizontalAccuracy:verticalAccuracy:course:speed:timestamp:` (page 46)

Initializes and returns a location object with the specified coordinate and course information.

Location Attributes

`coordinate` (page 41) *property*

The geographical coordinate information. (read-only)

`altitude` (page 41) *property*

The altitude measured in meters. (read-only)

`horizontalAccuracy` (page 42) *property*

The radius of uncertainty for the location, measured in meters. (read-only)

`verticalAccuracy` (page 44) *property*

The accuracy of the altitude value in meters. (read-only)

`timestamp` (page 43) *property*

The time at which this location was determined. (read-only)

– `description` (page 45)

Returns the location data in a formatted text string.

Measuring the Distance Between Coordinates

– `distanceFromLocation:` (page 45)

Returns the distance (in meters) from the receiver's location to the specified location.

– `getDistanceFrom:` (page 46)

Returns the distance (in meters) from the receiver's location to the specified location. (Deprecated. Use the `distanceFromLocation:` (page 45) method instead.)

Getting Speed and Course Information

`speed` (page 43) *property*
The instantaneous speed of the device in meters per second.

`course` (page 42) *property*
The direction in which the device is traveling.

Properties

altitude

The altitude measured in meters. (read-only)

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) CLLocationDistance altitude
```

Discussion
Positive values indicate altitudes above sea level. Negative values indicate altitudes below sea level.

Special Considerations
In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.

Availability
Available in iOS 2.0 and later.

See Also
`@property verticalAccuracy` (page 44)

Declared in
`CLLocation.h`

coordinate

The geographical coordinate information. (read-only)

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) CLLocationCoordinate2D coordinate
```

Discussion
When running in the simulator, Core Location assigns a fixed set of coordinate values to this property. You must run your application on an iOS-based device to get real location values.

Special Considerations

In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.

Availability

Available in iOS 2.0 and later.

Related Sample Code

Breadcrumb

GeocoderDemo

MapSearch

pARk

PhotosByLocation

Declared in

`CLLocation.h`

course

The direction in which the device is traveling.

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) CLLocationDirection course
```

Discussion

Course values are measured in degrees starting at due north and continuing clockwise around the compass. Thus, north is 0 degrees, east is 90 degrees, south is 180 degrees, and so on. Course values may not be available on all devices. A negative value indicates that the direction is invalid.

Special Considerations

In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.

Availability

Available in iOS 2.2 and later.

Related Sample Code

LocateMe

Declared in

`CLLocation.h`

horizontalAccuracy

The radius of uncertainty for the location, measured in meters. (read-only)

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) CLLocationAccuracy horizontalAccuracy
```

Discussion

The location’s latitude and longitude identify the center of the circle, and this value indicates the radius of that circle. A negative value indicates that the location’s latitude and longitude are invalid.

Special Considerations

In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.

Availability

Available in iOS 2.0 and later.

Related Sample Code
LocateMe

Declared in

`CLLocation.h`

speed

The instantaneous speed of the device in meters per second.

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) CLLocationSpeed speed
```

Discussion

This value reflects the instantaneous speed of the device in the direction of its current heading. A negative value indicates an invalid speed. Because the actual speed can change many times between the delivery of subsequent location events, you should use this property for informational purposes only.

Special Considerations

In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.

Availability

Available in iOS 2.2 and later.

Related Sample Code
LocateMe

Declared in

`CLLocation.h`

timestamp

The time at which this location was determined. (read-only)

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) NSDate *timestamp
```

Special Considerations

In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.

Availability

Available in iOS 2.0 and later.

Related Sample Code
GeocoderDemo

Simple Core Data Relationships

Declared in

`CLLocation.h`

verticalAccuracy

The accuracy of the altitude value in meters. (read-only)

```
@property(readonly, NS_NONATOMIC_IPHONEONLY) CLLocationAccuracy verticalAccuracy
```

Discussion

The value in the `altitude` property could be plus or minus the value indicated by this property. A negative value indicates that the altitude value is invalid.

Determining the vertical accuracy requires a device with GPS capabilities. Thus, on some earlier iOS-based devices, this property always contains a negative value.

Special Considerations

In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.

Availability

Available in iOS 2.0 and later.

See Also

`@property altitude` (page 41)

Related Sample Code
LocateMe

Declared in

`CLLocation.h`

Instance Methods

description

Returns the location data in a formatted text string.

```
- (NSString *)description
```

Return Value
A string of the form "<<*latitude*>, <*longitude*>> +/- <*accuracy*> m (speed <*speed*> kph / heading <*heading*>) @ <*date-time*>", where <*latitude*>, <*longitude*>, <*accuracy*>, <*speed*>, and <*heading*> are formatted floating point numbers and <*date-time*> is a formatted date string that includes date, time, and time zone information.

Discussion
The returned string is intended for display purposes only.

Availability
Available in iOS 2.0 and later.

Declared in
`CLLocation.h`

distanceFromLocation:

Returns the distance (in meters) from the receiver's location to the specified location.

```
- (CLLocationDistance)distanceFromLocation:(const CLLocation *)location
```

Parameters
`location`
The other location.

Return Value
The distance (in meters) between the two locations.

Discussion
This method measures the distance between the two locations by tracing a line between them that follows the curvature of the Earth. The resulting arc is a smooth curve and does not take into account specific altitude changes between the two locations.

Availability
Available in iOS 3.2 and later.

Related Sample Code
GeocoderDemo

Declared in
`CLLocation.h`

getDistanceFrom:

Returns the distance (in meters) from the receiver's location to the specified location. (Deprecated in iOS 3.2. Use the `distanceFromLocation:` (page 45) method instead.)

```
- (CLLocationDistance)getDistanceFrom:(const CLLocation *)location
```

Parameters
`location`
The other location.

Return Value
The distance (in meters) between the two locations.

Discussion
This method measures the distance between the two locations by tracing a line between them that follows the curvature of the Earth. The resulting arc is a smooth curve and does not take into account specific altitude changes between the two locations.

Availability
Available in iOS 2.0 and later.

Deprecated in iOS 3.2.

Declared in
`CLLocation.h`

initWithCoordinate:altitude:horizontalAccuracy:verticalAccuracy:course:speed: timestamp:

Initializes and returns a location object with the specified coordinate and course information.

```
- (id)initWithCoordinate:(CLLocationCoordinate2D)coordinate
altitude:(CLLocationDistance)altitude horizontalAccuracy:(CLLocationAccuracy)hAccuracy
 verticalAccuracy:(CLLocationAccuracy)vAccuracy course:(CLLocationDirection)course
 speed:(CLLocationSpeed)speed timestamp:(NSDate *)timestamp
```

Parameters

`coordinate`
: A coordinate structure containing the latitude and longitude values.

`altitude`
: The altitude value for the location.

`hAccuracy`
: The accuracy of the coordinate value. Specifying a negative number indicates that the coordinate value is invalid.

`vAccuracy`
: The accuracy of the altitude value. Specifying a negative number indicates that the altitude value is invalid.

`course`
: The direction of travel for the location.

`speed`
: The current speed associated with this location.

`timestamp`
: The time to associate with the location object. Typically, you would set this to the current time.

Return Value

A location object initialized with the specified information.

Discussion

Typically, you acquire location objects from the location service, but you can use this method to create new location objects for other uses in your application.

Availability

Available in iOS 4.2 and later.

Declared in

`CLLocation.h`

initWithCoordinate:altitude:horizontalAccuracy:verticalAccuracy:timestamp:

Initializes and returns a location object with the specified coordinate information.

```
- (id)initWithCoordinate:(CLLocationCoordinate2D)coordinate
altitude:(CLLocationDistance)altitude horizontalAccuracy:(CLLocationAccuracy)hAccuracy
 verticalAccuracy:(CLLocationAccuracy)vAccuracy timestamp:(NSDate *)timestamp
```

Parameters

`coordinate`
: A coordinate structure containing the latitude and longitude values.

`altitude`
: The altitude value for the location.

`hAccuracy`
: The accuracy of the coordinate value. Specifying a negative number indicates that the coordinate value is invalid.

`vAccuracy`
: The accuracy of the altitude value. Specifying a negative number indicates that the altitude value is invalid.

`timestamp`
: The time to associate with the location object. Typically, you would set this to the current time.

Return Value

A location object initialized with the specified information.

Discussion

Typically, you acquire location objects from the location service, but you can use this method to create new location objects for other uses in your application.

Availability

Available in iOS 2.0 and later.

Declared in

`CLLocation.h`

initWithLatitude:longitude:

Initializes and returns a location object with the specified latitude and longitude.

```
- (id)initWithLatitude:(CLLocationDegrees)latitude
longitude:(CLLocationDegrees)longitude
```

Parameters

`latitude`
: The latitude of the coordinate point.

`longitude`
: The longitude of the coordinate point.

Return Value

A location object initialized with the specified coordinate point.

Discussion

Typically, you acquire location objects from the location service, but you can use this method to create new location objects for other uses in your application. When using this method, the other properties of the object are initialized to appropriate values. In particular, the `altitude` and `horizontalAccuracy` properties are set to 0, the `verticalAccuracy` property is set to -1 to indicate that the altitude value is invalid, and the `timestamp` property is set to the time at which the instance was initialized.

Availability

Available in iOS 2.0 and later.

Related Sample Code

AVMovieExporter

Core Data Transformable Attributes

GeocoderDemo

pARk

Simple Core Data Relationships

Declared in

`CLLocation.h`

CLLocationManager Class Reference

Inherits from	NSObject
Conforms to	NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreLocation.framework
Availability	Available in iOS 2.0 and later.
Declared in	CLLocationManager.h
Companion guide	Location and Maps Programming Guide
Related sample code	Checking and Requesting Access to Data Classes in Privacy Settings GeocoderDemo LocateMe Regions Teslameter

Overview

The `CLLocationManager` class defines the interface for configuring the delivery of location- and heading-related events to your application. You use an instance of this class to establish the parameters that determine when location and heading events should be delivered and to start and stop the actual delivery of those events. You can also use a location manager object to retrieve the most recent location and heading data.

A location manager object provides support for the following location-related activities:

- Tracking large or small changes in the user's current location with a configurable degree of accuracy.
- Reporting heading changes from the onboard compass. (iOS only)
- Monitoring distinct regions of interest and generating location events when the user enters or leaves those regions.
- Deferring the delivery of location updates while the app is in the background. (iOS 6 and later only)
- Reporting the range to nearby beacons.

Some location services require the presence of specific hardware on the given device. For example, heading information is available only for devices that contain a hardware compass. This class defines several methods that you can use to determine which services are currently available.

> **Important:** In addition to hardware not being available, the user has the option of denying an application's access to location service data. During its initial uses by an application, the Core Location framework prompts the user to confirm that using the location service is acceptable. If the user denies the request, the `CLLocationManager` object reports an appropriate error to its delegate during future requests. You can also check the application's explicit authorization status using the `authorizationStatus` (page 65) method.

For the services you do use, you should configure any properties associated with that service accurately. The location manager object manages power aggressively by turning off hardware when it is not needed. For example, setting the desired accuracy for location events to one kilometer gives the location manager the flexibility to turn off GPS hardware and rely solely on the WiFi or cell radios. Turning off GPS hardware can lead to significant power savings.

To configure and use a `CLLocationManager` object to deliver events:

1. Always check to see whether the desired services are available before starting any services and abandon the operation if they are not.
2. Create an instance of the `CLLocationManager` class.
3. Assign a custom object to the `delegate` (page 58) property. This object must conform to the `CLLocationManagerDelegate` protocol.
4. Configure any additional properties relevant to the desired service.
5. Call the appropriate start method to begin the delivery of events.

All location- and heading-related updates are delivered to the associated delegate object, which is a custom object that you provide. For information about the delegate methods, see *CLLocationManagerDelegate Protocol Reference* .

Getting the User's Current Location

There are two options for configuring location-related services:

- Use the standard location services, which allow you to specify the desired accuracy of the location data and receive updates as the location changes. Standard location services are available in all versions of iOS and in OS X 10.6 and later.
- Request events for significant location changes only, which provides a more limited set of tracking options but offers tremendous power savings and the ability to receive location updates even if your application is not running. This service is available only in iOS 4.0 and later and requires a device with a cellular radio.

Start standard location services by calling the `startUpdatingLocation` (page 79) method. This service is most appropriate for applications that need more fine-grained control over the delivery of location events. Specifically, it takes into account the values in the `desiredAccuracy` (page 58) and `distanceFilter` (page 59) property to determine when to deliver new events. The precision of the standard location services are needed by navigation applications or any application where high-precision location data or a regular stream of updates is required. However, these services typically require the location-tracking hardware to be enabled for longer periods of time, which can result in higher power usage.

For applications that do not need a regular stream of location events, consider using the `startMonitoringSignificantLocationChanges` (page 76) method to start the delivery of events instead. This method is more appropriate for the majority of applications that just need an initial user location fix and need updates only when the user moves a significant distance. This interface delivers new events only when it detects changes to the device's associated cell towers, resulting in less frequent updates and significantly lower power usage.

Regardless of which location service you use, location data is reported to your application via the location manager's associated delegate object. Because it can take several seconds to return an initial location, the location manager typically delivers the previously cached location data immediately and then delivers more up-to-date location data as it becomes available. Therefore it is always a good idea to check the timestamp of any location object before taking any actions. If both location services are enabled simultaneously, they deliver events using the same set of delegate methods.

In iOS 6 and later, you can defer the delivery of location data when your app is in the background. It is recommended that you use this feature in situations where your app could process the data later without any problems. For example, an app that tracks the user's location on a hiking trail could defer updates until the user hikes a certain distance and then process the points all at once. Deferring updates helps save power by allowing your app to remain asleep for longer periods of time.

Using Regions to Monitor Boundary Crossings

In iOS 4.0 and later and OS X 10.8 and later, you can use the region-monitoring service to define the boundaries for multiple geographical regions. After registering a region using the `startMonitoringForRegion:` (page 74) method, the location manager tracks movement across the region's boundary and reports that movement to its delegate. You might use region monitoring to alert the user to approaching landmarks or to provide other relevant information. For example, upon approaching a dry cleaners, an application could notify the user to pick up any clothes that had been dropped off and are now ready.

In iOS, the regions you register with the location manager persist between launches of your application. If a region crossing occurs while your iOS app is not running, the system automatically wakes it up (or relaunches it) in the background so that it can process the event. When relaunched, all of the regions you configured previously are made available in the `monitoredRegions` (page 63) property of any location manager objects you create.

In OS X, region monitoring works only while the app is running (either in the foreground or background) and the user's system is awake. The system does not launch apps to deliver region-related notifications. Similarly, if the user puts the computer to sleep, the system does not deliver region monitoring notifications to your app. If the user wakes up the computer inside a monitored region, the system does deliver region notifications to your app if it is running. However, if the computer enters and exits the region before being woken up, no notification is delivered.

Note: The region monitoring interfaces were introduced in OS X 10.7 but support for monitoring actual regions is supported only in OS X 10.8 and later.

The region monitoring service operates independently of any location services in use by your application, and you may use it in conjunction with any of the other services. Region monitoring is not supported on all devices. Use the `regionMonitoringAvailable` (page 69) class method to determine if region monitoring can be used.

Configuring Heading-Related Services

In iOS, a device with the appropriate hardware may also report heading information. When the value in the `headingAvailable` (page 60) property is YES, you can use a location manager object to retrieve heading information. To begin the delivery of heading-related events, assign a delegate to the location manager object and call the location manager's `startUpdatingHeading` (page 78) method. If location updates are also enabled, the location manager returns both the true heading and magnetic heading values. If location updates are not enabled, the location manager returns only the magnetic heading value. These features are not available in OS X.

Tasks

Accessing the Delegate

`delegate` (page 58) *property*
The delegate object to receive update events.

Determining the Availability of Services

+ `authorizationStatus` (page 65)
: Returns the application's authorization status for using location services.

+ `locationServicesEnabled` (page 68)
: Returns a Boolean value indicating whether location services are enabled on the device.

+ `deferredLocationUpdatesAvailable` (page 66)
: Returns a Boolean value indicating whether the device supports deferred location updates.

+ `significantLocationChangeMonitoringAvailable` (page 70)
: Returns a Boolean value indicating whether significant location change tracking is available.

+ `headingAvailable` (page 67)
: Returns a Boolean value indicating whether the location manager is able to generate heading-related events.

+ `isMonitoringAvailableForClass:` (page 67)
: Returns a Boolean indicating whether the device supports region monitoring using the specified class.

+ `isRangingAvailable` (page 68)
: Returns a Boolean indicating whether the device supports ranging of Bluetooth beacons.

+ `regionMonitoringAvailable` (page 69) Deprecated in iOS 7.0
: Returns a Boolean value indicating whether region monitoring is supported on the current device.

Initiating Standard Location Updates

– `startUpdatingLocation` (page 79)
: Starts the generation of updates that report the user's current location.

– `stopUpdatingLocation` (page 82)
: Stops the generation of location updates.

`pausesLocationUpdatesAutomatically` (page 64) *property*
: A Boolean value indicating whether the location manager object may pause location updates.

`distanceFilter` (page 59) *property*
: The minimum distance (measured in meters) a device must move horizontally before an update event is generated.

`desiredAccuracy` (page 58) *property*
: The accuracy of the location data.

`activityType` (page 57) *property*
  The type of user activity associated with the location updates.

Initiating Significant Location Updates

– `startMonitoringSignificantLocationChanges` (page 76)
  Starts the generation of updates based on significant location changes.

– `stopMonitoringSignificantLocationChanges` (page 80)
  Stops the delivery of location events based on significant location changes.

Initiating Heading Updates

– `startUpdatingHeading` (page 78)
  Starts the generation of updates that report the user's current heading.

– `stopUpdatingHeading` (page 81)
  Stops the generation of heading updates.

– `dismissHeadingCalibrationDisplay` (page 73)
  Dismisses the heading calibration view from the screen immediately.

`headingFilter` (page 60) *property*
  The minimum angular change (measured in degrees) required to generate new heading events.

`headingOrientation` (page 61) *property*
  The device orientation to use when computing heading values.

Initiating Region Monitoring

– `startMonitoringForRegion:` (page 74)
  Starts monitoring the specified region.

– `stopMonitoringForRegion:` (page 80)
  Stops monitoring the specified region.

`monitoredRegions` (page 63) *property*
  The set of shared regions monitored by all location manager objects. (read-only)

`maximumRegionMonitoringDistance` (page 63) *property*
  The largest boundary distance that can be assigned to a region. (read-only)

Initiating Beacon Ranging Requests

– `startRangingBeaconsInRegion:` (page 77)
Starts the delivery of notifications for beacons in the specified region.

– `stopRangingBeaconsInRegion:` (page 81)
Stops the delivery of notifications for the specified beacon region.

– `requestStateForRegion:` (page 73)
Retrieves the state of a region asynchronously.

`rangedRegions` (page 65) *property*
The set of regions currently being tracked using ranging. (read-only)

Deferring Location Updates

– `allowDeferredLocationUpdatesUntilTraveled:timeout:` (page 71)
Asks the location manager to defer the delivery of location updates until the specified criteria are met.

– `disallowDeferredLocationUpdates` (page 73)
Cancels the deferral of location updates for this app.

Getting Recently Retrieved Data

`location` (page 61) *property*
The most recently retrieved user location. (read-only)

`heading` (page 59) *property*
The most recently reported heading. (read-only)

Describing Your Application's Services to the User

`purpose` (page 64) *property* Deprecated in iOS 6.0
An application-provided string that describes the reason for using location services.

Deprecated Properties and Methods

`locationServicesEnabled` (page 62) *property*
A Boolean value indicating whether location services are enabled on the device. (read-only) (Deprecated. Use the `locationServicesEnabled` (page 68) class method instead.)

`headingAvailable` (page 60) *property*
A Boolean value indicating whether the location manager is able to generate heading-related events. (read-only) (Deprecated. Use the `headingAvailable` (page 67) class method instead.)

`+ regionMonitoringEnabled` (page 69) Deprecated in iOS 6.0
Returns a Boolean value indicating whether region monitoring is currently enabled. (Deprecated. Use `regionMonitoringAvailable` (page 69) and `authorizationStatus` (page 65) instead.)

`– startMonitoringForRegion:desiredAccuracy:` (page 75) Deprecated in iOS 6.0
Starts monitoring the specified region for boundary crossings. (Deprecated. Use `startMonitoringForRegion:` (page 74) instead.)

Properties

activityType

The type of user activity associated with the location updates.

```
@property(assign, nonatomic) CLActivityType activityType
```

Discussion
The location manager uses the information in this property as a cue to determine when location updates may be automatically paused. Pausing updates gives the system the opportunity to save power in situations where the user's location is not likely to be changing. For example, if the activity type is `CLActivityTypeAutomotiveNavigation` (page 84) and no location changes have occurred recently, the radios might be powered down until movement is detected again.

The default value of this property is `CLActivityTypeOther` (page 84).

Availability
Available in iOS 6.0 and later.

Declared in
`CLLocationManager.h`

delegate

The delegate object to receive update events.

```
@property(assign, nonatomic) id<CLLocationManagerDelegate> delegate
```

Special Considerations

In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.

Availability

Available in iOS 2.0 and later.

Related Sample Code

LocateMe

MapSearch

Regions

Teslameter

Declared in

`CLLocationManager.h`

desiredAccuracy

The accuracy of the location data.

```
@property(assign, nonatomic) CLLocationAccuracy desiredAccuracy
```

Discussion

The receiver does its best to achieve the requested accuracy; however, the actual accuracy is not guaranteed.

You should assign a value to this property that is appropriate for your usage scenario. For example, if you need the current location only within a kilometer, you should specify `kCLLocationAccuracyKilometer` (page 131) and not `kCLLocationAccuracyBestForNavigation` (page 130). Determining a location with greater accuracy requires more time and more power.

When requesting high-accuracy location data, the initial event delivered by the location service may not have the accuracy you requested. The location service delivers the initial event as quickly as possible. It then continues to determine the location with the accuracy you requested and delivers additional events, as necessary, when that data is available.

The default value of this property is `kCLLocationAccuracyBest`.

This property is used only in conjunction with the standard location services and is not used when monitoring significant location changes.

Special Considerations

In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.

Availability

Available in iOS 2.0 and later.

Related Sample Code

LocateMe

Regions

Declared in

`CLLocationManager.h`

distanceFilter

The minimum distance (measured in meters) a device must move horizontally before an update event is generated.

```
@property(assign, nonatomic) CLLocationDistance distanceFilter
```

Discussion

This distance is measured relative to the previously delivered location. Use the value `kCLDistanceFilterNone` (page 132) to be notified of all movements. The default value of this property is `kCLDistanceFilterNone`.

This property is used only in conjunction with the standard location services and is not used when monitoring significant location changes.

Special Considerations

In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.

Availability

Available in iOS 2.0 and later.

Related Sample Code

LocateMe

Regions

Declared in

`CLLocationManager.h`

heading

The most recently reported heading. (read-only)

```
@property(readonly, nonatomic) CLHeading *heading
```

Discussion
The value of this property is `nil` if heading updates have never been initiated.

Availability
Available in iOS 4.0 and later.

Declared in
`CLLocationManager.h`

headingAvailable

A Boolean value indicating whether the location manager is able to generate heading-related events. (read-only) (Deprecated in iOS 4.0. Use the `headingAvailable` (page 67) class method instead.)

```
@property(readonly, nonatomic) BOOL headingAvailable
```

Discussion
Heading data may not be available on all iOS-based devices. You should check the value of this property before asking the location manager to deliver heading-related events.

Availability
Available in iOS 3.0 and later.

Deprecated in iOS 4.0.

See Also
– `startUpdatingHeading` (page 78)

Declared in
`CLLocationManager.h`

headingFilter

The minimum angular change (measured in degrees) required to generate new heading events.

```
@property(assign, nonatomic) CLLocationDegrees headingFilter
```

Discussion
The angular distance is measured relative to the last delivered heading event. Use the value `kCLHeadingFilterNone` (page 132) to be notified of all movements. The default value of this property is `1` degree.

Availability
Available in iOS 3.0 and later.

Related Sample Code
Teslameter

Declared in
`CLLocationManager.h`

headingOrientation

The device orientation to use when computing heading values.

```
@property(assign, nonatomic) CLDeviceOrientation headingOrientation
```

Discussion
When computing heading values, the location manager assumes that the top of the device in portrait mode represents due north (0 degrees) by default. For applications that run in other orientations, this may not always be the most convenient orientation. This property allows you to specify which device orientation you want the location manager to use as the reference point for due north.

Although you can set the value of this property to `CLDeviceOrientationUnknown` (page 125), `CLDeviceOrientationFaceUp` (page 126), or `CLDeviceOrientationFaceDown` (page 126), doing so has no effect on the orientation reference point. The original reference point is retained instead.

Changing the value in this property affects only those heading values reported after the change is made.

Availability
Available in iOS 4.0 and later.

Declared in
`CLLocationManager.h`

location

The most recently retrieved user location. (read-only)

```
@property(readonly, nonatomic) CLLocation *location
```

Discussion
The value of this property is `nil` if no location data has ever been retrieved.

In iOS 4.0 and later, this property may contain a more recent location object at launch time. Specifically, if significant location updates are running and your application is terminated, this property is updated with the most recent location data when your application is relaunched (and you create a new location manager object). This location data may be more recent than the last location event processed by your application.

It is always a good idea to check the timestamp of the location stored in this property. If the receiver is currently gathering location data, but the minimum distance filter is large, the returned location might be relatively old. If it is, you can stop the receiver and start it again to force an update.

Availability
Available in iOS 2.0 and later.

See Also
– `startUpdatingLocation` (page 79)

Related Sample Code
AVMovieExporter

Declared in
`CLLocationManager.h`

locationServicesEnabled

A Boolean value indicating whether location services are enabled on the device. (read-only) (Deprecated in iOS 4.0. Use the `locationServicesEnabled` (page 68) class method instead.)

```
@property(readonly, nonatomic) BOOL locationServicesEnabled
```

Discussion
In iOS, the user can enable or disable location services using the controls in Settings > Location Services. In OS X, the user can enable or disable location services from the Security & Privacy system preference.

You should check this property before starting location updates to determine whether the user has location services enabled for the current device. If this property contains the value `NO` and you start location updates anyway, the Core Location framework prompts the user with a confirmation alert asking whether location services should be reenabled.

Special Considerations
In iOS, this property is declared as `nonatomic`. In OS X, it is declared as `atomic`.

Availability
Available in iOS 2.0 and later.

Deprecated in iOS 4.0.

See Also

– `startUpdatingLocation` (page 79)

Related Sample Code

LocateMe

Declared in

`CLLocationManager.h`

maximumRegionMonitoringDistance

The largest boundary distance that can be assigned to a region. (read-only)

```
@property(readonly, nonatomic) CLLocationDistance maximumRegionMonitoringDistance
```

Discussion

This property defines the largest boundary distance allowed from a region's center point. Attempting to monitor a region with a distance larger than this value causes the location manager to send a `kCLErrorRegionMonitoringFailure` (page 128) error to the delegate.

If region monitoring is unavailable or not supported, the value in this property is `–1`.

Availability

Available in iOS 4.0 and later.

Declared in

`CLLocationManager.h`

monitoredRegions

The set of shared regions monitored by all location manager objects. (read-only)

```
@property(readonly, nonatomic) NSSet *monitoredRegions
```

Discussion

You cannot add regions to this property directly. Instead, you must register regions by calling the `startMonitoringForRegion:` (page 74) method. The regions in this property are shared by all instances of the `CLLocationManager` class in your application.

The objects in this set may not necessarily be the same objects you specified at registration time. Only the region data itself is maintained by the system. Therefore, the only way to uniquely identify a registered region is using its `identifier` (page 97) property.

The location manager persists region data between launches of your application. If your application is terminated and then relaunched, the contents of this property are repopulated with region objects that contain the previously registered data.

Availability
Available in iOS 4.0 and later.

Declared in
`CLLocationManager.h`

pausesLocationUpdatesAutomatically

A Boolean value indicating whether the location manager object may pause location updates.

```
@property(assign, nonatomic) BOOL pausesLocationUpdatesAutomatically
```

Discussion
Allowing the location manager to pause updates can improve battery life on the target device without sacrificing location data. When this property is set to `YES`, the location manager pauses updates (and powers down the appropriate hardware) at times when the location data is unlikely to change. For example, if the user stops for food while using a navigation app, the location manager might pause updates for a period of time. You can help the determination of when to pause location updates by assigning a value to the `activityType` property.

The default value of this property is `YES`.

Availability
Available in iOS 6.0 and later.

See Also
`@property activityType` (page 57)

Declared in
`CLLocationManager.h`

purpose

An application-provided string that describes the reason for using location services. (Deprecated in iOS 6.0.)

```
@property(copy, nonatomic) NSString *purpose
```

Discussion
If this property is not `nil` and the system needs to ask for the user's consent to use location services, it displays the provided string. You can use this string to explain why your application is using location services.

You must set the value of this property prior to starting any location services. Because the string is ultimately displayed to the user, you should always load it from a localized strings file.

Availability
Available in iOS 3.2 and later.

Deprecated in iOS 6.0.

Declared in
`CLLocationManager.h`

rangedRegions

The set of regions currently being tracked using ranging. (read-only)

```
@property (readonly, nonatomic) NSSet *rangedRegions
```

Discussion
The objects in the set are instances of the `CLBeaconRegion` class.

Availability
Available in iOS 7.0 and later.

See Also
– `startRangingBeaconsInRegion:` (page 77)

Declared in
`CLLocationManager.h`

Class Methods

authorizationStatus

Returns the application's authorization status for using location services.

```
+ (CLAuthorizationStatus)authorizationStatus
```

Return Value
A value indicating whether the application is authorized to use location services.

Discussion
The authorization status of a given application is managed by the system and determined by several factors. Applications must be explicitly authorized to use location services by the user and location services must themselves currently be enabled for the system. A request for user authorization is displayed automatically when your application first attempts to use location services.

Availability
Available in iOS 4.2 and later.

Declared in
`CLLocationManager.h`

deferredLocationUpdatesAvailable

Returns a Boolean value indicating whether the device supports deferred location updates.

```
+ (BOOL)deferredLocationUpdatesAvailable
```

Return Value
`YES` if the device supports deferred location updates or `NO` if it does not.

Discussion
Deferred location updates are a way for the location manager to avoid frequently waking up a background app to deliver location changes. Normally, when an app wants location updates in the background, the app must be woken up whenever a new event arrives. Waking up the app consumes power, which in some situations might be wasted if the app cannot do anything with the location information other than log it and go back to sleep anyway. Deferring location updates gives you the ability to wait until a time when your app can do something useful with the data and then process the updates all at once.

Deferred location updates require the presence of GPS hardware and may not be supported on all iOS devices.

Availability
Available in iOS 6.0 and later.

See Also
– `allowDeferredLocationUpdatesUntilTraveled:timeout:` (page 71)

Declared in
`CLLocationManager.h`

headingAvailable

Returns a Boolean value indicating whether the location manager is able to generate heading-related events.

```
+ (BOOL)headingAvailable
```

Return Value
`YES` if heading data is available; `NO` if it is not.

Discussion
Heading data may not be available on all iOS-based devices. You should check the value returned by this method before asking the location manager to deliver heading-related events.

Availability
Available in iOS 4.0 and later.

Declared in
`CLLocationManager.h`

isMonitoringAvailableForClass:

Returns a Boolean indicating whether the device supports region monitoring using the specified class.

```
+ (BOOL)isMonitoringAvailableForClass:(Class)regionClass
```

Parameters
`regionClass`
A region monitoring class from the Map Kit framework. This class must be descend from the `CLRegion` class.

Return Value
`YES` if the device is capable of monitoring regions using the specified class or `NO` if it is not.

Discussion
The availability of region monitoring support is dependent on the hardware present on the device. This method does not take into account the availability of location services or the fact that the user might have disabled them for the app or system; you must determine your app's authorization status separately.

Availability
Available in iOS 7.0 and later.

See Also
`+ authorizationStatus` (page 65)

Declared in
`CLLocationManager.h`

isRangingAvailable

Returns a Boolean indicating whether the device supports ranging of Bluetooth beacons.

```
+ (BOOL)isRangingAvailable
```

Return Value
`YES` if the device supports ranging or `NO` if it does not.

Availability
Available in iOS 7.0 and later.

Declared in
`CLLocationManager.h`

locationServicesEnabled

Returns a Boolean value indicating whether location services are enabled on the device.

```
+ (BOOL)locationServicesEnabled
```

Return Value
`YES` if location services are enabled; `NO` if they are not.

Discussion
The user can enable or disable location services from the Settings application by toggling the Location Services switch in General.

You should check the return value of this method before starting location updates to determine whether the user has location services enabled for the current device. Location services prompts users the first time they attempt to use location-related information in an app but does not prompt for subsequent attempts. If the user denies the use of location services and you attempt to start location updates anyway, the location manager reports an error to its delegate.

Availability
Available in iOS 4.0 and later.

See Also
`locationManager:didFailWithError:` (page 108)

`locationManager:monitoringDidFailForRegion:withError:` (page 112)

Declared in
`CLLocationManager.h`

regionMonitoringAvailable

Returns a Boolean value indicating whether region monitoring is supported on the current device. (Deprecated in iOS 7.0.)

```
+ (BOOL)regionMonitoringAvailable
```

Return Value
`YES` if region monitoring is available; `NO` if it is not.

Discussion
Support for region monitoring may not be available on all devices and models. You should check the value of this property before attempting to set up any regions or initiate region monitoring.

Even if region monitoring support is present on a device, it may still be unavailable because the user disabled it for the current application or for all applications.

Special Considerations
This class is deprecated in iOS 7 and later but is still supported in OS X.

Availability
Available in iOS 4.0 and later.

Deprecated in iOS 7.0.

See Also
`+ regionMonitoringEnabled` (page 69)

Declared in
`CLLocationManager.h`

regionMonitoringEnabled

Returns a Boolean value indicating whether region monitoring is currently enabled. (Deprecated in iOS 6.0. Use `regionMonitoringAvailable` (page 69) and `authorizationStatus` (page 65) instead.)

```
+ (BOOL)regionMonitoringEnabled
```

Return Value

`YES` if region monitoring is available and is currently enabled; `NO` if it is unavailable or not enabled.

Discussion

In iOS, the user can enable or disable location services (including region monitoring) using the controls in Settings > Location Services.

You should check the return value of this method before starting region monitoring updates to determine whether the user currently allows location services to be used at all. If this method returns `NO` and you start region monitoring updates anyway, the Core Location framework prompts the user to confirm asking whether location services should be reenabled.

This method does not check to see if region monitoring capabilities are actually supported by the device. Therefore, you should also check the return value of the `regionMonitoringAvailable` class method before attempting to start region monitoring services.

Availability

Available in iOS 4.0 and later.

Deprecated in iOS 6.0.

See Also

`+ regionMonitoringAvailable` (page 69)

Declared in

`CLLocationManager.h`

significantLocationChangeMonitoringAvailable

Returns a Boolean value indicating whether significant location change tracking is available.

```
+ (BOOL)significantLocationChangeMonitoringAvailable
```

Return Value

`YES` if location change monitoring is available; `NO` if it is not.

Discussion

This method indicates whether the device is able to report updates based on significant location changes only. (Significant location change monitoring primarily involves detecting changes in the cell tower currently associated with the device.) This capability provides tremendous power savings for applications that want to track a user's approximate location and do not need highly accurate position information.

Availability
Available in iOS 4.0 and later.

Declared in
`CLLocationManager.h`

Instance Methods

allowDeferredLocationUpdatesUntilTraveled:timeout:

Asks the location manager to defer the delivery of location updates until the specified criteria are met.

```
- (void)allowDeferredLocationUpdatesUntilTraveled:(CLLocationDistance)distance
timeout:(NSTimeInterval)timeout
```

Parameters

`distance`
: The distance (in meters) from the current location that must be travelled before event delivery resumes. To specify an unlimited distance, pass the `CLLocationDistanceMax` (page 83) constant.

`timeout`
: The amount of time (in seconds) from the current time that must pass before event delivery resumes. To specify an unlimited amount of time, pass the `CLTimeIntervalMax` (page 83) constant.

Discussion
Call this method in situations where you want location data with GPS accuracy but do not need to process that data right away. If your app is in the background and the system is able to optimize its power usage, the location manager tells the GPS hardware to store new locations internally until the specified distance or timeout conditions are met. When one or both criteria are met, the location manager ends deferred locations by calling the `locationManager:didFinishDeferredUpdatesWithError:` (page 109) method of its delegate and delivers the cached locations to the `locationManager:didUpdateLocations:` (page 111) method. If your app is in the foreground, the location manager does not defer the deliver of events but does monitor for the specified criteria. If your app moves to the background before the criteria are met, the location manager may begin deferring the delivery of events.

> **Important:** Because deferred updates use the GPS to track location changes, the location manager allows deferred updates only when GPS hardware is available on the device and when the desired accuracy is set to `kCLLocationAccuracyBest` (page 131) or `kCLLocationAccuracyBestForNavigation` (page 130). If the GPS hardware is not available, the location manager reports a `kCLErrorDeferredFailed` (page 129) error. If the accuracy is not set to one of the supported values, the location manager reports a `kCLErrorDeferredAccuracyTooLow` (page 129) error.
>
> In addition, the `distanceFilter` (page 59) property of the location manager must be set to `kCLDistanceFilterNone`. If it is set to any other value, the location manager reports a `kCLErrorDeferredDistanceFiltered` (page 129) error.

Start the delivery of location updates before calling this method. The most common place to call this method is in your delegate's `locationManager:didUpdateLocations:` method. After processing any new locations, call this method if you want to defer future updates until the distance or time criteria are met. If new events arrive and your app is in the background, the events are cached and their delivery is deferred appropriately.

Your delegate's `locationManager:didFinishDeferredUpdatesWithError:` method is called exactly once for each time you call this method. If you call this method twice in succession, the location manager cancels the previous deferral before starting the new one. Therefore, you should keep track of whether updates are currently deferred and avoid calling this method multiple times in succession. If you want to change the deferral criteria for any reason, and therefore call this method again, be prepared to receive a `kCLErrorDeferredCanceled` (page 129) error in your delegate's `locationManager:didFinishDeferredUpdatesWithError:` method.

After calling this method, the location manager may deliver location updates even if the specified distance and timeout criteria are not met. For example, if the caches used to store deferred samples become full, the location manager may deliver the cached samples so it can collect new ones. The delivery of samples does not automatically end deferred mode for your app. The location manager resumes deferred mode when it is able to do so.

Availability
Available in iOS 6.0 and later.

See Also
– `disallowDeferredLocationUpdates` (page 73)

Declared in
`CLLocationManager.h`

disallowDeferredLocationUpdates

Cancels the deferral of location updates for this app.

```
- (void)disallowDeferredLocationUpdates
```

Discussion
Call this method if you previously deferred location event delivery using the `allowDeferredLocationUpdatesUntilTraveled:timeout:` method and now want to resume the delivery of events at normal intervals.

Availability
Available in iOS 6.0 and later.

See Also
`– allowDeferredLocationUpdatesUntilTraveled:timeout:` (page 71)

Declared in
`CLLocationManager.h`

dismissHeadingCalibrationDisplay

Dismisses the heading calibration view from the screen immediately.

```
- (void)dismissHeadingCalibrationDisplay
```

Discussion
Core Location uses the heading calibration alert to calibrate the available heading hardware as needed. The display of this view is automatic, assuming your delegate supports displaying the view at all. If the view is displayed, you can use this method to dismiss it after an appropriate amount of time to ensure that your application's user interface is not unduly disrupted.

Availability
Available in iOS 3.0 and later.

Declared in
`CLLocationManager.h`

requestStateForRegion:

Retrieves the state of a region asynchronously.

```
- (void)requestStateForRegion:(CLRegion *)region
```

Parameters

`region`

The region whose state you want to know. This object must be an instance of one of the standard region subclasses provided by Map Kit. You cannot use this method to determine the state of custom regions you define yourself.

Discussion

This method performs the request asynchronously and delivers the results to the location manager's delegate. You must implement the `locationManager:didDetermineState:forRegion:` (page 106) method in the delegate to receive the results.

If the `region` parameter contains an unknown type of region object, this method does nothing.

Availability

Available in iOS 7.0 and later.

Declared in

`CLLocationManager.h`

startMonitoringForRegion:

Starts monitoring the specified region.

```
- (void)startMonitoringForRegion:(CLRegion *)region
```

Parameters

`region`

The region object that defines the boundary to monitor. This parameter must not be `nil`.

Discussion

You must call this method once for each region you want to monitor. If an existing region with the same identifier is already being monitored by the application, the old region is replaced by the new one. The regions you add using this method are shared by all location manager objects in your application and stored in the `monitoredRegions` (page 63) property.

Region events are delivered to the `locationManager:didEnterRegion:` (page 107) and `locationManager:didExitRegion:` (page 107) methods of your delegate. If there is an error, the location manager calls the `locationManager:monitoringDidFailForRegion:withError:` (page 112) method of your delegate instead.

An app can register up to 20 regions at a time. In order to report region changes in a timely manner, the region monitoring service requires network connectivity.

In iOS 6, regions with a radius between 1 and 400 meters work better on iPhone 4S or later devices. (In iOS 5, regions with a radius between 1 and 150 meters work better on iPhone 4S and later devices.) On these devices, an app can expect to receive the appropriate region entered or region exited notification within 3 to 5 minutes on average, if not sooner.

Availability

Available in iOS 5.0 and later.

See Also

– `stopMonitoringForRegion:` (page 80)

Declared in

`CLLocationManager.h`

startMonitoringForRegion:desiredAccuracy:

Starts monitoring the specified region for boundary crossings. (Deprecated in iOS 6.0. Use *`startMonitoringForRegion:`* *(page 74) instead.)*

```
– (void)startMonitoringForRegion:(CLRegion *)region
desiredAccuracy:(CLLocationAccuracy)accuracy
```

Parameters

`region`

The region object that defines the boundary to monitor. This parameter must not be `nil`.

`accuracy`

The distance past the border (measured in meters) at which to generate notifications. You can use this value to prevent the delivery of multiple notifications when the user is close to the border's edge.

Discussion

You must call this method separately for each region you want to monitor. If an existing region with the same identifier is already being monitored by the application, the old region is replaced by the new one. The regions you add using this method are shared by all location manager objects in your application and stored in the `monitoredRegions` (page 63) property.

If you begin monitoring a region and your application is subsequently terminated, the system automatically relaunches it into the background if the region boundary is crossed. In such a case, the options dictionary passed to the `application:didFinishLaunchingWithOptions:` method of your application delegate contains the key `UIApplicationLaunchOptionsLocationKey` to indicate that your application was launched because of a location-related event. In addition, creating a new location manager and assigning a delegate results in the delivery of the corresponding region messages. The newly created location manager's `location` (page 61) property also contains the current location even if location services are not enabled.

Region events are delivered to the `locationManager:didEnterRegion:` (page 107) and `locationManager:didExitRegion:` (page 107) methods of your delegate. If there is an error, the location manager calls the `locationManager:monitoringDidFailForRegion:withError:` (page 112) method of your delegate instead.

Availability
Available in iOS 4.0 and later.

Deprecated in iOS 6.0.

Related Sample Code
Regions

Declared in
`CLLocationManager.h`

startMonitoringSignificantLocationChanges

Starts the generation of updates based on significant location changes.

```
- (void)startMonitoringSignificantLocationChanges
```

Discussion
This method initiates the delivery of location events asynchronously, returning shortly after you call it. Location events are delivered to your delegate's `locationManager:didUpdateLocations:` (page 111) method. The first event to be delivered is usually the most recently cached location event (if any) but may be a newer event in some circumstances. Obtaining a current location fix may take several additional seconds, so be sure to check the timestamps on the location events in your delegate method.

After returning a current location fix, the receiver generates update events only when a significant change in the user's location is detected. For example, it might generate a new event when the device becomes associated with a different cell tower. It does not rely on the value in the `distanceFilter` (page 59) property to generate events. Calling this method several times in succession does not automatically result in new events being generated. Calling `stopMonitoringSignificantLocationChanges` in between, however, does cause a new initial event to be sent the next time you call this method.

If you start this service and your application is subsequently terminated, the system automatically relaunches the application into the background if a new event arrives. In such a case, the options dictionary passed to the `locationManager:didUpdateLocations:` method of your application delegate contains the key `UIApplicationLaunchOptionsLocationKey` to indicate that your application was launched because of a location event. Upon relaunch, you must still configure a location manager object and call this method to continue receiving location events. When you restart location services, the current event is delivered to your delegate immediately. In addition, the `location` (page 61) property of your location manager object is populated with the most recent location object even before you start location services.

In addition to your delegate object implementing the `locationManager:didUpdateLocations:` method, it should also implement the `locationManager:didFailWithError:` (page 108) method to respond to potential errors.

Note: Apps can expect a notification as soon as the device moves 500 meters or more from its previous notification. It should not expect notifications more frequently than once every five minutes. If the device is able to retrieve data from the network, the location manager is much more likely to deliver notifications in a timely manner.

Availability
Available in iOS 4.0 and later.

See Also
– `stopMonitoringSignificantLocationChanges` (page 80)

Declared in
`CLLocationManager.h`

startRangingBeaconsInRegion:

Starts the delivery of notifications for beacons in the specified region.

```
- (void)startRangingBeaconsInRegion:(CLBeaconRegion *)region
```

Parameters
`region`
The region object that defines the identifying information for the targeted beacons. The number of beacons represented by this region object depends on which identifier values you use to initialize it. Beacons must match all of the identifiers you specify. This method copies the region information it needs from the object you provide.

Discussion
Once registered, the location manager reports any encountered beacons to its delegate by calling the `locationManager:didRangeBeacons:inRegion:` (page 109) method. If there is an error registering the specified beacon region, the location manager calls its delegate's `locationManager:rangingBeaconsDidFailForRegion:withError:` (page 113) method and provides the appropriate error information.

Availability
Available in iOS 7.0 and later.

Declared in
`CLLocationManager.h`

startUpdatingHeading

Starts the generation of updates that report the user's current heading.

```
- (void)startUpdatingHeading
```

Discussion
This method returns immediately. Calling this method when the receiver is stopped causes it to obtain an initial heading and notify your delegate. After that, the receiver generates update events when the value in the `headingFilter` property is exceeded.

Before calling this method, you should always check the `headingAvailable` property to see whether heading information is supported on the current device. If heading information is not supported, calling this method has no effect and does not result in the delivery of events to your delegate.

Calling this method several times in succession does not automatically result in new events being generated. Calling `stopUpdatingHeading` in between, however, does cause a new initial event to be sent the next time you call this method.

If you start this service and your application is suspended, the system stops the delivery of events until your application starts running again (either in the foreground or background). If your application is terminated, the delivery of new heading events stops altogether and must be restarted by your code when the application is relaunched.

Heading events are delivered to the `locationManager:didUpdateHeading:` (page 110) method of your delegate. If there is an error, the location manager calls the `locationManager:didFailWithError:` (page 108) method of your delegate instead.

Availability
Available in iOS 3.0 and later.

See Also
`– stopUpdatingHeading` (page 81)
`@property headingAvailable` (page 60)

Declared in
`CLLocationManager.h`

startUpdatingLocation

Starts the generation of updates that report the user's current location.

```
- (void)startUpdatingLocation
```

Discussion

This method returns immediately. Calling this method causes the location manager to obtain an initial location fix (which may take several seconds) and notify your delegate by calling its `locationManager:didUpdateLocations:` (page 111) method. (In iOS 5 and earlier, the location manager calls the `locationManager:didUpdateToLocation:fromLocation:` method instead.) After that, the receiver generates update events primarily when the value in the `distanceFilter` property is exceeded. Updates may be delivered in other situations though. For example, the receiver may send another notification if the hardware gathers a more accurate location reading.

Calling this method several times in succession does not automatically result in new events being generated. Calling `stopUpdatingLocation` in between, however, does cause a new initial event to be sent the next time you call this method.

If you start this service and your application is suspended, the system stops the delivery of events until your application starts running again (either in the foreground or background). If your application is terminated, the delivery of new location events stops altogether. Therefore, if your application needs to receive location events while in the background, it must include the `UIBackgroundModes` key (with the `location` value) in its `Info.plist` file.

In addition to your delegate object implementing the `locationManager:didUpdateLocations:` method, it should also implement the `locationManager:didFailWithError:` (page 108) method to respond to potential errors.

Availability

Available in iOS 2.0 and later.

See Also

`– stopUpdatingLocation` (page 82)
`@property locationServicesEnabled` (page 62)
`@property distanceFilter` (page 59)

Related Sample Code
LocateMe

Declared in
`CLLocationManager.h`

stopMonitoringForRegion:

Stops monitoring the specified region.

```
- (void)stopMonitoringForRegion:(CLRegion *)region
```

Parameters

`region`

The region object currently being monitored. This parameter must not be `nil`.

Discussion

If the specified region object is not currently being monitored, this method has no effect.

Availability

Available in iOS 4.0 and later.

See Also

`– startMonitoringForRegion:` (page 74)

Declared in

`CLLocationManager.h`

stopMonitoringSignificantLocationChanges

Stops the delivery of location events based on significant location changes.

```
- (void)stopMonitoringSignificantLocationChanges
```

Discussion

Use this method to stop the delivery of location events that was started using the `startMonitoringSignificantLocationChanges` method.

Availability

Available in iOS 4.0 and later.

See Also

`– startMonitoringSignificantLocationChanges` (page 76)

Declared in

`CLLocationManager.h`

stopRangingBeaconsInRegion:

Stops the delivery of notifications for the specified beacon region.

```
- (void)stopRangingBeaconsInRegion:(CLBeaconRegion *)region
```

Parameters

`region`

The region that identifies the beacons. The object you specify need not be the exact same object that you registered but the beacon attributes should be the same.

Availability

Available in iOS 7.0 and later.

Declared in

`CLLocationManager.h`

stopUpdatingHeading

Stops the generation of heading updates.

```
- (void)stopUpdatingHeading
```

Discussion

Call this method whenever your code no longer needs to receive heading-related events. Disabling event delivery gives the receiver the option of disabling the appropriate hardware (and thereby saving power) when no clients need location data. You can always restart the generation of heading updates by calling the `startUpdatingHeading` method again.

Availability

Available in iOS 3.0 and later.

See Also

– `startUpdatingHeading` (page 78)

Related Sample Code
Teslameter

Declared in

`CLLocationManager.h`

stopUpdatingLocation

Stops the generation of location updates.

```
- (void)stopUpdatingLocation
```

Discussion
Call this method whenever your code no longer needs to receive location-related events. Disabling event delivery gives the receiver the option of disabling the appropriate hardware (and thereby saving power) when no clients need location data. You can always restart the generation of location updates by calling the `startUpdatingLocation` method again.

Availability
Available in iOS 2.0 and later.

See Also
– `startUpdatingLocation` (page 79)

Related Sample Code
LocateMe

MapSearch

Declared in
`CLLocationManager.h`

Constants

CLAuthorizationStatus

These constants indicate whether the application is authorized to use location services.

```
typedef enum {
   kCLAuthorizationStatusNotDetermined = 0,
   kCLAuthorizationStatusRestricted,
   kCLAuthorizationStatusDenied,
   kCLAuthorizationStatusAuthorized
} CLAuthorizationStatus;
```

Constants

`kCLAuthorizationStatusNotDetermined`

The user has not yet made a choice regarding whether this application can use location services.

Available in iOS 4.2 and later.

Declared in `CLLocationManager.h`.

`kCLAuthorizationStatusRestricted`

This application is not authorized to use location services. The user cannot change this application's status, possibly due to active restrictions such as parental controls being in place.

Available in iOS 4.2 and later.

Declared in `CLLocationManager.h`.

`kCLAuthorizationStatusDenied`

The user explicitly denied the use of location services for this application or location services are currently disabled in Settings.

Available in iOS 4.2 and later.

Declared in `CLLocationManager.h`.

`kCLAuthorizationStatusAuthorized`

This application is authorized to use location services.

Available in iOS 4.2 and later.

Declared in `CLLocationManager.h`.

Deferred Update Constants

Constants for specifying maximum values during deferred updates.

```
extern const CLLocationDistance CLLocationDistanceMax;
extern const NSTimeInterval CLTimeIntervalMax;
```

Constants

`CLLocationDistanceMax`

A value representing an unlimited distance.

Available in iOS 6.0 and later.

Declared in `CLLocation.h`.

`CLTimeIntervalMax`

A value representing an unlimited amount of time.

Available in iOS 6.0 and later.

Declared in `CLLocation.h`.

CLActivityType

These constants indicate the type of activity associated with location updates.

```
enum {
   CLActivityTypeOther = 1,
   CLActivityTypeAutomotiveNavigation,
   CLActivityTypeFitness,
   CLActivityTypeOtherNavigation,
};
typedef NSInteger CLActivityType;
```

Constants

`CLActivityTypeOther`

The location manager is being used for an unknown activity.

Available in iOS 6.0 and later.

Declared in `CLLocationManager.h`.

`CLActivityTypeAutomotiveNavigation`

The location manager is being used specifically during vehicular navigation to track location changes to the automobile. This activity might cause location updates to be paused only when the vehicle does not move for an extended period of time.

Available in iOS 6.0 and later.

Declared in `CLLocationManager.h`.

`CLActivityTypeFitness`

The location manager is being used to track any pedestrian-related activity. This activity might cause location updates to be paused only when the user does not move a significant distance over a period of time.

Available in iOS 6.0 and later.

Declared in `CLLocationManager.h`.

`CLActivityTypeOtherNavigation`

The location manager is being used to track movements for other types of vehicular navigation that are not automobile related. For example, you would use this to track navigation by boat, train, or plane. Do not use this type for pedestrian navigation tracking. This activity might cause location updates to be paused only when the vehicle does not move a significant distance over a period of time.

Available in iOS 6.0 and later.

Declared in `CLLocationManager.h`.

CLPlacemark Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSSecureCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreLocation.framework
Availability	Available in iOS 5.0 and later.
Declared in	CLPlacemark.h
Companion guide	Location and Maps Programming Guide
Related sample code	GeocoderDemo

Overview

A `CLPlacemark` object stores placemark data for a given latitude and longitude. Placemark data includes information such as the country, state, city, and street address associated with the specified coordinate. It can also include points of interest and geographically related data. Placemark objects are typically generated by a `CLGeocoder` object, although you can also create them explicitly yourself.

Tasks

Initializing a Placemark Object

– `initWithPlacemark:` (page 93)
    Initializes and returns a placemark object from another placemark object.

Accessing the Location Data

`location` (page 90) *property*
: The location object containing latitude and longitude information. (read-only)

Accessing the Placemark Attributes

`name` (page 90) *property*
: The name of the placemark. (read-only)

`addressDictionary` (page 87) *property*
: A dictionary containing the Address Book keys and values for the placemark. (read-only)

`ISOcountryCode` (page 89) *property*
: The abbreviated country name. (read-only)

`country` (page 88) *property*
: The name of the country associated with the placemark. (read-only)

`postalCode` (page 91) *property*
: The postal code associated with the placemark. (read-only)

`administrativeArea` (page 88) *property*
: The state or province associated with the placemark. (read-only)

`subAdministrativeArea` (page 92) *property*
: Additional administrative area information for the placemark. (read-only)

`locality` (page 89) *property*
: The city associated with the placemark. (read-only)

`subLocality` (page 92) *property*
: Additional city-level information for the placemark. (read-only)

`thoroughfare` (page 93) *property*
: The street address associated with the placemark. (read-only)

`subThoroughfare` (page 92) *property*
: Additional street-level information for the placemark. (read-only)

`region` (page 91) *property*
: The geographic region associated with the placemark. (read-only)

Accessing Geographic Information

`inlandWater` (page 89) *property*
The name of the inland water body associated with the placemark. (read-only)

`ocean` (page 91) *property*
The name of the ocean associated with the placemark. (read-only)

Accessing Landmark Information

`areasOfInterest` (page 88) *property* Deprecated in iOS 4.0
The relevant areas of interest associated with the placemark. (read-only)

Properties

addressDictionary

A dictionary containing the Address Book keys and values for the placemark. (read-only)

```
@property(nonatomic, readonly) NSDictionary *addressDictionary
```

Discussion
The keys in this dictionary are those defined by the Address Book framework and used to access address information for a person. For a list of the strings that can be in this dictionary, see the "Address Property" constants in *ABPerson Reference*.

You can format the contents of this dictionary to get a full address string as opposed to building the address yourself. To format the dictionary, use the `ABCreateStringWithAddressDictionary` function as described in *Address Book UI Functions Reference*.

Availability
Available in iOS 5.0 and later.

Related Sample Code
GeocoderDemo

Declared in
`CLPlacemark.h`

administrativeArea

The state or province associated with the placemark. (read-only)

```
@property(nonatomic, readonly) NSString *administrativeArea
```

Discussion
The string in this property can be either the spelled out name of the administrative area or its designated abbreviation, if one exists. If the placemark location is Apple's headquarters, for example, the value for this property would be the string "CA" or "California".

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

areasOfInterest

The relevant areas of interest associated with the placemark. (read-only)

```
@property(nonatomic, readonly) NSArray *areasOfInterest
```

Discussion
Examples of an area of interest are the name of a military base or large national park or an attraction such as Eiffel Tower, Disneyland, or Golden Gate Park.

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

country

The name of the country associated with the placemark. (read-only)

```
@property(nonatomic, readonly) NSString *country
```

Discussion
If the placemark location is Apple's headquarters, for example, the value for this property would be the string "United States".

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

inlandWater

The name of the inland water body associated with the placemark. (read-only)

```
@property(nonatomic, readonly) NSString *inlandWater
```

Discussion
For coordinates that lie over an inland body of water, this property contains the name of that water body—the name of a lake, stream, river, or other waterway.

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

ISOcountryCode

The abbreviated country name. (read-only)

```
@property(nonatomic, readonly) NSString *ISOcountryCode
```

Discussion
This string is the standard abbreviation used to refer to the country. For example, if the placemark location is Apple's headquarters, the value for this property would be the string "US".

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

locality

The city associated with the placemark. (read-only)

```
@property(nonatomic, readonly) NSString *locality
```

Discussion
If the placemark location is Apple's headquarters, for example, the value for this property would be the string "Cupertino".

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

location

The location object containing latitude and longitude information. (read-only)

```
@property(nonatomic, readonly) CLLocation *location
```

Discussion
This object is used to initialize the placemark object.

Availability
Available in iOS 5.0 and later.

Related Sample Code
GeocoderDemo

MapSearch

Declared in
`CLPlacemark.h`

name

The name of the placemark. (read-only)

```
@property(nonatomic, readonly) NSString *name
```

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

ocean

The name of the ocean associated with the placemark. (read-only)

```
@property(nonatomic, readonly) NSString *ocean
```

Discussion
For coordinates that lie over an ocean, this property contains the name of the ocean.

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

postalCode

The postal code associated with the placemark. (read-only)

```
@property(nonatomic, readonly) NSString *postalCode
```

Discussion
If the placemark location is Apple's headquarters, for example, the value for this property would be the string "95014".

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

region

The geographic region associated with the placemark. (read-only)

```
@property(nonatomic, readonly) CLRegion *region
```

Availability
Available in iOS 5.0 and later.

Related Sample Code
GeocoderDemo

Declared in
`CLPlacemark.h`

subAdministrativeArea

Additional administrative area information for the placemark. (read-only)

```
@property(nonatomic, readonly) NSString *subAdministrativeArea
```

Discussion
Subadministrative areas typically correspond to counties or other regions that are then organized into a larger administrative area or state. For example, if the placemark location is Apple's headquarters, the value for this property would be the string "Santa Clara", which is the county in California that contains the city of Cupertino.

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

subLocality

Additional city-level information for the placemark. (read-only)

```
@property(nonatomic, readonly) NSString *subLocality
```

Discussion
This property contains additional information, such as the name of the neighborhood or landmark associated with the placemark. It might also refer to a common name that is associated with the location.

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

subThoroughfare

Additional street-level information for the placemark. (read-only)

```
@property(nonatomic, readonly) NSString *subThoroughfare
```

Discussion
Subthroughfares provide information such as the street number for the location. For example, if the placemark location is Apple's headquarters (1 Infinite Loop), the value for this property would be the string "1".

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

thoroughfare

The street address associated with the placemark. (read-only)

```
@property(nonatomic, readonly) NSString *thoroughfare
```

Discussion
The street address contains the street name. For example, if the placemark location is Apple's headquarters, the value for this property would be the string "Infinite Loop".

Availability
Available in iOS 5.0 and later.

Declared in
`CLPlacemark.h`

Instance Methods

initWithPlacemark:

Initializes and returns a placemark object from another placemark object.

```
- (id)initWithPlacemark:(CLPlacemark *)placemark
```

Parameters
`placemark`

The placemark object to use as the source of the data for the new object.

Return Value
A new placemark object.

Discussion
You can use this method to transfer information from one placemark object to another placemark object.

Availability
Available in iOS 5.0 and later.

Declared in

`CLPlacemark.h`

CLRegion Class Reference

Inherits from	NSObject
Conforms to	NSCopying NSSecureCoding NSObject (NSObject)
Framework	/System/Library/Frameworks/CoreLocation.framework
Availability	Available in iOS 4.0 and later.
Declared in	CLLocation.h
Companion guide	Location and Maps Programming Guide
Related sample code	GeocoderDemo Regions

Overview

The `CLRegion` class defines an abstract area that can be tracked. In iOS, you do not create instances of this class directly; instead, you instantiate subclasses that define specific types of regions. In OS X, you create instances of this class and use them to store the region information. After you create a region, you must register it with a `CLLocationManager` object. The location manager generates appropriate events whenever the user crosses the boundaries of the region.

To use this class, create an instance of it and use the `startMonitoringForRegion:` (page 74) method of a `CLLocationManager` object to begin monitoring it.

Tasks

Initializing a Region in OS X

– `initCircularRegionWithCenter:radius:identifier:` (page 99) Deprecated in iOS 6.0 Deprecated in iOS 7.0

Initializes and returns a region object defining a circular area.

Accessing a Region's Attributes

`identifier` (page 97) *property*

The identifier for the region object. (read-only)

`center` (page 96) *property* Deprecated in iOS 7.0

The center point of the region. (read-only)

`radius` (page 98) *property* Deprecated in iOS 4.0 Deprecated in iOS 7.0

The radius (measured in meters) that defines the region's outer boundary. (read-only)

Configuring Notification Delivery

`notifyOnEntry` (page 97) *property*

A Boolean indicating that notifications are generated upon entry into the region.

`notifyOnExit` (page 98) *property*

A Boolean indicating that notifications are generated upon exit from the region.

Hit Testing in a Region

– `containsCoordinate:` (page 99) Deprecated in iOS 7.0

Returns a Boolean value indicating whether the region contains the specified coordinate.

Properties

center

The center point of the region. (read-only) (Deprecated in iOS 7.0.)

```
@property(readonly, nonatomic) CLLocationCoordinate2D center
```

Special Considerations
In iOS, use a `CLCircularRegion` object to manage geographic regions.

Availability
Available in iOS 4.0 and later.

Deprecated in iOS 7.0.

Related Sample Code
Regions

Declared in
`CLRegion.h`

identifier

The identifier for the region object. (read-only)

```
@property(readonly, nonatomic) NSString *identifier
```

Discussion
This is a value that you specify and can use to identify this region inside your application.

Availability
Available in iOS 4.0 and later.

Declared in
`CLRegion.h`

notifyOnEntry

A Boolean indicating that notifications are generated upon entry into the region.

```
@property (nonatomic, assign) BOOL notifyOnEntry
```

Discussion
When this property is `YES`, a device crossing from outside the region to inside the region triggers the delivery of a notification. If the property is `NO`, a notification is not generated. The default value of this property is `YES`.

If the app is not running when a boundary crossing occurs, the system launches the app into the background to handle it. Upon launch, your app must configure new location manager and delegate objects to receive the notification. The notification is sent to your delegate's `locationManager:didEnterRegion:` (page 107) method.

Availability
Available in iOS 7.0 and later.

Declared in
`CLRegion.h`

notifyOnExit

A Boolean indicating that notifications are generated upon exit from the region.

```
@property (nonatomic, assign) BOOL notifyOnExit
```

Discussion
When this property is `YES`, a device crossing from inside the region to outside the region triggers the delivery of a notification. If the property is `NO`, a notification is not generated. The default value of this property is `YES`.

If the app is not running when a boundary crossing occurs, the system launches the app into the background to handle it. Upon launch, your app must configure new location manager and delegate objects to receive the notification. The notification is sent to your delegate's `locationManager:didExitRegion:` (page 107) method.

Availability
Available in iOS 7.0 and later.

Declared in
`CLRegion.h`

radius

The radius (measured in meters) that defines the region's outer boundary. (read-only) (Deprecated in iOS 7.0.)

```
@property(readonly, nonatomic) CLLocationDistance radius
```

Special Considerations
In iOS, use a `CLCircularRegion` object to manage geographic regions.

Availability
Available in iOS 4.0 and later.

Deprecated in iOS 7.0.

Related Sample Code
Regions

Declared in
`CLRegion.h`

Instance Methods

containsCoordinate:

Returns a Boolean value indicating whether the region contains the specified coordinate. (Deprecated in iOS 7.0.)

```
- (BOOL)containsCoordinate:(CLLocationCoordinate2D)coordinate
```

Parameters

`coordinate`
    The coordinate to test against the region.

Return Value
`YES` if the coordinate lies within the region's boundaries or `NO` if it does not.

Special Considerations
In iOS, use a `CLCircularRegion` object to manage geographic regions.

Availability
Available in iOS 4.0 and later.

Deprecated in iOS 7.0.

Declared in
`CLRegion.h`

initCircularRegionWithCenter:radius:identifier:

Initializes and returns a region object defining a circular area. (Deprecated in iOS 7.0.)

```
- (id)initCircularRegionWithCenter:(CLLocationCoordinate2D)center
radius:(CLLocationDistance)radius identifier:(NSString *)identifier
```

Parameters

`center`
    The center point of the region.

`radius`
    The distance (measured in meters) from the center point that marks the boundary of the region.

`identifier`
: A unique identifier to associate with the region object. You use this identifier to differentiate regions within your application. This value must not be `nil`.

Return Value

An initialized region object.

Special Considerations

In iOS, use a `CLCircularRegion` object to manage geographic regions.

Availability

Available in iOS 4.0 and later.

Deprecated in iOS 7.0.

Related Sample Code
GeocoderDemo

Regions

Declared in

`CLRegion.h`

Constants

CLRegionState

Constants that reflect the relationship of the current location to the region boundaries.

```
typedef {
   CLRegionStateUnknown,
   CLRegionStateInside,
   CLRegionStateOutside
} CLRegionState;
```

Constants

`CLRegionStateUnknown`
: It is unknown whether the location is inside or outside of the region.

  Available in iOS 7.0 and later.

  Declared in `CLRegion.h`.

`CLRegionStateInside`

The location is inside the given region.

Available in iOS 7.0 and later.

Declared in `CLRegion.h`.

`CLRegionStateOutside`

The location is outside of the given region.

Available in iOS 7.0 and later.

Declared in `CLRegion.h`.

Protocols

CLLocationManagerDelegate Protocol Reference

Conforms to	NSObject
Framework	/System/Library/Frameworks/CoreLocation.framework
Availability	Available in iOS 2.0 and later.
Declared in	CLLocationManagerDelegate.h
Companion guide	Location and Maps Programming Guide
Related sample code	GeocoderDemo LocateMe Regions Simple Core Data Relationships Teslameter

Overview

The `CLLocationManagerDelegate` protocol defines the methods used to receive location and heading updates from a `CLLocationManager` object.

Upon receiving a successful location or heading update, you can use the result to update your user interface or perform other actions. If the location or heading could not be determined, you might want to stop updates for a short period of time and try again later. You can use the `stopUpdatingLocation` (page 82), `stopMonitoringSignificantLocationChanges` (page 80), `stopUpdatingHeading` (page 81), or `stopMonitoringForRegion:` (page 80) methods of `CLLocationManager` to stop location, heading, and region updates.

The methods of your delegate object are called from the thread in which you started the corresponding location services. That thread must itself have an active run loop, like the one found in your application's main thread.

Tasks

Responding to Location Events

– `locationManager:didUpdateLocations:` (page 111)
  Tells the delegate that new location data is available.

– `locationManager:didFailWithError:` (page 108)
  Tells the delegate that the location manager was unable to retrieve a location value.

– `locationManager:didFinishDeferredUpdatesWithError:` (page 109)
  Tells the delegate that updates will no longer be deferred.

– `locationManager:didUpdateToLocation:fromLocation:` (page 112)
  Tells the delegate that a new location value is available. (Deprecated. Use `locationManager:didUpdateLocations:` (page 111) instead.)

Pausing Location Updates

– `locationManagerDidPauseLocationUpdates:` (page 114)
  Tells the delegate that location updates were paused. (required)

– `locationManagerDidResumeLocationUpdates:` (page 114)
  Tells the delegate that the delivery of location updates has resumed. (required)

Responding to Heading Events

– `locationManager:didUpdateHeading:` (page 110)
  Tells the delegate that the location manager received updated heading information.

– `locationManagerShouldDisplayHeadingCalibration:` (page 115)
  Asks the delegate whether the heading calibration alert should be displayed.

Responding to Region Events

– `locationManager:didEnterRegion:` (page 107)
  Tells the delegate that the user entered the specified region.

– `locationManager:didExitRegion:` (page 107)
  Tells the delegate that the user left the specified region.

– `locationManager:didDetermineState:forRegion:` (page 106)
  Tells the delegate about the state of the specified region. (required)

– `locationManager:monitoringDidFailForRegion:withError:` (page 112)
  Tells the delegate that a region monitoring error occurred.

– `locationManager:didStartMonitoringForRegion:` (page 110)
  Tells the delegate that a new region is being monitored.

Responding to Ranging Events

– `locationManager:rangingBeaconsDidFailForRegion:withError:` (page 113)
  Tells the delegate that an error occurred while gathering ranging information for a set of beacons. (required)

– `locationManager:didRangeBeacons:inRegion:` (page 109) Deprecated in iOS 3.2
  Tells the delegate that one or more beacons are in range. (required)

Responding to Authorization Changes

– `locationManager:didChangeAuthorizationStatus:` (page 105)
  Tells the delegate that the authorization status for the application changed.

Instance Methods

locationManager:didChangeAuthorizationStatus:

Tells the delegate that the authorization status for the application changed.

```
- (void)locationManager:(CLLocationManager *)manager
didChangeAuthorizationStatus:(CLAuthorizationStatus)status
```

Parameters

`manager`
  The location manager object reporting the event.

`status`
  The new authorization status for the application.

Discussion

This method is called whenever the application's ability to use location services changes. Changes can occur because the user allowed or denied the use of location services for your application or for the system as a whole.

Availability

Available in iOS 4.2 and later.

Declared in

`CLLocationManagerDelegate.h`

locationManager:didDetermineState:forRegion:

Tells the delegate about the state of the specified region. (required)

```
- (void)locationManager:(CLLocationManager *)manager
didDetermineState:(CLRegionState)state forRegion:(CLRegion *)region
```

Parameters

`manager`
  The location manager object reporting the event.

`state`
  The state of the specified region. For a list of possible values, see the `CLRegionState` (page 100) type.

`region`
  The region whose state was determined.

Discussion

The location manager calls this method whenever there is a boundary transition for a region. It calls this method in addition to calling the `locationManager:didEnterRegion:` and `locationManager:didExitRegion:` methods. The location manager also calls this method in response to a call to its `requestStateForRegion:` (page 73) method, which runs asynchronously.

Availability

Available in iOS 7.0 and later.

See Also

– `locationManager:didEnterRegion:` (page 107)
– `locationManager:didExitRegion:` (page 107)

Declared in

`CLLocationManagerDelegate.h`

locationManager:didEnterRegion:

Tells the delegate that the user entered the specified region.

```
- (void)locationManager:(CLLocationManager *)manager didEnterRegion:(CLRegion *)region
```

Parameters

`manager`
: The location manager object reporting the event.

`region`
: An object containing information about the region that was entered.

Discussion

Because regions are a shared application resource, every active location manager object delivers this message to its associated delegate. It does not matter which location manager actually registered the specified region. And if multiple location managers share a delegate object, that delegate receives the message multiple times.

The region object provided may not be the same one that was registered. As a result, you should never perform pointer-level comparisons to determine equality. Instead, use the region's identifier string to determine if your delegate should respond.

Availability

Available in iOS 4.0 and later.

Declared in

`CLLocationManagerDelegate.h`

locationManager:didExitRegion:

Tells the delegate that the user left the specified region.

```
- (void)locationManager:(CLLocationManager *)manager didExitRegion:(CLRegion *)region
```

Parameters

`manager`
: The location manager object reporting the event.

`region`
: An object containing information about the region that was exited.

Discussion

Because regions are a shared application resource, every active location manager object delivers this message to its associated delegate. It does not matter which location manager actually registered the specified region. And if multiple location managers share a delegate object, that delegate receives the message multiple times.

The region object provided may not be the same one that was registered. As a result, you should never perform pointer-level comparisons to determine equality. Instead, use the region's identifier string to determine if your delegate should respond.

Availability
Available in iOS 4.0 and later.

Declared in
`CLLocationManagerDelegate.h`

locationManager:didFailWithError:

Tells the delegate that the location manager was unable to retrieve a location value.

```
- (void)locationManager:(CLLocationManager *)manager didFailWithError:(NSError *)error
```

Parameters

`manager`
The location manager object that was unable to retrieve the location.

`error`
The error object containing the reason the location or heading could not be retrieved.

Discussion
Implementation of this method is optional but recommended.

The location manager calls this method when it encounters an error trying to get the location or heading data. If the location service is unable to retrieve a location right away, it reports a `kCLErrorLocationUnknown` (page 127) error and keeps trying. In such a situation, you can simply ignore the error and wait for a new event. If a heading could not be determined because of strong interference from nearby magnetic fields, this method returns `kCLErrorHeadingFailure` (page 127).

If the user denies your application's use of the location service, this method reports a `kCLErrorDenied` (page 127) error. Upon receiving such an error, you should stop the location service.

Availability
Available in iOS 2.0 and later.

Declared in
`CLLocationManagerDelegate.h`

locationManager:didFinishDeferredUpdatesWithError:

Tells the delegate that updates will no longer be deferred.

```
- (void)locationManager:(CLLocationManager *)manager
didFinishDeferredUpdatesWithError:(NSError *)error
```

Parameters

`manager`

The location manager object that generated the update event.

`error`

The error object containing the reason deferred location updates could not be delivered.

Discussion

The location manager object calls this method to let you know that it has stopped deferring the delivery of location events. The manager may call this method for any number of reasons. For example, it calls it when you stop location updates altogether, when you ask the location manager to disallow deferred updates, or when a condition for deferring updates (such as exceeding a timeout or distance parameter) is met.

Availability

Available in iOS 6.0 and later.

See Also

`disallowDeferredLocationUpdates` (page 73)
`allowDeferredLocationUpdatesUntilTraveled:timeout:` (page 71)

Declared in

`CLLocationManagerDelegate.h`

locationManager:didRangeBeacons:inRegion:

Tells the delegate that one or more beacons are in range. (required)

```
- (void)locationManager:(CLLocationManager *)manager didRangeBeacons:(NSArray
*)beacons inRegion:(CLBeaconRegion *)region
```

Parameters

`manager`

The location manager object reporting the event.

`beacons`

An array of `CLBeacon` objects representing the beacons currently in range. You can use the information in these objects to determine the range of each beacon and its identifying information.

`region`
    The region object containing the parameters that were used to locate the beacons.

Discussion
The location manager calls this method whenever a beacon comes within range or goes out of range. The location manager also calls this method when the range of the beacon changes; for example, when the beacon gets closer.

Availability
Available in iOS 7.0 and later.

Declared in
`CLLocationManagerDelegate.h`

locationManager:didStartMonitoringForRegion:

Tells the delegate that a new region is being monitored.

```
- (void)locationManager:(CLLocationManager *)manager
didStartMonitoringForRegion:(CLRegion *)region
```

Parameters
`manager`
    The location manager object reporting the event.

`region`
    The region that is being monitored.

Availability
Available in iOS 5.0 and later.

Declared in
`CLLocationManagerDelegate.h`

locationManager:didUpdateHeading:

Tells the delegate that the location manager received updated heading information.

```
- (void)locationManager:(CLLocationManager *)manager didUpdateHeading:(CLHeading
*)newHeading
```

Parameters

`manager`
: The location manager object that generated the update event.

`newHeading`
: The new heading data.

Discussion

Implementation of this method is optional but expected if you start heading updates using the `startUpdatingHeading` (page 78) method.

The location manager object calls this method after you initially start the heading service. Subsequent events are delivered when the previously reported value changes by more than the value specified in the `headingFilter` (page 60) property of the location manager object.

Availability

Available in iOS 3.0 and later.

Declared in

`CLLocationManagerDelegate.h`

locationManager:didUpdateLocations:

Tells the delegate that new location data is available.

```
- (void)locationManager:(CLLocationManager *)manager
        didUpdateLocations:(NSArray *)locations
```

Parameters

`manager`
: The location manager object that generated the update event.

`locations`
: An array of `CLLocation` objects containing the location data. This array always contains at least one object representing the current location. If updates were deferred or if multiple locations arrived before they could be delivered, the array may contain additional entries. The objects in the array are organized in the order in which they occurred. Therefore, the most recent location update is at the end of the array.

Discussion

Implementation of this method is optional but recommended.

Availability

Available in iOS 6.0 and later.

Declared in
`CLLocationManagerDelegate.h`

locationManager:didUpdateToLocation:fromLocation:

Tells the delegate that a new location value is available. (Deprecated in iOS 6.0. Use `locationManager:didUpdateLocations:` (page 111) instead.)

```
- (void)locationManager:(CLLocationManager *)manager didUpdateToLocation:(CLLocation
 *)newLocation fromLocation:(CLLocation *)oldLocation
```

Parameters

`manager`
    The location manager object that generated the update event.

`newLocation`
    The new location data.

`oldLocation`
    The location data from the previous update. If this is the first update event delivered by this location manager, this parameter is `nil`.

Discussion
By the time this message is delivered to your delegate, the new location data is also available directly from the `CLLocationManager` object. The `newLocation` parameter may contain the data that was cached from a previous usage of the location service. You can use the `timestamp` (page 43) property of the location object to determine how recent the location data is.

Availability
Available in iOS 2.0 and later.

Deprecated in iOS 6.0.

Declared in
`CLLocationManagerDelegate.h`

locationManager:monitoringDidFailForRegion:withError:

Tells the delegate that a region monitoring error occurred.

```
- (void)locationManager:(CLLocationManager *)manager
monitoringDidFailForRegion:(CLRegion *)region withError:(NSError *)error
```

Parameters

`manager`
: The location manager object reporting the event.

`region`
: The region for which the error occurred.

`error`
: An error object containing the error code that indicates why region monitoring failed.

Discussion

If an error occurs while trying to monitor a given region, the location manager sends this message to its delegate. Region monitoring might fail because the region itself cannot be monitored or because there was a more general failure in configuring the region monitoring service.

Although implementation of this method is optional, it is recommended that you implement it if you use region monitoring in your application.

Availability

Available in iOS 4.0 and later.

Declared in

`CLLocationManagerDelegate.h`

locationManager:rangingBeaconsDidFailForRegion:withError:

Tells the delegate that an error occurred while gathering ranging information for a set of beacons. (required)

```
- (void)locationManager:(CLLocationManager *)manager
rangingBeaconsDidFailForRegion:(CLBeaconRegion *)region withError:(NSError *)error
```

Parameters

`manager`
: The location manager object reporting the event.

`region`
: The region object that encountered the error.

`error`
: An error object containing the error code that indicates why ranging failed.

Discussion

Errors occur most often when registering a beacon region failed. If the region object itself is invalid or if it contains invalid data, the location manager calls this method to report the problem.

Availability
Available in iOS 7.0 and later.

Declared in
`CLLocationManagerDelegate.h`

locationManagerDidPauseLocationUpdates:

Tells the delegate that location updates were paused. (required)

```
- (void)locationManagerDidPauseLocationUpdates:(CLLocationManager *)manager
```

Parameters
`manager`
The location manager object that paused the delivery of events.

Discussion
When the location manager detects that the device's location is not changing, it can pause the delivery of updates in order to shut down the appropriate hardware and save power. When it does this, it calls this method to let your app know that this has happened.

Availability
Available in iOS 6.0 and later.

See Also
`– locationManagerDidResumeLocationUpdates:` (page 114)

Declared in
`CLLocationManagerDelegate.h`

locationManagerDidResumeLocationUpdates:

Tells the delegate that the delivery of location updates has resumed. (required)

```
- (void)locationManagerDidResumeLocationUpdates:(CLLocationManager *)manager
```

Parameters
`manager`
The location manager that resumed the delivery of events.

Discussion

When location updates are paused and need to be resumed (perhaps because the user is moving again), the location manager calls this method to let your app know that it is about to begin the delivery of those updates again.

Availability

Available in iOS 6.0 and later.

See Also

– `locationManagerDidPauseLocationUpdates:` (page 114)

Declared in

`CLLocationManagerDelegate.h`

locationManagerShouldDisplayHeadingCalibration:

Asks the delegate whether the heading calibration alert should be displayed.

```
- (BOOL)locationManagerShouldDisplayHeadingCalibration:(CLLocationManager *)manager
```

Parameters

`manager`

The location manager object coordinating the display of the heading calibration alert.

Return Value

`YES` if you want to allow the heading calibration alert to be displayed; `NO` if you do not.

Discussion

Core Location may call this method in an effort to calibrate the onboard hardware used to determine heading values. Typically, Core Location calls this method at the following times:

- The first time heading updates are ever requested
- When Core Location observes a significant change in magnitude or inclination of the observed magnetic field

If you return `YES` from this method, Core Location displays the heading calibration alert on top of the current window immediately. The calibration alert prompts the user to move the device in a particular pattern so that Core Location can distinguish between the Earth's magnetic field and any local magnetic fields. The alert remains visible until calibration is complete or until you explicitly dismiss it by calling the `dismissHeadingCalibrationDisplay` (page 73) method. In the latter case, you can use this method to set up a timer and dismiss the interface after a specified amount of time has elapsed.

> **Note:** The calibration process is able to filter out only those magnetic fields that move with the device. To calibrate a device that is near other sources of magnetic interference, the user must either move the device away from the source or move the source in conjunction with the device during the calibration process.

If you return `NO` from this method or do not provide an implementation for it in your delegate, Core Location does not display the heading calibration alert. Even if the alert is not displayed, calibration can still occur naturally when any interfering magnetic fields move away from the device. However, if the device is unable to calibrate itself for any reason, the value in the `headingAccuracy` (page 34) property of any subsequent events will reflect the uncalibrated readings.

Availability
Available in iOS 3.0 and later.

Declared in
`CLLocationManagerDelegate.h`

Functions

Core Location Functions Reference

Declared in	CLLocation.h
Companion guide	Location and Maps Programming Guide

Overview

The Core Location framework provides functions to help you work with coordinate values.

Functions

CLLocationCoordinate2DIsValid

Returns a Boolean indicating whether the specified coordinate is valid.

```
BOOL CLLocationCoordinate2DIsValid(
   CLLocationCoordinate2D coord)
```

Parameters

`coord`

A coordinate containing latitude and longitude values.

Return Value

`YES` if the coordinate is valid or `NO` if it is not.

Discussion

A coordinate is considered invalid if it meets at least one of the following criteria:

- Its latitude is greater than 90 degrees or less than -90 degrees.
- Its longitude is greater than 180 degrees or less than -180 degrees.

Availability

Available in iOS 4.0 and later.

Related Sample Code
GeocoderDemo

KMLViewer

PhotosByLocation

Declared in
`CLLocation.h`

CLLocationCoordinate2DMake

Formats a latitude and longitude value into a coordinate data structure format.

```
CLLocationCoordinate2D CLLocationCoordinate2DMake(
   CLLocationDegrees latitude,
   CLLocationDegrees longitude)
```

Parameters

`latitude`

The latitude for the new coordinate.

`longitude`

The longitude for the new coordinate.

Return Value

A coordinate structure encompassing the latitude and longitude values.

Availability

Available in iOS 4.0 and later.

Related Sample Code
KMLViewer

Regions

Declared in
`CLLocation.h`

Data Types

Core Location Data Types Reference

Framework	CoreLocation/CoreLocation.h

Overview

This document describes the data types found in the Core Location framework.

Data Types

CLLocationAccuracy

Represents the accuracy of a coordinate value in meters.

```
typedef double CLLocationAccuracy;
```

Availability
Available in iOS 2.0 and later.

Declared in
`CLLocation.h`

CLLocationCoordinate2D

A structure that contains a geographical coordinate using the WGS 84 reference frame.

```
typedef struct {
   CLLocationDegrees latitude;
   CLLocationDegrees longitude;
} CLLocationCoordinate2D;
```

Fields

`latitude`
The latitude in degrees. Positive values indicate latitudes north of the equator. Negative values indicate latitudes south of the equator.

`longitude`
The longitude in degrees. Measurements are relative to the zero meridian, with positive values extending east of the meridian and negative values extending west of the meridian.

Availability
Available in iOS 2.0 and later.

Declared in
`CLLocation.h`

CLLocationDegrees

Represents a latitude or longitude value specified in degrees.

```
typedef double CLLocationDegrees;
```

Availability
Available in iOS 2.0 and later.

Declared in
`CLLocation.h`

CLLocationDirection

Represents a direction that is measured in degrees relative to true north.

```
typedef double CLLocationDirection;
```

Discussion
Direction values are measured in degrees starting at due north and continue clockwise around the compass. Thus, north is 0 degrees, east is 90 degrees, south is 180 degrees, and so on. A negative value indicates an invalid direction.

Availability
Available in iOS 2.2 and later.

Declared in
`CLLocation.h`

CLLocationDistance

A distance measurement (in meters) from an existing location.

```
typedef double CLLocationDistance;
```

Availability
Available in iOS 2.0 and later.

Declared in
`CLLocation.h`

CLLocationSpeed

Represents the speed at which the device is moving in meters per second.

```
typedef double CLLocationSpeed;
```

Availability
Available in iOS 2.2 and later.

Declared in
`CLLocation.h`

Constants

Core Location Constants Reference

Framework	CoreLocation/CoreLocation.h

Overview

This document describes the constants found in the Core Location framework.

Constants

CLDeviceOrientation

The physical orientation of the device.

```
typedef enum {
   CLDeviceOrientationUnknown = 0,
   CLDeviceOrientationPortrait,
   CLDeviceOrientationPortraitUpsideDown,
   CLDeviceOrientationLandscapeLeft,
   CLDeviceOrientationLandscapeRight,
   CLDeviceOrientationFaceUp,
   CLDeviceOrientationFaceDown
} CLDeviceOrientation;
```

Constants

`CLDeviceOrientationUnknown`

The orientation is currently not known.

Available in iOS 4.0 and later.

Declared in `CLLocationManager.h`.

`CLDeviceOrientationPortrait`

The device is in portrait mode, with the device held upright and the home button at the bottom.

Available in iOS 4.0 and later.

Declared in `CLLocationManager.h`.

`CLDeviceOrientationPortraitUpsideDown`
: The device is in portrait mode but upside down, with the device held upright and the home button at the top.

  Available in iOS 4.0 and later.

  Declared in `CLLocationManager.h`.

`CLDeviceOrientationLandscapeLeft`
: The device is in landscape mode, with the device held upright and the home button on the right side.

  Available in iOS 4.0 and later.

  Declared in `CLLocationManager.h`.

`CLDeviceOrientationLandscapeRight`
: The device is in landscape mode, with the device held upright and the home button on the left side.

  Available in iOS 4.0 and later.

  Declared in `CLLocationManager.h`.

`CLDeviceOrientationFaceUp`
: The device is held parallel to the ground with the screen facing upwards.

  Available in iOS 4.0 and later.

  Declared in `CLLocationManager.h`.

`CLDeviceOrientationFaceDown`
: The device is held parallel to the ground with the screen facing downwards.

  Available in iOS 4.0 and later.

  Declared in `CLLocationManager.h`.

kCLErrorDomain

The domain for Core Location errors.

```
extern NSString *const kCLErrorDomain;
```

Constants

`kCLErrorDomain`
: The domain for Core Location errors. This value is used in the `NSError` class.

  Available in iOS 2.0 and later.

  Declared in `CLErrorDomain.h`.

CLError

Error codes returned by the location manager object.

```
typedef enum {
   kCLErrorLocationUnknown  = 0,
   kCLErrorDenied,
   kCLErrorNetwork,
   kCLErrorHeadingFailure,
   kCLErrorRegionMonitoringDenied,
   kCLErrorRegionMonitoringFailure,
   kCLErrorRegionMonitoringSetupDelayed,
   kCLErrorRegionMonitoringResponseDelayed,
   kCLErrorGeocodeFoundNoResult,
   kCLErrorGeocodeFoundPartialResult,
   kCLErrorGeocodeCanceled,
   kCLErrorDeferredFailed,
   kCLErrorDeferredNotUpdatingLocation,
   kCLErrorDeferredAccuracyTooLow,
   kCLErrorDeferredDistanceFiltered,
   kCLErrorDeferredCanceled,
   kCLErrorRangingUnavailable,
   kCLErrorRangingFailure,
} CLError;
```

Constants

`kCLErrorLocationUnknown`

The location manager was unable to obtain a location value right now.

Available in iOS 2.0 and later.

Declared in `CLError.h`.

`kCLErrorDenied`

Access to the location service was denied by the user.

Available in iOS 2.0 and later.

Declared in `CLError.h`.

`kCLErrorNetwork`

The network was unavailable or a network error occurred.

Available in iOS 3.0 and later.

Declared in `CLError.h`.

`kCLErrorHeadingFailure`

The heading could not be determined.

Available in iOS 3.0 and later.

Declared in `CLError.h`.

`kCLErrorRegionMonitoringDenied`

Access to the region monitoring service was denied by the user.

Available in iOS 4.0 and later.

Declared in `CLError.h`.

`kCLErrorRegionMonitoringFailure`

A registered region cannot be monitored. Monitoring can fail if the app has exceeded the maximum number of regions that it can monitor simultaneously. Monitoring can also fail if the region's radius distance is too large.

Available in iOS 4.0 and later.

Declared in `CLError.h`.

`kCLErrorRegionMonitoringSetupDelayed`

Core Location could not initialize the region monitoring feature immediately.

Available in iOS 4.0 and later.

Declared in `CLError.h`.

`kCLErrorRegionMonitoringResponseDelayed`

Core Location will deliver events but they may be delayed. Possible keys in the user information dictionary are described in "Error User Info Keys" (page 130).

Available in iOS 5.0 and later.

Declared in `CLError.h`.

`kCLErrorGeocodeFoundNoResult`

The geocode request yielded no result.

Available in iOS 5.0 and later.

Declared in `CLError.h`.

`kCLErrorGeocodeFoundPartialResult`

The geocode request yielded a partial result.

Available in iOS 5.0 and later.

Declared in `CLError.h`.

`kCLErrorGeocodeCanceled`

The geocode request was canceled.

Available in iOS 5.0 and later.

Declared in `CLError.h`.

`kCLErrorDeferredFailed`

The location manager did not enter deferred mode for an unknown reason. This error can occur if GPS is unavailable, not active, or is temporarily interrupted. If you get this error on a device that has GPS hardware, the solution is to try again.

Available in iOS 6.0 and later.

Declared in `CLError.h`.

`kCLErrorDeferredNotUpdatingLocation`

The location manager did not enter deferred mode because location updates were already disabled or paused.

Available in iOS 6.0 and later.

Declared in `CLError.h`.

`kCLErrorDeferredAccuracyTooLow`

Deferred mode is not supported for the requested accuracy. The accuracy must be set to `kCLLocationAccuracyBest` (page 131) or `kCLLocationAccuracyBestForNavigation` (page 130).

Available in iOS 6.0 and later.

Declared in `CLError.h`.

`kCLErrorDeferredDistanceFiltered`

Deferred mode does not support distance filters. Set the distance filter to `kCLDistanceFilterNone` (page 132).

Available in iOS 6.0 and later.

Declared in `CLError.h`.

`kCLErrorDeferredCanceled`

The request for deferred updates was canceled by your app or by the location manager. This error is returned if you call the `disallowDeferredLocationUpdates` (page 73) method or schedule a new deferred update before the previous deferred update request is processed. The location manager may also report this error too. For example, if the app is in the foreground when a new location is determined, the location manager cancels deferred updates and delivers the location data to your app.

Available in iOS 6.0 and later.

Declared in `CLError.h`.

`kCLErrorRangingUnavailable`

Ranging is disabled. This might happen if the device is in Airplane mode or if Bluetooth or location services are disabled.

Available in iOS 7.0 and later.

Declared in `CLError.h`.

`kCLErrorRangingFailure`

A general ranging error occurred.

Available in iOS 7.0 and later.

Declared in `CLError.h`.

Discussion

Errors are delivered to the delegate using an `NSError` object.

Error User Info Keys

Keys used in error user information dictionaries.

```
extern NSString *const kCLErrorUserInfoAlternateRegionKey;
```

Constants

`kCLErrorUserInfoAlternateRegionKey`

A key in the user information dictionary of an `kCLErrorRegionMonitoringResponseDelayed` (page 128) error whose value is a `CLRegion` object that the location services can more effectively monitor.

Available in iOS 5.0 and later.

Declared in `CLError.h`.

Accuracy Constants

Constant values you can use to specify the accuracy of a location.

```
extern const CLLocationAccuracy kCLLocationAccuracyBestForNavigation;
extern const CLLocationAccuracy kCLLocationAccuracyBest;
extern const CLLocationAccuracy kCLLocationAccuracyNearestTenMeters;
extern const CLLocationAccuracy kCLLocationAccuracyHundredMeters;
extern const CLLocationAccuracy kCLLocationAccuracyKilometer;
extern const CLLocationAccuracy kCLLocationAccuracyThreeKilometers;
```

Constants

`kCLLocationAccuracyBestForNavigation`

Use the highest possible accuracy and combine it with additional sensor data. This level of accuracy is intended for use in navigation applications that require precise position information at all times and are intended to be used only while the device is plugged in.

Available in iOS 4.0 and later.

Declared in `CLLocation.h`.

`kCLLocationAccuracyBest`

Use the highest-level of accuracy.

Available in iOS 2.0 and later.

Declared in `CLLocation.h`.

`kCLLocationAccuracyNearestTenMeters`

Accurate to within ten meters of the desired target.

Available in iOS 2.0 and later.

Declared in `CLLocation.h`.

`kCLLocationAccuracyHundredMeters`

Accurate to within one hundred meters.

Available in iOS 2.0 and later.

Declared in `CLLocation.h`.

`kCLLocationAccuracyKilometer`

Accurate to the nearest kilometer.

Available in iOS 2.0 and later.

Declared in `CLLocation.h`.

`kCLLocationAccuracyThreeKilometers`

Accurate to the nearest three kilometers.

Available in iOS 2.0 and later.

Declared in `CLLocation.h`.

Specifying an Invalid Coordinate

Use this constant whenever you want to indicate that a coordinate is invalid.

```
const CLLocationCoordinate2D kCLLocationCoordinate2DInvalid
```

Constants

`kCLLocationCoordinate2DInvalid`

An invalid coordinate value.

Available in iOS 4.0 and later.

Declared in `CLLocation.h`.

Distance Filter Value

This constant indicates the minimum distance required before an event is generated.

```
extern const CLLocationDistance kCLDistanceFilterNone;
```

Constants

`kCLDistanceFilterNone`

All movements are reported.

Available in iOS 2.0 and later.

Declared in `CLLocation.h`.

Heading Filter Value

This constant indicates the minimum heading change before an event is generated.

```
const CLLocationDegrees kCLHeadingFilterNone;
```

Constants

`kCLHeadingFilterNone`

All heading changes are reported.

Available in iOS 3.0 and later.

Declared in `CLHeading.h`.

Document Revision History

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

Date	Notes
2013-09-18	Added classes introduced in iOS 7.
2011-10-12	Updated for iOS 5.0.
2010-11-04	Added the geocoding classes plus new constants and data types documents.
2010-05-11	Added classes and functions introduced in iOS 4.0.
2009-07-28	Updated the document to reflect the availability of the interfaces in OS X v10.6.
2009-05-07	Added the CLHeading class.
2008-03-12	New document that describes the classes and protocols for configuring and scheduling the delivery of location-related events.