7-Core Location and Map Kit

Applicazioni basate sulla posizione

Le informazioni sulla posizione in iOS includono:

- Servizi di localizzazione, forniti dal framework "Core Location", includono
  - API Objuective-C
  - Informazioni relative alla posizione e alla direzione dell'utente
- Mappe, fornite dal framework "MapKit", include:
  - Supporto per la visualizzazione e l'annotazione delle mappe

---

Core Location

Servizi di localizzazione

Le applicazioni iOS sono destinate ad essere eseguiti su dispositivi mobili, quindi i servizi di locazione:

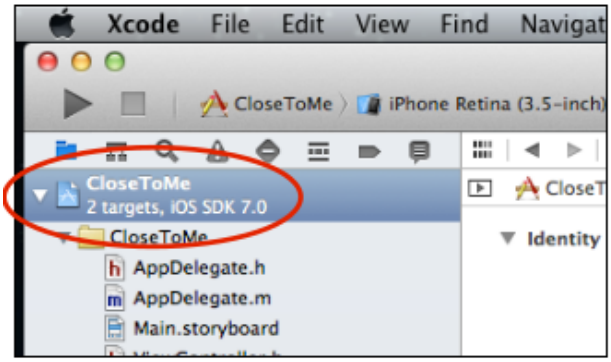
- Forniscono il supporto per la mobilità
- Monitorano la posizione dell'utente e generare aggiornamenti

Il "Core location" framework fornisce il supporto per i servizi di localizzazione e deve essere collegato al progetto così da permettere l'accesso a tutte le interfacce che fornisce.

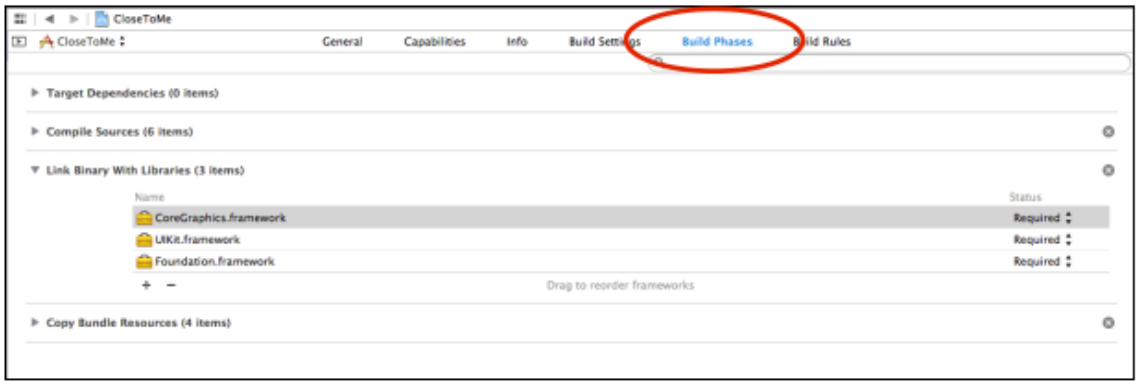
> Nota
> In qualunque momento che una classe o un protocollo definito nel framework è utilizzato deve essere importato all'interno del file con la direttiva `#import <CoreLocation/CoreLocation.h>`

Collegare il framework al progetto

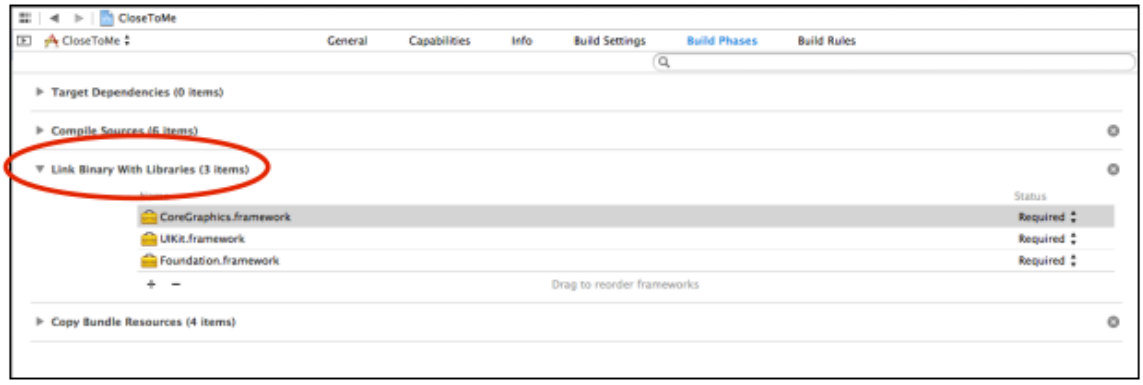
Seleziona il progetto



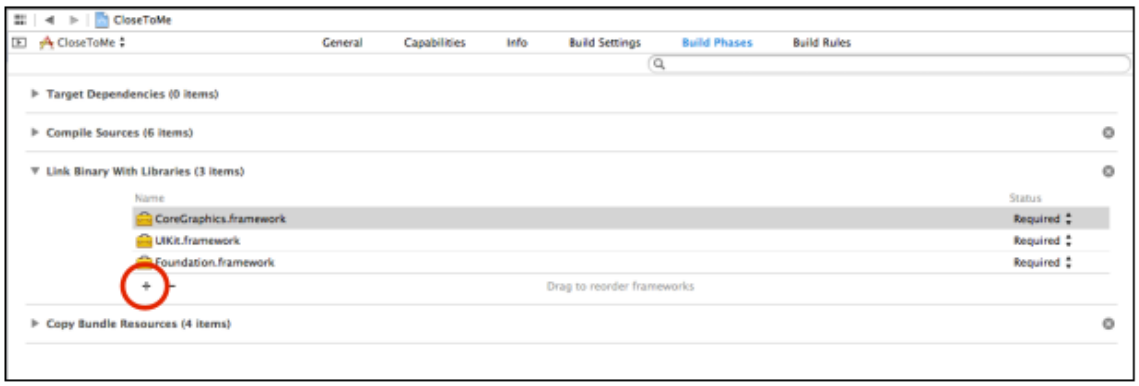
Seleziona la fase di costruzione



Collega i binari con le librerie

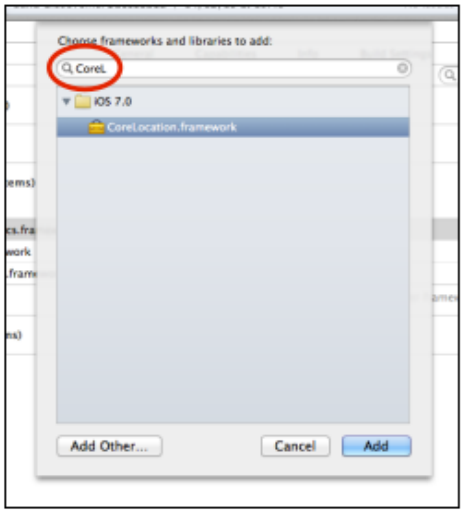


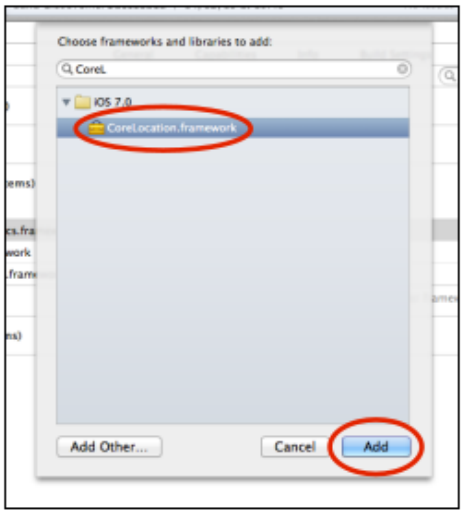
Aggiungi il framework



Cerca all'interno del menù

"my/sistemi_mobili/ios/src/scelta-framework--coreLocation.png" non è ancora stato creato. Clicca per crearlo.





Richiedi i servizi di localizzazione nell'applicazione

Se l'applicazione richiede i servizi di localizzazione per funzionare correttamente, il file `Info.plist` del progetto di Xcode deve contenere una voce per la "Required Device Capabilities" con il valore "location-serveces".

Prendere la posizione dell'utente

Il framework fornisce meccanismi per localizzare la posizione corrente del dispositivo ed utilizzare le informazioni nell'applicazione. Le informazioni provengono da hardware built-in nei cellulari, wi-fi o gps per triangolare la posizione del dispositivo.
I servizi riportano la posizione del dispositivo all'applicazione e possono essere impostati per generare aggiornamenti periodici per riceve dati aggiornati.

Sono presenti due servizi differenti che possono essere utilizzati per ottenere la posizione corrente:

- servizi di localizzazione standard, configurabili e con meccanismi general-purpose
- servizi di cambiamento-significativo, sistemi di localizzazione a basso consumo che possono risvegliare/avviare applicazioni sospese/non in esecuzione (disponibile da iOS 4+)

> Nota
> Bisogna fare attenzione perché i servizi di localizzazione richiedono molta batteria, quindi bisogna gestirli in modo appropriato

Configurazione dei servizi

I servizi sono gestiti da un istanza di `CLLocationManager` che è la classe che gestisce i dati sulla localizzazione e notifica l'applicazione in caso di aggiornamenti.

Questo gestore può essere configurato in modo da ottenere il comportamento desiderato. Una sua istanza (creato con alloc/init) può essere utilizzato in 2 modi:

- per configurare i parametri che determinano quando eventi devono essere consegnati
- iniziare e terminare eventuali consegne

La configurazione impostando determinate proprietà:

```
CLLocationAccuracy desiredAccuracy;
CLLocationDistance distanceFilter;
CLActivityType activityType;
```

Impostare l'accuratezza desiderata

La proprietà `desiredAccuracy` può essere utilizzata per impostare l'accuratezza di una posizione data. Questo è un suggerimento che è dato ai servizi di localizzazione, il ricevitore farà del suo meglio, non garantendo nulla, per ottenere l'accuratezza richiesta.
È importante assegnare un valore appropriato in base allo scenario di utilizzo.

> Nota
> Maggiora sarà la precisione richiesta, maggiore sarà la potenza/energia richiesta

I valori accettati sono di tipo `CCLocationAccuracy`:

Valore	Descrizione
`kCLLocationAccuracyBestForNavigation`	Utilizza il più alto livello di accuratezza possibile e la combina con dati aggiuntivi ottenuti dai sensori
`kCLLocationAccuracyBest`	Utilizza il più alto livello di accuratezza
`kCCLocationAccuracyNearestTenMeters`	Utilizza un livello di accuratezza con precisione di 10 metri
`kCLLocationAccuracyHundredMeters`	Utilizza un livello di accuratezza con precisione di 100 metri
`kCLLocationAccuracyKilometer`	Utilizza un livello di accuratezza con precisione di 1 km
`kCLLocationAccuracyThreeKilometers`	Utilizza un livello di accuratezza con precisione di 3 km

Impostare il filtro della distanza

La proprietà `distanceFilter` può essere usata per configurare la minima distanza in metri per cui un dispositivo deve muoversi per invocare un evento di aggiornamento.

> Nota
> La distanza viene misurata relativamente all'ultima posizione presa in considerazione

I valore costante, predefinito, `kCLDistanceFilterNone` può essere utilizzato per ricevere notifica di tutti gli eventi senza che sia impostato alcun filtro.

> Nota
> Questa proprietà è utilizzata solo con servizi di localizzazione standard e non con con servizi di localizzazione a cambiamento significativo

Tipi di attività

La proprietà `activityType` può essere utilizzata per configurare il tipo di attività che l'utente sta svolgendo, questa informazione può essere utilizzata dai servizi di localizzazione per determinare se gli aggiornamenti della posizione possono essere automaticamente messi in pausa.

> Nota
> Fermare gli aggiornamenti forniscono al sistema l'opportunità di risparmiare energia in situazione in cui la posizione dell'utente non sembra star cambiando.

Il valore di default di questa proprietà è `CLActivityTypeOther`, ma altre costanti possono essere usate:

```
CLActivityTypeAutomotiveNavigation
CLActivityTypeOtherNavigation
CLActivityTypeFitness
```

Monitoraggio della posizione e dei cambiamenti di direzione

Un istanza di `CLLocationManager` può essere avviata/terminata monitorando la posizione e i cambiamenti di direzione con i metodi:

```
[locationManager startUpdatingLocation];
[locationManager startUpdatingHeading];
[locationManager stopUpdatingLocation];
[locationManager stopUpdatingHeading];
```

Regioni di monitoraggio

Un istanza di `CLLocationManager` può anche monitorare eventi relativi a regioni specifiche nello spazio, come l'ingresso/uscita da una determinata regione. I servizi di monitoraggio delle regioni possono essere utilizzati per definire i confini di più regioni geografiche.
Per avviare/terminare il monitoraggio per specifiche regioni si utilizzano i metodi:

```
[locationManager startMonitoringForRegion:region];
[locationManager stopMonitoringForRegion:region];
```

Essere avvisati

La classe `CLLocationManagerDelegate` definisce la proprietà `delegate` che può essere utilizzata per impostare un oggetto delegato, che riceverà una notifica quando un evento sulla localizzazione verrà generato.

Il delegato deve essere un istanza di una classe conforme al protocollo `CLLocationManagerDelegate`, che definisce i metodi usati per ricevere aggiornamenti sulla posizione e sulla direzione da un `CLLocationManager`.

La classe `CLLocation` rappresenta i dati sulla localizzazione generati da un `CLLocationManager`. Una sua istanza include le coordinate geografiche e altitudine della posizione del dispositivo con valori indicanti l'accuratezza delle misurazioni e quanto queste misurazione sono fatte (includendo anche direzione e velocità).

Conformarsi al protocollo

La `CLLocationManagerDelegate` definisce i metodi usati per ricevere aggiornamenti sulla localizzazione e direzione da un `CLLocationManager`

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

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

- (void)locationManagerDidPauseLocationUpdates:(CLLocationManager *)manager

- (void)locationManagerDidResumeLocationUpdates:(CLLocationManager *)manager

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

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

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

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

Esempi di monitoraggio della posizione

```
@interface ViewController ()<CLLocationManagerDelegate>

@property (nonatomic,strong) CLLocationManager *locationManager;

@end

@implementation

// Esempio di lazy loading, si postricipa l'allocazione della emmoria a quando è realemente
necessario
// È una pratica molto buona per proprietà forti

- (CLLocationManager *)locationManager{
        if(!_locationManager)
                _locationManager = [[CLLocationManager alloc] init];
        return _locationManager;
}

- (void)viewDidLoad{
        [super viewDidLoad];
        // configura il gestiore di localizzazione, imposta il delgato e inizia a ricevere
gli aggiornamenti
        self.locationManager.desiredAccuracy = kCLLocationAccuracyBest;
        self.locationManager.distanceFilter = 100;
        self.locationManager.delegate = self;
        [self.locationManager startUpdatingLocation];
}

// Questo metodo callback è chaiamto dal gestore di localizzazione ogni volta che la
posizione cambia
```

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

        CLLocation *currentLocation = [locations lastObject];
        // ..
}

@end
```

Simulare la posizione

È possibile simulare la posizione all'interno del simulatore, così da non dover testare il codice su un dispositivo.
Per fare ciò bisogna aggiungere al progetto uno o più file `GPX`

> Nota
> I file `GPX` sono file `XML` che hanno uno specifico formato

```
<?xml version="1.0" encoding="UTF-8" standalone="no"?>
<gpx
xmlns="http://www.topografix.com/GPX/1/1"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="http://www.topografix.com/GPX/1/1
http://www.topografix.com/GPX/1/1/gpx.xsd"
version="1.1"
creator="gpx-poi.com">
<wpt lat="44.801700" lon="10.328012">
<time>2013-12-10T16:52:28Z</time>
<name>Parma centro</name>
</wpt>
</gpx>
```

Dopo che un file `GPX` è stato aggiunto al progetto è possibile avviare la simulazione in una specifica posizione `Debug -> Simulate Location`.

> Nota
> Alcune posizione sono già fornite da Xcode

Geocoding

La classe `CLGeocoder` fornisce i servizi per convertire le coordinate in una rappresentazione più user-friendly, che consiste in:

- strade, città, stati e continenti
- Punti di interesse, punti di riferimento

Un oggetto di questa classe funziona con un servizio basato sulla rete per cercare informazioni sul segnaposto per il relativo valore di coordinate specifiche. Questo avviene mediante 2 tipi di richieste:

- Reverse-geocoding, questo tipo di richiesta prende i valori di latitudine e longitudine e trova un indirizzo leggibile dall'utente
- Forward-geocoding, questo tipo di richiesta prende un indirizzo leggibile dall'utente e trova i corrispondenti valori di latitudine e longitudine
  Il risultato di entrambe le richieste ritorna un oggetto di tipo `PLPlacemark`.

> Nota
> Bisogna tener presente che le richieste di geocodifica sono limitate per ciascuna applicazione.

Reverese-geocoding

Questo tipo di richiesta può essere effettuata mediante i metodi di un istanza di `CLGeocoder`:

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

Dove il tipo `CLGeocodeCompletitionHandler` è definito come

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

> Nota
> Il blocco del gestore di completamento verrà eseguito sul thread principale

> Nota
> Per la maggior parte delle richieste di geocoding, l'array del placemark passato conterrò una sola entità

Esempio

```
CLGeocoder *geocoder = [[CLGeocoder alloc] init];
[geocoder reverseGeocodeLocation:location
                            completionHandler:^(NSArray *placemarks, NSError *error) {
        // enumerateObjectsUsingBlock è un modo conveniente per iterare all'interno di una
collezione
        // (oggetto, indice, flag)
        [placemarks enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
                if([obj isKindOfClass:[CLPlacemark class]]){
                        CLPlacemark *pm = (CLPlacemark *)obj;
                        //...
                }
        }];
}];
```

Map Kit

Il framework Map Kit consente di incorporare un'interfaccia mappa, a livello stradale e satellitare, completamente funzionale nell'applicazione.

> Nota
> Le mappe possono essere ingrandite e spostate in modo programmatico

Il framework fornisce il supporto automatico per gli eventi touch che consentono agli utenti di eseguire lo zoom e la panoramica della mappa.

> Nota
> Le mappe possono essere annotate con informazioni personalizzate

In qualsiasi momento che una classe o un protocollo definito in questo framework viene utilizzato bisogna importarlo con la direttiva `#import <MapKit/MapKit.h>`.

Aggiungere una mappa all'interfaccia utente

La classe `MKMapView`, sottoclasse di `UIView` è un'interfaccia autonoma per la presentazione dei dati cartografici. Può essere aggiunta in 2 modi:

- Programmaticamente mediante `alloc/initWithFrame`
- Tramite lo storyboard, trascinandola dalla palette degli oggetti

L'area visibile di una vista di questo tipo può essere configurata impostando la proprietà `region` con una struct di tipo `MKCoordinateRegion`

```
MKCoordinateRegion mapRegion;
mapRegion.center = location.coordinate;
mapRegion.span.latitudeDelta = 0.1;
mapRegion.span.longitudeDelta = 0.1;
[self.mapView setRegion:mapRegion animated:YES];
```

Ingrandimento e panoramica

Per eseguire la panoramica (senza ingrandimento), è sufficiente impostare la proprietà `centerCoordinate` della map-view.

```
CLLocationCoordinate2D newCenter = CLLocationCoordinate2DMake(loc.latitude,loc.longitude);
self.mapView.centerCoordinate = newCenter;
```

Per ingrandire bisogna resettare la proprietà `region`

```
MKCoordinateRegion mapRegion;
mapRegion.center = location.coordinate;
mapRegion.span.latitudeDelta = 0.1;
mapRegion.span.longitudeDelta = 0.1;
[self.mapView setRegion:mapRegion animated:YES];
```

La posizione dell'utente e annotazioni

Per mostrare la posizione dell'utente nella map-view è necessario impostare la proprietà della mappa `sowUserLocation` a `YES`

```
self.mapView.showsUserLocation = YES;
```

Come detto sopra una map-view può avere delle annotazioni, che includono:

- Latitudine
- Longitudine
- Titolo
- Sottotitolo
  Queste sono mostrate mediante un istanza di `MKAnnotationView`.

È importante notare che le annotazioni possono mostrare un richiamo (callout), il quale, di default, presenta titolo e sottotitolo. Un richiamo può presentare delle accessory-view per personalizzare ulteriormente il suo contenuto:

- `leftCalloutAccessoryView`
- `rightCalloutAccessoryView`

Le annotazioni che possono essere aggiunte alla map-view sono oggi conformi al protocollo `MKAnnotation`.

> Nota
> La map-view ha una proprietà accessibile in sola lettura `annotation` tiene traccia delle annotazioni attualmente visibili

Un oggetto `MKAnnotation` ha associato un `MKAnnotationView` che è attualmente mostrato. Il protocollo `MKAnnotation` definisce:

- Una proprietà richiesta `coordinate` di tipo `CLLocationCoordinate2D`
- 2 proprietà opzionali `title` e `subtitle` di tipo `NSString*`

Utilizzo delle categorie per le annotazioni

Un ottimo modo per fornire funzionalità di annotazione a una classe è l'utilizzo delle categorie, facendo così è necessario aggiungere un metodo richiesto dal protocollo `MKAnnotation` senza toccare la classe originale

MDCheckin.h

```
@interface MDCheckin : NSObject
@property (nonatomic,strong,readonly) MDPoi *poi;
@end
```

MDCheckin+Annotation.h

```
#import "MDCheckin.h"
#import <MapKit/MapKit.h>

// MKAnnotation rappresenta una semplice dichiarazione della categoria per essere conforme al
protocollo MKANnotation
@interface MDCheckin (Annotation)<MKAnnotation>
@end
```

MDCheckin+Annotation.m

```
#import "MDCheckin+Annotation.h"
@implementation MDCheckin (Annotation)

// implementazione del metodo richiesto @required
- (CLLocationCoordinate2D)coordinate{
        CLLocationCoordinate2D coordinate;
        coordinate.latitude = self.poi.latitude;
        coordinate.longitude = self.poi.longitude;
        return coordinate;
}
// implementazione del metodo opzionale #optional
- (NSString *)title{
        return self.poi.name;
}
@end
```

Poiché la proprietà `annotation` di una map-view è di sola lettura, è necessario utilizzare metodi appropriati per aggiungere e rimuovere annotazioni:

```
- (void)addAnnotation:(id<MKAnnotation>)annotation
- (void)addAnnotations:(NSArray *)annotations
```

```
- (void)removeAnnotation:(id<MKAnnotation>)annotation
- (void)removeAnnotations:(NSArray *)annotations
```

> Nota
> Per motivi di performance, una buona pratica è di inserire subito, se possibile, tutte le annotazioni sulla mappa

Le istanze di `MKAnnotationView` sono riutilizzate da `MKMapView`, similmente a come `UITableViews` fa con `UITableViewCell`.

Il metodo standard per mettere un annotazione sulla mappa avviene utilizzando la classe, o una qualsiasi sottoclasse di `MKPinAnnotationView` per personalizzare l'aspetto della annotation-view. Dopo che una annotazione è stata toccata, un callout deve essere mostrato se la sua proprietà `canShowCallout` è stata impostata su `YES`

Il delegato di MKMapView

La classe `MKMapView` ha una proprietà `delegate` che può essere usata per impostare un oggetto delegato che in opzionalmente riceverà gli eventi relativi alla map-view e sarà responsabile di fornire concrete istanza di `MKAnnotationView` per una data annotazione.

Il delegato deve essere conforme al protocollo `MKMapVIewDelegate`, quindi presentare i metodi per:

- Rispondere a cambiamento di posizione della mappa
- Rispondere a caricamento degli eventi dei dati della mappa
- Rispondere a cambiamenti di posizione dell'utente
- Gestire annotation-view
- Rispondere a eventi di trascinamento di annotation-view
- Rispondere a elezione di annotation-view

Personalizzare le annotation-view

```
- (MKAnnotationView *)mapView:(MKMapView *)mapView
                                            viewForAnnotation:(id<MKAnnotation>)annotation
```

Questo metodo può essere chiamato in qualsiasi momento in cui una annotazione deve essere posizionata su una map-view, ciò richiede di fornire un `MKAnnotationView` per l'annotazione data.

> Nota
> Ciò avviene in modo molto simile a `cellForRowAtIndexPath:` in `UITableViewDataSource`

Esempio

```
- (MKAnnotationView *)mapView:(MKMapView *)mapView
                                            viewForAnnotation:
(id<MKAnnotation>)annotation{

        static NSString *AnnotationIdentifier = @"ViewController";

        MKAnnotationView *view = [mapView

dequeueReusableAnnotationViewWithIdentifier:AnnotationIdentifier];
        if(!view){
                view = [[MKPinAnnotationView alloc] initWithAnnotation:annotation
                reuseIdentifier:AnnotationIdentifier];
                view.canShowCallout = YES;
        }

        view.annotation = annotation;
        UIImageView *imageView = [[UIImageView alloc] initWithFrame:CGRectMake(0, 0, 36,
36)];
        imageView.image = [UIImage imageNamed:@"photo"];
        view.leftCalloutAccessoryView = imageView;
        view.rightCalloutAccessoryView = [UIButton buttonWithType:UIButtonTypeInfoDark];

        return view;
}
```

Il delegato riceve la notifica quando un annotazione viene selezionata, generalmente se la proprietà `canShowCallout` viene impostata a `YES` un call'out verrà successivamente mostrato.

> Nota
> Questo è un buon posto per eseguire l'inizializzazione pigra del contenuto del callout in modo che venga

> caricato solo quando richiesto

```
- (void)mapView:(MKMapView *)mapView didSelectAnnotationView:(MKAnnotationView *)view{
        if([view.leftCalloutAccessoryView isKindOfClass:[UIImageView class]]){
                UIImageView *imageView = (UIImageView *)view.leftCalloutAccessoryView;
                imageView.image = ...; //caricato dalla rete utilizzando il GCD (no main
queue)
        }
}
```

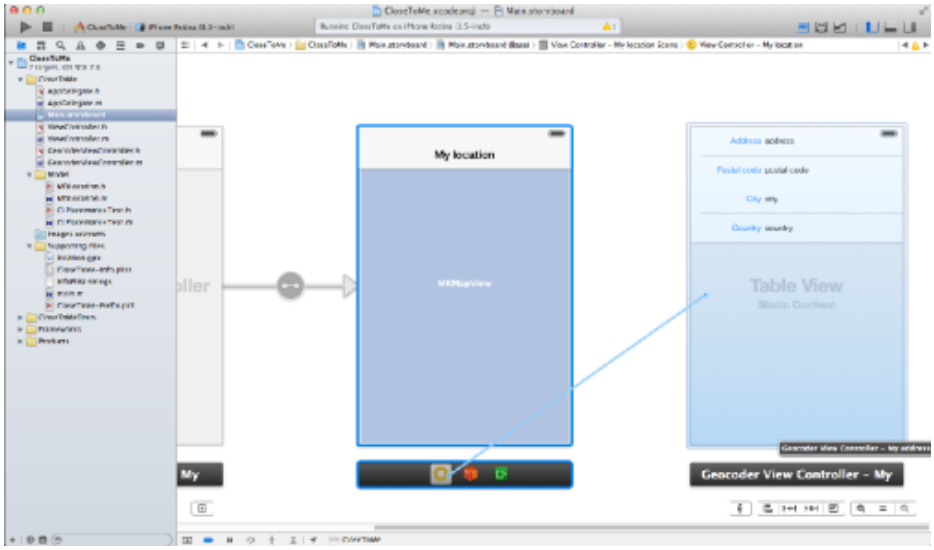
Se un accessory-view di una callout è un `UIControl` il delegato riceve una notifica, con il metodo seguente, quando il controllo è toccato

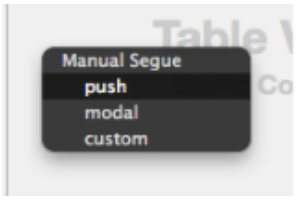
```
- (void)mapView:(MKMapView *)mapView
                annotationView:(MKAnnotationView *)view
                calloutAccessoryControlTapped:(UIControl *)control
```

Seguire programmaticamente

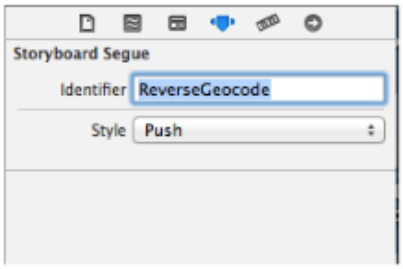
Spesso non è definire un seguito direttamente dalla storyboard, questo perché l'elemento da cui si sta provenendo non è presente un quest'ultima, ad esempio il callout di un'annotazione. Per questo motivo è necessario impostare il proseque mediante codice, la procedura è:

1. Nella storyboard, creare manualmente un seguito ctrl-dragging il controllo dal controller della vista di origine (non un controllo specifico nella vista) al controller della vista di destinazione





2. Assegnare un identificatore al seguito in modo da avere la possibilità di farci riferimento all'interno del codice



3. Eseguire il metodo `performSegueWithIdentifier:sender:` di `UIViewController`, specificando l'identificativo inserto nel punto 2 ed impostando il mandante corretto

```
- (void)mapView:(MKMapView *)mapView
                                annotationView:(MKAnnotationView *)view
                                calloutAccessoryControlTapped:(UIControl *)control{
```

```
        [self performSegueWithIdentifier:@"ReverseGeocode" sender:view];
}
```

Lavorare con JSON

Lavorare con JSON in iOS è abbastanza semplice, è presente la classe `NSJSONSerialization` che fornisce i metodi per:

- Serializzare un oggetto a una stringa JSON
- Deserializzare una stringa JSON in un oggetto

È molto importante però che un oggetto che debba essere convertito abbia le seguenti proprietà:

- L'oggetto più esterno sia un `NSArray` o un `NSDictionary`
- Tutti gli oggetti contenuti sono istanze di `NSString`, `NSNumber`, `NSArray`, `NSDictionary` o `NSNull`
- Tutte le chiavi del dizionario sono istanze di `NSString`
- I numeri contenuti non sono né `NaN` né `infinty`

Serializzare

```
id obj = ...;
NSError * error;
// isValidJSONObject -> controllo se l'oggetto è serializzabile
if([NSJSONSerialization isValidJSONObject:obj]){
        // il metodo ritorna o l'oggetto JSON, codificato in UTF-8 o nil
        NSData *data = [NSJSONSerialization dataWithJSONObject:obj
                                        options:0
                                        error:&error]; // in caso di errore error manterrà
l'errore avvenuto
}
```

Deserializzare

```
NSData *data = ...;
NSError * error;
// ritorna un NSArray* o un NSDictionary* in caso di successo o nil in caso di errore
id obj = [NSJSONSerialization JSONObjectWithData:data

options:NSJSONReadingAllowFragment
                                                        error:&error]; // in caso di
errore error manterrà l'errore avvenuto
```