Mobile Application Development

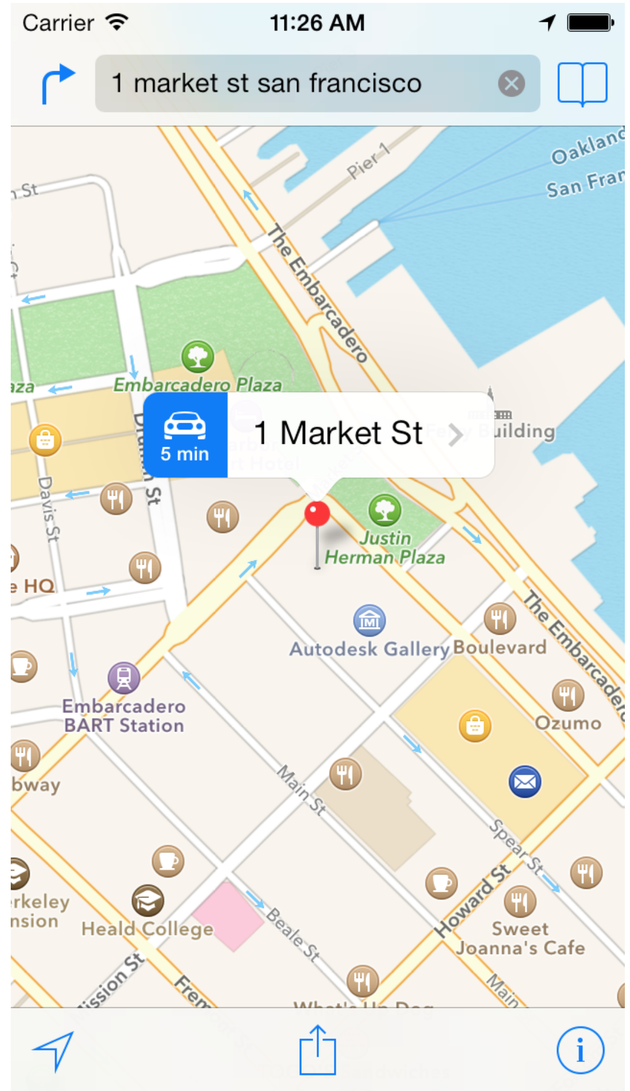
Lecture 21
Core Location and Map Kit

Lecture Summary

- Core Location
- Getting the user's location
- Geocoding
- Map Kit
- Annotating maps
- Segueing programmatically
- Working with JSON
- DEMO

Location-based applications

- Location-based information in iOS involve:
  - Location services
  - Maps
- Location services are provided by the **Core Location** framework
  - Objective-C APIs
  - Provides information related to the user's location and heading
- Maps are provided by the **Map Kit** framework
  - Support for displaying and annotating maps

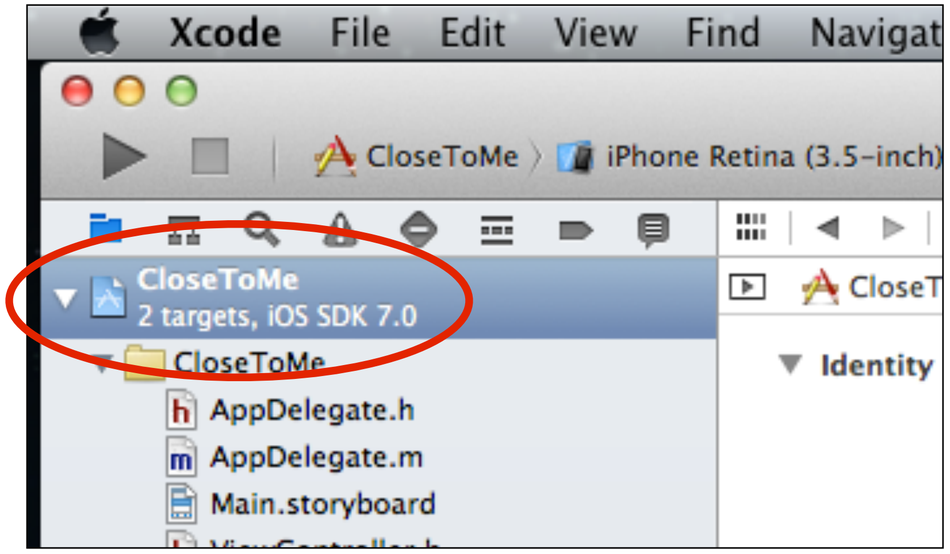


Location services

- iOS applications are intended to be executed on mobile devices
- Location services provide support for mobility
- Location services monitor the user's position and generate updates
- The Core Location framework provides support for location services and must be linked to the project in order to gain access to all the interfaces it provides
- Anytime a class or protocol defined in the Core Location framework is used, the framework must be imported into the file with the following import directive

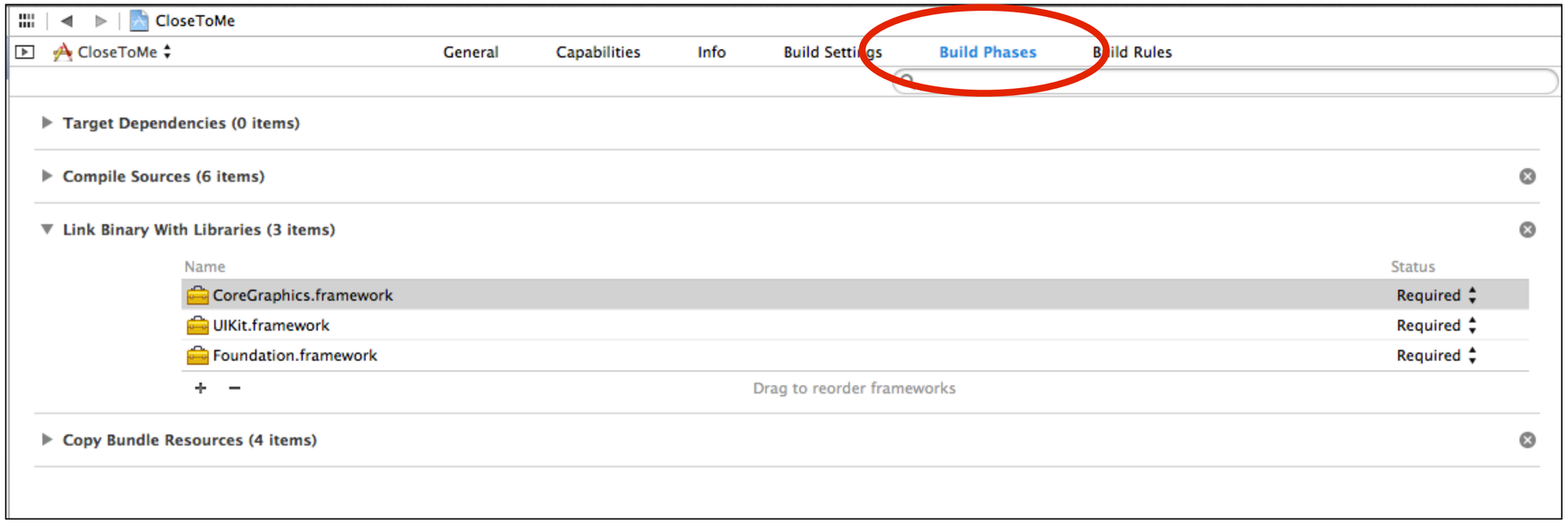
```
#import <CoreLocation/CoreLocation.h>
```

Linking the Core Location framework to your project



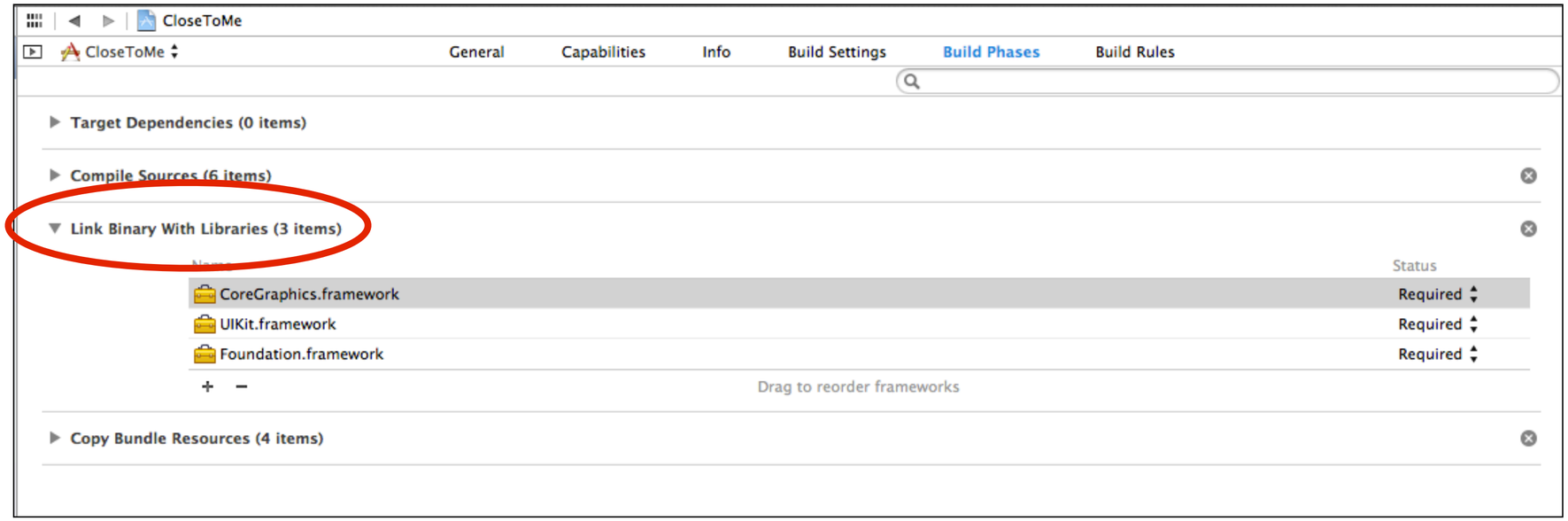
Select the project file in the navigator

Linking the Core Location framework to your project



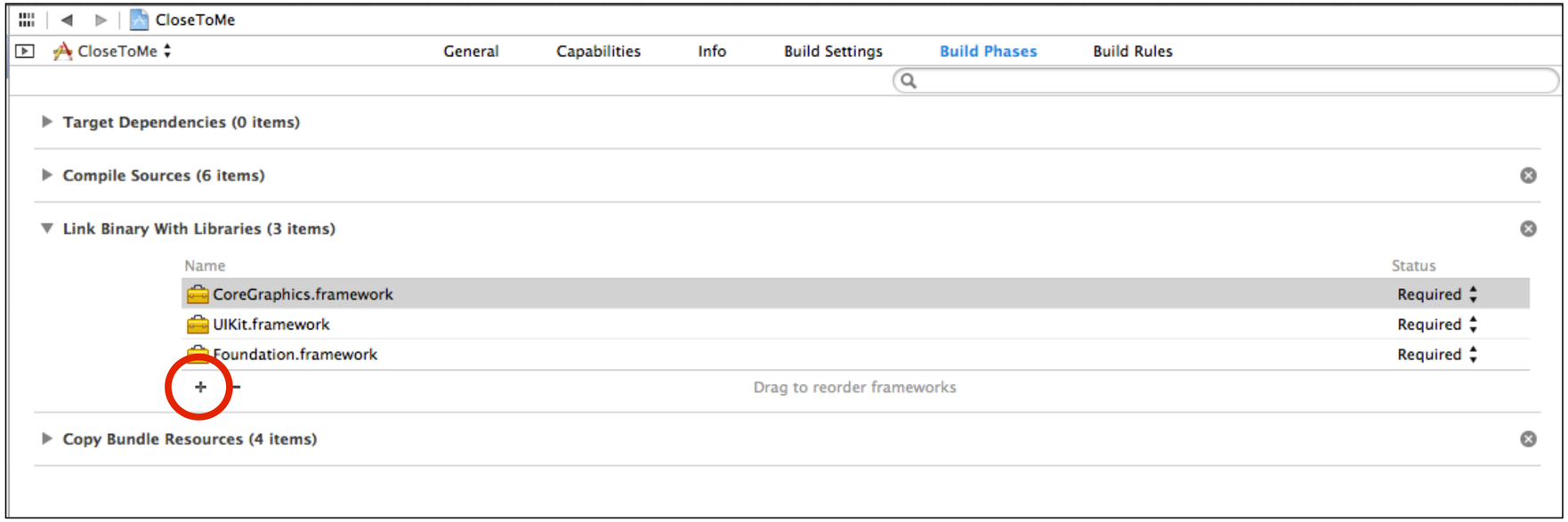
Select the **Build Phases** tab

Linking the Core Location framework to your project



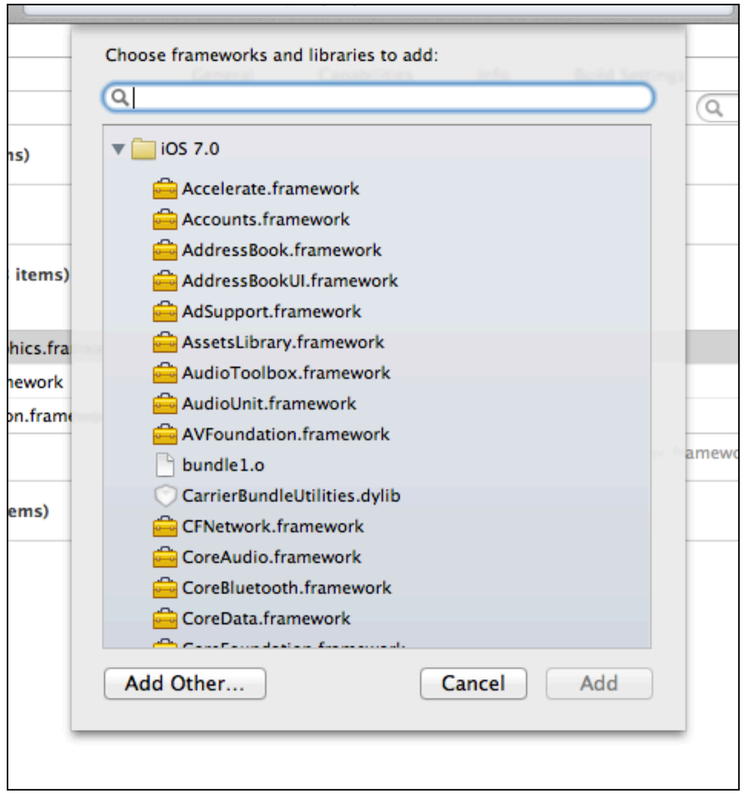
Expand the **Link Binary With Libraries** section

Linking the Core Location framework to your project



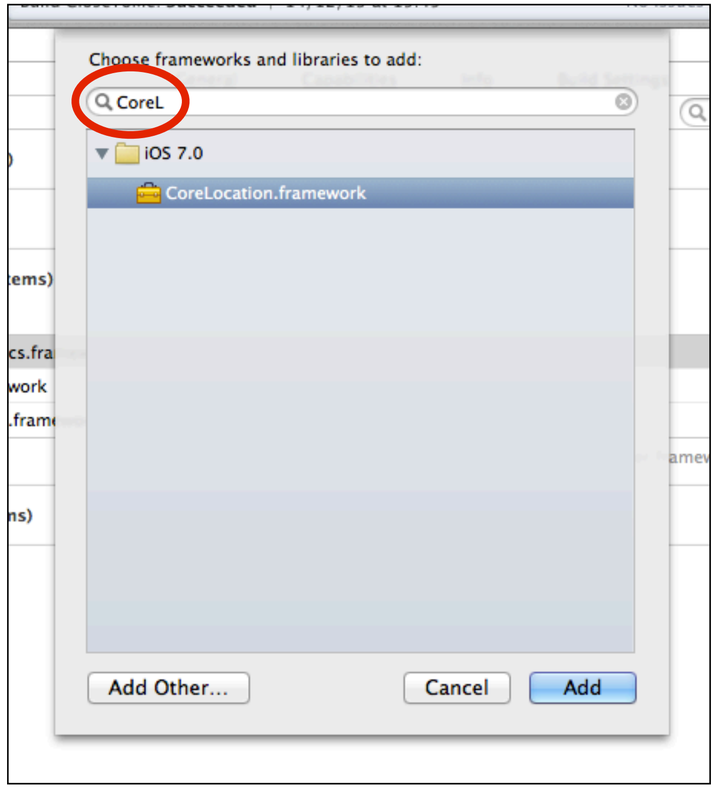
Click on the ‘+’ button to add a framework

Linking the Core Location framework to your project



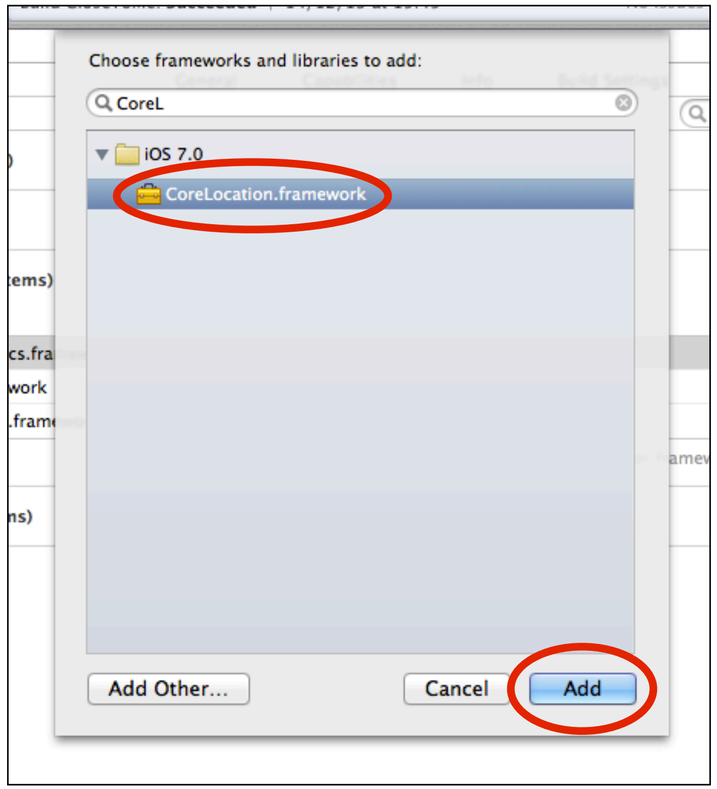
A selection menu appears

Linking the Core Location framework to your project



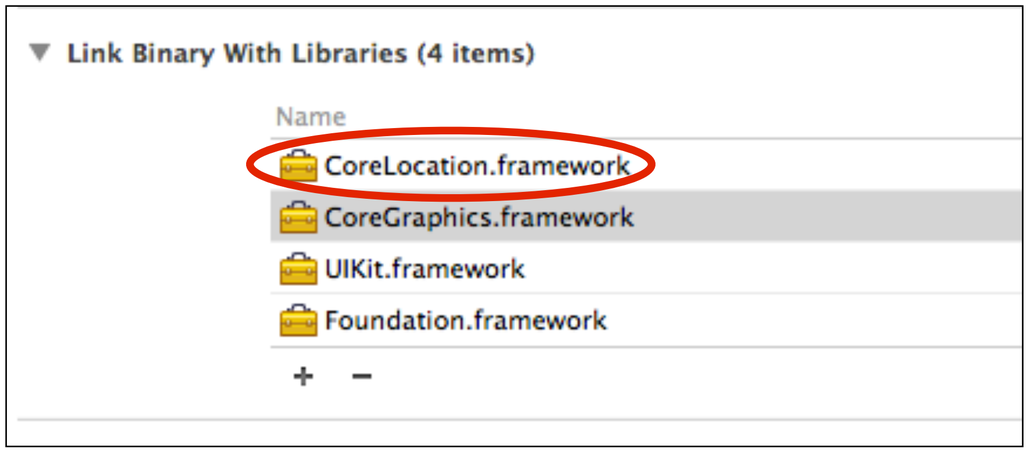
Start typing in the search box for ‘Core Location’

Linking the Core Location framework to your project



Select ‘CoreLocation.framework’ and then click ‘Add’

Linking the Core Location framework to your project



CoreLocation.framework is now linked with your project

Requiring location services in your app

- If the app requires location services to run properly, the **Info.plist** file of the Xcode project should contain an entry for the **Required Device Capabilities** key with the **location-services** value

Key	Type	Value
▼ Information Property List	Dictionary	(14 items)
Localization native development r...	String	Italy
Bundle display name	String	${PRODUCT_NAME}
Executable file	String	${EXECUTABLE_NAME}
Bundle identifier	String	it.unipr.ce.mobdev.${PRODUCT_NAME:rfc1034identifier}
InfoDictionary version	String	6.0
Bundle name	String	${PRODUCT_NAME}
Bundle OS Type code	String	APPL
Bundle versions string, short	String	1.0
Bundle creator OS Type code	String	????
Bundle version	String	1.0
Application requires iPhone envir...	Boolean	YES
Main storyboard file base name	String	Main
▼ Required device capabilities	Array	(2 items)
Item 0	String	armv7
Item 1	String	location-services
▶ Supported interface orientations	Array	(3 items)

Getting the user's location

- The Core Location framework provides mechanisms to locate the current position of the device and use that information in the application
- Location information are obtained from the built-in cellular, Wi-Fi, or GPS hardware to triangulate a location fix for the device
- The Core Location service reports the device location to the app and can be configured to generate periodic updates as it receives new or improved data
- There are two different services that can be used to get the device's current location:
  - **standard location service**: configurable, general-purpose mechanism (most common)
  - **significant-change location service**: low-power location service which can wake up an app that is suspended or not running (available only in iOS 4+)
- Be aware that location services are power-intensive and can drain the device battery significantly, so they must be managed properly according to the app's requirements

Configuring the Core Location service

- The Core Location service is managed by an instance of `CLLocationManager`, which is the class that handles location data and notifies the application of location and heading updates
- The `CLLocationManager` can be configured in order to tune in the desired behavior
- An instance of `CLLocationManager` (created with `alloc`/`init`) can be used:
  1. to configure the parameters that determine when location and heading events should be delivered
  2. to start and stop the actual delivery of those events
- The configuration is performed by setting a number of properties:

```
@property (assign, nonatomic) CLLocationAccuracy desiredAccuracy;
@property (assign, nonatomic) CLLocationDistance distanceFilter;
@property (assign, nonatomic) CLActivityType activityType;
```

Setting the desired accuracy

- The `desiredAccuracy` property can be used to set the accuracy of the location data
- This is a hint that is given to the location services: the receiver does its best to achieve the requested accuracy but the actual accuracy is not guaranteed
- It is important to assign a value to this property that is appropriate for your usage scenario: determining a location with greater accuracy requires more time and more power
- Accepted values are of type `CLLocationAccuracy`:

`kCLLocationAccuracyBestForNavigation`	Use the highest possible accuracy and combine it with additional sensor data
`kCLLocationAccuracyBest`	Use the highest-level of accuracy
`kCLLocationAccuracyNearestTenMeters`	Accurate to within ten meters of the desired target
`kCLLocationAccuracyHundredMeters`	Accurate to within one hundred meters
`kCLLocationAccuracyKilometer`	Accurate to the nearest kilometer
`kCLLocationAccuracyThreeKilometers`	Accurate to the nearest three kilometers

Distance filter

- The `distanceFilter` property can be used to configure the minimum distance in meters that a device must travel before an update event is generated
- This distance is measured relative to the previously delivered location
- The constant value `kCLDistanceFilterNone` (default) can be used to be notified of all events with no filter being set
- This property is used only with standard location service and not with significant-change location service

Activity type

- The `activityType` property can be used to configure the type of activity that the user is performing
- This information can be used by the location service 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
- The default value is the constant `CLActivityTypeOther`
- Other constants that can be used:
  - `CLActivityTypeAutomotiveNavigation`
  - `CLActivityTypeOtherNavigation`
  - `CLActivityTypeFitness`

Monitoring location and heading changes

- A `CLLocationManager` instance can start/stop monitoring location and heading updates with the following methods:

```
[locationManager startUpdatingLocation];

[locationManager startUpdatingHeading];

[locationManager stopUpdatingLocation];

[locationManager stopUpdatingHeading];
```

Monitoring regions

- A `CLLocationManager` instance can also monitor events related to specific regions in space, such as entering or exiting a certain region
- The region-monitoring service can be used to define the boundaries for multiple geographical regions
- To start and stop monitoring for a specific region, the following methods can be used:

```
[locationManager startMonitoringForRegion:region];

[locationManager stopMonitoringForRegion:region];
```

Being notified: CLLocationManagerDelegate

- The `CLLocationManager` class defines a `delegate` property can be used to set a delegate object which will be notified when a location-related event has been generated
- The delegate must be an instance of a class conforming to the `CLLocationManagerDelegate` protocol
- The `CLLocationManagerDelegate` defines the methods used to receive location and heading updates from a `CLLocationManager`
- The `CLLocation` class represents the location data generated by a `CLLocationManager`
- `CLLocation` instances include 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, as well as heading and speed information

Conforming to CLLocationManagerDelegate

- The `CLLocationManagerDelegate` defines the methods used to receive location and heading updates from a `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
```

Example of location monitoring

```
@interface ViewController ()<CLLocationManagerDelegate>

@property (nonatomic,strong) CLLocationManager *locationManager;

@end

@implementation

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

- (void)viewDidLoad{
    [super viewDidLoad];
    self.locationManager.desiredAccuracy = kCLLocationAccuracyBest;
    self.locationManager.distanceFilter = 100;
    self.locationManager.delegate = self;
    [self.locationManager startUpdatingLocation];
}

- (void)locationManager:(CLLocationManager *)manager didUpdateLocations:(NSArray *)locations{
    CLLocation *currentLocation = [locations lastObject];
    ...
}

@end
```

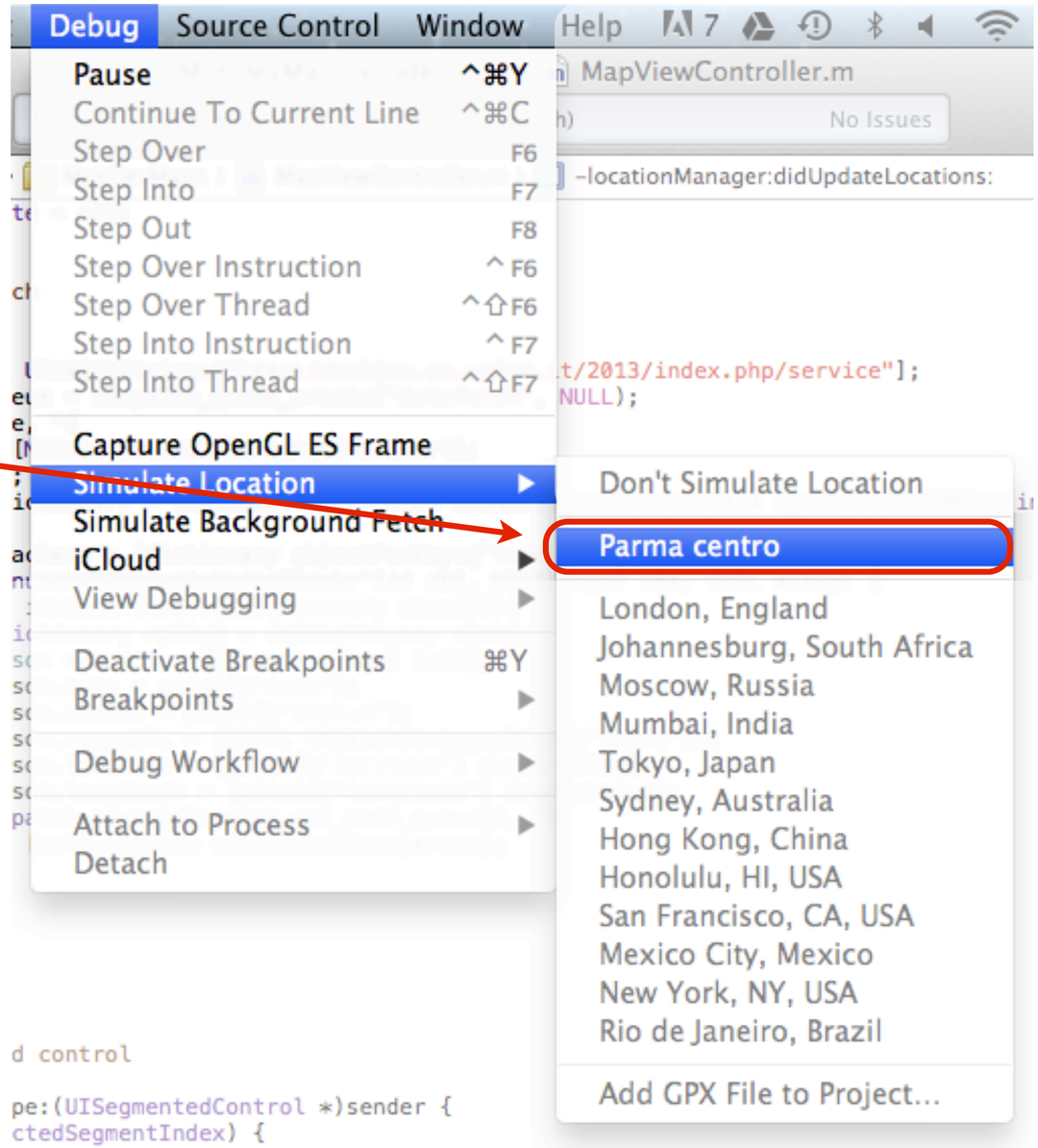
Simulating locations in the simulator

- It is possible to simulate locations in the simulator, thus avoiding to test your code on the device
- One or more GPX files must added to the project
- GPX files are XML files that have a specific document format

```
<?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>
```

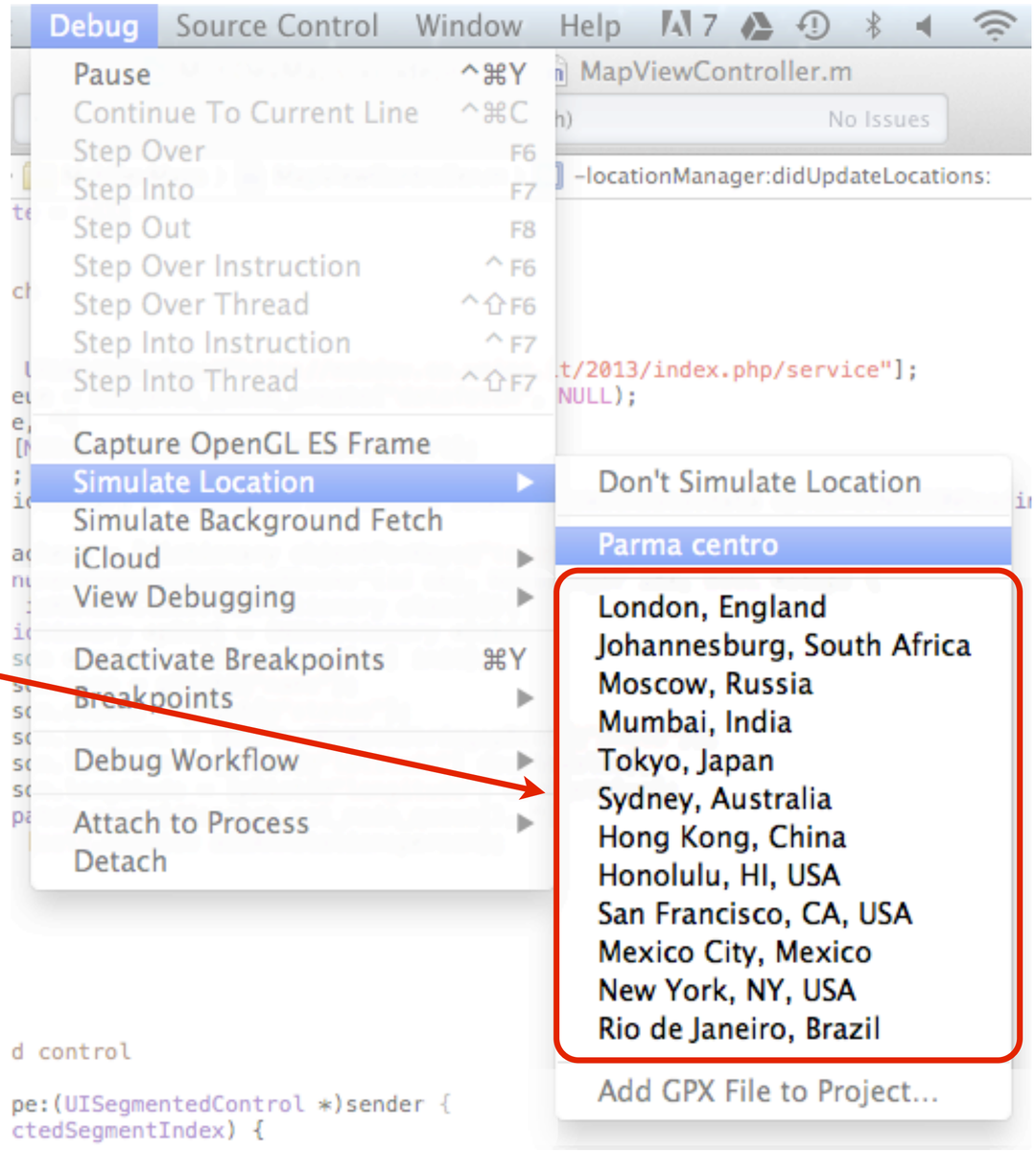
Simulating locations in the simulator

- Once a GPX file has been added to the project it is possible to simulate the user's location in that specific location by using the Debug → Simulate Location menu and then selecting the desired location



Simulating locations in the simulator

- Some locations are already provided by Xcode



Geocoding

- The `CLGeocoder` class provides services for converting between a coordinate (specified as a latitude and longitude) and the user-friendly representation of that coordinate, consisting of:
  - street, city, state, and country
  - a relevant point of interest, landmarks, or other identifying information
- A geocoder object is a works with a network-based service to look up placemark information for its specified coordinate value
- **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
- The results are returned using a `CLPlacemark` object
- Be aware that geocoding requests are rate-limited for each app

Reverse geocoding

- Reverse geocoding can be performed with the following method of an instance of `CLGeocoder`

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

- The `CLGeocodeCompletionHandler` type is defined as

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

- This method submits the specified location data to the geocoding server asynchronously and returns
- The completion handler block will be executed on the main thread
- For most geocoding requests, the `placemark` array passed in the completion handler should contain only one entry

Reverse geocoding example

```
CLGeocoder *geocoder = [[CLGeocoder alloc] init];

[geocoder reverseGeocodeLocation:location
               completionHandler:^(NSArray *placemarks, NSError *error) {

    [placemarks enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {

        if([obj isKindOfClass:[CLPlacemark class]]){

            CLPlacemark *pm = (CLPlacemark *)obj;
            ...

        }

    }];

}];
```

Map Kit

- The Map Kit framework allows to embed a fully functional map interface into your app
- Map Kit can be used to display street-level and satellite maps
- Maps can be zoomed and panned programmatically
- The framework provides automatic support for the touch events that let users zoom and pan the map
- Maps can be annotated with custom information
- Anytime a class or protocol defined in the Map Kit framework is used, the framework must be imported into the file with the following import directive

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

Add a map to the user interface

- The `MKMapView` class (subclass of `UIView`) is a self-contained interface for presenting map data
- A `MKMapView` can be added:
  - programmatically with `alloc/initWithFrame:`
  - in storyboard, by dragging it from the object palette
- A map view's visible area can be configured by setting the `region` property with a `MKCoordinateRegion` struct:

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

Zooming and panning

- To pan (without zooming), it is sufficient to set the `centerCoordinate` property of the map view

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

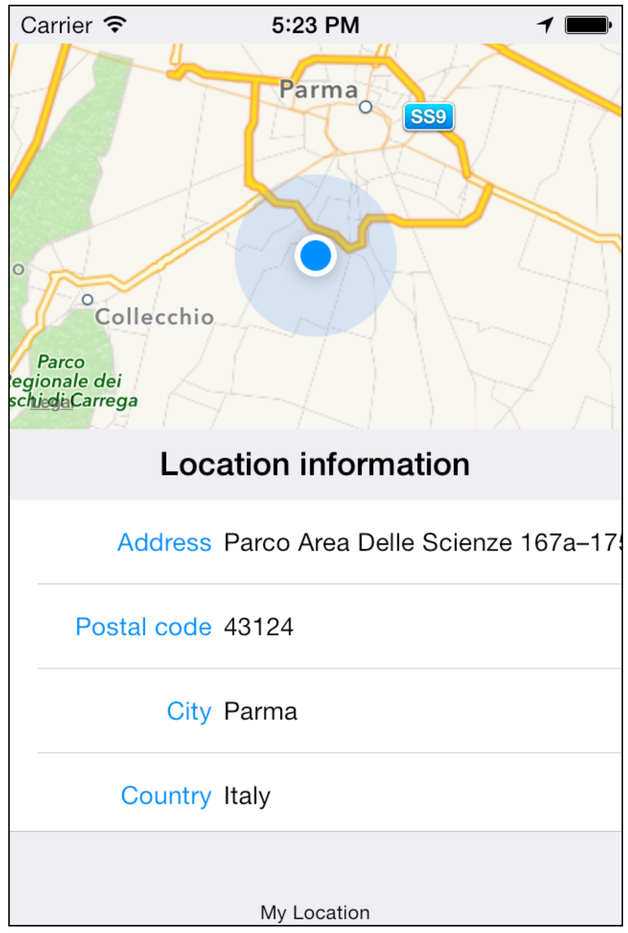
- To zoom, the region property must be reset

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

User location in map view

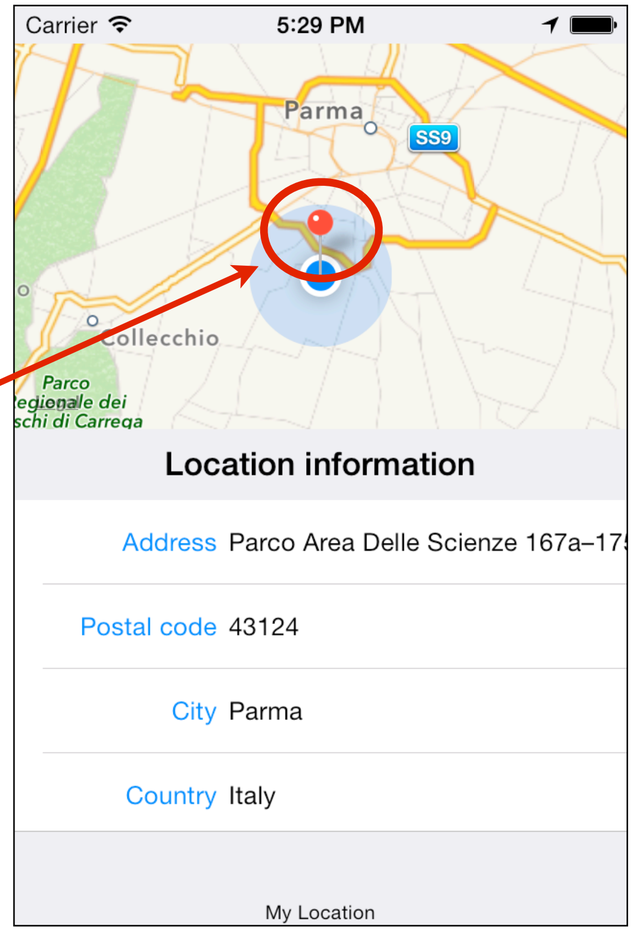
- To display the user location in the map view, the `showUserLocation` property of the map view must be set to YES

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



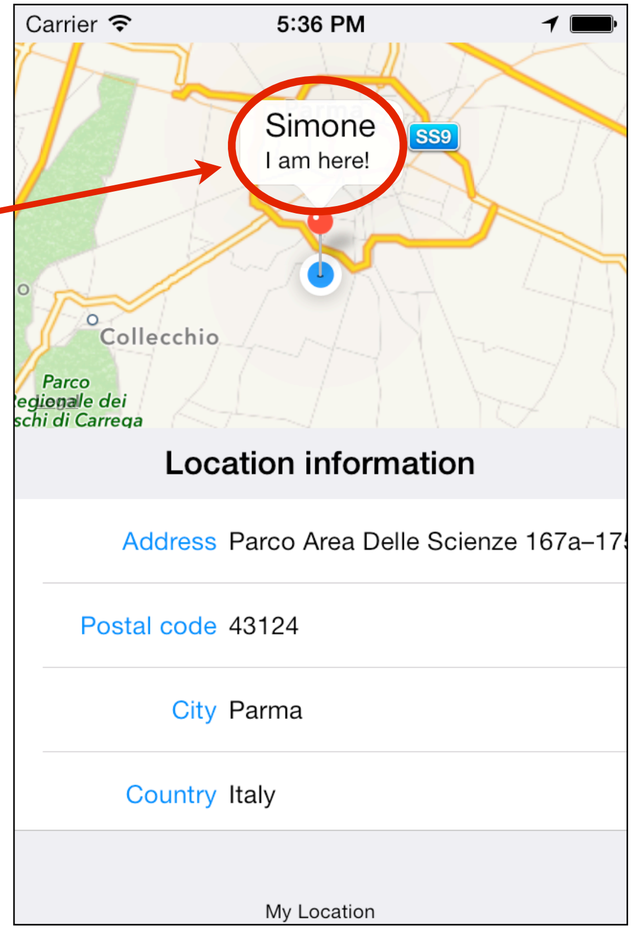
Annotating maps

- A map view can have annotations
- An annotation include a coordinate (latitude and longitude), a title, and a subtitle
- Annotations are displayed using an instance of `MKAnnotationView`



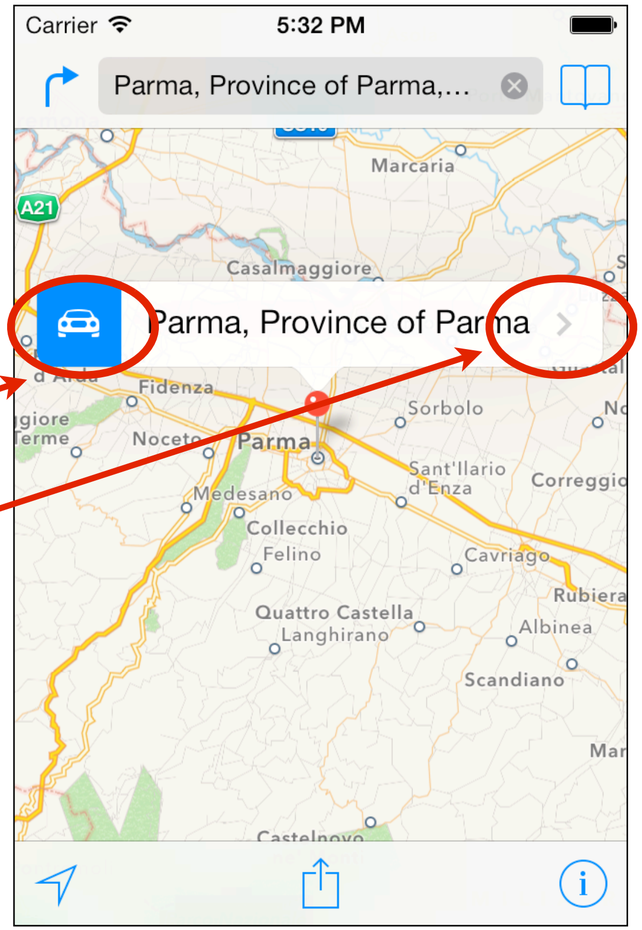
Annotating maps

- Annotations can also display a callout
- A callout displays the title and subtitle by default



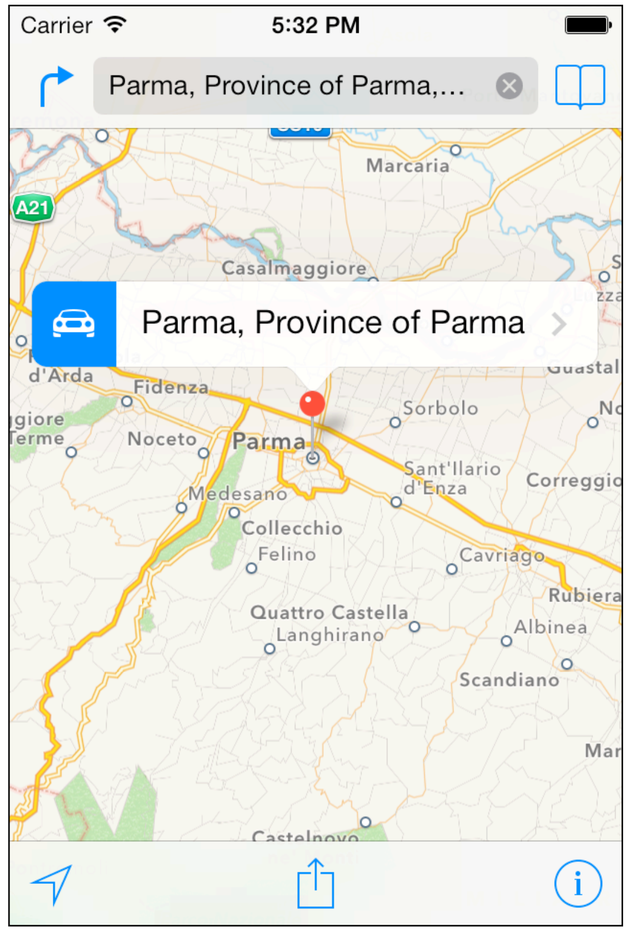
Annotating maps

- Annotations can also display a callout
- A callout displays the title and subtitle by default
- A callout has also accessory view to further customize its contents:
  - `leftCalloutAccessoryView`
  - `rightCalloutAccessoryView`



Annotating maps

- The annotations that can be added to a map view are object that conform to the `MKAnnotation` protocol
- Map view has a readonly property called `annotation` which holds all the annotations currently in the map
- An `MKAnnotation` has an associated `MKAnnotationView` which is actually displayed on the screen
- The `MKAnnotation` protocol defines:
  - a required property `coordinate` of type `CLLocationCoordinate2D`
  - two optional properties `title` and `subtitle` of type `NSString*`



Annotating maps

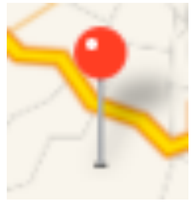
- Since the map view's **annotation** property is readonly, proper methods must be used to add them to and remove them from a map view

```
- (void)addAnnotation:(id<MKAnnotation>)annotation

- (void)addAnnotations:(NSArray *)annotations

- (void)removeAnnotation:(id<MKAnnotation>)annotation

- (void)removeAnnotations:(NSArray *)annotations
```

- For performance reasons, a good practice is to put all annotations on the map right away (if possible)
- **MKAnnotationView** instances are reused by **MKMapView**, similarly to what **UITableViews** do with **UITableViewCells**

Annotating maps

- The standard way to put annotation on a map is by using the `MKPinAnnotationView`



- It is possible to use any subclass of `MKAnnotationView` to customize the look of the annotation view
- When an annotation is tapped, a callout might be displayed if its `canShowCallout` property has been set to `YES`

MKMapViewDelegate

- The `MKMapView` has a `delegate` property that can be used to set a delegate object that can optionally receive events related to the map view and be responsible to provide concrete `MKAnnotationView` instances for a given annotation
- The delegate must conform to the `MKMapViewDelegate` protocol
- The methods of the protocol allow to:
  - respond to map position changes
  - respond to loading map data events
  - respond to user location changes
  - manage annotation views (can be used to customize the annotation's callout)
  - respond to annotation views drag events
  - respond to selection of annotation views (can be used to lazily initialize the callout's subviews)

Customizing annotation views

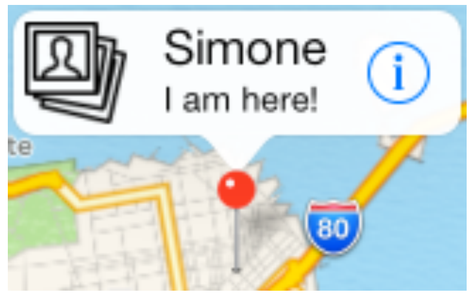
```
- (MKAnnotationView *)mapView:(MKMapView *)mapView
              viewForAnnotation:(id<MKAnnotation>)annotation
```

- This method gets called any time an annotation must put on the map view
- It asks to provide a custom MKAnnotationView for the given annotation
- It is very similar to `cellForRowAtIndexPath:` in `UITableViewDataSource`
- This is where it is possible to customize the annotation view

Customizing annotation views

```
- (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;
}
```

Customizing annotation views




```
- (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;
}
```

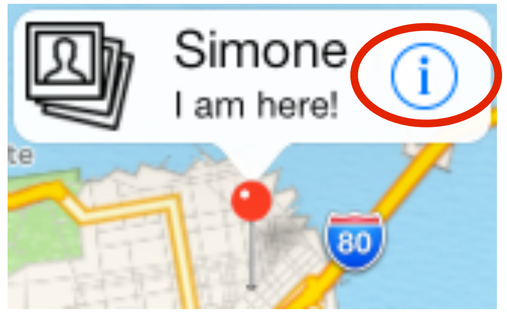
Responding to selection of an annotation view

- The delegate gets notified when an annotation (the pin) is selected
- Typically, if the `canShowCallout` property has been set to `YES`, a callout will then be shown
- This is a good place to perform lazy initialization of the content of the callout so that it is loaded only when requested

```
- (void)mapView:(MKMapView *)mapView didSelectAnnotationView:(MKAnnotationView *)view{

    if([view.leftCalloutAccessoryView isKindOfClass:[UIImageView class]]){
        UIImageView *imageView = (UIImageView *)view.leftCalloutAccessoryView;
        imageView.image = ...; //load from network using GCD (not in main queue!)
    }

}
```

be aware of annotation reuse!

Responding to tap in the callout



- If an accessory view of the callout is a UIControl, the delegate gets notified when the control is tapped with the following method

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

- At this point it is possible to segue to another view controller (e.g. to show some details) but this cannot be done in storyboard as usual since there is no object in storyboard that represents this callout

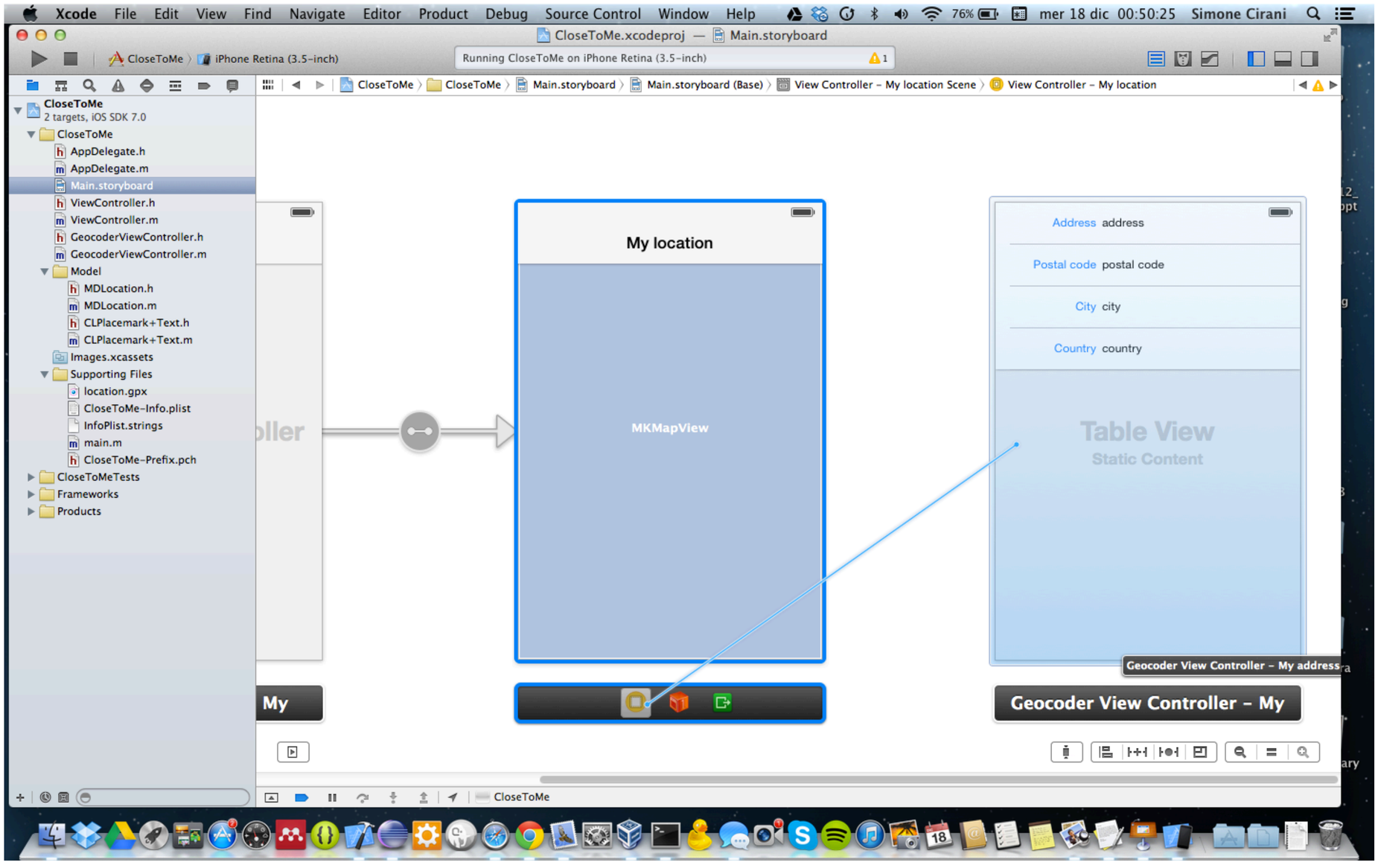
- To cope with this problem, we need to segue programmatically

Segueing programmatically

- Sometimes it is not possible to define a segue directly in storyboard just because the element that we are segueing from does not exist in storyboard, such as a callout of a map annotation
- It is of course possible to segue from such an element, but it has to be done programmatically
- The procedure to segue from code is:

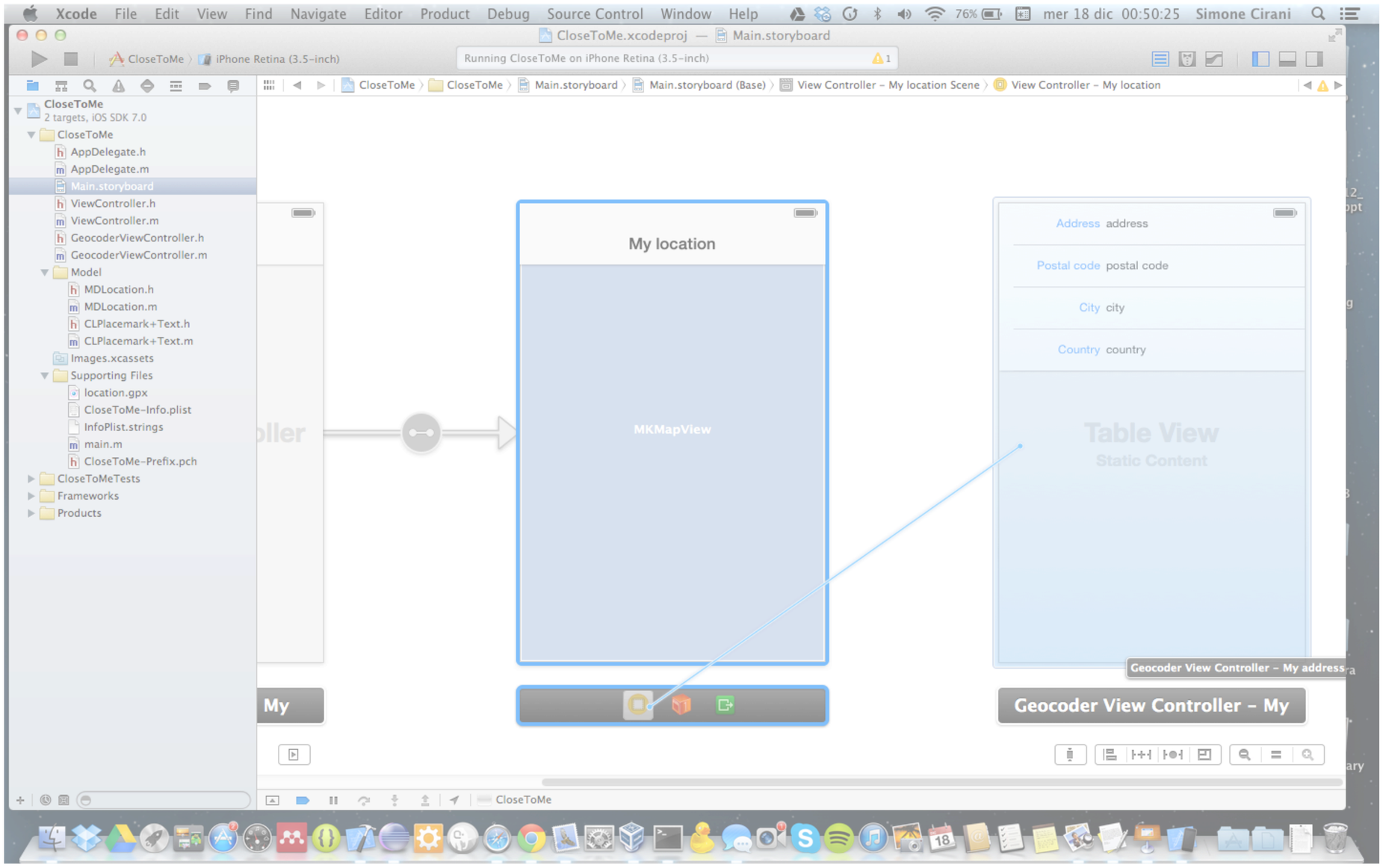
1. in storyboard, create a (manual) segue by control-dragging from the origin view controller (not a specific control in the view) to the destination view controller
2. assign an identifier to the segue in order to be able to refer to the segue in code
3. execute the `UIViewController`'s `performSegueWithIdentifier:sender:` method specifying the identifier given in step 2 and setting the correct sender

Segueing programmatically

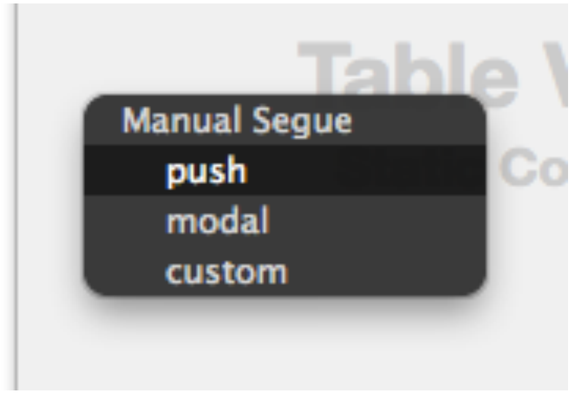


In storyboard, create a (manual) segue by control-dragging from the origin view controller to the destination view controller

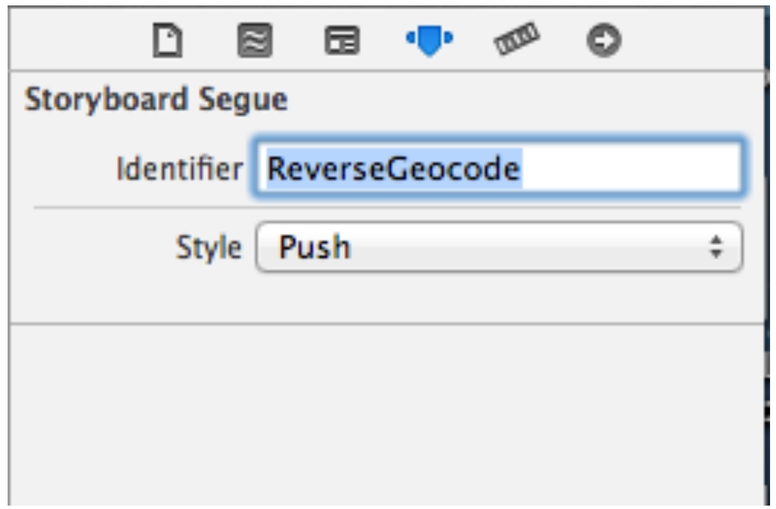
Segueing programmatically



In storyboard, create a (manual) segue by control-dragging from the origin view controller to the destination view controller



Segueing programmatically



Assign an identifier to the segue in order to be able to refer to the segue in code

Segueing programmatically

Call `performSegueWithIdentifier:sender:` method specifying the identifier and setting the correct sender

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

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

}
```

Working with JSON

- Working with JSON in iOS is pretty easy
- The `NSJSONSerialization` class provides methods to serialize objects to JSON strings and deserialize JSON strings to objects
- An object that may be converted to JSON must have the following properties:
  - The top level object is an `NSArray` or `NSDictionary`
  - All objects are instances of `NSString`, `NSNumber`, `NSArray`, `NSDictionary`, or `NSNull`
  - All dictionary keys are instances of `NSString`
  - Numbers are not NaN or infinity

Serialize to JSON

```
id obj = ...;
NSError * error;
if([NSJSONSerialization isValidJSONObject:obj]){

    NSData *data = [NSJSONSerialization dataWithJSONObject:obj
                                                   options:0
                                                     error:&error];


}
```

- It is best to check whether an object can be serialized with the `isValidJSONObject:` method
- The method return JSON data for `obj`, or `nil` if an error occurs
- The resulting data is a encoded in UTF-8
- If an error occurs, the `error` variable holds the error

Deserialize from JSON

```
NSData *data = ...;
NSError * error;
id obj = [NSJSONSerialization JSONObjectWithData:data
                                         options:NSJSONReadingAllowFragment
                                           error:&error];
```

- The returned object may be a `NSArray*` or a `NSDictionary*`, or `nil` if an error occurs
- If an error occurs, the `error` variable holds the error

Deserialization example

- The following URL exposes a service which returns some JSON data:

http://mobdev.ce.unipr.it/2013/index.php/service

- The JSON data has the following format:

```
{
timestamp: 1387350841,
course: "Mobile Application Development 2013",
teachers:
    [
        {
        name: "Simone Cirani",
        status: "Teaching iOS",
        photo: "http://www.tlc.unipr.it/cirani/images/simone.png",
        latitude: 44.12345,
        longitude: 10.12345
        },
        {
        name: "Marco Picone",
        status: "Teaching Android",
        photo: "http://wasnlab.tlc.unipr.it/people/picone/img/picone_150_200.png",
        latitude: 44.12345,
        longitude: 10.12345
        }
    ]
}
```

Deserialization example

- Suppose that we are interested in parsing the “teachers” field and create objects of a class `Person` defined as follows:

```
@interface Person : NSObject

@property (nonatomic,strong) NSString *name;
@property (nonatomic,strong) NSString *status;
@property (nonatomic,strong) NSURL *imageURL;
@property (nonatomic,readwrite) double latitude;
@property (nonatomic,readwrite) double longitude;

@end
```

Deserialization example

- The JSON object and the class are mapped as follows:

```
{
timestamp: 1387350841,
course: "Mobile Application Development 2013",
teachers:
    [
        {
        name: "Simone Cirani",
        status: "Teaching iOS",
        photo: "http://www.tlc.unipr.it/cirani/images/
simone.png",
        latitude: 44.12345,
        longitude: 10.12345
        },
        {
        name: "Marco Picone",
        status: "Teaching Android",
        photo: "http://wasnlab.tlc.unipr.it/people/
picone/img/picone_150_200.png",
        latitude: 44.12345,
        longitude: 10.12345
        }
    ]
}
```

```
@interface Person : NSObject

@property (nonatomic,strong) NSString *name;
@property (nonatomic,strong) NSString *status;
@property (nonatomic,strong) NSURL *imageURL;
@property (nonatomic,readwrite) double latitude;
@property (nonatomic,readwrite) double longitude;

@end
```

Deserialization example

- The following snippet shows how to deserialize to the instance of `Person`:

```
NSData *data = [NSData dataWithContentsOfURL:url];
NSError * error;
NSDictionary *dictionary = [NSJSONSerialization JSONObjectWithData:data
options:NSJSONReadingAllowFragments error:&error];
if(!error){
    NSArray *teachers = [dictionary objectForKey:@"teachers"];
    [teachers enumerateObjectsUsingBlock:^(id obj, NSUInteger idx, BOOL *stop) {
        if([obj isKindOfClass:[NSDictionary class]]){
            NSDictionary *pDict = (NSDictionary *)obj;
            Person *person = [[Person alloc] init];
            person.name = pDict[@"name"];
            person.status = pDict[@"status"];
            person.imageURL = [NSURL URLWithString:pDict[@"photo"]];
            person.latitude = [pDict[@"latitude"] doubleValue];
            person.longitude = [pDict[@"longitude"] doubleValue];

        }
    }];
}
```

Subscripting

```
person.name = pDict[@"name"];

person.status = pDict[@"status"];

person.imageURL = [NSURL URLWithString:pDict[@"photo"]];

person.latitude = [pDict[@"latitude"] doubleValue];

person.longitude = [pDict[@"longitude"] doubleValue];
```

This is called “subscripting”: a faster way to access elements of arrays and dictionaries without the need to use `objectAtIndex:` or `objectForKey:`

Subscripting

Subscripting with `NSDictionary`

```
person.name = pDict[@"name"];
```

```
person.name = [pDict objectForKey:@"name"];
```

Subscripting with `NSArray`

```
id obj = locations[0];
```

```
id obj = [locations objectAtIndex:0];
```

Mobile Application Development

Lecture 21
Core Location and Map Kit