Фреймворк – набор связных классов, которые можно использовать в target. Cacoa touch – это колллекция фреймворков.

По умолчанию при создании IOS приложения в target (Build Phases в графе Link Binary with Labraries) три фреймворка – 1). CoreGraphics – графическая библиотека. 2). UIKit – классы, которые составляют пользовательский интерфейс. 3). Foundation – содержит базовые классы типа NSString NSArray и тому подобное.

Создаваемому приложению также необходим фреймворк Core Location – содержит классы для определения местоположения устройства.

Core Location

Данный фреймфорк содержит классы позволяющие приложению определять географическое местоположение устройства.

Неважно какой тип IOS устройства используется.

Классы в Core location начинаются с префикса CL (у UIKIT фреймворка это префикс UI у Faundation - ТЫ)

При использовании классов данного фреймворка необходимо включить его в заголовочный файл.

Подключаем заголовочный файл фреймворка в делегат приложения и создаем экземпляр класса

CLLocationManager (взаимодействует с железом в устройстве для определения местоположения)

У него есть свойства, два из них меняем 1). distanceFilter – указывает расстояние в метрах, при смещении устройства на которое приложению будет сообщаться, что местоположение изменилось. 2). Свойство desiredAccuracy – сообщает (менеджеру местоположения) с какой точностью должно быть определено местоположение.

Данные свойства необходимы для нахождения компромисса между точностью, временем и уровнем заряда батареи необходимым для определения местоположения.

В конечном счете точность зависит от типа устройства, наличия сотовых вышек, спутников, известных точек wi-fi.

В .m делегата приложения (в методе по инициализации) создаю locationmanager = [[ alloc] init]

И устанавливаю его свойства для получения наиболее точного местоположения с частым обновлением.

**Receiving updates from CLLovationManager**

Если на данном шаге запустить данный код, то locationManager определит местоположение, но вы нигде не удидете данную информацию.

При каждом определении своего местоположения CLLocationManager отсылает своему делегату сообщение:

locationManager:didUpdateToLocation:FromLocation:

(одним из аргументов данного сообщения является экземпляр CLLocation (содержит широту и долготу устройства и высоту над уровнем моря)

(Figere 4.4)

В нашем случае делегатом для CLLocationManager будет делегат приложения

Поэтому пишем:

[locationmanager setDelegat:self];

Далее прописываем реализацию метода locationManager:didUpdateToLocation:FromLocation:

(но самое удивительное, что не объявляем в хедере делегата приложения, что мы поддерживает протоком LCLocationManager)

**ГЛАВА 5**

**Диаграммы объектов**

Схема стр 98

1). Несколько экземпляров MKAnnotationView – значки на MKMapView

2). MKMapView – отображает карту и метки в известных местах

3). UIActivityIndicatorView – показывает.ю что устройство работает

4). UITextField – окно для ввода текста для метки текущего местоположения

Моделями на низком уровне являются CLLocationManager (взаимодействует с железом для определения местоположения) и MapPoint

В середине находится контроллер – делегат приложения также являющийся делегатом MKMapView, UiTextField, CLLocationManager. Отвечает за обработку запросов от объектов, обновление пользовательского интерфейса, обработку обновлений.

**MapKit FrameWork**

Core Location – говорит где мы находимся, а MapKit – показывает нам “мир”. Экземпляры Класса MKMapKit – отображают карту, отображают аннотацию.

В приложении подключаем фреймворк MapKit (и добавляем хедер в хедер делегата приложения)

Объявляем все элементы, что будут отображаться на дисплее с добавочным(IBOutlet) – это говорит, что я буду использовать xib (в смысле не руками прописывать, а добавлять на визуальные модели)

IBOutlet MKMapView \*dsfsf

IBOutlet UIActivityIndicatorView \*fgfdgrfg

IBOutlet UITextField \*fgfbfg

После этого создаю xib (FN+N user interface window)

Далее открываю свой xib в file’s Owner в custom Class ставлю UIApplication

Далее добавляю объект (справа внизу) и ставлю custom class мой делегат приложения

Далее добавляю справа снизу все view которые надо далее щелкаю на объект(в моем случае делегат приложения) созданный в предыдущем пункте правым кликом и добавляю связи с моими view, а на сами view (визуальные) тоже щелкаю правым кликом и устанавливаю их delegeta если надо.

Далее перехожу к делегату MAPVIEW мне надо, чтобы на карте при запуске приложения показывалось текущее местоположение пользователя для этого в делегате приложения в методе инициализации пишу:

[worldMap setShowsUserLocation:YES]; и убираю предыдующую строчку (была нужна для определения местоположения ранее)

[locationmanager startUpdatelocation]; - она лишняя так как MaPView \*worldMap знает как работать с Location manager

Далее будем разбираться как сделать увеличение масштаба карты при определении местоположения (устанавливаем поддержку протокола MKMapViewDelegate делегатом приложения)

**Работа с документацией**

Документация состоит из четырех частей: Справочник по API (API Reference), системы guides (system guides), инструменты guides (tools guides), примеры кода (sample code).

Справочник по API (API Reference) – содержит информацию по каждому классу, протоколу, функции, структуре методу и так далее и тому подобное используемому в Cocoa Touch

Гайды по системе (system guides) – содержат обзоры высокого уровня (как видит пользователь) и обсуждение концепций (как это может выглядеть)

Гайды по инструментарию (tools guides) – содоржат справочную информацию по XCOde и остальной части пакета инструментов для разработчика.

Ищу метод в протоколе для MapView – надо производить зуммирование при определениии местоположения. Данный метод mapView: didUpdateUserLocation:

Добавляю его в .m делегата приложения

Смысл данного метода: когда MKMapView определяет местоположение пользователя, то он посылает своему делегату данное сообщение содержащее экземпляр класса MKUserLovation.

Теперь нам надо только описать данный метод, а именно сделать зуммирование места на карте с текущим местоположением пользователя.

Когда MKMapView определяет местоположение он автоматом не увеличивает его, но логично, что в нем должен быть метод для осуществления данных действий для этого лезем в документацию по MKMapView нахожу пункт Manipulating the Visible Portion of the Map и открываю свойство (MKCoordinateRegion) щелкаю по нему и оказывается это структура содержащая две переменные:

Typedef struct {

CLLocationCoordinate2D central; \\ сами координаты в центре карты

MccoordinateSpane spane; [\\определяет](file:///\\определяет) уровень зуммирования

} MKCoordinateRegion

Чтобы получить координаты региона мне надо опять залезть в документацию и найти функции для MKCoorfinateRegion (просто вбиваю это в поиск в reference и смотрю существующие функции) одной из них оказывается MKCoordinateRegionMakeDistance – позволяет указать область с CLLocationCoordinate2D и зуммирование по направлениям север-юг, восток-запад (В моем случае будет использовано 250 на 250) но для координации мне надо получить местоположение пользователя для этого:

Смотрим по документации

Когда MKMapView определяет текущее местоположение оно отправляет сообщение делегату (писал ранее) содержащее MKUserLocation которое содержит свойство CLLocation которое в свою очередь имеет свойство CLLocationCoordinate2D А это то что нам нужно для зуммирования:

По логике получается, что мы может использовать информацию получаемую в сообщении отправляемом делегату MKMapView при определении местоположения для передачи ее в свойство MKCoordinateRegion чтобы дальше отправить ее в функцию MKCoordinateRegionMakeDistance которая и осуществит зуммирование по двум направлениям (север-юг, восток-запад)

Исходя из всего выше сказанного получается следующий ход действий для каждого обновления местоположения пользователя то есть для каждой отправки делегату приложения сообщения mapView: didUpdateUserLocation:

1). Надо вытащить из данных отправляемых с данным сообщением (MKUserLocation) необходимые нам данные CLLocationCoordinate2D

-(void)mapView (MKMapView \*)mv didUpdateUserLocation: (MKUserLocation \*) u

{

CLLocationCoordinate2d loc = [u coordinate];

2). Надо создать переменную в которую будем заносить результат функции MKCoordinateRegionMakeDistance – она и будет осуществлять зуммирование

MKCoordinateRegion region = MKCoordinateRegionMakeDistance(loc, 250, 250);

3). Нашей view карте отправляем сообщение с полученным результатом функции

[worldview setRegion: region animated:YES];

**Your own MKAnnotation**

**MKAnnotationView Хранит указатель на MKAnnotation**

**Мы создаем новый класс MapPoin который соответствует MKAnnotation, когда пользователь делает тег места будет создаваться экземпляр MapPoint, который будет представлен на карте**