Core Location

Core Location

Core Location is a framework that provides the following services for a device to determine:

- its geographic location
- Its altitude
- Its orientation
- Its position relative to a nearby beacon.

The framework uses all available onboard hardware to do this.

- Wi-fi and/or cellular
- GPS
- Bluetooth
- Magnetometer
- Barometer

Core Location (cont.)

To begin using Core Location services, you must first import the framework, and then create a `CLLocationManager` object to start, stop, and manage the delivery of location-related events to your app.

```
import CoreLocation
     . . .
var locationManager = CLLocationManager()
```

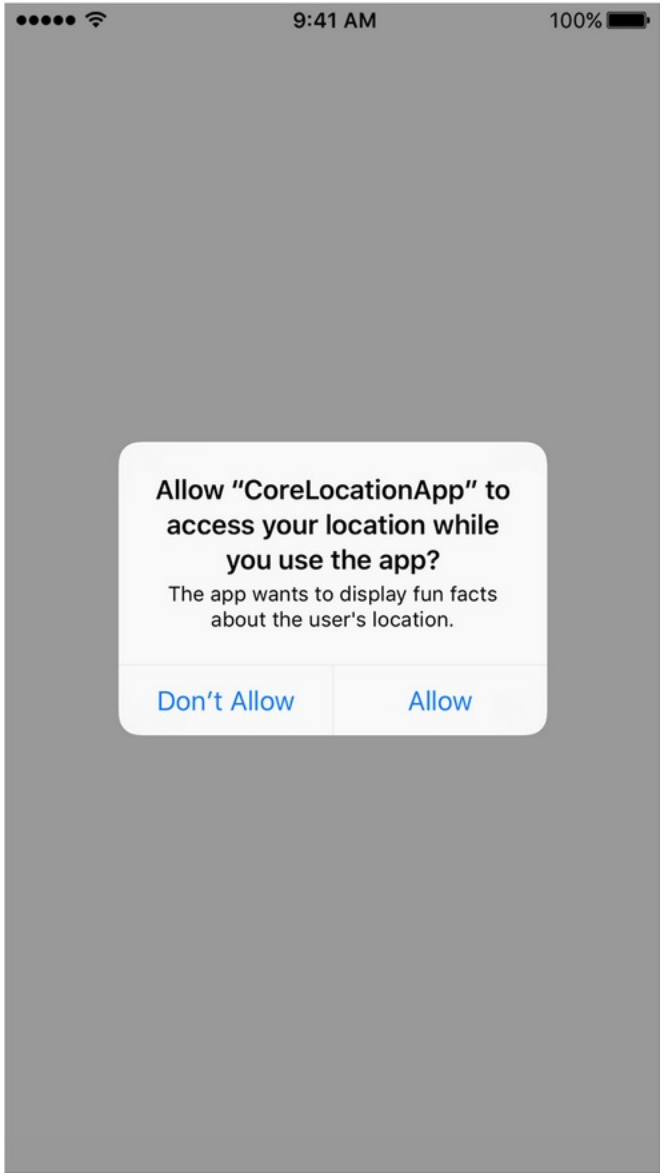
Then you must ask the user permission to use Location Services.

```
locationManager?.requestWhenInUseAuthorization()
```

Core Location (cont.)

This will cause the system to bring up an alert requesting access from the user.

Once the user responds, your app records the user's response, and will not display this alert again.



MapKit

`MapKit` is an API that makes it easy to display maps, plot locations, draw routes and other shapes on top of the map.

- Elements drawn on top of the map are called *overlays.*

You can choose what kind of map is rendered:

- Standard
- Satellite
- Hybrid

You can define pins (annotations) for locations of interest.

- You can define the color of pins.
- You can define *callouts* that show some basic identifying information when the pin is touched.

MapKit (cont.)

In order to display a map, you must:

- Import `MapKit`
- Include an `MKMapView` somewhere in a view controller

There are methods available for a number of things, including:

- Adding overlays
- Setting center of the map and the currently visible region of the map
- Changing the map type
- Using coordinates (latitude, longitude) for pin location and/or for a travel path
- Adding annotations

Core Audio

Core Audio

Core Audio refers to a group of services that allow you to play audio in your application.

Just like with everything else, Apple provides a number of these services for you; or, you can use third-party frameworks.

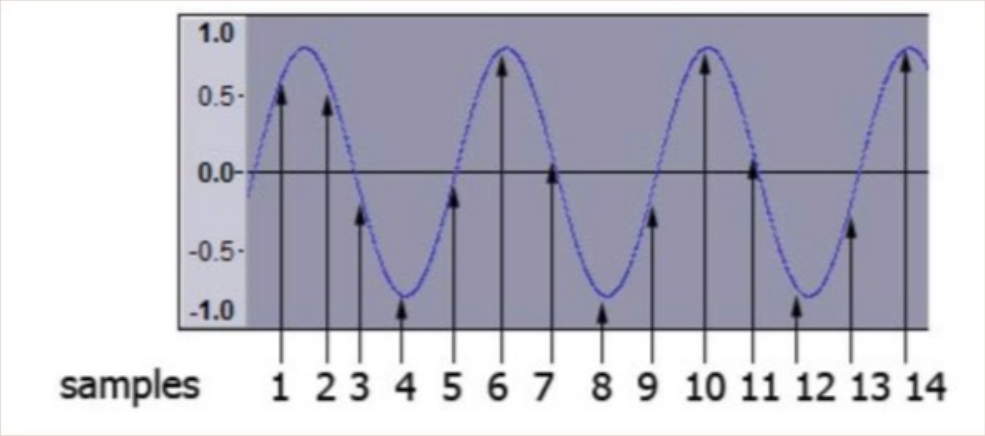
In order to use these frameworks, you need to understand:

- Bit rates and sample rates
- File formats and data formats

Sample Rates

Sample rates refer to how often a real-world (analog) sound wave is sampled to make a digital signal

- A sample is a measure of a sound, at a particular point in time
- Higher sample rates produce more accurate representations of the original sound
- 44,100 Hz (cycles per second) is used for CD audio. This is twice the rate of human hearing (20-20,000 Hz)
- Can handle some gaps via interpolation



Bit Rates

CD-quality audio is also called 16-bit audio, because that's how many bits are used to represent each sample. The more bits, the more accurate the representation. So you have to take into account both the sample rate (samples per unit time) and the number of bits per sample, giving you a total number of bits per unit time.

This is referred to as the *bit rate.*

- Usually expressed in bits-per-second (bps)
- Higher bit rates produce larger files but higher quality
- Lower bit rates produce smaller files but lower quality
- Choose a bit rate appropriate for the type of audio file; (the lowest bit rate necessary to satisfy the quality goals of your app)

File and Data Formats

File formats encapsulate the audio data. Each file format has a list of data formats it can encapsulate.

The *file* format of most interest to us as iOS programmers is CAF – Core Audio File Format.

The *data* formats of most interest to us are:

- Linear PCM – Linear Pulse-Code Modulation
  - Uncompressed data
  - Most efficient playing
  - The preferred variant for the iPhone is *little-endian integer 16-bit,* or LEI16 for short
- MP3

Core Audio Frameworks

Frameworks that allow you to play audio in your application:

- System Audio Services
  - Collection of C functions
  - Used to play sounds of 30 seconds or less
- AV Audio Player
  - Recommended for typical audio-only tasks
  - Objective-C library
  - Can handle interruptions such as incoming phone calls
- Media Player
  - Handles both audio and video
- OpenAL
  - Third-party cross-platform audio API
  - Offers maximum control, but there is a big learning curve

Creating Audio in an App

Steps to add audio to an app:

- Create or copy audio files
- The recommended format is CAF. If necessary, convert audio files to CAF format
- Add a framework to the project
- Add audio files to project
- Add code to play / stop / manipulate audio

Tools to create audio:

- Audacity – free download at http://www.audacityteam.org/download/
- GarageBand – free (preinstalled on Apple computers)

Apple Tools to Manipulate Audio Files

Command-line tools on the Mac platform:

`afplay` – plays an audio file

```
$ afplay myFile.wav
```

`afconvert` – converts an audio file from one format to another

```
afconvert -d [out data format]
          -f [out file format ]
          [inFile] [outFile]
```

```
$ afconvert -d LEI16 -f 'caff' myFile.wav myFile.caf
```

`afinfo` – displays information about an audio file

Resources

For additional information:

iOS Multimedia Programming Guide: Using Audio

https://developer.apple.com/library/ios/documentation/audiovideo/conceptual/multimediapg/usingaudio/usingaudio.html