# AVFoundation

* AVFoundation is under UIKit
* AVFoundation can be broken to the following:
  + Playback
  + Capture
  + Edit
  + Read Write
* Playback
  + In this you take data from a movie file, playing through file and display on the screen
  + AVAsset – This is the movie file
  + AVPlayer - AV controls the playback itself
  + AVPlayerLayer – Main class involved in displaying images
* AVAsset
  + Properties of AVAsset
    - This is a timed audiovisual media
    - Video, movie, song.
    - It can be on the device or can be fetched over a network
    - It can be finite or unbounded like in webcam.
  + Where to get one
    - Application bundle
    - Media Library
    - Photo Library
    - Network URL
  + Structure of AVAsset
    - AVAsset is made of tracks called AVAssetTract like video or audio track
    - AVAssetTrackSegment is in each track
  + It may take time to load the asset because it might be large or be a network file
    - Asynchronous key-value loading protocol tells when the asset is available
* AVPlayer
  + Controller objects
    - Play
    - Pause
    - Rate
  + AVPlayer cannot be directly connected with AVAsset
    - AVPlayerItem is used to connect them
* AVPlayerLayer
  + Subclass of UIView
  + Shows AVPlayer on screen
* Playback Steps
  + Create an asset
  + Ask for the tracks of asset to be loaded
  + Once loaded, create a player item for the asset
  + Attach the item to a player
  + Attach the player to a player layer
  + Wait until the item is ready to play then play

# Code of AVFoundation for video:

* Create an asset:
  + AVURLAsset \*asset = [AVURLASSet URLAssetWithURL:fileURL options:nil];
* Ask for tracks to be loaded:
  + // To ask for tracks and other things to be loaded, the following array is made. These are the property names.
  + NSArray \*requestedKeys = [NSArray arrayWithObjects:@tracks, @”playable”, nil];
  + [asset loadValuesAsynchronouslyForKeys:
  + ^{
    - // Main\_queue for AVPlayer
    - dispatch\_async( dispatch\_get\_main\_queue(),
    - ^{
      * // Completion block goes here
    - });
  + }];
* Once loaded, create a player item for the asset
  + // The following code comes in main queue
  + ^{
    - NSError \*error = nil;
    - // Check status of track
    - AVKeyValueStatus status = [asset statusOfValueForKey:@”tracks error:&error];
    - // If the track is loaded
    - if (status == AVKeyValueStatusLoaded) {
      * // The following line creates a player item. Only the properties are loaded
      * self.playerItem = [AVPlayerItem playerItemWithAsset:asset]
* Attach the item to a player
  + - * // this is continued from the top.
      * // once we have the player item, we attach it to the player
      * self.player = [AVPlayer playerWithPlayerItem:playerItem];
* Attach the player to a player layer
  + - * // this is continued from the top
      * // attach the player to player layer.
      * [playerView setPlayer:player];
    - else {
      * // Didn't load, need to do something
    - }
  + };
* Wait until the item is ready to play
  + // Now we have objects hooked up. But the player item is not necessarily ready to play yet.
  + // We need to observe the status property of player item so we can detect when item is ready
  + // this is a file global
  + static void \*PlayerItemStatusContext = &PlayerItemStatusContext;
  + // this is in the completion handler
  + [playerItem addObserver:self forKeyPath:@"status" options:0 context: PlayerItemStatusContext];
* Update UI when playback is ready
  + - (void)observeValueForKeyPath:(NSString \*)keyPath ofObject:(id)object change:(NSDictionary \*)change context
  + {
    - if (context == PlayerItemStatusContext) {
      * dispatch\_async(dispatch\_get\_main\_queue(),
        + ^{

// Turn on your 'play' button...

// if status is AVPPlayerItemStatusReadyToPlay

* + - * + });
    - } else {
      * [super observeValueForKeyPath:keyPath ofObject:object change:change context:context];
    - }
  + }
* Play and Pause
  + - (IBAction)play:(id)sender
  + {
    - [player play];
  + }
  + - (IBAction)pause:(id)sender
  + {
    - [player pause];
  + }
* How to associate AVplayerLayer with UIView
  + @interface PlayerView:UIView {
  + }
  + @property (nonatomic, retain) AVPlayer \*player;
  + @end
  + @implementation PlayerView
  + // We make a UIView subclass and we make its layer class be AVPlayerLayer.
  + + (Class)layerClass {
    - return [AVPlayerLayer class]; // UIView will instantiate AVPlayerLayer
  + }
  + // When you fetch the view's player, we return the layer associated with AVPlayerLayer
  + - (AVPlayer\*)player {
    - return [AVPlayerLayer \*)[self layer] player];
  + }
  + // The view's player is being set as the layer of AVPlayerLayer
  + - (void)setPlayer:(AVPlayer \*)player {
    - [(AVPlayerLayer \*)[self layer] setPlayer:player];
  + }
  + @end
* When the playback has ended
  + // the completion handler
  + [[NSNotifinationCenter defaultCenter] addObserver:self selector:@selector(playerItemDidReachEnd;) name:AVPlayerItemDidPlayToEndTimeNotification object:playerItem];
  + - (void)playerItemDidReachEnd:(NSNotification \*)notification {
    - // This method has to be made and every time the end notification is release it will come to this method.
    - // In case of rewind set time to start etc.AVURLasset