last updated : January 31, 2014

INTRODUCTION

Apple's video playing and streaming solution is very limited, it supports only playing specific video file formats, http streaming & h264 video codec and its source is closed rendering the task of streaming and playing video a very hard one.
If you need a custom video player that play divx, mkv, flv, etc… or uses other popular streaming protocols such as MMS, RTSP, RTMP, or if you play video files/stream from an http server but using different audio/video codecs or you play video files/stream from a server which you can use also the apple API (MPMoviePlayerController or AVPlayerItem), but need to modify the video player, like buffering duration, audio or video raw data and so on then that means you will need **VideoKit**

FEATURES

- Play most popular media files locally that Apple doesn't support
- Stream from popular protocols (http, mms, rtsp & rtmp)
- Supports all popular audio & video codecs
- Supports mjpeg streams - mostly for ipcams
- Supports animated GIF files with full transparency
- 720p HD streams are supported by iPad 1 and above.
- Supports real time video effects with using pixel shaders
- Successful Audio & video syncing
- Very easy to use (very similar to Apple's MPMoviePlayerViewController API & MPMoviePlayerController)
- Look & feel like Apple's MPMoviePlayerViewController & MPMoviePlayerController
- Works with Wifi & 3G
- Shows detailed information about stream (audio & video codecs,total streamed data in bytes, connection type)
- Works on all iOS(3GS, iPhone 4/4S, iPhone 5 and all iPads) devices & supports all screen types and rotations
- Supports pausing stream
- Supports streaming in background
- Robust error handling
- Supports audio resampling
- Supports seeking in local & remote files
- Airplay
- Showing files/streams both in fullscreen and embedded
- Supports multiple players on same view
- Player is now an NSObject instance, so can be used without UI
- Supports transition from embedded/fullscreen to fullscreen/embedded
- Supports fullscreen, embedded and non UI control modes
- Supports initial playback time for files
- Supports looping for files
- Supports adjusting volume level for each player
- Supports changing audio streams in realtime
- Supports disabling audio stream in file/stream

REQUIREMENTS

iOS **5.0 SDK** or above is needed to compile VideoKit however, if you need under 5.0, VideoKit can be built for iOS 4.0 with small modifications.

The VideoKit SDK requires additionally the following Apple frameworks:

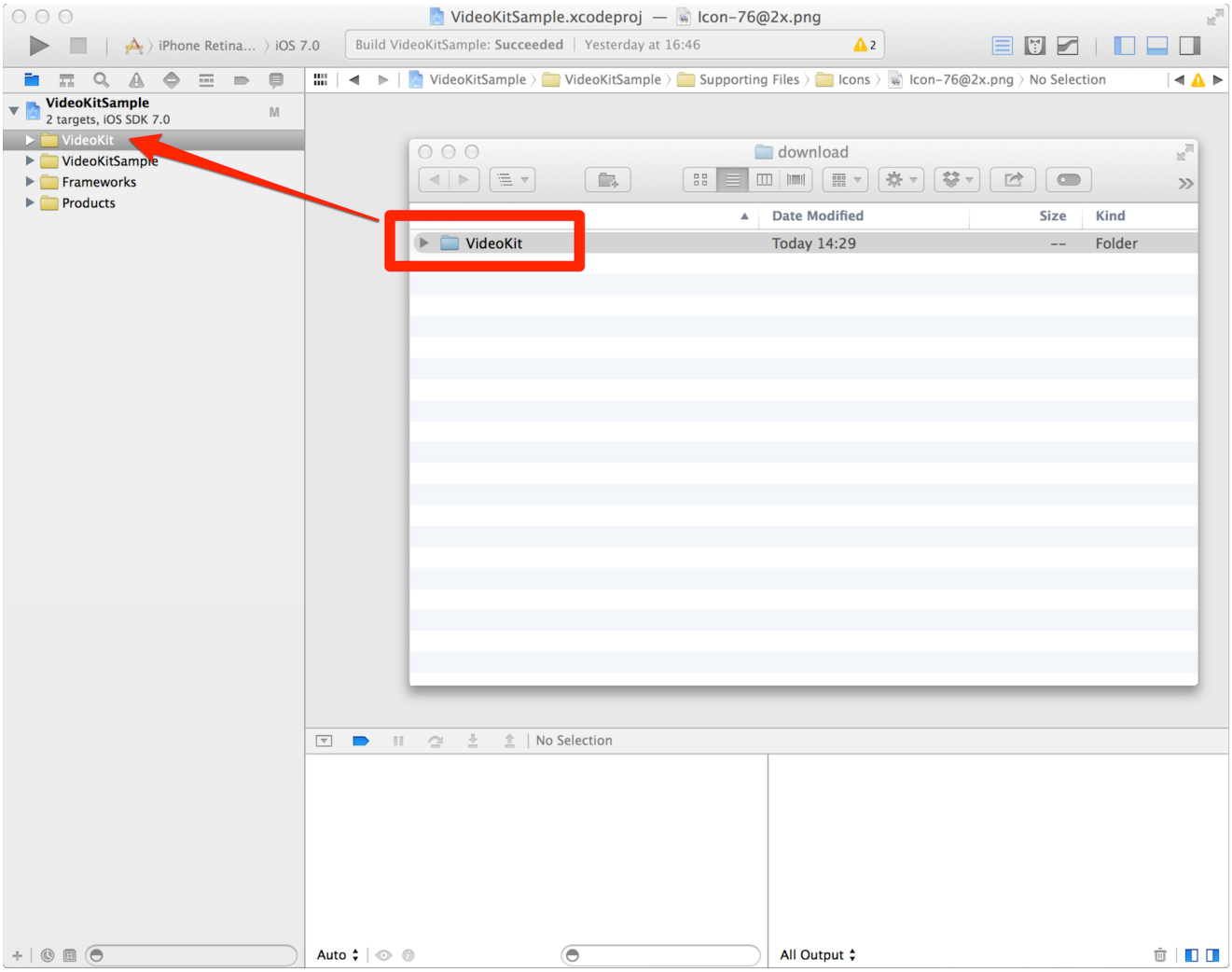
- SystemConfiguration (used in **Reachability** framework by Apple)
- MediaPlayer (needed by **MPVolumeView**)
- AudioToolbox (needed by **AudioUnit**)
- AVFoundation (needed by **AVAudioSession**)
- OpenGLES (using when **rendering** pictures to screen)
- QuartzCore (using to draw **StreamInfo view**)

and VideoKit also needs to these frameworks (for connecting to stream, decoding Audio&Video packets, etc...):

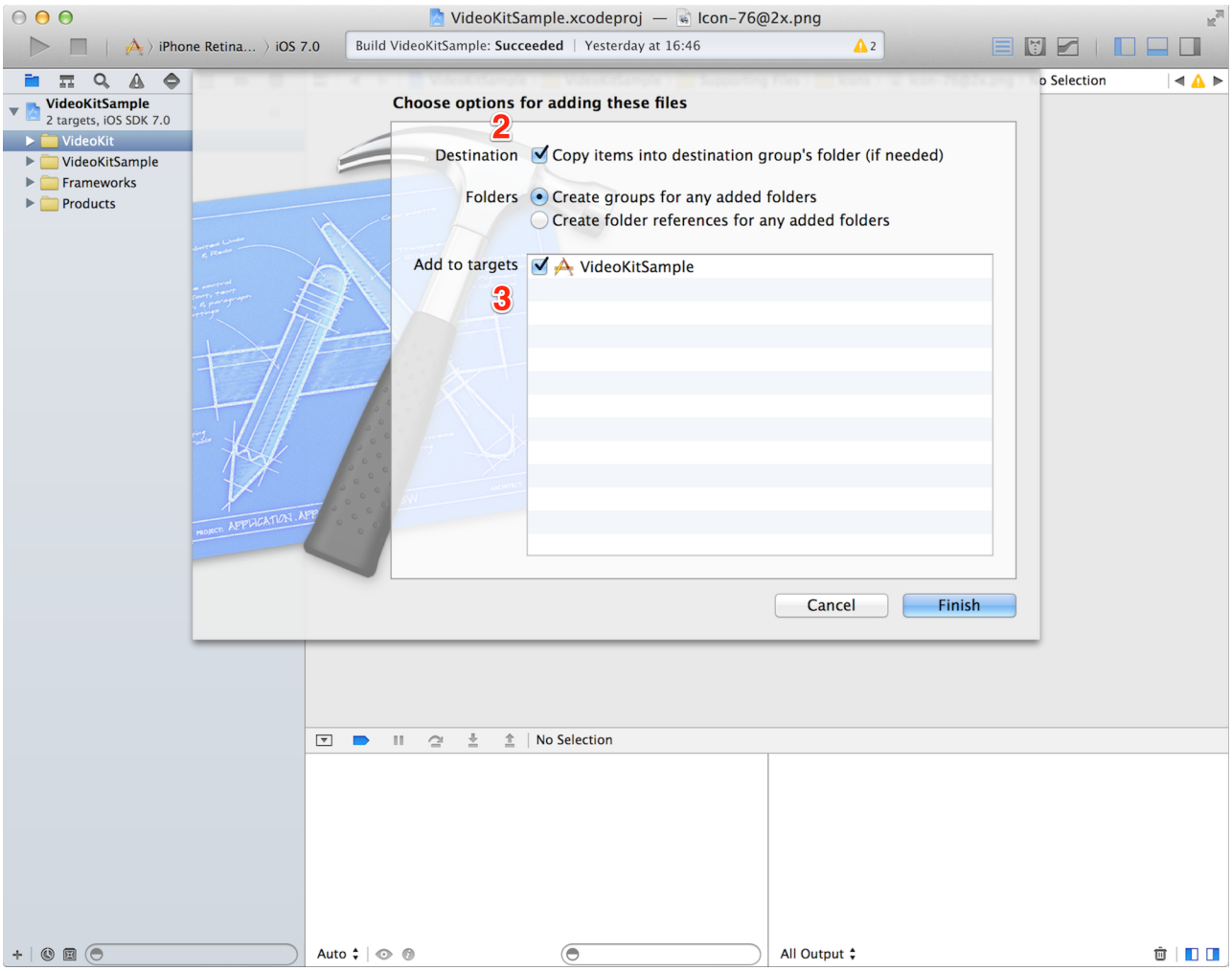
- libz
- libiconv
- libavcodec
- libavformat
- libavutil
- libswscale
- libswresample

INTEGRATION

Adding VideoKit to your Xcode project

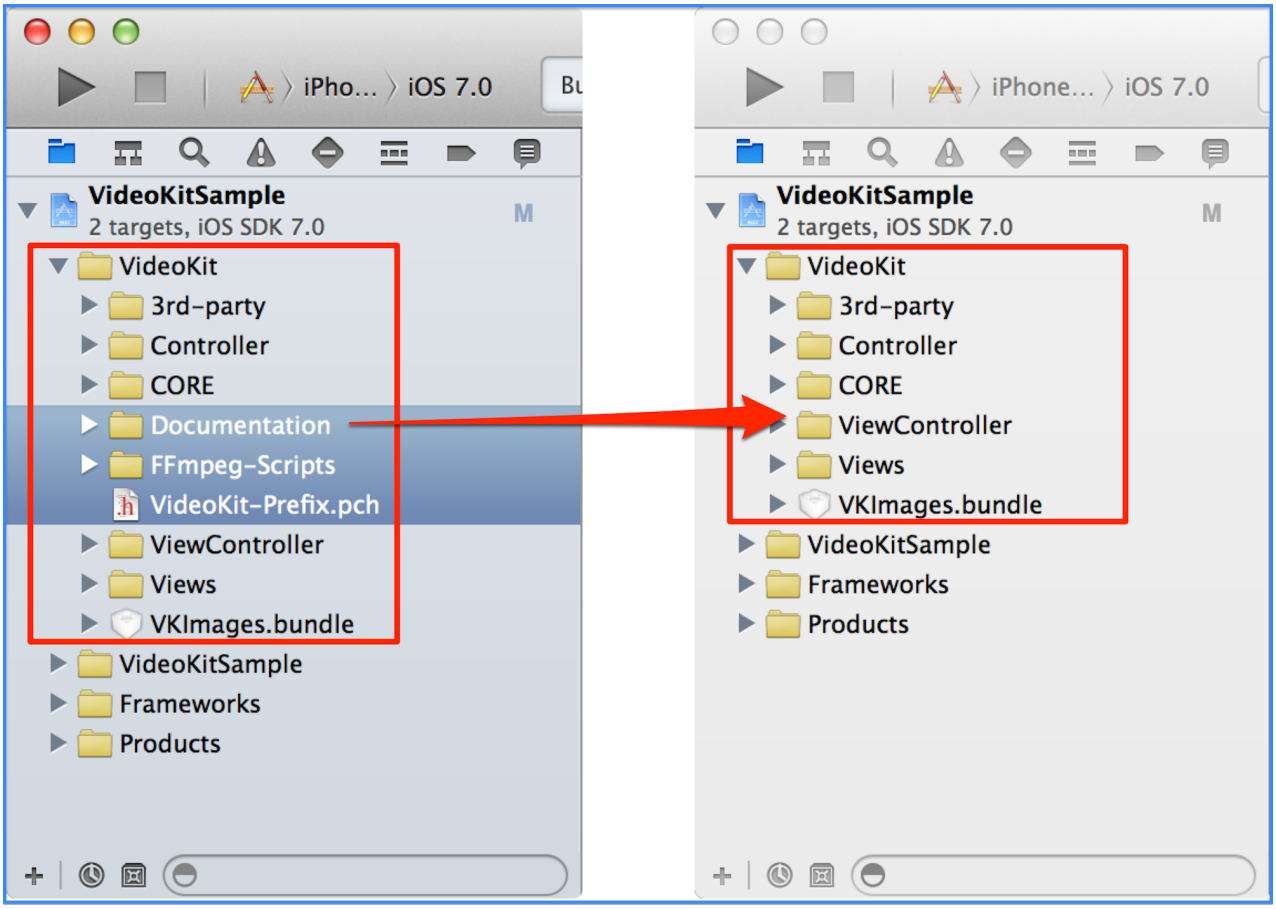


1. Go to VideoKit folder, and move VideoKit folder to your Xcode project. (Please note that, VideoKit folder must be added to **root** not under any folder)

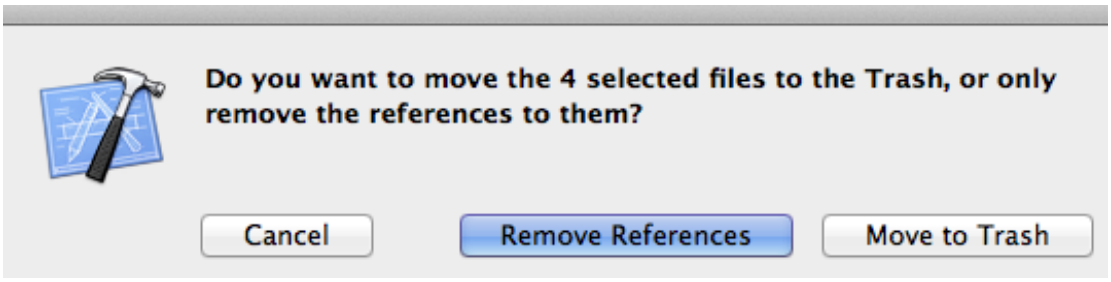


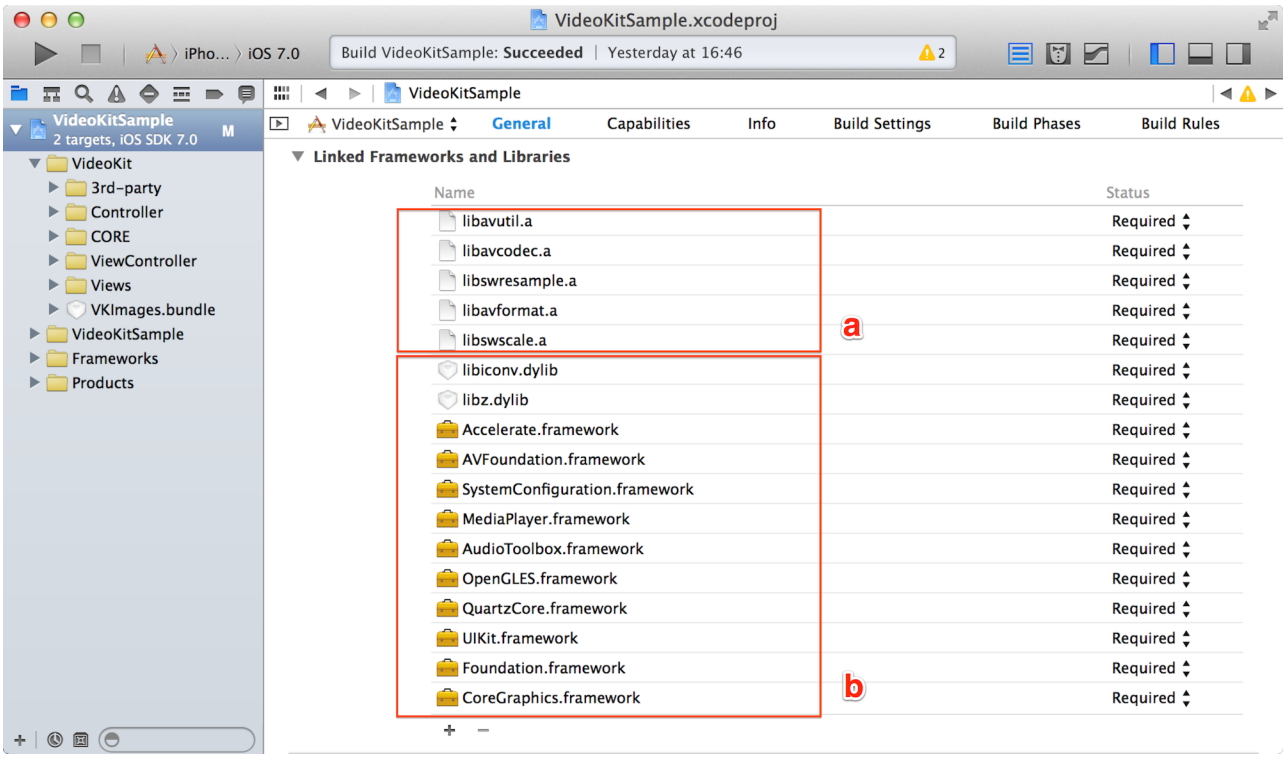
2. Be sure that ‘Copy items into destination group’s folder’ is checked

3. Be sure that ‘VideoKitSample’ target is checked



4. Remove selected unnecessary folders (Documentation, FFmpeg-Scripts) and the file (VideoKit-Prefix.pch). When asking about remove process, select ‘Remove References’. After doing this, our VideoKit folder is seen like above (at right).



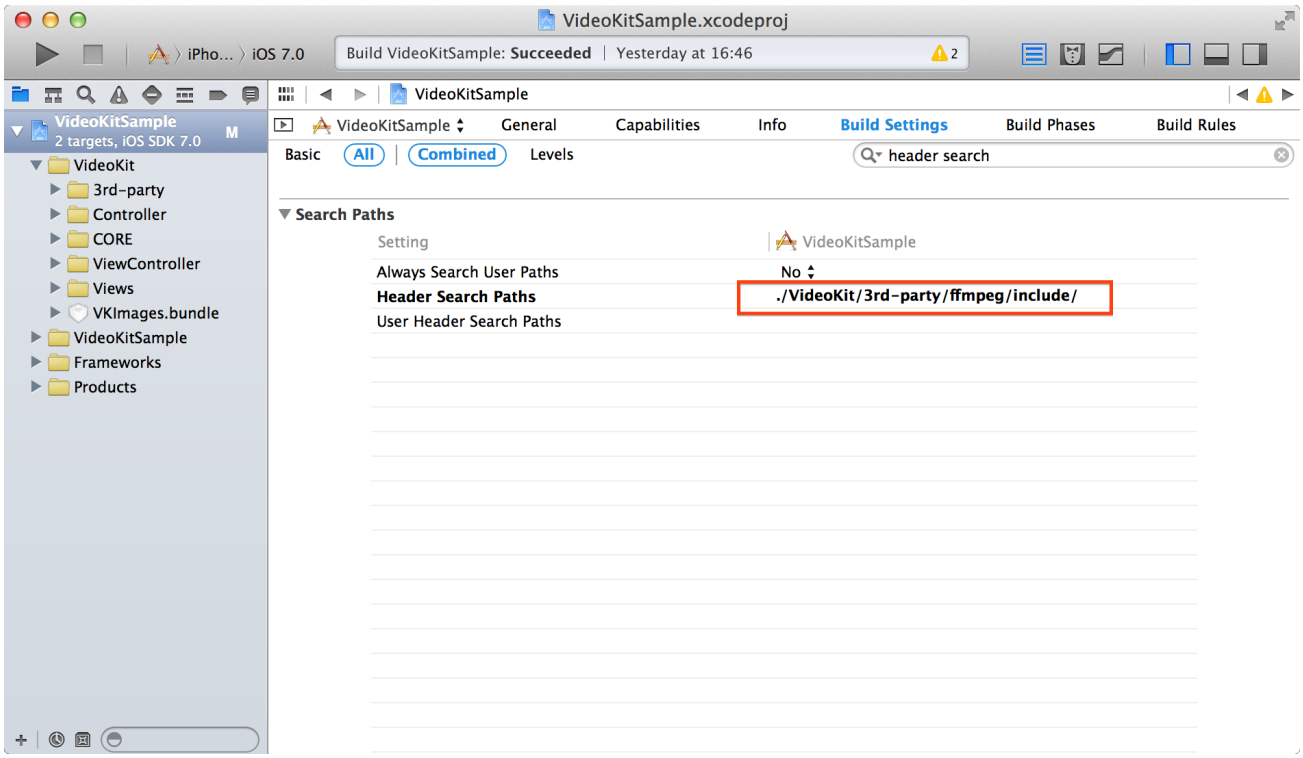


5. Select your target, and go to the **General** tab.

a. Those libraries (located in rectangle labelled as “a” in above picture) are **automatically added** when you add VideoKit folder to your xcode project.

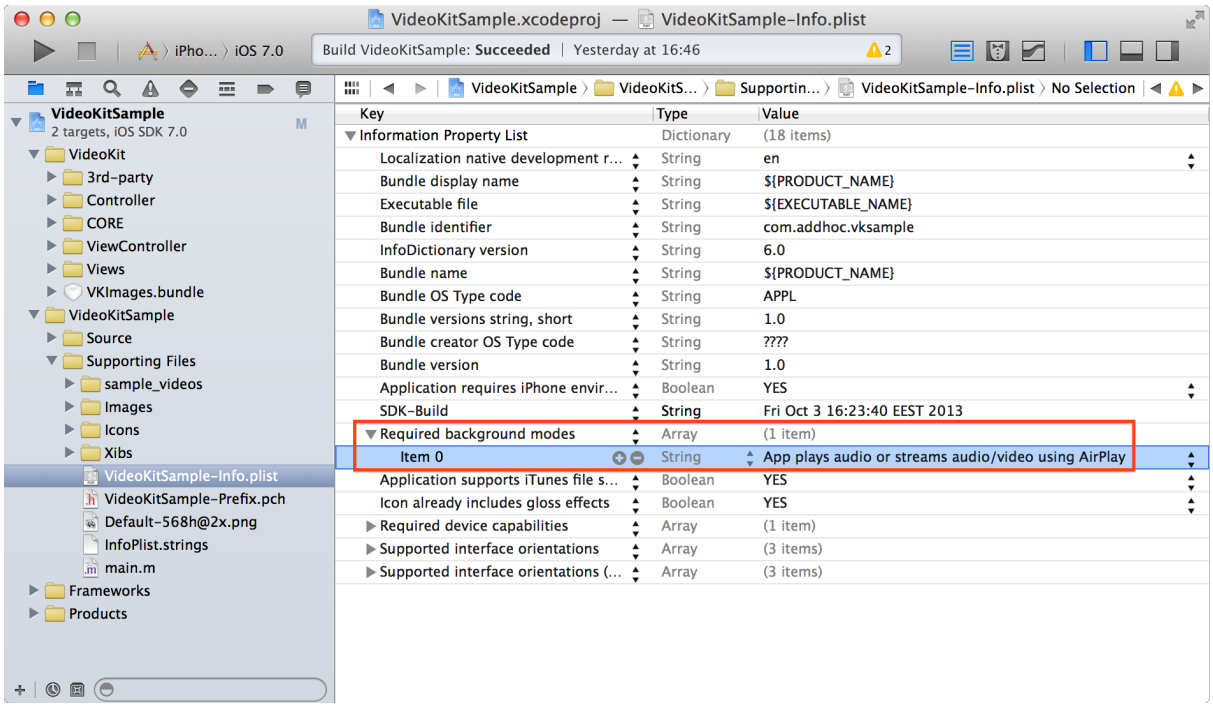
b. Also add these frameworks/libraries

1. libz
2. libiconv
2. QuartzCore
3. SystemConfiguration
4. MediaPlayer
5. OpenGLES
6. AudioToolbox
7. AVFoundation



6. Select your target again, and go to **Build Settings** tab

Find "**Header Search Paths",** set it as below
"Header Search Paths" | **./VideoKit/3rd-party/ffmpeg/include**



7. Now, add the 'Required Background Modes' key to your plist file and set item 0's value to **"App plays audio"**

Integration is done, your project should be compiled without any error now. (If your project is ARC enabled, then please see section 4 (**ARC Supported Projects**) in below)

TRIAL VERSION

VideoKit TRIAL is same as the paid one except, TRIAL has a logo on screen and a limited time playing duration (for now, the time limitation is 15 mins)

UNLOCK TRIAL VERSION (IF PURCHASED A LICENSE)

VideoKit is distributed as trial version as default but license holders have the source code, and so they can unlock the Trial version and disable the Trial version's limitations. To do that please follow the steps below,

1. Find "VKDecodeManager.h" file in VideoKit/VideoKit/Core & open it
2. Find the line "#define TRIAL 1" & uncomment it as below

```
//#define TRIAL  1
```

3. Then rebuilt the project

HOW TO USE

Using VideoKit is a very easy task, it's similar to Apple's MPMoviePlayerViewController & MPMoviePlayerController API.

A. Using **VKPlayerViewController** (Full screen player like Apple's MPMoviePlayerViewController)

1. First of all, add **include file (#import "VKPlayerViewController.h")** to your ViewController

2. Then call VKPlayerViewController with a valid URL as below

```
VKPlayerViewController *playerVc = [[[VKPlayerViewController alloc] initWithURL:urlString
decoderOptions:NULL] autorelease];
playerVc.barTitle = [channel name];
playerVc.statusBarHidden = YES;
[self.navigationController presentViewController:playerVc animated:YES completion:NULL];
```

B. Using **VKPlayerController** (Embedded player like Apple's MPMoviePlayerController)

1. First of all, add **include file (#import "VKPlayerController.h")** to your ViewController

2. Then call VKPlayerController, add as subview and play with a valid URL as below

```
VKPlayerController *_player  = [[[VKPlayerController alloc] initWithURLString:urlString]
autorelease];
_player.view.frame = CGRectMake(28.0, 38.0, 264.0, 167.0);
 [self.view addSubview:_player.view];
_player.decoderOptions = options;
[_player play];
```

VKDecoder supports some options for some specific protocols and formats, the options can be passed to VSDecoder via VKPlayerViewController during initialization and via options property which is settable for VKPlayerController.
Using options and other protocol related infos are shown below,

A. RTSP

A1. RTSP Transport Options

RTSP is not technically a protocol handler in libavformat,
so it must use a lower transport protocol to connect to streaming server,
The following options are supported:

udp
Use UDP as lower transport protocol.

tcp
Use TCP (interleaving within the RTSP control channel) as lower transport protocol.

udp_multicast
Use UDP multicast as lower transport protocol.

http
Use HTTP tunneling as lower transport protocol, which is useful for passing proxies.

```
//a sample option for rtsp - setting transport layer as TCP
NSDictionary *option = [NSDictionary
dictionaryWithObject:VKDECODER_OPT_VALUE_RTSP_TRANSPORT_TCP
forKey:VKDECODER_OPT_KEY_RTSP_TRANSPORT];

VKPlayerViewController *playerVc = [[[VKPlayerViewController alloc] initWithURL:[NSURL
URLWithString:[urlString
stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]]
decoderOptions:option] autorelease];
  ...
```

A2. RTSP Secured Streams & Authentication

VideoKit supports RTSP streams with username and password authentication. There must provide username and your password as below (general format is rtsp://USERNAME:PASSWORD@IPADDRESS:PORT/...) and must set the RTSP transport layer(transport layer can be "UDP" or "TCP", "UDP" is used in below sample).

```
//a sample for secure stream
NSDictionary *option = [NSDictionary
dictionaryWithObject:VSDECODER_OPT_VALUE_RTSP_TRANSPORT_UDP
forKey:VSDECODER_OPT_KEY_RTSP_TRANSPORT];
NSString *urlString = @"rtsp://username:password@192.168.1.5:554/11";

VKPlayerViewController *playerVc = [[[VKPlayerViewController alloc] initWithURL:[NSURL
URLWithString:[urlString
stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]]
decoderOptions:option] autorelease];
  ...
```

B. MJPEG Streams

```
//for mjpeg streams, below option is a must
NSDictionary *option = [NSDictionary dictionaryWithObject:[NSNumber numberWithBool:YES]
forKey:VSDECODER_OPT_KEY_FORCE_MJPEG];

VKPlayerViewController *playerVc = [[[VKPlayerViewController alloc] initWithURL:[NSURL
URLWithString:[urlString
stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]]
decoderOptions:option] autorelease];
  ...
```

C. RTMP

VideoKit supports rtmp streaming, ffmpeg has many server configuration parameters for rtmp streaming protocol. These parameters can be passed through ffmpeg by using VSDECODER_OPT_KEY_PASS_THROUGH as below,

An example of rtmp streaming url,
rtmp://95.211.148.203/live/aflam4youddd?id=152675 -rtmp_swfurl http://mips.tv/content/scripts/eplayer.swf -rtmp_live live -rtmp_pageurl http://mips.tv/embedplayer/aflam4youddd/1/600/380 -rtmp_conn S:OK

```
//Add sample channel to channel list with using VKDECODER_OPT_KEY_PASS_THROUGH
Channel *c14 = [Channel channelWithName:@"Aflam Live TV"
addr:@"rtmp://95.211.148.203/live/aflam4youddd?id=152675 -rtmp_swfurl
http://mips.tv/content/scripts/eplayer.swf -rtmp_live live -rtmp_pageurl
```

```
http://mips.tv/embedplayer/aflam4youddd/1/600/380 -rtmp_conn S:OK" description:@"Pass through params - ffplay style" localFile:NO options:[NSDictionary dictionaryWithObject:@"1" forKey:VKDECODER_OPT_KEY_PASS_THROUGH]];
 [_streamList addObject:c14];
```

C. OTHERS

C1. Multiple Audio Supported Streams

```
//for multiple audio streams, a default one can be set before playing by stream index OR by language string
// If both is set, index option is high priority to other.

NSDictionary *opt1 = [NSDictionary dictionaryWithObject:[NSNumber numberWithInt:2] forKey:VSDECODER_OPT_KEY_AUD_STRM_DEF_IDX];

NSDictionary *opt2 = [NSDictionary dictionaryWithObject:@"eng" forKey:VSDECODER_OPT_KEY_AUD_STRM_DEF_STR];

VKPlayerViewController *playerVc = [[[VKPlayerViewController alloc] initWithURL:[NSURL URLWithString:[urlString stringByAddingPercentEscapesUsingEncoding:NSUTF8StringEncoding]] decoderOptions:option] autorelease];
   ...
```

C2. Logs

VideoKit supports advanced & configurable log mechanism. As Kit consists of some layers, each related logs can be enabled or disabled by setting log level..
To set log level,

```
//to set log level
_decodeManager = [[VSDecodeManager alloc] init];
_decodeManager.delegate = self;
[_decodeManager setLogLevel:kVSLogLevelStateChanges];
```

3. Also, you can get notifications from VKPlayerViewController & VKPlayerControleller about state changes and error handling. (This is optional, if you do not need to handle state change events or error handling than **skip this step**)

```
VKPlayerViewController *playerVc = [[[VKPlayerViewController alloc] initWithURLString:urlString
decoderOptions:options] autorelease];
playerVc.delegate = self;
```

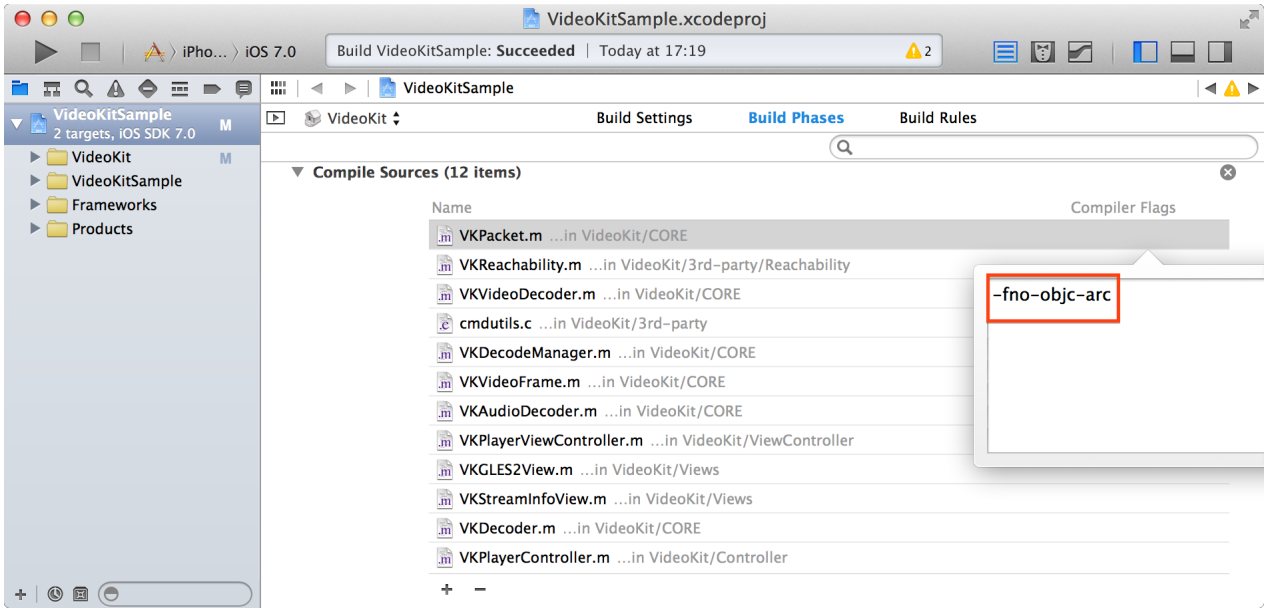
Then implement below method to handle state change events and/or errors

```
- (void)onPlayerViewControllerStateChanged:(VKDecoderState)state errorCode:(VKError)errCode {
    if (state == kVKDecoderStateConnecting) {
    } else if (state == kVKDecoderStateConnected) {
    } else if (state == kVKDecoderStateInitialLoading) {
    } else if (state == kVKDecoderStateReadyToPlay) {
    } else if (state == kVKDecoderStateBuffering) {
    } else if (state == kVKDecoderStatePlaying) {
    } else if (state == kVKDecoderStatePaused) {
    } else if (state == kVKDecoderStateStoppedByUser) {
    } else if (state == kVKDecoderStateConnectionFailed) {
    } else if (state == kVKDecoderStateStoppedWithError) {
        if (errCode == kVKErrorStreamReadError) {
        }
    }
}
```

4. ARC Supported Projects

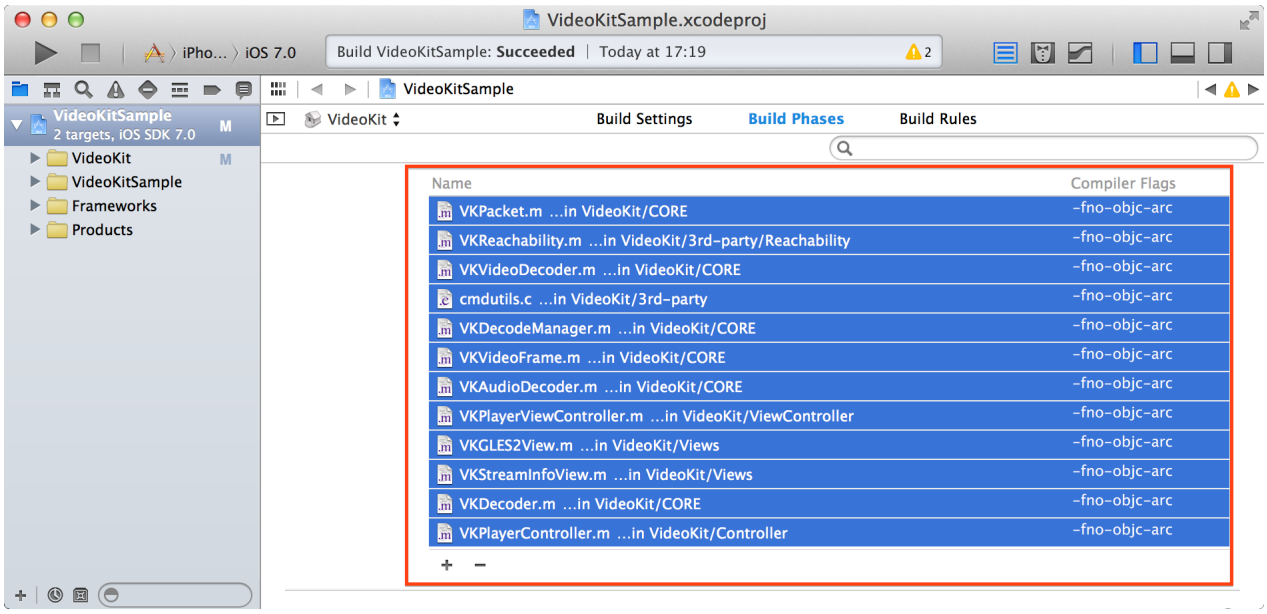
VideoKit is a non ARC library but it can used any ARC or non-ARC projects. Kit can be added to non-ARC project without doing anything.
For ARC enabled projects, after adding VideoKit to project,

> Go to Targets -> Build Phases -> Compile Sources
> double click on the right column of the row under Compiler Flags (It can also be added it to multiple files by holding the cmd button to select the files and then pressing enter to bring up the flag edit box.)



-fno-objc-arc

> Write above compiler flag into the box



> Do this for all VideoKit source files.

5. How to use Airplay feature

AirPlay is the term Apple uses for the protocol that lets you stream your Apple devices to your Apple TV
But, Apple has, bafflingly, not given developers any way to add a Dual Screen AirPlay button inside their apps – users must (themselves) open the multi-tasking tray(<= iOS 6)/control center(iOS 7), swipe to/find the AirPlay icon, select the Apple TV and turn on Mirroring.
(more info at apple site - iOS: How to use AirPlay Mirroring - http://support.apple.com/kb/HT5209)

After enabling mirroring, when player starts playing, dual screen is automatically detected and video plays on Apple TV screen. (This behaviour can be disabled by setting allowAirPlay to NO, default value is YES)

CHANGELOG

Version: 1.10 – Release Date: 01/31/2014

- RTP protocol is supported
- Color formats other than YUV are supported (like RGB, BGR ...)
- Animated GIF files are supported with full transparency
- New decoder parameter, which is VKDECODER_OPT_KEY_PASS_THROUGH, is added to support passing different server parameters to ffmpeg.
- ffmpeg build script is updated for supporting arm64 arch

Version: 1.02 – Release Date: 01/05/2014

- Removed unnecessary header files

Version: 1.01 – Release Date: 12/10/2013

- VideoKit crashes on iOS 5.x devices on runtime, FIXED
- Removed Right Swipe Gesture recognizer which is unused
- Detecting file stream algorithm is broken, FIXED

Version: 1.0 – Release Date: 12/01/2013

- First initial release