Im360 SDK iOS v6.x and newer

Introduction

The im360 iOS SDK is meant for use as a platform to create im360-Player-enabled iOS apps, using Objective-C/C++. This document is designed to assist developers in the creation of such apps.

Requirements

- im360 SDK for iOS *(This folder will be supplied by Immersive Media)*
- im360 iOS examples *(This folder will be supplied by Immersive Media)*
- iOS Device (Important: The im360 iOS SDK will not run in iOS Simulator. A real device must be used.)
- Apple Xcode

Example Projects (im360.examples.(sdk.v0.6.6)-ios)

The example projects can be found in the *im360 iOS examples* package. Please review the full source code for complete information.

1. SpriteTap

   Receives touch events within the im360 video player, and creates a sprite at the point of touch.
2. ControllerRotations

   Demonstrates how to keep the im360 video scene upright when the iOS device orientation changes between landscape and portrait.
3. ImageProjections

   Switches the im360 video player between 2D and 3D display modes. (For switching between 360-degree panoramic video, and regular 2D video)
4. VideoSwitch

   Switches the im360 player's video source.

Running the im360 SDK iOS v6.x example projects (im360.examples.(sdk.v0.6.6)-ios)

- Copy the im360 SDK framework (im360.framework) into your examples folder. It should be placed at the same level as the README.examples file.
- Open the examples.workspace file in Xcode.
- You should now be able to run the examples. (Make sure you run the examples on an actual iOS device device, and not an emulator.)

Extra Examples

These example projects can be provided by Immersive Media.

- PlaybarAndMediaEvents:

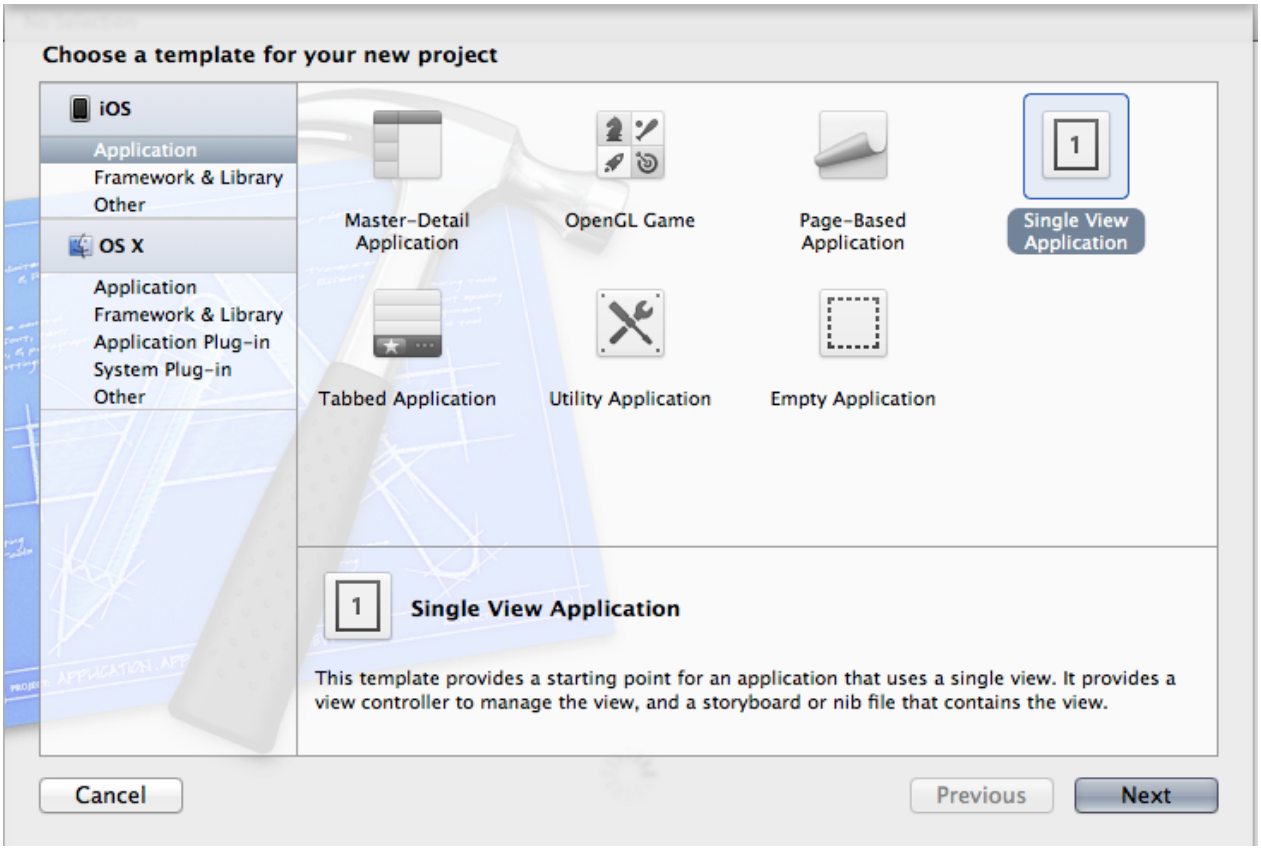
  Demonstrates how to:

  - Receive media events and get media information, such as *video loaded* and *video playback time*.
  - Implement a video play bar.
  - Allow your iOS app to enter the background gracefully, while a video is playing.

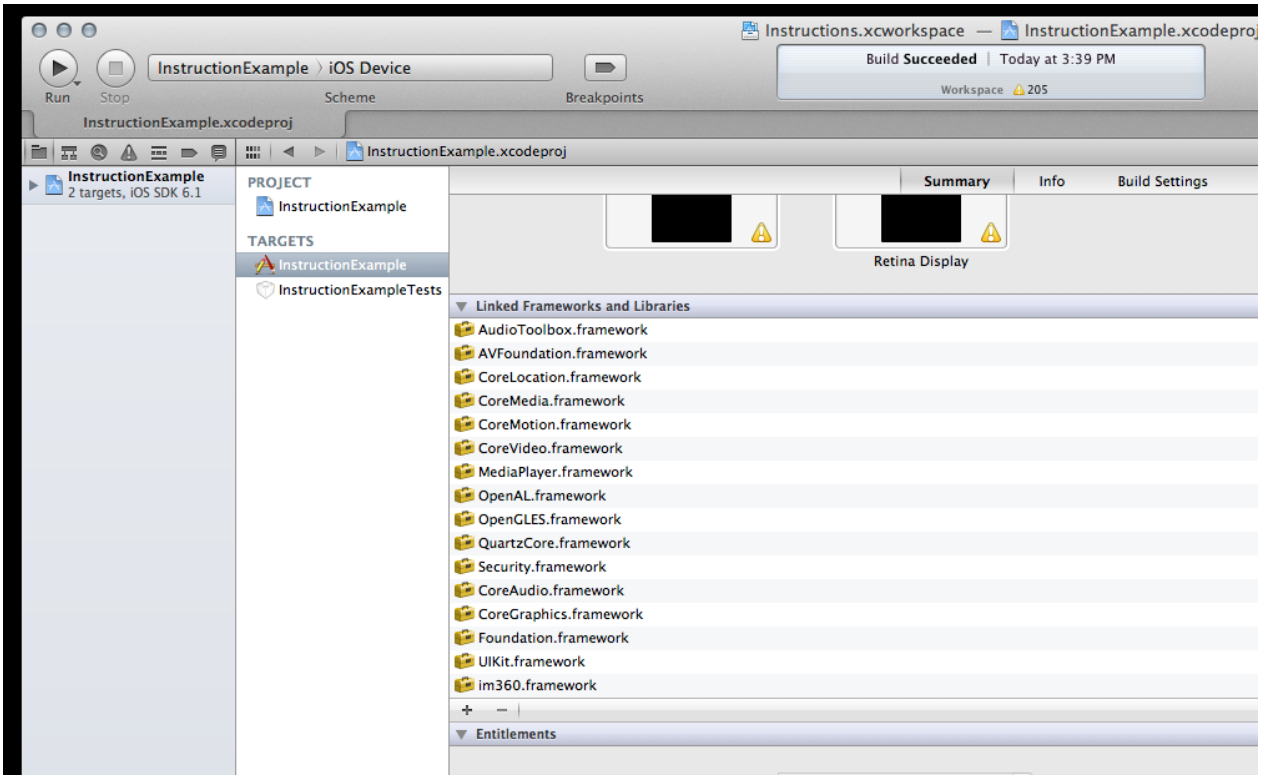
Setting up a new Xcode Project

If you would like to create a fresh Xcode project and integrate the im360 iOS SDK:

1. Create a new iOS project
2. Select *iOS->Application->Single View Application*

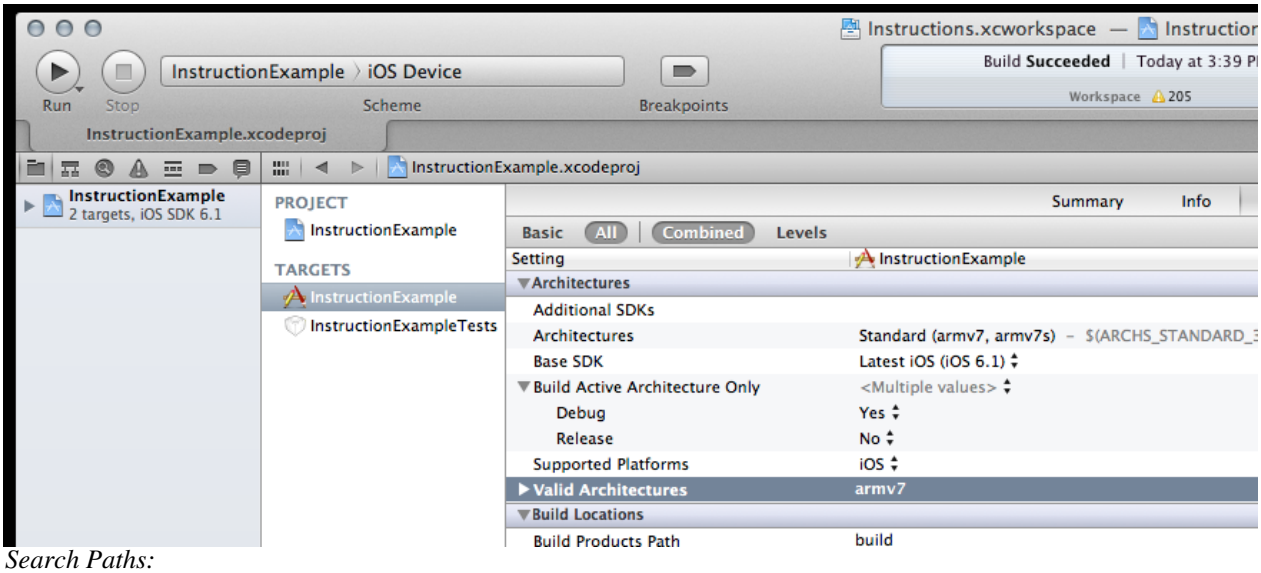


1. Go to *Targets->[your target]->General Tab->Linked Frameworks and Libraries*. Add the following frameworks:



- im360.framework *(The framework directory can be found in: im360.SDK_vXXX-ios/im360.framework)*
- AVFoundation
- CoreMedia
- MediaPlayer
- Security
- CoreVideo
- AudioToolbox
- CoreAudio
- CoreLocation
- CoreMotion
- OpenAL
- OpenGLES
- QuartzCore
- UIKit
- Foundation
- CoreGraphics

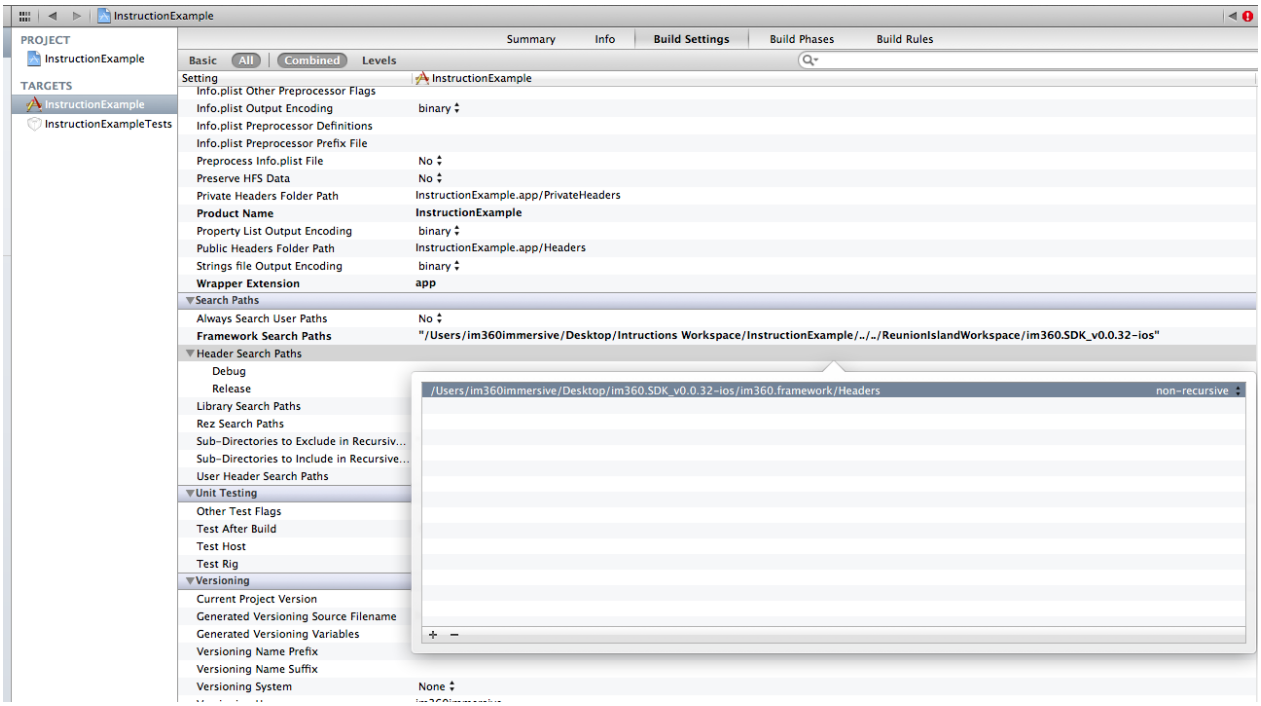
2. Go to *Targets->[your target]->Build Setting*, and change the following values:
    1. *Architectures*
        - *Valid Architectures*:
            - **armv7**



2. *Search Paths:*

   - *Header Search Paths:*

   (Add a reference to your copy of the directory: ***im360.SDK_vXXX-ios/im360.framework/Headers***)



1. *Apple LLVM 6.0 - Language:*

   - *C Language Dialect:*

     - **GNU99 [-std=gnu99]**

2. *Apple LLVM 6.0 - Language C++:*

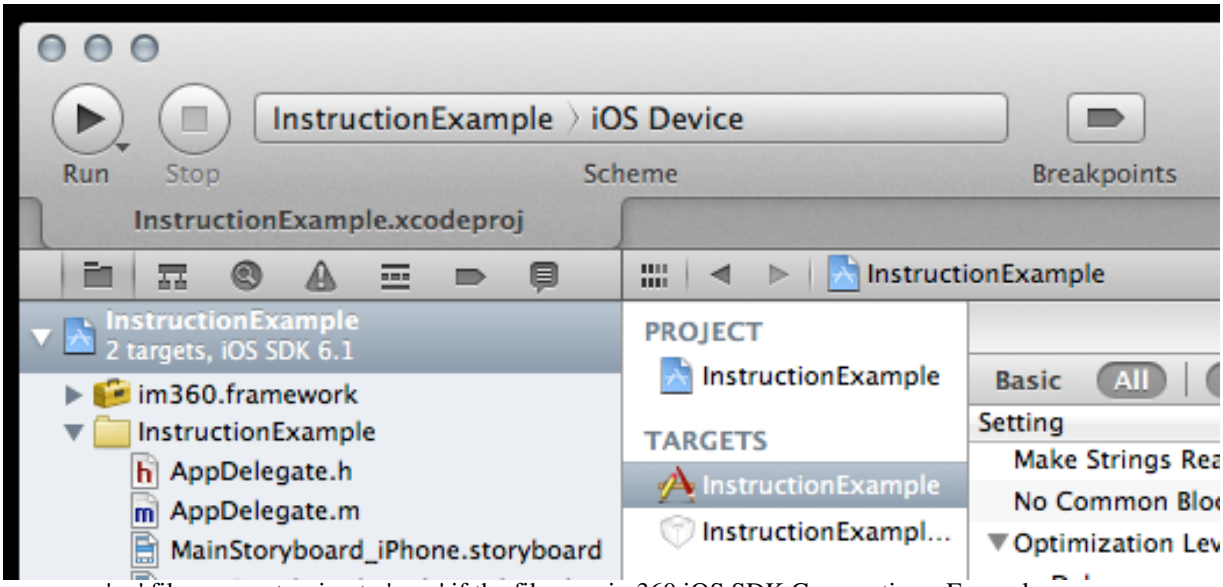
   - C++ Language Dialect:

     - **GNU++98 [-std=gnu++98]**

   - C++ Standard Library:

     - **libstdc++ (GNU C++ standard library)**

3. Change the *Active Scheme* to **iOS Device** *(The im360 iOS SDK will not run in iOS Simulator. A real device must be used.)*



4. Change any '.m' filename extension to '.mm' if the file uses im360 iOS SDK C++ portions. Example: ***ViewController.mm***

Provisioning Your iOS Device

Login using your Apple developer account to the iOS Dev Center Provisioning Portal to create and download certificates & provisioning profiles:

https://developer.apple.com/account/ios/overview.action

Refer to Apple's instructions for setting up a new iOS device for XCode development

https:/ / developer. apple. com/ library/ ios/ documentation/ NetworkingInternet/ Conceptual/ RemoteNotificationsPG/Chapters/ProvisioningDevelopment.html

You can check the state of your XCode provisioning setup in the XCode->Preferences menu (under the 'accounts' tab)

Key Interfaces & Classes

The interfaces and classes you will likely be working with most often. Please see the included header files for a complete list of properties and methods.

- **IM360View:** *im360.framework/Headers/im360/ios/IM360View.h*
  - To be used as your *UIView* for im360 video
  - Derived from *UIView*
  - Will automatically create the *Player* instance
- **IM360ViewDelegate:** *im360.framework/Headers/im360/ios/IM360View.h*
  - Passes events from *BasicScene* for use by any listening *EventListener*s
  - To be used in the declaration of your *UIViewController* class

```
@interface ViewController : UIViewController<IM360ViewDelegate> {
     IM360View *                    _im360View;
     im360::player::Player::pointer  _player;
}
```

  - Please register your *UIViewController* instance with your *IM360View* instance

```
- (void)viewDidAppear:(BOOL)animated
{
    CGRect rect = self.view.frame;

     _im360View = [[IM360View alloc] initWithFrame:rect];
     _im360View.im360ViewDelegate = self;
}
```

- **EventListener:** *im360.framework/Headers/im360/event/core/EventListener.h*
  - Listens for events dispatched by your Player object
  - *More precisely: EventListeners provide an interface to listen for events generated by EventDispatchers. Event are dispatched directly to their listeners as well as to their parent dispatchers, this allows you to listen for events on an object directly or further up the event chain which will often include an accumulation of multiple dispatcher events. Furthermore all events are dispatched by default during the player rendering cycle which for iOS is on the main UI thread. This can be disabled by calling setDispatchEventsEnabled(false) on your player instance and choosing to call EventManager::instance().dispatchEvents() at your discretion; this however needs to be done on the same thread the player is rendering on.*

```
class PlayerProxy : public event::EventListener    //EventListener
{
public:
    typedef core::SharedPointer<PlayerProxy>::pointer pointer;

    PlayerProxy() : EventListener()
    {

    }

    virtual ~PlayerProxy()
    {

    }

    void onEvent(event::Event::pointer event)
    {
        if( event->getPointer<event::MediaEvent>() ) {

            if (!owner){
                return;
            }
            [owner
onMediaPlayStateChange:(im360::event::MediaEvent::EventId)event->getPointer<event::MediaEvent>()->getEventId()];
        }
        else if( event->getPointer<event::TimeChangeEvent>() ) {
            [owner onMediaTimeChange:event->getPointer<event::TimeChangeEvent>()];
        }

    }
```

```
    PlayerViewController * owner;
};
```

- **BasicScene:** *im360.framework/Headers/im360/scene/BasicScene.h*
  - Load, play, and pause videos
  - Load and display images
  - Get and set video playhead time position
  - Set the view mode (2D or 3D)
  - Set 'nadir' logo image *(The image at the bottom of a 360 video)*
  - Dispatch media events:
    - *onMediaLoaded*
    - *onMediaFrame*
    - *onMediaTimeChange*
    - *onMediaDurationChange*
    - *onMediaFormatChange*
    - *onMediaPlayStateChange*
    - *onMediaError*
    - *onMediaMetaData*
  - Dispatch scene events:
    - *onNodeClick*
    - *onWorldClick*
- **Player:** *im360.framework/Headers/im360/player/Player.h*
  - Load a video *Player->loadMedia(const std::string & url) is the easiest way to load a video.*
  - Load a scene json file

Basic Concepts

- Playing a video:

  A video can be played in any of the following ways:

  1. *BasicScene->loadVideo(videoUrl,* audioUrl*)*

     This method will load a video file (mp4/webm). The *audioUrl* is optional for mp4 but required for webm files.
  2. *Player->loadScene(sceneFile)*

     A scene file is the way to package up all media elements to be presented together. The scene file must reference a video (mp4/webm)and optional audio file (if it is separate) but then can have many optional other elements. Scenes are defined in a single json file. The json scene file also sets various other properties, such as nadir logo image, and information for rendering the 360 panoramic scene. *An example scene file is included with the sdk example package.*
  3. *Player->loadMedia(const std::string & url)*

     This method will load a video file (mp4/webm).

- Encoding videos and metadata

The Immersive mobile SDK supports videos that are h.264/AAC in an mp4 file or HLS m3u8 manifest stream. Most iOS devices require the baseline profile but some support Main. We usually stick to baseline for widest compatibility. Our player, by default, assumes the video to be an equirectangular layout and will stretch (asymmetrically if needed) the video to a 2:1 aspect ratio and cover the sphere with it (phi range from +90 to -90 and theta from -180 to +180). So even if the video is 1920x1080, which is not 2:1, we will stretch to fit so its important

the encoder supports the asymmetric scaling because our original captured video will be in the 2:1 ratio. Immersive's capture and post processing tools all support this naturally. There is no requirement of any particular video size or bitrate since we can asymmetrically scale anything. So small ads or lower power mobiles may prefer 720p or lower resolution. Bitrates for full screen videos start to look good on mobiles above 2Mbps but look really good above 3Mbps.

There are several ways to serve assets to the mobile SDK. If the videos are full spherical and can use the players default behavior then they can be pulled from your servers like any other video. This is the easiest approach but is limited to playing full spherical videos. So, if the video is a panoramic or partial sphere or inverted or was filmed with a fisheye camera (instead of a spherical camera) then a set of metadata needs to be loaded with the video to override the defaults and specify how best to paint and play the video. There are several ways to do this:

1. Place the metadata file on your video server at the same URL path and filename but with a ".meta" extension. The player always looks for this file automatically.
2. Instead of loading the video, load a "video entity" record. This is a JSON formatted file that specifies the video to be played and the metadata associated with it. These are small files so they can be loaded from a separate server from where the video is served from. This is how our im360 video server works (see below).
3. After loading the video URL in the player, then make direct function calls to set the various presentation parameters (phi, theta, segments, vstart, etc) manually.

Here's a sample "video entity" record read from our im360 server to show the file format.


```
 "videoId": 271,
 "sourceId": 422,
 "videoPath":
"/mnt/hgfs/im360ws/mediacat/store/videos/000/000/271.mp4",
 "videoFile": "271.mp4",
 "videoFilesize": 27247698,
 "videoUrl": "http://192.168.75.40/store/videos/000/000/271.
mp4?cache=1377213637985",
 "videoFormat": "mp4",
 "videoWidth": 3200,
 "videoHeight": 1600,
 "videoBitrate": 0,
 "videoCodec": "h264",
 "videoFps": 15,
 "videoDuration": 85.933334,
 "videoAudioChannels": 0,
 "videoAudioBitrate": 0,
 "videoAudioFrequency": 0,
 "videoMetaComment":
"{\\\"cameraType\\\":\\\"quattro\\\",\\\"phiEnd\\\":-60,\\\"phiStart\\\":60,\\\"proj\\\":
 "videoDateAdded": "2013-08-22 18:20:37.985957",
 "videoDateUpdated": "2013-08-22 18:20:37.985957"
```

Troubleshooting

- ***[tr1/memory]*** Error while targeting iOS 6 or greater:

  Change all compiler references from *C++11* to *C++98*. Please see full description in #Setting up a new Xcode Project

- ***Parse Issue*** Error:

  Ensure that all '.m' filename extensions have been changed to to '.mm' if the files use or include im360 iOS SDK C++ portions. Example: *ViewController.mm*

- ***gpus_ReturnNotPermittedKillClient*** Error:

  Please clean up any im360 SDK objects when your app enters the background, as shown in the example project #Extra Examples

Article Sources and Contributors

Im360 SDK iOS v6.x and newer *Source*: https://50.112.113.82/wiki/index.php?oldid=1183 *Contributors*: Tshirley

Image Sources, Licenses and Contributors

File:Template.png *Source*: https://50.112.113.82/wiki/index.php?title=File:Template.png *License*: unknown *Contributors*: Tshirley
File:frameworks.png *Source*: https://50.112.113.82/wiki/index.php?title=File:Frameworks.png *License*: unknown *Contributors*: Tshirley
File:arm.png *Source*: https://50.112.113.82/wiki/index.php?title=File:Arm.png *License*: unknown *Contributors*: Tshirley
File:headers.png *Source*: https://50.112.113.82/wiki/index.php?title=File:Headers.png *License*: unknown *Contributors*: Tshirley
File:Active_scheme.png *Source*: https://50.112.113.82/wiki/index.php?title=File:Active_scheme.png *License*: unknown *Contributors*: Tshirley