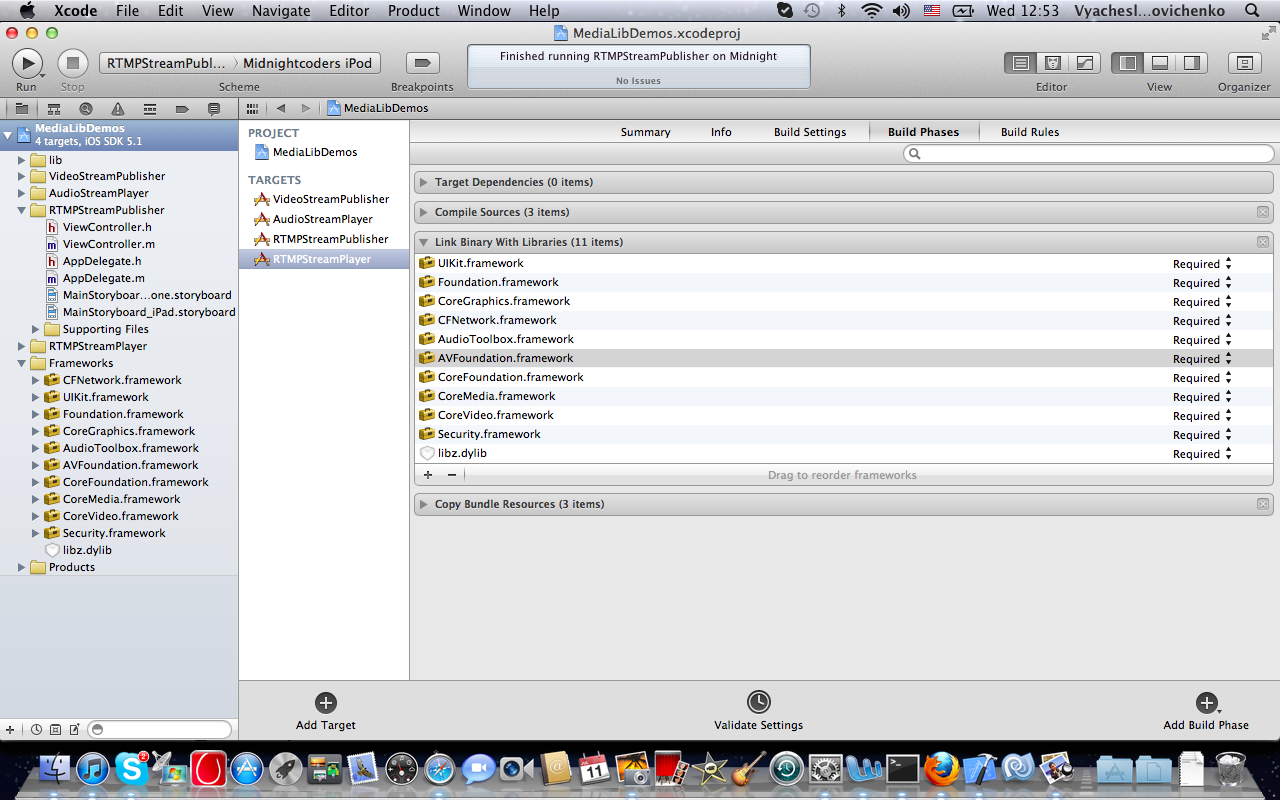
Setup Libraries

Copy CommLibiOS and MediaLibiOS libraries from Download section of site, load zip files to some folder, for example - “lib”, unpack them and then delete zip files.

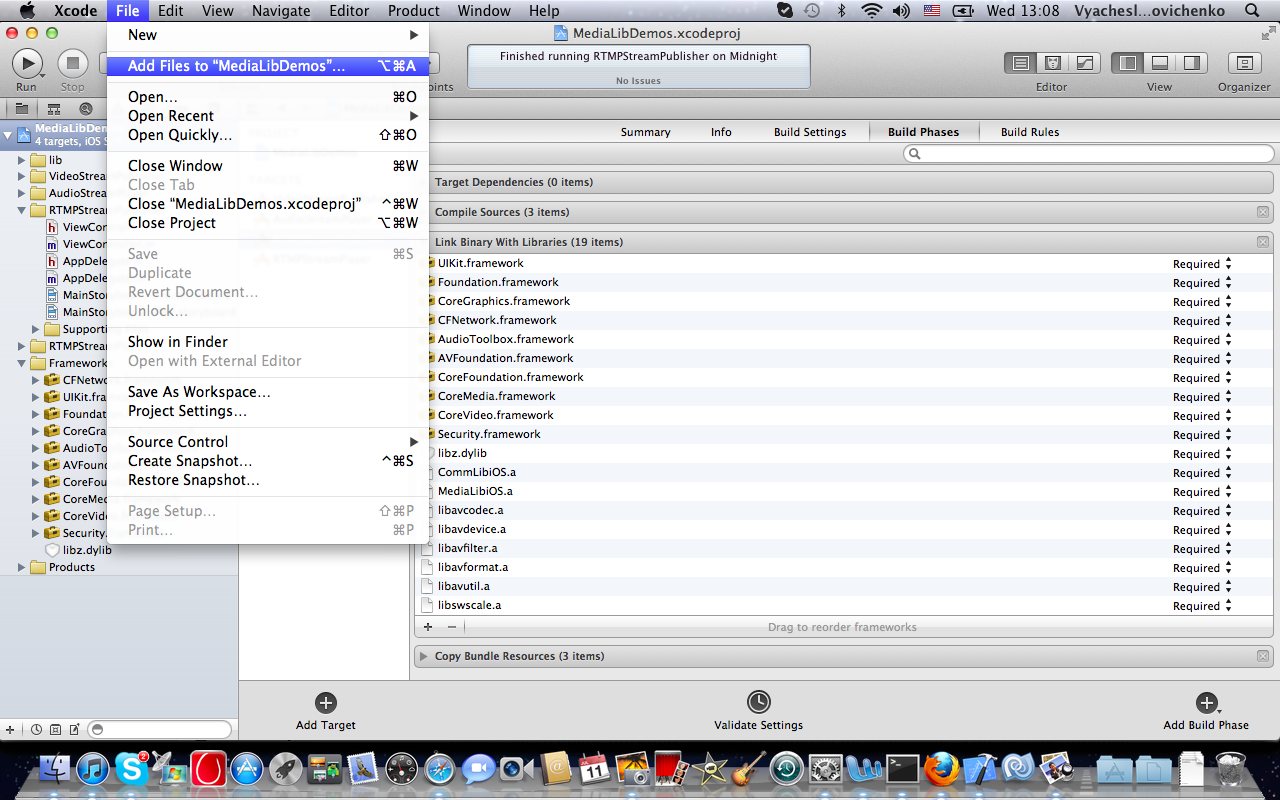
Add Libraries and Frameworks to the Project

Just after creating the project you should add libraries and frameworks to it. Using the CommLibiOS and MediaLibiOS libraries requires to add the following frameworks: CFNetwork, AudioToolbox, AVFoundation, CoreFoundation, CoreMedia, CoreVideo, Security, and library libz.dylib.

Open the settings for your target by clicking on the blue bar at the very top of the Xcode sidebar, then switch the “Build Phases” tab, expand the box “Link Binary With Libraries”, then click the plus button and add needed frameworks and library:



Then add CommLibiOS and MediaLibiOS libraries to the project. Open the settings for your target by clicking on the blue bar at the very top of the Xcode sidebar, then choose menu item “File -> Add Files to …” and add folder “lib”, where CommLibiOS and MediaLibiOS libraries are. You will see the appeared libraries in box “Link Binary With Libraries”.



Walkthrough: Building Media Stream Publisher App

Here is the description of the simple application for streaming of the video and audio to media server by RTMP. For this purpose some controls and data viewers are needed: two text fields for entering server URL and stream Name, image view for previewing the publishing stream, and three buttons – “Connect/Disconnect”, “Toggle Cameras”, “Start/Pause” for process control.

Create a New Project

Launch Xcode and choose “File > New > New Project > iOS > Application > Single View Application”. Click Next. Enter **RTMPStreamPublisher** as the project name, and if you have not yet done so, set up your company identifier. Set the Device Family to Universal, check Use Storyboards and Use Automatic Reference Counting, and leave Include Unit Tests unchecked. Click Next. Specify where to save your project and click Create. Xcode opens a new browser containing your project.

Now add frameworks and libraries (see the section “Add Libraries and Frameworks to the Project” above).

It is time for some programming. Open RTMPStreamPublisher folder on Xcode sidebar, choose ViewController.h and add the following text:

#import "BroadcastStreamClient.h"

@interface ViewController : UIViewController <UITextFieldDelegate, IMediaStreamEvent> {

BroadcastStreamClient \*upstream;

IBOutlet UITextField \*hostTextField;

IBOutlet UITextField \*streamTextField;

IBOutlet UIView \*previewView;

IBOutlet UIBarButtonItem \*btnConnect;

IBOutlet UIBarButtonItem \*btnToggle;

IBOutlet UIBarButtonItem \*btnPublish;

}

-(IBAction)connectControl:(id)sender;

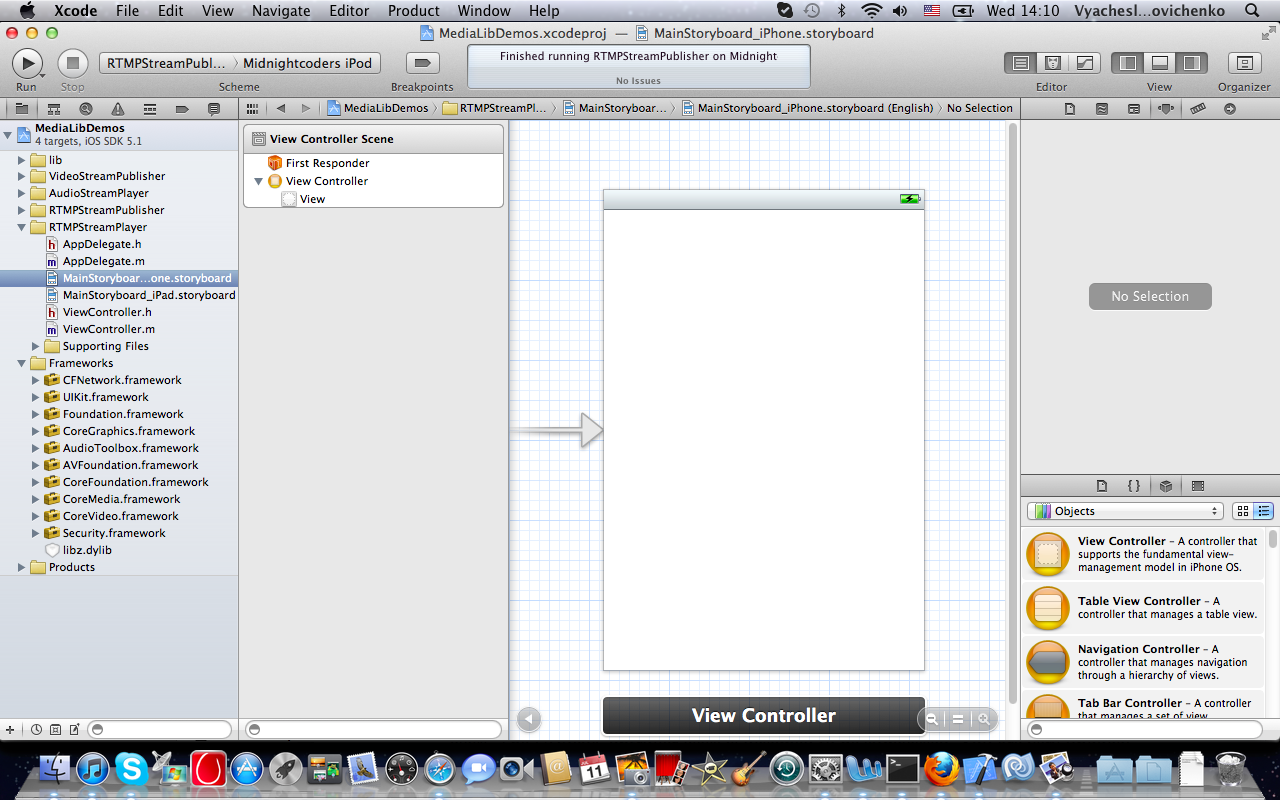
-(IBAction)publishControl:(id)sender;

-(IBAction)camerasToggle:(id)sender;

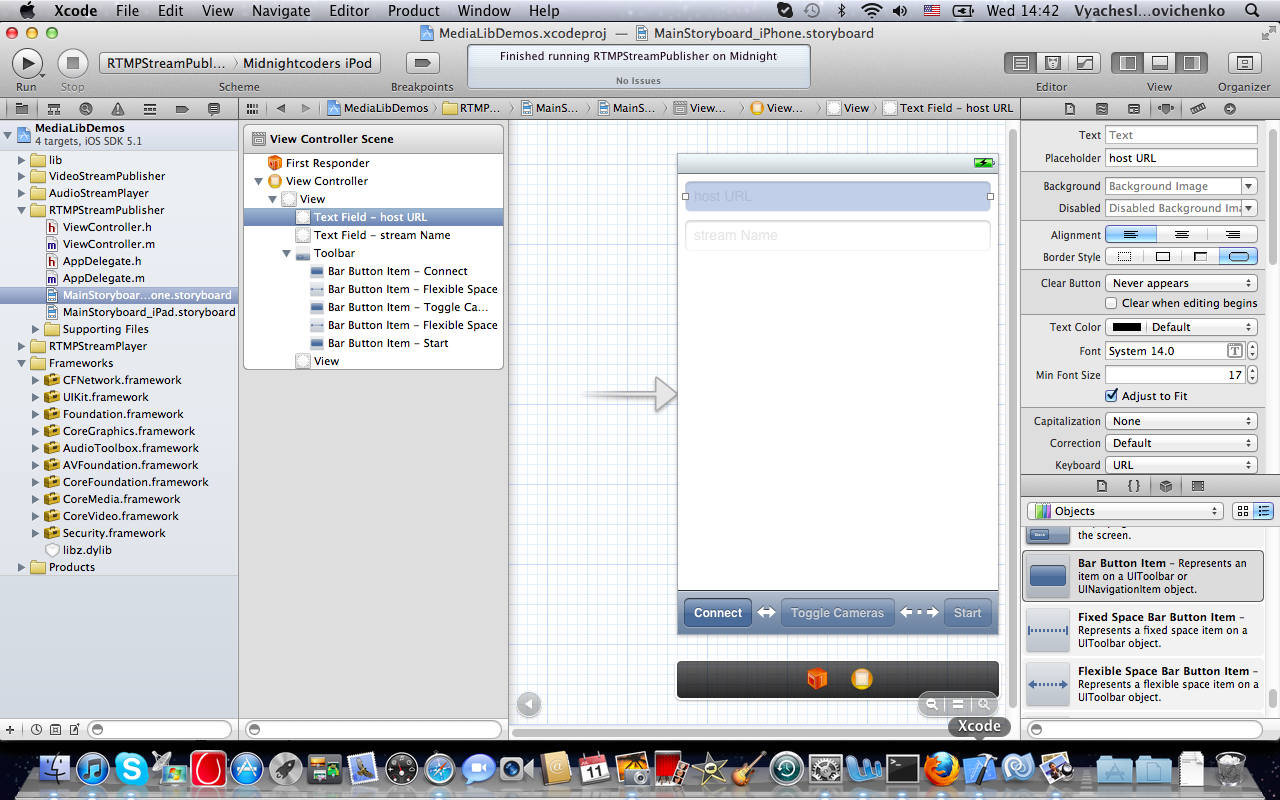
@end

Build an User Interface

The project has two storyboards: one for iPhone, one for iPad, as well as an app delegate class and a view controller class. Click MainStoryboard\_iPhone.storyboard to open the storyboard in Interface Builder. With the default settings, the storyboard consists of a single scene made up of a basic view controller.



Drag-and-drop two Text Fields on top of root View, Toolbar on bottom of root View and View on all room above Toolbar, then put on Toolbar in series three Bar Button Items with two Flexible Space Bar Button Items between them.



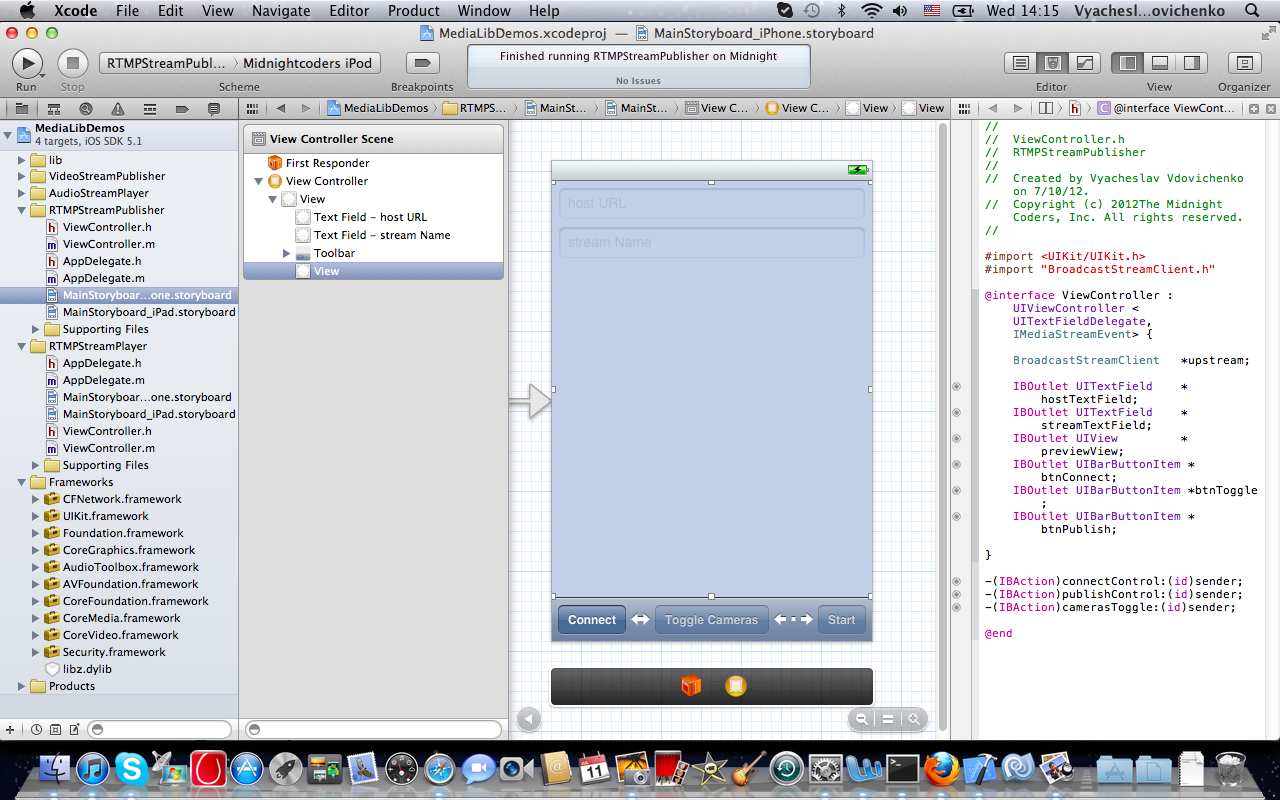
Open Attribute Inspector (View -> Utilites -> Show Attribute Inspector), and set the additional attributes in UI elements:

* for first Text Field : Placeholder = “host URL”, Keyboard = “URL”, Return Key = “Done”;
* for second Text Field: Placeholder = “stream Name”, Return Key = “Done”;
* for first Bar Button Item: Title = “Connect”;
* for second Bar Button Item: Title = “Toggle Cameras”, Enabled = off;
* for third Bar Button Item: Title = “Start”, Enabled = off;
* for View: hidden = on.

And now it is some magic – UI will be linked with code. Hide the Utility pane (View -> Utilities -> Hide Utilities). Then choose ViewController.h source code file on Xcode sidebar, right-click on the text and click “Open in Assistant Editor”.

Holding “control” button on keyboard, drag IBOutlets and IBAction to the appropriate UI elements (let it be “control-drag” operation):

* choose first Text Field (on View Controller Scene pane) and control-drag IBOutlet UITextField \*hostTextField to it;
* choose second Text Field and control-drag IBOutlet UITextField \*streamTextField to it;
* choose View and control-drag IBOutlet UIView \*previewView to it;
* choose first Bar Button Item and control-drag IBOutlet UIBarButtonItem \*btnConnect to it, then control-drag -(IBAction)connectionControl:(id)sender to it;
* choose second Bar Button Item and control-drag IBOutlet UIBarButtonItem \*btnToggle to it, then control-drag -(IBAction)camerasToggle:(id)sender to it;
* choose third Bar Button Item and control-drag IBOutlet UIBarButtonItem \*btnPublish to it, then control-drag -(IBAction)publishControl:(id)sender to it.



That is all … for iPhone. Click MainStoryboard\_iPad.storyboard – and repeat all this for iPad ☺

Correct the Default Code

Click ViewController.m. Add two line in method -(void)viewDidLoad:

-(void)viewDidLoad {

[super viewDidLoad];

hostTextField.delegate = self;

streamTextField.delegate = self;

}

It allows to get UITextFieldDelegate protocol for keyboard control.

-(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {

return (interfaceOrientation == UIInterfaceOrientationPortrait);

}

It lets only portrait device orientation.

Add the New Code

Add this code below the default code:

#pragma mark -

#pragma mark Private Methods

-(void)showAlert:(NSString \*)message {

UIAlertView \*av = [[UIAlertView alloc] initWithTitle:@"Receive" message:message delegate:self

cancelButtonTitle:@"Ok" otherButtonTitles:nil];

[av show];

}

-(void)doConnect {

upstream = [[BroadcastStreamClient alloc] init:hostTextField.text];

upstream.delegate = self;

[upstream setPreviewLayer:previewView orientation:AVCaptureVideoOrientationPortrait];

[upstream stream:streamTextField.text publishType:PUBLISH\_LIVE];

btnConnect.title = @"Disconnect";

}

-(void)doDisconnect {

[upstream disconnect];

upstream = nil;

btnConnect.title = @"Connect";

btnToggle.enabled = NO;

btnPublish.title = @"Start";

btnPublish.enabled = NO;

hostTextField.hidden = NO;

streamTextField.hidden = NO;

previewView.hidden = YES;

}

Method showAlert: is used for notification about error situation.

Method doConnect creates instance “upstream” of BroadcastStreamClient class, which implements network and media processing. The “self” is declared as delegate of upstream – so it must implement IMediaStreamEvent protocol (below). Then upstream gets previewView as view for preview of publishing stream. Stream is opened with name from streamTextFild and “live” publish type. “Connect” button gets “Disconnect” title.

Method doDisconnect deallocates upstream instance and makes appropriate changes in visable/active attributes and titles of UI elements.

#pragma mark -

#pragma mark Public Methods

// ACTIONS

-(IBAction)connectControl:(id)sender {

NSLog(@"connectControl: host = %@", hostTextField.text);

(!upstream) ? [self doConnect] : [self doDisconnect];

}

-(IBAction)publishControl:(id)sender {

NSLog(@"publishControl: stream = %@", streamTextField.text);

(upstream.state != STREAM\_PLAYING) ? [upstream start] : [upstream pause];

}

-(IBAction)camerasToggle:(id)sender {

NSLog(@"camerasToggle:");

if (upstream.state == STREAM\_PLAYING)

[upstream switchCameras];

}

There are button callbacks. Each of them is called by pushing appropriate button:

* method connectControl: makes connect/disconnect operations;
* method publishControl: starts/resumes stream publishing ([upstream start]) or makes a pause ([upstream pause]);
* method camerasToggle: switches the publishing from front or back cameras.

#pragma mark -

#pragma mark UITextFieldDelegate Methods

-(BOOL)textFieldShouldReturn:(UITextField \*)textField {

[textField resignFirstResponder];

return YES;

}

This is the implementation of UITextDelegate protocol. It lets the keyboard to disappear after text editing have been finished.

#pragma mark -

#pragma mark IMediaStreamEvent Methods

-(void)stateChanged:(MediaStreamState)state description:(NSString \*)description {

NSLog(@" $$$$$$ <IMediaStreamEvent> stateChangedEvent: %d = %@", (int)state, description);

switch (state) {

case CONN\_DISCONNECTED: {

[self doDisconnect];

[self showAlert:[NSString stringWithString:description]];

break;

}

case CONN\_CONNECTED: {

if (![description isEqualToString:@"RTMP.Client.isConnected"])

break;

[self publishControl:nil];

hostTextField.hidden = YES;

streamTextField.hidden = YES;

previewView.hidden = NO;

btnPublish.enabled = YES;

break;

}

case STREAM\_PAUSED: {

btnPublish.title = @"Start";

btnToggle.enabled = NO;

break;

}

case STREAM\_PLAYING: {

btnPublish.title = @"Pause";

btnToggle.enabled = YES;

break;

}

default:

break;

}

}

-(void)connectFailed:(int)code description:(NSString \*)description {

NSLog(@" $$$$$$ <IMediaStreamEvent> connectFailedEvent: %d = %@\n", code, description);

[self doDisconnect];

[self showAlert:(code == -1) ?

[NSString stringWithFormat:@"Unable to connect to the server. Make sure the hostname/IP address and port number are valid\n"] :

[NSString stringWithFormat:@"connectFailedEvent: %@ \n", description]];

}

This is the implementation of IMediaStreamEvent protocol, upstream can call its methods for notification about state changes and error situations:

* method stateChanged:description: makes the reaction to appropriate state change – it calls doDisconnect and showAlert for CONN\_DISCONNECTED state, and makes appropriate changes in visable/active attributes and titles of UI elements for other states;
* method connectFailed:description: reacts for error situations during the net connection process - it calls doDisconnect and showAlert.

So, build, run – and have fun!

Walkthrough: Building Media Stream Player App

Here is the description of the simple application for playback of the video and audio streams from media server by RTMP. For this purpose some controls and data viewers are needed: two text fields for entering server URL and stream Name, image view for viewing the playing stream, and two buttons – “Connect/Disconnect and “Start/Pause” for process control.

Create a New Project

Launch Xcode and choose “File > New > New Project > iOS > Application > Single View Application”. Click Next. Enter **RTMPStreamPlayer** as the project name, and if you have not yet done so, set up your company identifier. Set the Device Family to Universal, check Use Storyboards and Use Automatic Reference Counting, and leave Include Unit Tests unchecked. Click Next. Specify where to save your project and click Create. Xcode opens a new browser containing your project.

Now add frameworks and libraries (see the section “Add Libraries and Frameworks to the Project” above).

It is time for some programming. Open RTMPStreamPlayer folder on Xcode sidebar, choose ViewController.h and add the following text:

#import "MediaStreamPlayer.h"

@interface ViewController : UIViewController <UITextFieldDelegate, IMediaStreamEvent> {

MediaStreamPlayer \*player;

IBOutlet UITextField \*hostTextField;

IBOutlet UITextField \*streamTextField;

IBOutlet UIImageView \*previewView;

IBOutlet UIBarButtonItem \*btnConnect;

IBOutlet UIBarButtonItem \*btnPlay;

}

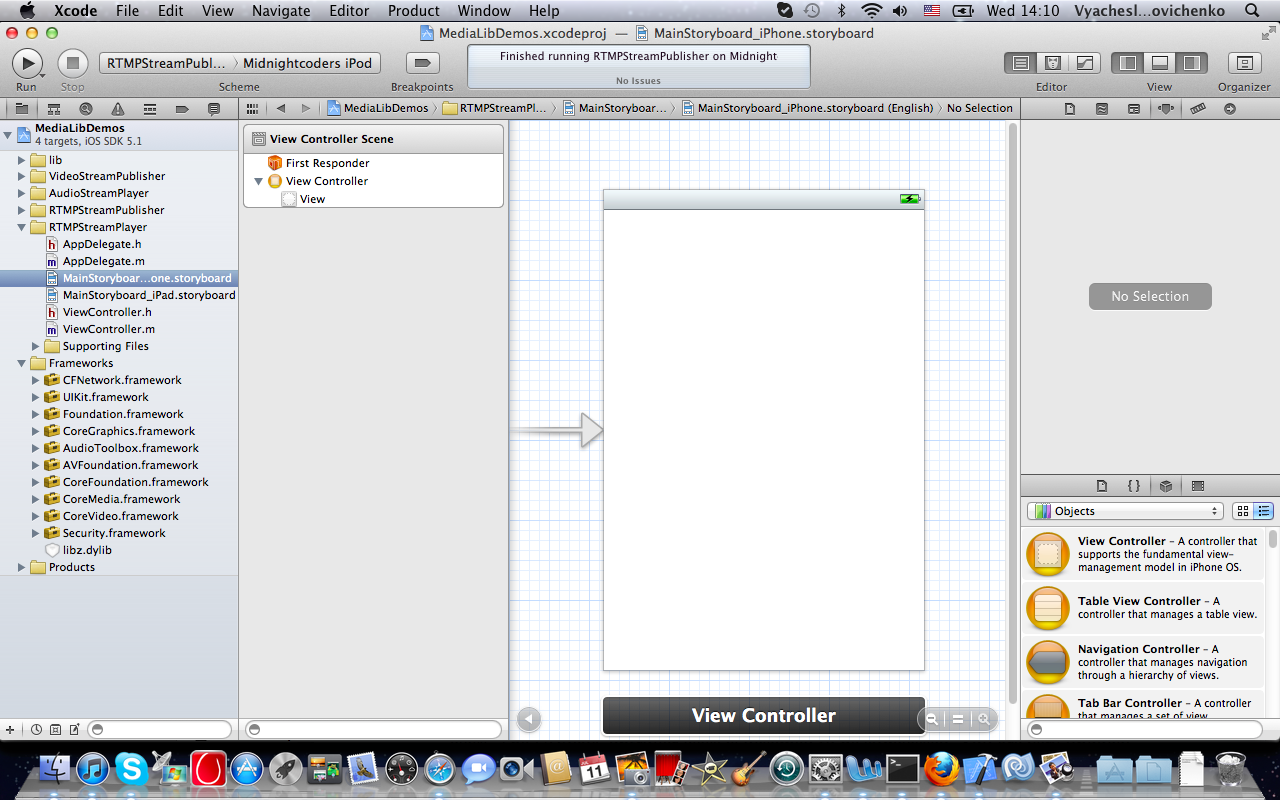
-(IBAction)connectControl:(id)sender;

-(IBAction)playControl:(id)sender;

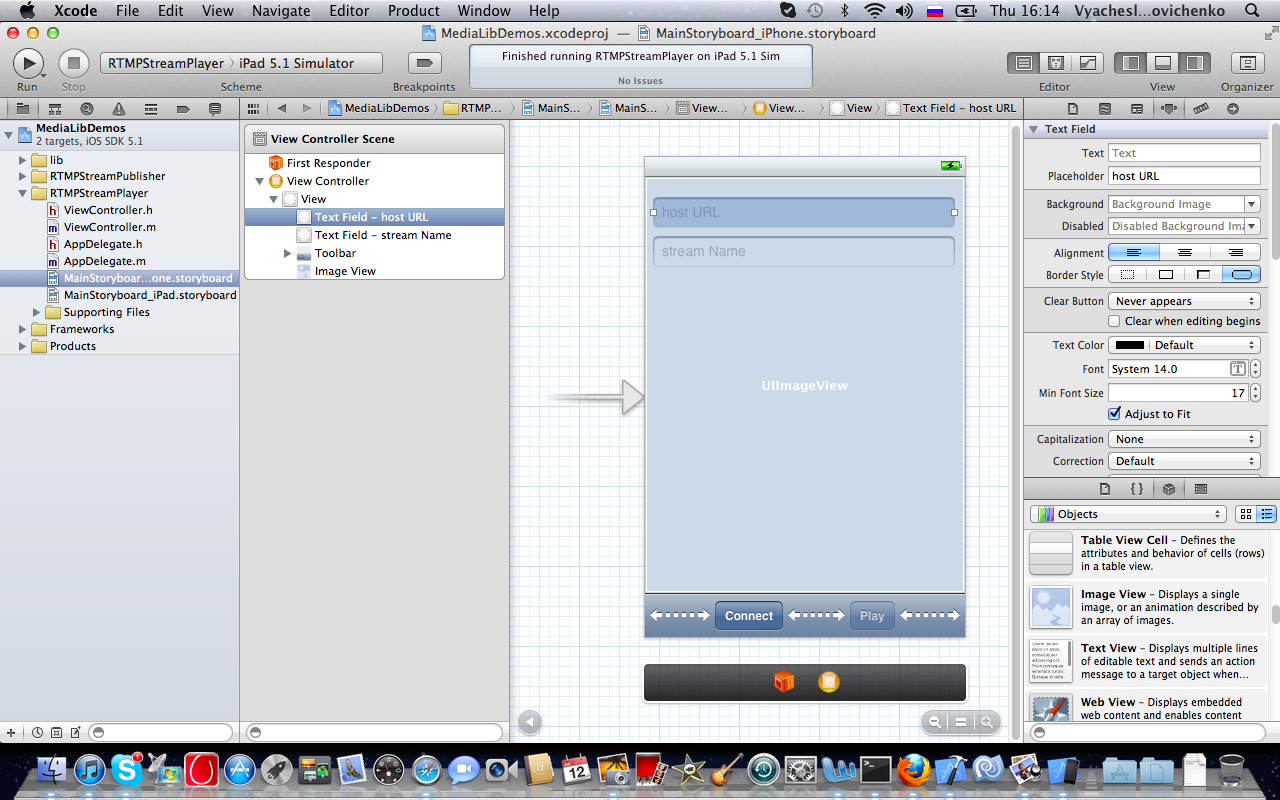
@end

Build an User Interface

The project has two storyboards: one for iPhone, one for iPad, as well as an app delegate class and a view controller class. Click MainStoryboard\_iPhone.storyboard to open the storyboard in Interface Builder. With the default settings, the storyboard consists of a single scene made up of a basic view controller.



Drag-and-drop two Text Fields on top of root View, Toolbar on bottom of root View and Image View on all room above Toolbar, then put on Toolbar in series three Flexible Space Bar Button Items with two Bar Button Items between them.



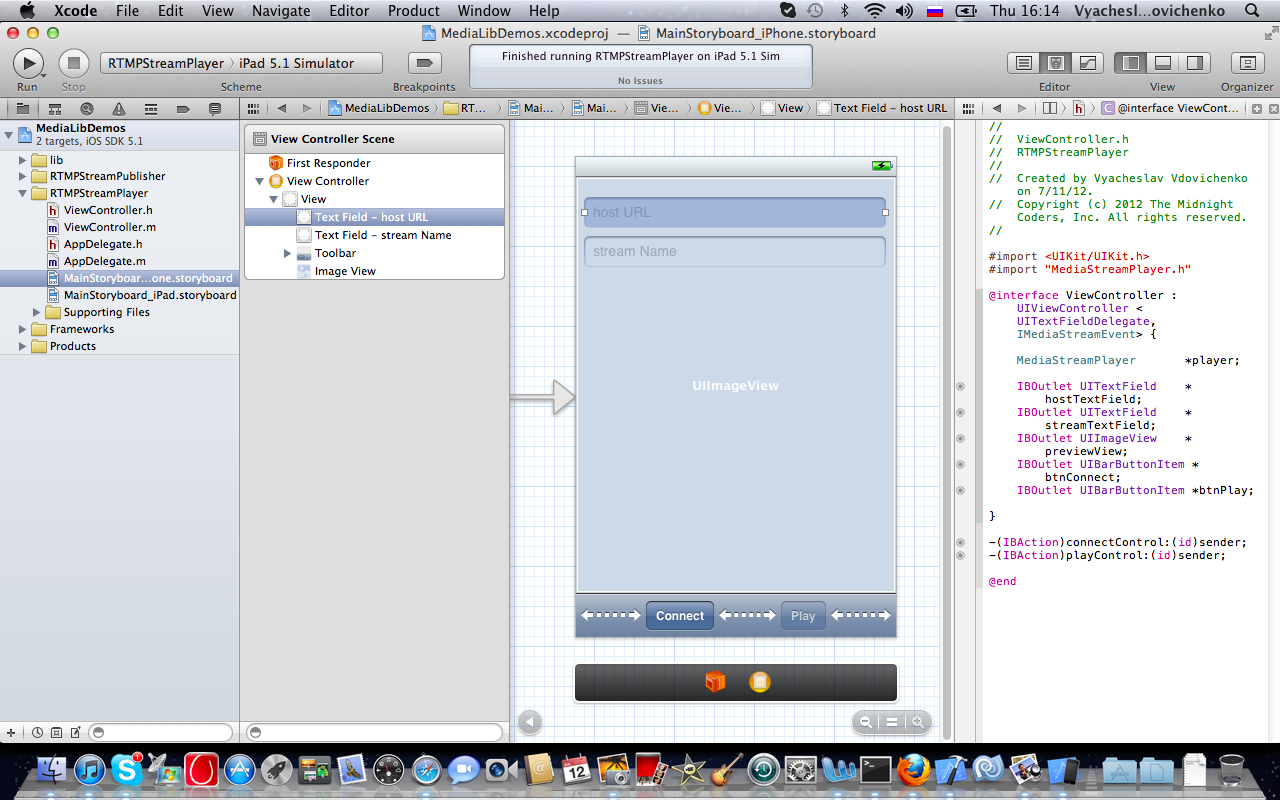
Open Attribute Inspector (View -> Utilites -> Show Attribute Inspector), and set the additional attributes in UI elements:

* for first Text Field : Placeholder = “host URL”, Keyboard = “URL”, Return Key = “Done”;
* for second Text Field: Placeholder = “stream Name”, Return Key = “Done”;
* for first Bar Button Item: Title = “Connect”;
* for second Bar Button Item: Title = “Start”, Enabled = off;
* for Image View: hidden = on.

Now UI will be linked with code. Hide the Utility pane (View -> Utilities -> Hide Utilities). Then choose ViewController.h source code file on Xcode sidebar, right-click on the text and click “Open in Assistant Editor”.

Holding “control” button on keyboard, drag IBOutlets and IBAction to the appropriate UI elements (let it be “control-drag” operation):

* choose first Text Field (on View Controller Scene pane) and control-drag IBOutlet UITextField \*hostTextField to it;
* choose second Text Field and control-drag IBOutlet UITextField \*streamTextField to it;
* choose Image View and control-drag IBOutlet UIImageView \*previewView to it;
* choose first Bar Button Item and control-drag IBOutlet UIBarButtonItem \*btnConnect to it, then control-drag -(IBAction)connectionControl:(id)sender to it;
* choose second Bar Button Item and control-drag IBOutlet UIBarButtonItem \*btnPlay to it, then control-drag -(IBAction)playControl:(id)sender to it.



That is all for iPhone. Click MainStoryboard\_iPad.storyboard – and repeat all this for iPad .

Correct the Default Code

Click ViewController.m. Add two line in method -(void)viewDidLoad:

-(void)viewDidLoad {

[super viewDidLoad];

hostTextField.delegate = self;

streamTextField.delegate = self;

}

It allows to get UITextFieldDelegate protocol for keyboard control.

-(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interfaceOrientation {

return (interfaceOrientation == UIInterfaceOrientationPortrait);

}

It lets only portrait device orientation.

Add the New Code

Add this code below the default code:

#pragma mark -

#pragma mark Private Methods

-(void)showAlert:(NSString \*)message {

UIAlertView \*av = [[UIAlertView alloc] initWithTitle:@"Receive" message:message delegate:self

cancelButtonTitle:@"Ok" otherButtonTitles:nil];

[av show];

}

-(void)doConnect {

player = [[MediaStreamPlayer alloc] init:hostTextField.text];

player.delegate = self;

player.player = [[FramesPlayer alloc] initWithView:previewView];

[player stream:streamTextField.text];

btnConnect.title = @"Disconnect";

}

-(void)doDisconnect {

[player disconnect];

player = nil;

btnConnect.title = @"Connect";

btnPlay.title = @"Start";

btnPlay.enabled = NO;

hostTextField.hidden = NO;

streamTextField.hidden = NO;

previewView.hidden = YES;

}

Method showAlert: is used for notification about error situation.

Method doConnect creates instance “player” of MediaStreamPlayer class, which implements network and media processing. The “self” is declared as delegate of player – so it must implement IMediaStreamEvent protocol (below). Then player gets previewView as view for playing stream (it uses build-in FramesPlayer class). Stream is opened with name from streamTextFild . “Connect” button gets “Disconnect” title.

Method doDisconnect deallocates player instance and makes appropriate changes in visable/active attributes and titles of UI elements.

#pragma mark -

#pragma mark Public Methods

// ACTIONS

-(IBAction)connectControl:(id)sender {

NSLog(@"connectControl: host = %@", hostTextField.text);

(!player) ? [self doConnect] : [self doDisconnect];

}

-(IBAction)playControl:(id)sender; {

NSLog(@"playControl: stream = %@", streamTextField.text);

(player.state != STREAM\_PLAYING) ? [player start] : [player pause];

}

There are button callbacks. Each of them is called by pushing appropriate button:

* method connectControl: makes connect/disconnect operations;
* method playControl: starts/resumes stream playing ([player start]) or makes a pause ([player pause).

#pragma mark -

#pragma mark UITextFieldDelegate Methods

-(BOOL)textFieldShouldReturn:(UITextField \*)textField {

[textField resignFirstResponder];

return YES;

}

This is the implementation of UITextDelegate protocol. It lets the keyboard to disappear after text editing have been finished.

#pragma mark -

#pragma mark IMediaStreamEvent Methods

-(void)stateChanged:(MediaStreamState)state description:(NSString \*)description {

NSLog(@" $$$$$$ <IMediaStreamEvent> stateChangedEvent: %d = %@", (int)state, description);

switch (state) {

case CONN\_DISCONNECTED: {

[self doDisconnect];

[self showAlert:[NSString stringWithString:description]];

break;

}

case STREAM\_CREATED: {

[player start];

hostTextField.hidden = YES;

streamTextField.hidden = YES;

previewView.hidden = NO;

btnPlay.enabled = YES;

break;

}

case STREAM\_PAUSED: {

btnPlay.title = @"Start";

break;

}

case STREAM\_PLAYING: {

if ([description isEqualToString:@"NetStream.Play.StreamNotFound"]) {

[player stop];

[self showAlert:[NSString stringWithString:description]];

break;

}

btnPlay.title = @"Pause";

break;

}

default:

break;

}

}

-(void)connectFailed:(int)code description:(NSString \*)description {

NSLog(@" $$$$$$ <IMediaStreamEvent> connectFailedEvent: %d = %@\n", code, description);

[self doDisconnect];

[self showAlert:(code == -1) ?

[NSString stringWithFormat:@"Unable to connect to the server. Make sure the hostname/IP address and port number are valid\n"] :

[NSString stringWithFormat:@"connectFailedEvent: %@ \n", description]];

}

This is the implementation of IMediaStreamEvent protocol, player can call its methods for notification about state changes and error situations:

* method stateChanged:description: makes the reaction to appropriate state change – it calls doDisconnect and showAlert for CONN\_DISCONNECTED state, it calls [player start] for STREAM\_CREATED state, it calls [player stop] and ShowAlert for STREAM\_PLAYING state with description = “NetStream.Play.StreamNotFound”, and makes appropriate changes in visable/active attributes and titles of UI elements for all this states;
* method connectFailed:description: reacts for error situations during the net connection process - it calls doDisconnect and showAlert.

Now you can build and run this app – good luck!