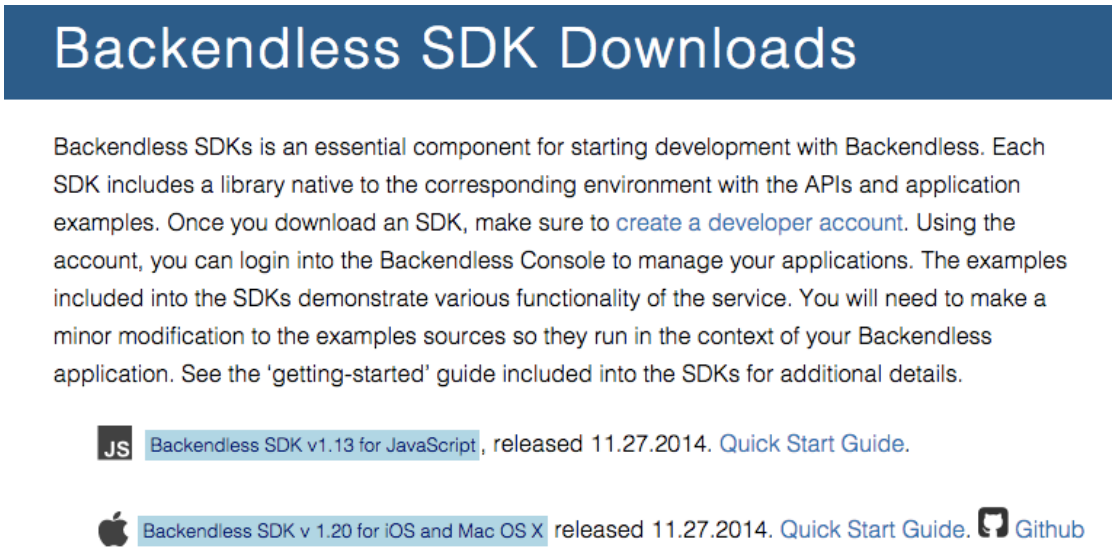
Developing an iOS Video Recorder and Player on Swift in 10 minutes from the scratch

Video broadcasting and streaming is one of the coolest features of Backendless. Our Media Service API enables client-server functionality for working with live and on-demand audio and video content. An application using the Media Service API can broadcast audio and video from the devices cameras and microphone and Backendless automatically handles streaming to other clients as well recording of the content on the server. The API also supports the capability to stream the pre-recorded (on-demand) content managed by the Media Service. Details are available in the documentation.

This post describes how to build an iOS application which can record a video on the server and then subsequently play it back.

Getting Backendless SDK

1. Login to your Backendless account or register to create a new one.
2. Download Backendless SDK for iOS from the Backendless SDK Downloads page and unzip it.



Getting Started

1. Start up Xcode and go to *File->New->Project*. Select *iOS->Application->Single View Applicatio"*, and click *Next*.

2. Enter *VideoService* for the *Product Name*, set the *Language* to *Swift*, and *Devices* to *iPhone*. Make sure *Use Core Data* is *not checked*, and click *Next*.

3. Choose a directory to save your project, and click *Create*.

Let's see what Xcode has built for you. In the upper left corner of Xcode, select the *iPhone 6 Simulator* and click *Play* to test your app.

You should see a blank white screen appear. Xcode has created a single blank screen in your app.

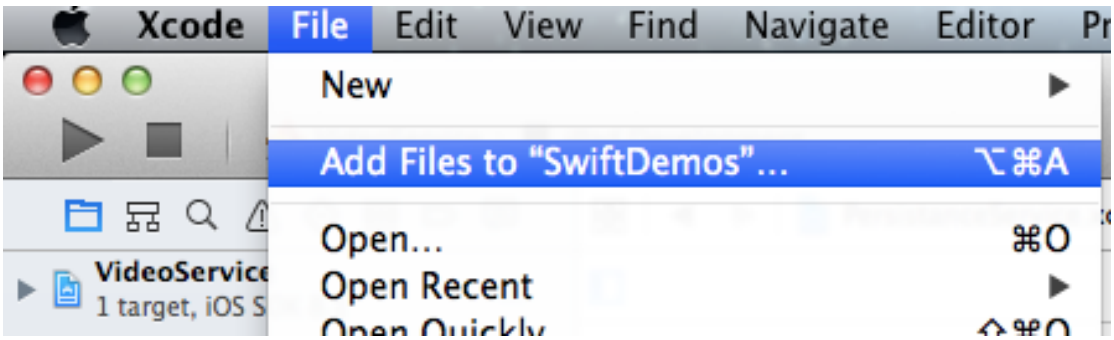
Add the Libraries and Frameworks to the project

1. Choose the target VideoService, go to Build Phases->Link Binary With Libraries, push "+", check the following iOS frameworks and libraries:

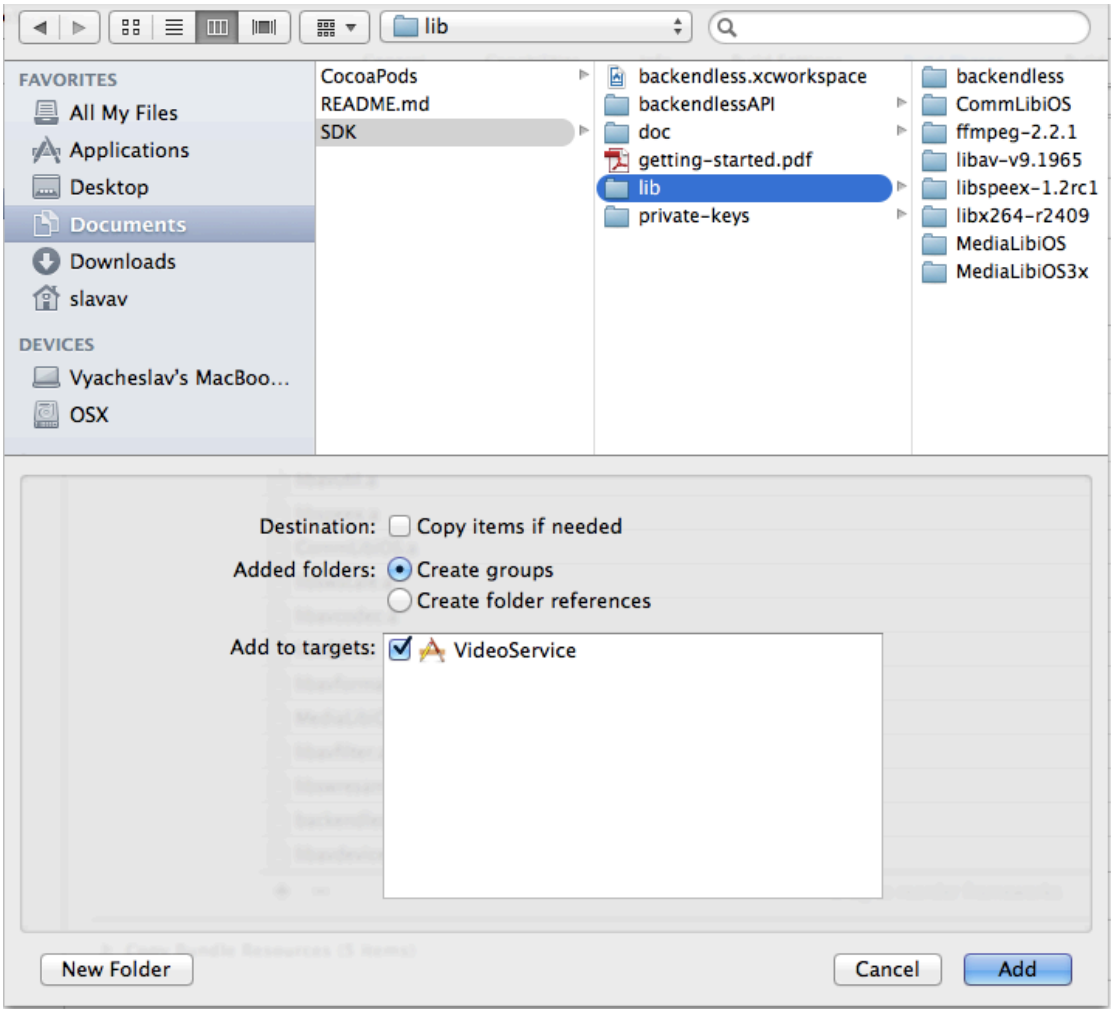
- SystemConfiguration.framework
- libsqlite3.dylib
- libz.dylib

Push "Add" button.

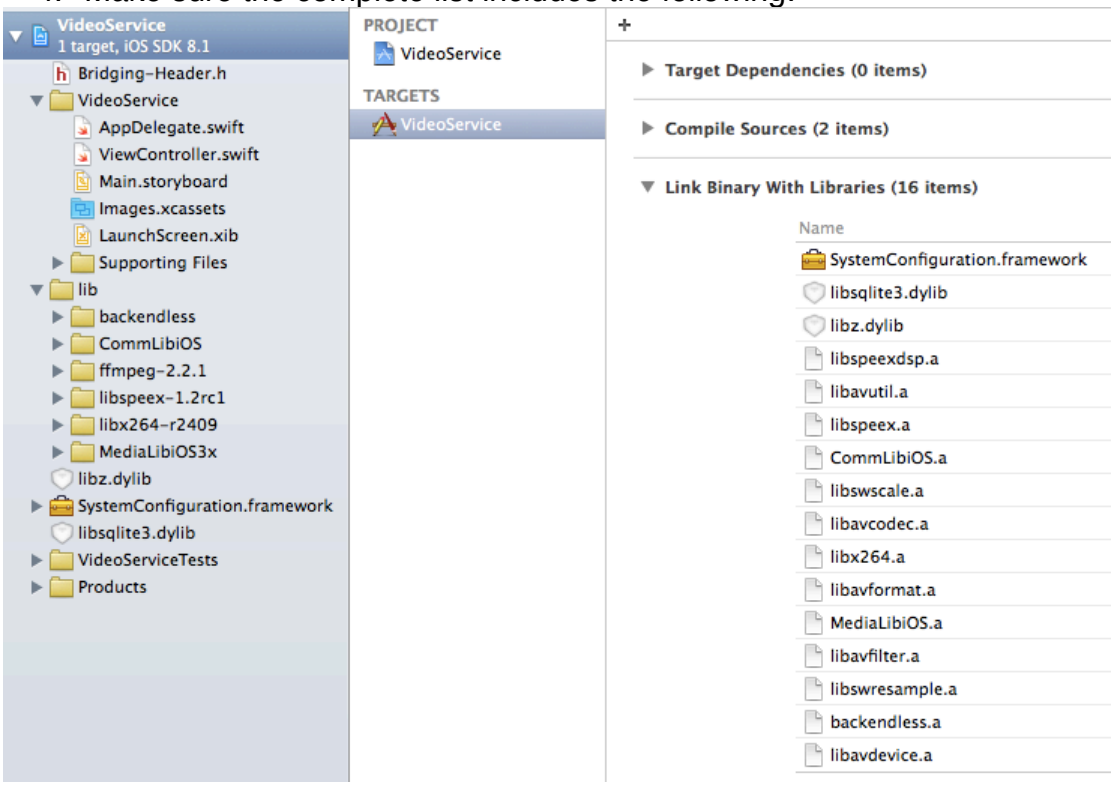
2. Mark the project and choose File->"Add Files to …" menu item:



3. In window choose the "lib" folder from the SDK folder. Please note that the "Add to targets" checkbox must be checked. Push "Add" button.



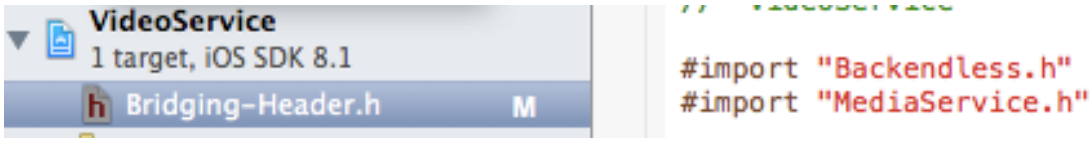
4. Make sure the complete list includes the following:



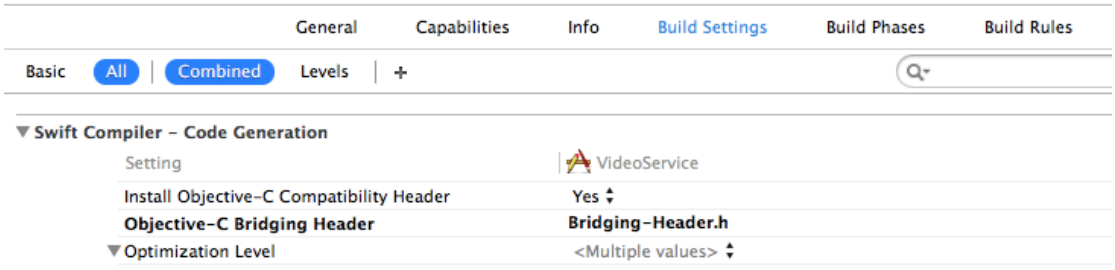
Add the Bridging Header File

For using Objective-C static libs on Swift the Bridging Header File is needed:

1. Add Bridging-Header.h to project and include the following code to it:

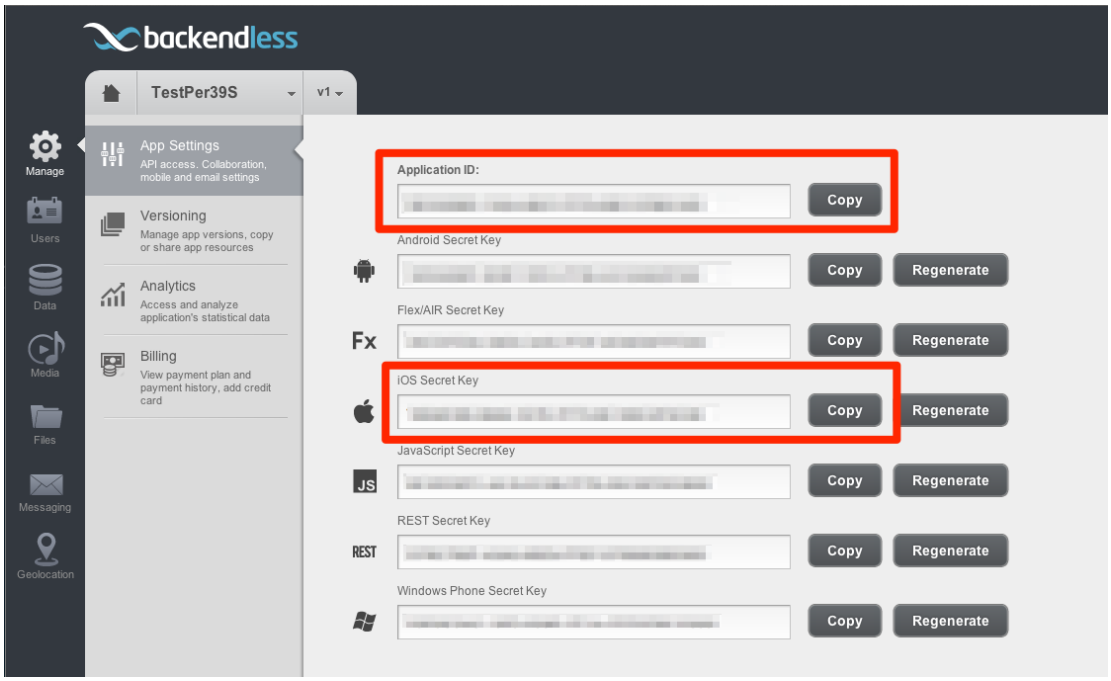


2. Add Bridging-Header.h to Build Settings:



Add Backendless Application Id & Secret Key

1. Get your Backendless application and secret keys for iOS from the Backendless Console. The keys can be found on the Manage -> App Settings section:



2. Add Backendless application initialization code block to AppDelegate.swift:

```
@UIApplicationMain
class AppDelegate: UIResponder, UIApplicationDelegate {

    let APP_ID = "-YOUR-APPLICATION-ID-”
    let SECRET_KEY = "-YOUR-APPLICATION-IOS-SECRET-KEY-"
    let VERSION_NUM = "v1"

    var backendless = Backendless.sharedInstance()

    var window: UIWindow?

    func application(application: UIApplication, didFinishLaunchingWithOptions
launchOptions: [NSObject: AnyObject]?) -> Bool {

        //DebLog.setIsActive(true)

        backendless.initApp(APP_ID, secret:SECRET_KEY, version:VERSION_NUM)
        backendless.setThrowException(false)
        backendless.mediaService = MediaService()

        return true
    }
 . . .
}
```

Introduction to Storyboards and Interface Builder

You create the user interface for your iOS apps in something called a Storyboard. Xcode comes with a built-in tool called Interface Builder that allows you to edit Storyboards in a nice, visual way.

With Interface Builder, you can lay out all of your buttons, text fields, labels, and other controls in your app (called Views) as simply as dragging and dropping.

Go ahead and click on Main.storyboard in the left side of Xcode to reveal the Storyboard in Interface Builder.

There’s a lot of stuff to cover here, so let’s go over each section of the screen one at a time.

1. On the far left is your Project Navigator, where your can see the files in your project.
2. On the left of Interface Builder is your Document Outline, where you can see at a glance the views inside each “screen” of your app (view

controllers). Be sure to click the “down” arrows next to each item to fully expand the document outline. Right now your app only has one view controller, with only one empty white view. You’ll be adding things into this soon.

3. There’s an arrow to the left of your view controller. This indicates that this is the initial view controller, or the view controller that is first displayed when the app starts up. You can change this by dragging the arrow to a different view controller, or by clicking the “Is Initial View Controller” property on a different view controller in the Attributes.

4. On the bottom of the Interface Builder you’ll see something that says “w Any”, “h Any”. This means that you are editing the layout for your app in a way that should work on any sized user interface. You can do this through the power of something called Auto Layout. By clicking this area, you can switch to editing the layout for devices of specific size classes.

5. On the top of your view controller you’ll see three small icons, which represent the view controller itself and two other items: First Responder, and Exit. If you’ve been developing in Xcode for a while, you’ll notice that these have moved (they used to be below the view controller).

6. On the bottom right of Interface Builder are four icons related to Auto Layout. You will learn more about these below.

7. On the upper right of Interface Builder are the Inspectors for whatever you have selected in the Document Outline. If you do not see the inspectors, go to View->Utilities->Show Utilities. Note there are several tabs of inspectors. You will be using these a lot to configure the views you add to this project.

8. On the bottom right of Interface Builder are the Libraries. This is a list of different types of views or view controllers you can add to your app. Soon you will be dragging items from your library into your view controller to lay out your app.

Creating the Views

The application should have the following UI elements:

- Text Field for stream name typing,
- Switch with label “Live” to control live/record streaming mode,
- View for published stream reviewing,
- Image View for playback stream viewing,
- the four Buttons: “Publish”, “Playback”, “Stop” and “Switch Cameras”.

Let’s build this user interface one piece at a time.

1. Navigation Bar. Rather than adding a navigation bar directly, select your view controller and go to Editor->Embed In->Navigation Controller. This will set up a Navigation Bar in your view controller. Double click the Navigation Bar (the one inside your view controller), and set the text to “Backendless Video Service”.
2. Text Field. From the Object Library, drag a Text Field into your view controller. In the Attributes Inspector (the Inspector‘s fourth tab) set Placeholder=”Stream Name”. In the Size Inspector (the Inspector‘s fifth tab) set X=19, Y=70, Width=192 and Height=30.
3. Label. From the Object Library, drag a Label into your view controller. Double click the label and set its text to “Live”. Select the label, and in the Size Inspector set X=219, Y=70, Width=42 and Height=30.
4. Switch. From the Object Library, drag a Switch into your view controller. In the Attribute Inspector, set State=On. In the Size Inspector set X=257 and Y=70.
5. View. From the Object Library, drag a View into your view controller. In the Attributes Inspector set Hidden=switch on. In the Size Inspector set X=0, Y=110, Width=320 and Height=450.
6. Image View. From the Object Library, drag a Image View into your view controller. In the Attributes Inspector set Hidden=switch on. In the Size Inspector set X=0, Y=110, Width=320 and Height=450.
7. Button “Publish”. From the Object Library, drag a Button into your view controller. Double click the Button, and set the text to “Publish”. In the Size Inspector set X=16, Y=562, Width=65 and Height=30.
8. Button “Playback”. From the Object Library, drag a Button into your view controller. Double click the Button, and set the text to “Playback”. In the Size Inspector set X=116, Y=562, Width=69 and Height=30.

9. Button “Stop”. From the Object Library, drag a Button into your view controller. Double click the Button, and set the text to “Stop”. In the Attributes Inspector set Hidden=switch on. In the Size Inspector set X=73, Y=562, Width=46 and Height=30.
10. Button “Switch Cameras”. From the Object Library, drag a Button into your view controller. Double click the Button, and set the text to “Switch Cameras”. In the Attributes Inspector set Hidden=switch on. In the Size Inspector set X=188, Y=562, Width=118 and Height=30.
11. Activity Indicator View. From the Object Library, drag an Activity Indicator View into your view controller. In the Attributes Inspector set Style=Gray, Hides When Stopped=switch on and Hidden=switch on. In the Size Inspector set X=150, Y=274, Width=20 and Height=20.
12. Tap Gesture Recognizer. From the Object Library, drag a Tap Gesture Recognizer onto your main view. This will be used to tell when the user taps the view to dismiss the keyboard.
13. Auto Layout. Interface Builder can often do a great job setting up reasonable Auto Layout constraints for you automatically; and it definitely can in this case. To do this, click on the third button in the lower left of the Interface Builder (which looks like a Tie Fighter) and select Add Missing Constraints.

Build and run on your iPhone 6 simulator, and you should see a basic user interface working already!

Connecting View Controller to Views

Let’s add some properties for its subviews, and hook them up in interface builder.

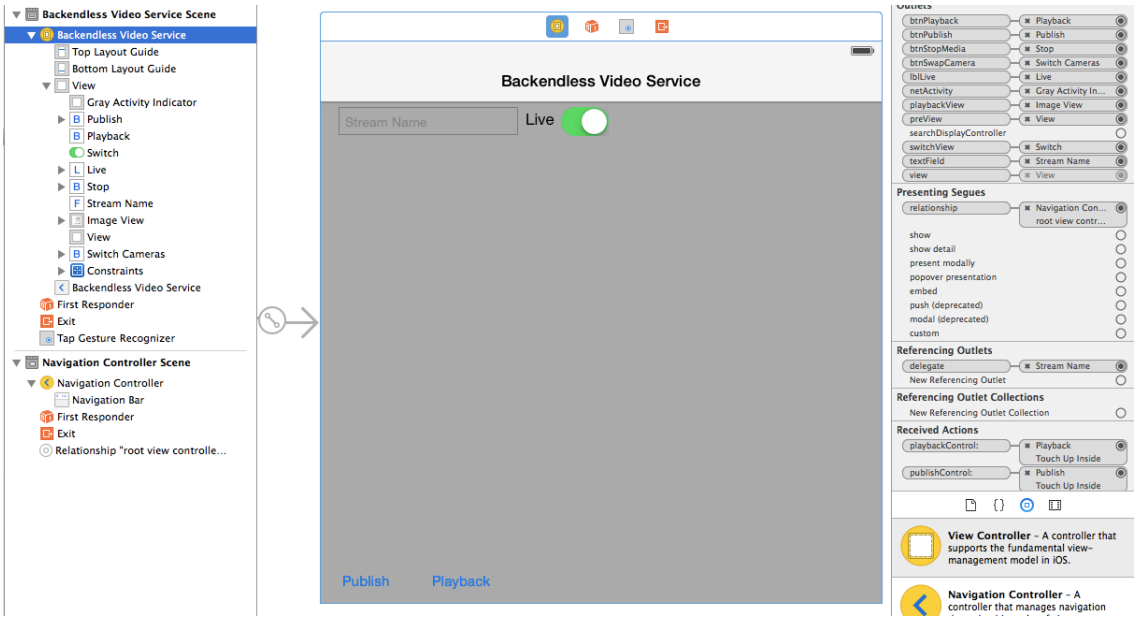
To do this, add these following properties to your ViewController class right before viewDidLoad() method:

```
@IBOutlet var btnPublish : UIButton!
@IBOutlet var btnPlayback : UIButton!
@IBOutlet var btnStopMedia : UIButton!
@IBOutlet var btnSwapCamera : UIButton!
@IBOutlet var preView : UIView!
@IBOutlet var playbackView : UIImageView!
@IBOutlet var textField : UITextField!
@IBOutlet var lblLive : UILabel!
@IBOutlet var switchView : UISwitch!
```

```
@IBOutlet var netActivity : UIActivityIndicatorView!
```

Let's try connecting these properties to the user interface elements.

Open *Main.storyboard* and select your View Controller named "Backendless Video Service" in the Document Outline. Open the *Connections Inspector* (the *Inspector*'s sixth tab), and you will see all of the properties you created listed in the *Outlets* section.



You'll notice a small circle to the right of btnPlayback. Control-drag from that button down to the "Playback" button, and release to connect your Swift property to this view.

Now repeat this for the other properties, connecting each one to the appropriate UI element:

- btnPublish to "Publish" button,
- btnStopMedia to "Stop" button,
- btnSwapCamera to "Switch Cameras" button,
- lblLive to "Live" label,
- netActivity to Gray Activity Indicator,
- playbackView to Image View,
- preview to View,

- switchView to Switch,
- textField to “Stream Name” text field.

And finally, in Referencing Outlets section you can see delegate property. Control-drag from its small circle to “Stream Name” text field allowing the View Controller to process UITextFieldDelegate callbacks.

Connecting Actions to View Controller

Just like you connected views to properties on your view controller, you want to connect certain actions from your views (such as a button click) to methods on your view controller.

To do this, open ViewController.swift and add these five new methods in the bottom of class:

```
@IBAction func switchCamerasControl(sender: AnyObject) {
}
@IBAction func stopMediaControl(sender: AnyObject) {
}
@IBAction func playbackControl(sender: AnyObject) {
}
@IBAction func publishControl(sender: AnyObject) {
}
@IBAction func viewTapped(sender: AnyObject) {
}
```

To make Interface Builder notice your new methods, you need to mark these methods with the *@IBAction* .

Next, switch back to Main.storyboard and make sure that your view controller is selected in the Document Outline. Make sure the Connections Inspector is open (the Inspector‘s sixth tab) and you will see your new methods listed in the Received Actions section.

Find the circle to the right of playbackControl:, and drag a line from that circle up to the “Playback” button. In the popup that appears,

choose “Touch Up Inside:”. This is effectively saying “when the user releases their finger from the screen when over the button, call my method playbackControl:“.

Now repeat this for the other methods:

- Drag a line from publishControl: to the “Publish” button, and choose “Touch Up Inside:”,
- Drag a line from stopMediaControl: to the “Stop” button, and choose “Touch Up Inside:”,
- Drag a line from switchCamerasControl: to the “Switch Cameras” button, and choose “Touch Up Inside:”,
- Drag from viewTapped: to the Tap Gesture Recognizer in the document outline. There are no actions to choose from for gesture recognizers; your method will simply be called with the recognizer is triggered.

Add Backendless MediaService elements to View Controller

1. Add the *UITextFieldDelegate* and I*MediaStreamerDelegate* prorocols to ViewController class just after UIViewController class name:

```
class ViewController: UIViewController, UITextFieldDelegate, IMediaStreamerDelegate {
```

2. Declare the following variables and constant just after outlets declaration:

```
    var backendless = Backendless.sharedInstance()

    var _publisher: MediaPublisher?
    var _player: MediaPlayer?

    let VIDEO_TUBE = "videoTube"
```

3. Implement publishControl action to start the live/record stream publishing:

```
    @IBAction func publishControl(sender: AnyObject) {


        var options: MediaPublishOptions
        if (switchView.on) {
            options = MediaPublishOptions.liveStream(self.preView) as
MediaPublishOptions
        }
```

```
        else {
            options = MediaPublishOptions.recordStream(self.preView) as
MediaPublishOptions
        }

        options.orientation = .Portrait
        options.resolution = RESOLUTION_CIF

        _publisher = backendless.mediaService.publishStream(textField.text,
tube:VIDEO_TUBE, options:options, responder:self)

        self.btnPublish.hidden = true
        self.btnPlayback.hidden = true
        self.textField.enabled = false
        self.switchView.enabled = false

        self.netActivity.startAnimating()
    }
```

An “options” instance of MediaPublishOptions sets the publishing mode (live or record) according to switchView state, orientation and resolution values for the stream, and the UIView instance reference that will show the video being published.

The publisher constructor accepts stream name, a tube name, options and a responder. Since the responder is set to self, the ViewController class must implement *IMediaStreamerDelegate* protocol.

4. Implement playbackControl action to start the stream playing:

```
    @IBAction func playbackControl(sender: AnyObject) {


        var options: MediaPlaybackOptions
        if (switchView.on) {
            options = MediaPlaybackOptions.liveStream(self.playbackView) as
MediaPlaybackOptions
        }
        else {
            options = MediaPlaybackOptions.recordStream(self.playbackView) as
MediaPlaybackOptions
        }

        options.orientation = .Up
        options.isRealTime = switchView.on

        _player = backendless.mediaService.playbackStream(textField.text,
tube:VIDEO_TUBE, options:options, responder:self)

        self.btnPublish.hidden = true
        self.btnPlayback.hidden = true
        self.textField.enabled = false
        self.switchView.enabled = false

        self.netActivity.startAnimating()
    }
```

An “options” instance of MediaPlaybackOptions sets the playing mode (live or record) according to switchView state, orientation and isRealTime values for the stream, and the UIImageView instance reference that will

show the played video. The player constructor accepts stream name, a tube name, options and a responder.

5. Implement switchCamerasControl action to switch the publishing stream between the front and back cameras:

```
@IBAction func switchCamerasControl(sender: AnyObject) {
     _publisher?.switchCameras()
   }
```

6. Implement stopMediaControl action to stop the stream publishing / playing:

```
@IBAction func stopMediaControl(sender: AnyObject) {


        if (_publisher != nil) {

            _publisher?.disconnect()
            _publisher = nil;

            self.preView.hidden = true
            self.btnStopMedia.hidden = true
            self.btnSwapCamera.hidden = true
        }

        if (_player != nil)
        {
            _player?.disconnect()
            _player = nil;
            self.playbackView.hidden = true
            self.btnStopMedia.hidden = true
        }

        self.btnPublish.hidden = false
        self.btnPlayback.hidden = false
        self.textField.enabled = true
        self.switchView.enabled = true

        self.netActivity.stopAnimating()
    }
```

When the user pushes “Stop” button, this action checks if mode is currently publishing or playing the video, then disconnect from the stream and set the publisher or player to nil.

7. Implement viewTapped() action and textFieldShouldReturn() method of *UITextFieldDelegate* protocol to dismiss the keyboard after stream name typing is finished:

```
    @IBAction func viewTapped(sender: AnyObject) {
        textField.resignFirstResponder()
    }

    // UITextFieldDelegate protocol methods
```

```
    func textFieldShouldReturn(_textField: UITextField) {
        textField.resignFirstResponder()
    }
```

8. And finally, implement *IMediaStreamerDelegate* protocol methods to handle the stream state changes and errors:

```
    func streamStateChanged(sender: AnyObject!, state: Int32, description:
String!) {

        switch state {

        case 0: //CONN_DISCONNECTED

            stopMediaControl(sender)
            return

        case 1: //CONN_CONNECTED
            return

        case 2: //STREAM_CREATED

            self.btnStopMedia.hidden = false
            return

        case 3: //STREAM_PLAYING

            // PUBLISHER
            if (_publisher != nil) {

                if (description != "NetStream.Publish.Start") {
                    stopMediaControl(sender)
                    return
                }

                self.preView.hidden = false
                self.btnSwapCamera.hidden = false
                netActivity.stopAnimating()
            }

            // PLAYER
            if (_player != nil) {

                if (description == "NetStream.Play.StreamNotFound") {
                    stopMediaControl(sender)
                    return
                }

                if (description != "NetStream.Play.Start") {
                    return
                }

                self.playbackView.hidden = false
                netActivity.stopAnimating()
            }

            return

        case 4: //STREAM_PAUSED

            if (description == "NetStream.Play.StreamNotFound") {
            }
```

```
            stopMediaControl(sender)
            return

        default:
            return
        }
    }

    func streamConnectFailed(sender: AnyObject!, code: Int32, description:
String!) {
        stopMediaControl(sender)
    }
```

You can download the sample code here.

That's all, enjoy