

13 장. 음악 재생하고 녹음하기

AVAudioPlayer 를 이용하여 오디오 파일을 재생, 일시 정지 및 정지하는 방법과 볼륨을 조절하는 방법 그리고 녹음하는 방법을 알아보자.



13-1 AVAudioPlayer 란?

아이폰에서는 대부분의 정보를 화면을 통해 제공하지만 간혹 소리를 이용해 정보를 제공하기도 한다,

예를 들어 운전 중일 때 화면을 통한 정보 제공은 위험한다. 이 때는 소리를 이용한 정보 전달이 가장 효과적인 방법일 것이다.

iOS에서는 기본적으로 음악 재생 앱과 녹음 앱을 제공한다. 오디오 파일을 재생할 수 있다면 벨소리나 알람과 같이 각종 소리와 관련된 다양한 작업을 할 수 있다. 또한 일정관리 앱에 녹음 기능을 추가해 목소리로 메모를 하는 등 메인 기능이 아닌 서브 기능으로도 사용할 수 있다. 오디오를 재생하는 방법 중 가장 쉬운 방법은 AVAudioPlayer를 사용하는 것이다.

오디오 포맷 / 코덱

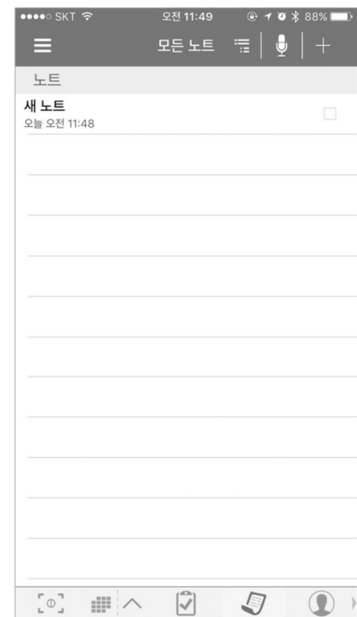
AAC(MPEG-4 Advanced Audio Coding)	ALAC(Apple Lossless Audio Codec)
HE-AAC(MPEG-4 High Efficiency AAC)	AMR(Adaptive Multirate)
Linear PCM(Linear Pulse Code Modulation)	iLBC(Internet Low Bit Rate Codec)
	MP3(MPEG-1 audio layer3)



음성 메모 앱



음악 앱

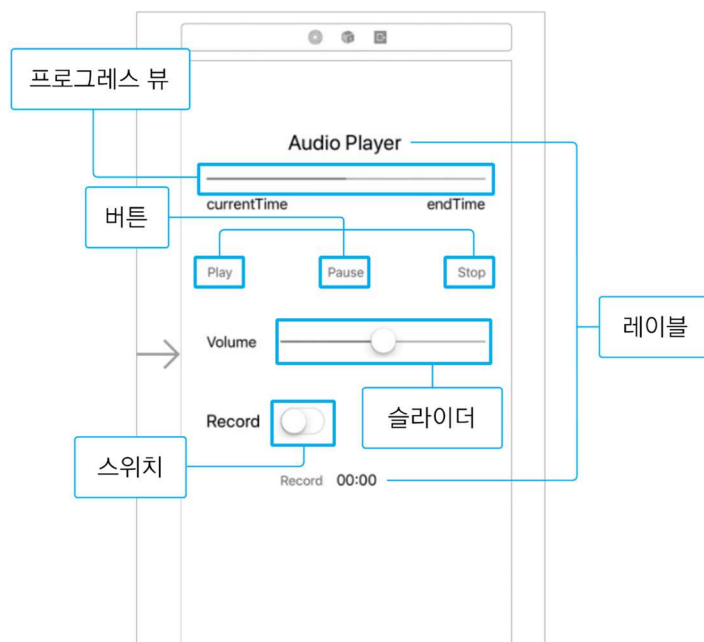


포켓 인포먼트(Pocket Informant) 앱

13-2 오디오재생및녹음앱을위한기본환경구성하기

오디오 재생 앱은 다양한 형태로 만들 수 있다. 하지만 이 장에서는 가장 기본적인 기능을 수행하는 형태의 앱을 만들 것이다, 앞에서 사용했던 버튼과 레이블 외에 프로그레스 뷰, 슬라이더도 만들어보자,

이제 각종 타이틀을 쓰기 위한 레이블(Label), 오디오 재생 및 녹음을 위한 버튼(Button), 오디오 재생 정도를 보여 줄 프로그레스 뷰(Progress View), 볼륨 조절을 위한 슬라이더(Slider) 그리고 재생 모드와 녹음 모드를 선택할 스위치(Switch)를 추가하여 스토리보드를 꾸며 보겠다.



1. 새프로젝트 만들기

Xcode 를 실행한 후 'Audio'라는 이름으로 프로젝트를 만든다.

2. 뷰 컨트롤러 크기 조절하기

아이폰 모양의 뷰 컨트롤러 크기를 상황에 맞게 조절한다,

3. 스토리보드 꾸미기

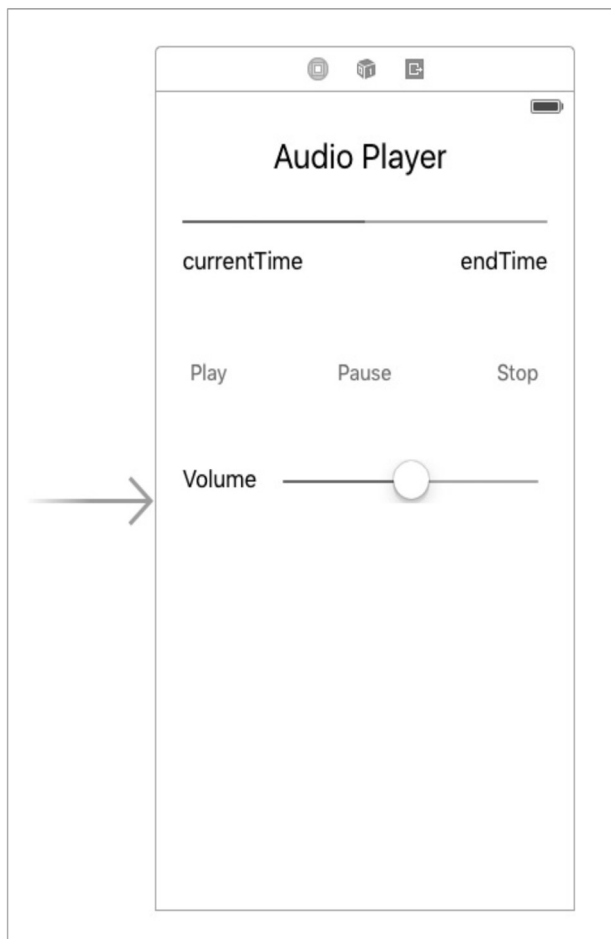
오브젝트 라이브러리에서 검색란에 'la'를 입력하여 검색한 후 [레이블(Label)]을 찾아 스토리 보드에 끌어다 놓는다. "Audio Player"로 text 를 변경한다,

4. 오브젝트 라이브러리에서 'pr'을 입력하여 검색한 후 [프로그레스 뷰(progress view)]를 찾아 스토리보드의 [Audio Player]아래에 배치한다.

5. 'Progress View' 아래쪽에 [레이블(Label)] 두개를 추가한다.그리고 내용을 'CurrentTime'과 'endTime'으로 수정한다. 이때 'endTime'은 오른쪽 정렬한다.

6. [currentTime]과 [endTime] 아래에 [버튼(Button)] 세개를 배치한 후 내용을 'Play', 'Pause' 그리고 'Stop'으로 변경한다. 그리고 그 버튼들 아래에는 레이블을 왼쪽에 놓고 'Volume'으로 수정한다.

7. 오브젝트 라이브러리에서 검색란에 'sl'을 입력하여 검색한 후 [슬라이더(Slider)]를 [Volume]의 오브젝트에 배치한다.



13-3 오디오 재생을 위한 아웃렛 변수와 액션 함수 추가하기

1. 오른쪽 윗부분의 [show the Assistant editor] 버튼을 클릭하여 보조 편집기 영역을 열면 화면이 두 영역으로 나뉘어진다. 오른쪽 보조 편집기 영역의 뷰 컨트롤러 클래스 선언문 바로 아래에 아웃렛 변수를 추가할 수 있도록 공간을 확보한다.

2. [프로그레스 뷰(Progress View)]를 마우스 왼쪽 버튼으로 선택한 후 마우스 버튼으로 드래그해서 오른쪽 보조 편집기 영역의 확보한 공간에 갖다 놓는다.

3. 설정창에서 아웃렛 변수의 이름을 'pvProgressPlay'로 입력하여 아웃렛 변수를 추가한다.

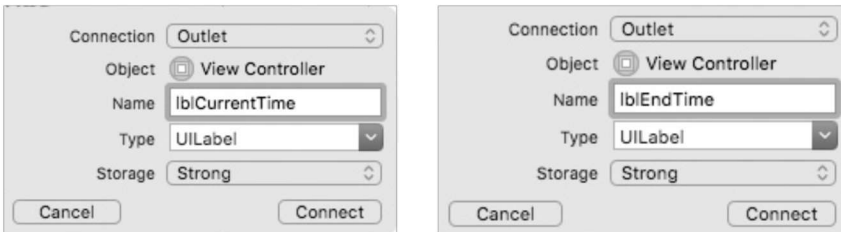
위치	뷰 컨트롤러의 클래스 선언문 바로 아래
연결(Connection)	Outlet
이름(Name)	pvProgressPlay
유형(Type)	UIProgressView



```
@IBOutlet var pvProgressPlay: UIProgressView!
```

4. 같은 방법으로 왼쪽 창에서 [currentTime]과 [endTime]을 선택한 후 오른쪽 보조 편집기 영역에서 생성한 아웃렛 변수 바로 아래에 끌어다 놓는다.

설정 창에서 아웃렛 변수의 이름은 각각 'lblCurrentTime', 'lblEndTime'으로 입력한다.

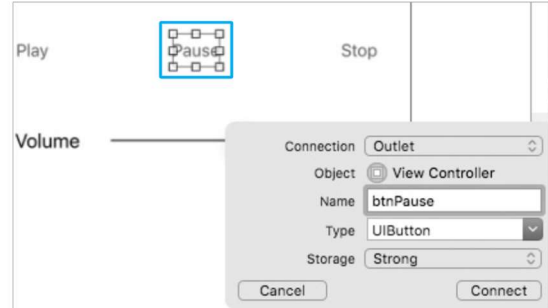
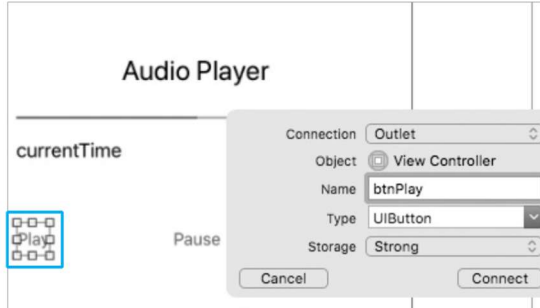


위치	앞에서 추가한 아웃렛 변수 바로 아래
연결(Connection)	Outlet
이름(Name)	- currentTime 레이블: lblCurrentTime - endTime 레이블: lblEndTime
유형(Type)	UILabel

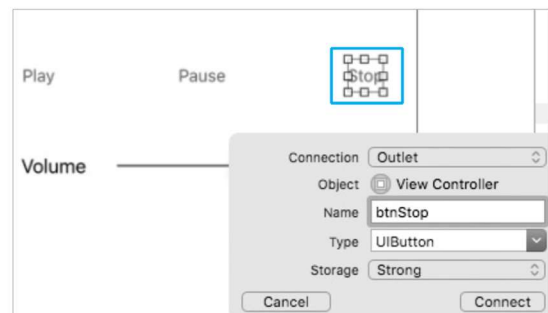
```
@IBOutlet var lblCurrentTime: UILabel!  
@IBOutlet var lblEndTime: UILabel!
```

5. 버튼 세 개의 아웃렛 변수를 추가한다. 방법은 앞에서 한 것과 동일하다.

마우스 오른쪽 버튼으로 버튼 세 개를 각각 끌어야 편집기 영역에 갖다 놓는다. 위치와 설정은 다음표를 참고한다.



위치	앞에서 추가한 아웃렛 변수 바로 아래
연결(Connection)	Outlet
이름(Name)	- Play 버튼: btnPlay - Pause 버튼: btnPause - Stop 버튼: btnStop
유형(Type)	UIButton

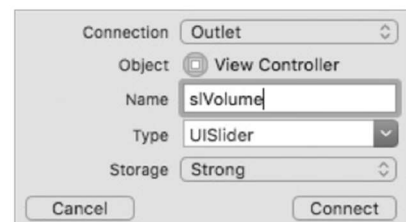


```
@IBOutlet var btnPlay: UIButton!
@IBOutlet var btnPause: UIButton!
@IBOutlet var btnStop: UIButton!
```

6. 계속해서 [슬라이더(Slider)]를 클릭한 후 앞에서 추가한 아웃렛 변수의 바로 아래에 끌어다 놓는다.

7. 설정 창에서 아웃렛 변수의 이름(Name)을 'slVolume'으로 입력한 후 [Connect]버튼을 클릭한다.

위치	앞에서 추가한 아웃렛 변수 바로 아래
연결(Connection)	Outlet
이름(Name)	slVolume
유형(Type)	UISlider



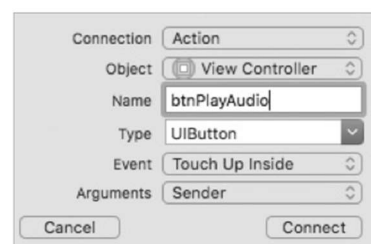
```
@IBOutlet var slVolume: UISlider!
```

8. 아웃렛 변수는 모두 추가했으니 이제 액션 함수를 추가한다. 이번에는 소스의 가장 아래쪽 닫힘 중괄호 '}' 바로 위에 추가한다.

9. 왼쪽 창에서 [Play]버튼을 마우스 오른쪽 버튼으로 클릭한 루 드래그해서 소스의 가장 아래쪽 '}' 바로 위에 갖다 놓는다.

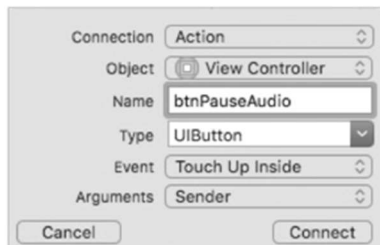
10. 설정 창에서 연결(Connection)을 [Action]으로 변경한다. 그리고 이름(Name)을 'btnPlayAudio'로 입력하고, 유형(Type)은 기기의 액션을 추가하는 것이므로 [UIButton]으로 변경한다. 변경을 완료한 후 [Connect]버튼을 클릭하여 추가한다.

위치	소스의 가장 아래쪽 '}' 바로 위
연결(Connection)	Action
이름(Name)	btnPlayAudio
유형(Type)	UIButton



```
@IBAction func btnPlayAudio(_ sender: UIButton) {
}
```

11. 나머지 버튼[Pause]와 [Stop]의 액션 함수도 같은 방법으로 추가한다. 액션 함수의 위치와 설정은 다음 표를 참고 한다.



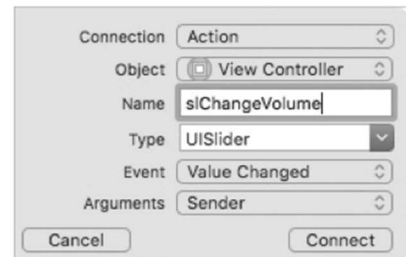
위치	앞에서 추가한 액션 함수 바로 아래
연결(Connection)	Action
이름(Name)	- Pause 버튼: btnPauseAudio - Stop 버튼: btnStopAudio
유형(Type)	UIButton

```
@IBAction func btnPauseAudio(_ sender: UIButton) {
}
@IBAction func btnStopAudio(_ sender: UIButton) {
}
```

12. 왼쪽 창에서 [슬라이더(Slider)]를 버튼으로 클릭한 후 드래그해서 줌전에 추가한 액션 함수 아래에 갖다 놓는다

연결 설정 창에서 연결(Connection)을 [Action]으로 선택한다. 그리고 이름(Name)을 'slChangeVolume'으로 입력하고, 유형(Type)은 슬라이더의 액션을 추가하는 것이므로 [UISlider]를 선택한다. 변경을 완료한 후 [Connect]버튼을 클릭하여 추가한다.

위치	앞에서 추가한 액션 함수 바로 아래
연결(Connection)	Action
이름(Name)	slChangeVolume
유형(Type)	UISlider



```
@IBAction func slChangeVolume(_ sender: UISlider) {  
}
```


13-4 오디오 재생을 위한 초기화하기

실제로 오디오를 재생하려면 오디오 파일을 불러오고 추가 설정도 해야 한다. 추가 설정이 필요한 부분은 소리의 크기를 조절하기 위한 볼륨, 볼륨을 표시할 슬라이더, 재생 시간을 표시하기 위한 타이머, 재생 정도를 표시할 프로그레스 뷰 등이다.

또한 오디오를 재생하려면 '초기화'라는 중요한 단계를 거쳐야 한다. 여기서 초기화란 오디오를 재생하기 위한 준비 과정뿐만 아니라 재생 시간을 맨 처음으로 돌이키고 버튼의 활성화 또는 비 활성화를 설정하는 과정까지 말한다. 이 초기화는 재생뿐만 아니라 녹음할 때도 필요하므로 개념을 잘 이해해 두기 바란다.

1. 오디오 재생을 위한 상수와 변수 추가하기

코딩을 위해 모드를 수정한 후 왼쪽의 내비게이터 영역에서 [ViewController.swift]를 선택한다, 오디오를 재생하려면 헤더 파일과 델리게이트가 필요하므로 'AVFoundation'을 불러오고, 'AVAudioPlayerDelegate'선언을 추가한다.

```
import UIKit
import AVFoundation

class ViewController: UIViewController, AVAudioPlayerDelegate {
```

2. 클래스에서 사용할 변수와 상수를 선언하겠다. 여기서 사용된 변수와 상수의 의미는 다음과 같다.

```
1 import UIKit
2 import AVFoundation
3
4 class ViewController: UIViewController, AVAudioPlayerDelegate {
5
6     var audioPlayer : AVAudioPlayer!
7     var audioFile : URL!
8
9     let MAX_VOLUME : Float = 10.0
10
11     var progressTimer : Timer!
12 }
```

7 행은 AVAudioPlayer 인스턴스 변수이다

8 행은 재생할 오디오의 파일명 변수

10 행은 최대 볼륨, 실수형 상수

12 행은 타이틀을 위한 변수

3. 오디오 파일 추가하기

재생할 음악인 'Sicilian_Breeze.mp3'를 내비게이터 영역에서 끌어다 추가한다. 그리고

[Target Membership]에 추가되었는지 확인한다,

4. viewDidLoad 함수의 audioFile 변수를 방금 추가한 'Sicilian_Breeze.mp3'로 설정한다,

```
override func viewDidLoad() {  
    audioFile = Bundle.main.url(forResource:  
        "Sicilian_Breeze", withExtension: "mp3")  
}
```

5. 오디오 재생을 위한 초기화하기

오디오 재생을 초기화하는 과정을 따로 함수로 만들겠다.

viewDidLoad 함수에 작성해도 좋지만 뒤에서 '재생 모드'와 '녹음 모드'로 변경할 때에 대비해서 '오디오 재생 초기화 과정'과 '녹음 초기화 과정'을 분리해 놓아야 편리하다.

따라서 이 함수를 viewDidLoad 함수 아래에 따로 추가한다.

```
func initPlay() {  
}
```

6. viewDidLoad 함수에 방금 작성한 initPlay 함수를 4 번 과정에서 추가한 audioFile 변수 아래에 추가한다.

```
override func viewDidLoad() {  
    audioFile = Bundle.main.url(forResource:  
        "Sicilian_Breeze", withExtension: "mp3")  
    initPlay()  
}
```

7. initPlay 함수에 앞에서 초기화한 audioFile 을 URL 로 하는 audioPlayer 인스턴스를 생성한다. 이때 AVAudioPlayer 함수는 입력 파라미터인 오디오 파일이 없을 때에 대비하여 try ~ catch 문을 사용한다.

```
func initPlay() {  
    do {  
        audioPlayer = try AVAudioPlayer(contentsOf: audioFile)  
    } catch let error as NSError {  
        print("Error-initPlay : \(error)")  
    }  
}
```

8. 이제 오디오를 재생할 때 필요한 모든 값을 초기화한다.

```
1 func initPlay() {  
2     do {  
3         audioPlayer = try AVAudioPlayer(contentsOf: audioFile)  
4     } catch let error as NSError {  
5         print("Error-initPlay : \(error)")  
6     }  
7     siVolume.maximumValue = MAX_VOLUME  
8     siVolume.value = 1.0  
9     pvProgressPlay.progress = 0
```

9	<code>audioPlayer.delegate = self</code>
10	<code>audioPlayer.prepareToPlay()</code>
11	<code>audioPlayer.volume = siVolume.value</code>
12	<code>}</code>
13	
14	

7 행은 슬라이더의 최대 볼륨을 상수 MAX_VOLUME 인 10.0 초기화한다.

8 행은 슬라이더의 볼륨을 1.0 으로 초기화한다.

9 행은 프로그레스 뷰의 진행을 0 으로 초기화한다,

11 행은 audioPlayer 의 델리게이트를 self 로 한다.

12 행은 prepareToPlay()를 실행한다.

13 행은 audioPlayer 의 볼륨을 방금 앞에서 초기화한 슬라이더의 볼륨값 1.0 으로 초기화한다.

13-5 재생 시간 초기화하기

오디오 재생을 위한 초기화에 관여하는 것은 '재생 시간'과 '버튼'이다. 먼저 재생 시간을 초기화해 보겠다.

1. 'endTime'레이블인 lblEndTime 에 총 재생 시간(오디오 꼭 길이)을 나타내기 위해 lblEndTime 을 초기화한다. 이 때 오디오의 총 재생 시간인 audioPlayer.duration 을 직접 사용하고 싶지만 시간 형태가 초 단위 실수 값이므로 "00:00"형태로 바꾸는 함수를 만든다. 다음 코드는 아직 완성되지 않았기 때문에 왼쪽에 빨간색 경고등이 켜지는데, 이것은 코드를 완성하고 나면 사라질 것이다.

2. "00:00"형태로 바꾸기 위해 TimeInterval 값을 받아 문자열(String)로 돌아보내는 함수 convertNSTimeInterval2String 를 생성한다

3. convertNSTimeInterval2String 함수안에 구체적인 코드를 입력한다,

```
1 func convertNSTimeInterval2String(_ time:TimeInterval) -> String {  
2     let min = Int(time/60)  
3     let sec = Int(time.truncatingRemainder(dividingBy: 60))  
4     let strTime = String(format: "%02d:%02d", min, sec)  
5     return strTime  
6 }  
7
```

3 행은 재생시간의 매개변수인 Time 값을 60 으로 나눈 '몫'을 정수값으로 반환하여 상수 min 값에 초기화한다.

4 행은 time 값을 60 으로 나눈 '나머지'값을 정수 값으로 변환하여 상수 sec 값에 초기화한다.

5 행은 이 두 값을 활용해 "%02d:%02d"형태의 문자열(String)로 변환하여 상수 strTime 에 초기화한다.

6 행은 이 값을 호출한 함수로 돌려본다.

4. 초기화한 값을 'endTime'레이블인 lblEndTime 에 나타낸다.

```
1 func initPlay() {  
2     ...  
3     pvProgressPlay.progress = 0
```

4	
5	audioPlayer.delegate = self
6	audioPlayer.prepareToPlay()
7	audioPlayer.volume = sIVolume.value
8	lblEndTime.text = convertNSTimeInterval2String(audioPlayer.duration)
	lblCurrentTime.text = convertNSTimeInterval2String(0)
9	}
10	

8 행은 오디오 파일의 재생시간인 audioPlyer.duration 값을

convertNSTimeInterval@String 함수를 이용해 lblEndTime 의 텍스트에 출력한다.

9 행은 lblCurrentTime 의 텍스트에는 convertNSTimeInterval2String 함수를 이용해 00:00
가 출력되도록 0 의 값을 입력한다.

13-6 재생, 일시정지및정지버튼제어하기

재생 시간을 초기화했으니 이제는 버튼을 제어해 본다.

오디오를 재생, 일시 정지, 정지했을 때 각 버튼의 활성화 및 비 활성화가 어떻게 해야 할 지 상상해 보자.

1. [Play]버튼은 오디오를 재생하는 역할을 하고 다른 두 버튼은 오디오를 멈추게 한다. 그러므로 재생에 관한 함수인 `initPlay` 함수에 [Play]버튼은 활성화, 나머지 두 버튼은 비 활성화하도록 코드를 추가한다.

```
1 func initPlay() {
2
3     ...
4     pvProgressPlay.progress = 0
5
6     audioPlayer.delegate = self
7     audioPlayer.prepareToPlay()
8     audioPlayer.volume = siVolume.value
9     lblEndTime.text = convertNSTimeInterval2String(audioPlayer.duration)
10    lblCurrentTime.text = convertNSTimeInterval2String(0)
11    btnPlay.isEnabled = true
12    btnPause.isEnabled = false
13    btnStop.isEnabled = false
14 }
```

2. [Play],[Pause] 그리고 [Stop]버튼의 동작 여부를 설정하는 부분은 앞으로도 계속 사용해야 하므로 함수를 따로 만들겠다. 이렇게 만든 함수에 '재생(Play)'/일시 정지(Pause)' 그리고 '정지(Stop)'의 순으로 `true`, `false` 값을 주면서 각각 설정할 것이다.

```
1 func setPlayButtons(_ play:Bool, pause:Bool, stop:Bool) {
2     btnPlay.isEnabled = play
3     btnPause.isEnabled = pause
4     btnStop.isEnabled = stop
5 }
```

3. 과정 1 에서 추가한 코드인 `btnPlay.isEnabled = true`, `btnPause.isEnabled = false` 그리고 `btnStop.isEnabled = false` 를 삭제하고 `setPlayButtons` 함수를 사용하여 다음과 같이 대체한다.

```
1 func initPlay() {  
2  
3     ...  
4     audioPlayer.volume = slVolume.value  
5     lblEndTime.text = convertNSTimeInterval2String(audioPlayer.duration)  
6     lblCurrentTime.text = convertNSTimeInterval2String(0)  
7     setPlayButtons(true, pause: false, stop: false)  
8 }
```

4. 오디오 재생하기

오디오가 재생 중일 때의 모습을 상상해 보자. [Play]버튼을 눌러 정상적으로 재생된다면 [Play]버튼은 비활성화되고 나머지 두 버튼은 활성화되어야 한다.

음악을 재생해야 하므로 `btnPlayAudio` 함수를 수정한다.

```
1 @IBAction func btnPlayAudio(_ sender: UIButton) {  
2     audioPlayer.play()  
3     setPlayButtons(false, pause: true, stop: true)  
4 }
```

2 행은 `audioPlayer.play` 함수를 실행해 오디오를 재생한다.

3 행은 [Play]버튼은 비활성화, 나머지 두 버튼은 활성화한다.

5. 오디오 일시 정지하기

‘재생’을 구현한 것과 비슷한 방식으로 ‘일시 정지’를 구현해 보겠다. 오디오를 잠시 멈추도록(일시 정지하도록) `btnPauseAudio` 함수를 수정한다. 일시정지 중이므로 `audioPlayer.pause` 함수를 실행하고 [Pause]버튼은 비활성화, 나머지 두 버튼은 활성화한다.

```
1 @IBAction func btnPauseAudio(_ sender: UIButton) {  
2     audioPlayer.pause()  
3     setPlayButtons(true, pause: false, stop: true)  
4 }
```

6. 오디오 정지하기

오디오를 멈추도록 `btnStopAudio` 함수를 수정한다. 정지 상태이므로 `audioPlayer.stop` 함수를 실행하고 [Play]버튼은 활성화, 나머지 두 버튼은 비활성화한다.

```
1 @IBAction func btnStopAudio(_ sender: UIButton) {  
2     audioPlayer.stop()  
3 }
```

3	<code>setPlayButtons(true, pause: false, stop: false)</code>
4	<code>}</code>

7. 결과 확인하기

총 재생 시간은 01:55 초로 표시되고, 재생시간은 00:00 으로 표시된다.

[Play]버튼을 클릭하면 오디오가 재생되고 [Play]버튼은 비활성화, 나머지 버튼은 활성화 된다. [Pause]와 [Stop]버튼도 잘 동작한다. 하지만 재생시간은 00:00 으로 변화가 없다.

13-7 재생 시간 표시하고 볼륨 제어하기

타이머(NSTimer)를 이용하여 재생 시간이 제대로 작동되도록 구현해 보겠다.

1. 우선 `btnPlayAudio` 함수를 함수를 수정하자. 프로그레스 타이머 (`ProgressTimer.scheduledTimer` 함수를 사용하여 0.1 초 간격으로 타이머를 생성하도록 구성한다. 셀렉터(selector)는 앞에서 선언한 상수를 `timePlayerSelector` 를 사용한다.

1	<code>@IBAction func btnPlayAudio(_ sender: UIButton) {</code>
2	<code>...</code>
3	
4	<code> progressTimer = Timer.scheduledTimer(timeInterval: 0.1, target: self,</code>
5	<code> selector: timePlayerSelector, userInfo: nil, repeats: true)</code>
6	<code>}</code>
7	

2. 아웃렛 변수를 선언한 위치 바로 위에 재생 타이머를 위한 상수를 추가한다.

1	<code>let MAX_VOLUME : Float = 10.0</code>
2	<code>var progressTimer : Timer!</code>
3	
4	<code>let timePlayerSelector:Selector =</code>
5	<code> #selector(ViewController.updatePlayTime)</code>

4~5 행을 추가한다.

3. `updatePlayTime` 함수를 생성한다.

앞에서 만든 타이머에 의해 0.1 초 간격으로 이 함수가 실행되는데, 그 때마다 `audioPlayer.currentTime`, 즉 재생 시간을 레이블 'lblCurrentTime'과 프로그레스 뷰 (Progress View)에 나타낸다.

1	<code>@objc func updatePlayTime() {</code>
---	--------------------------------------------


```

2      lblCurrentTime.text =
3          convertNSTimeInterval2String(audioPlayer.currentTime)
4      pvProgressPlay.progress =
5          Float(audioPlayer.currentTime/audioPlayer.duration)
6  }

```

2 행은 재생 시간인 `audioPlayer.currentTime` 을 레이블 'lblCurrentTime'에 나타낸다.

4 행은 프로그레스 뷰(Progress View)인 `pvProgressPlay` 의 진행 상황에 `audioPlayer.currentTime` 을 `audioPlayer.duration` 으로 나눈 값으로 표시한다.

4. 재생 중일 때 시간이 표시되도록 만들었으니 이번에는 정지했을 때 시간이 00:00 이 되도록 만들자. '정지'했을 때의 상황이므로 `btnStopAudio` 함수를 수정한다.

```

1  @IBAction func btnStopAudio(_ sender: UIButton) {
2      ...
3      audioPlayer.currentTime = 0
4      lblCurrentTime.text = convertNSTimeInterval2String(0)
5      ...
6      progressTimer.invalidate()
7  }

```

3 행은 오디오를 정지하고 다시 재생하면 처음부터 재생해야 하므로 `audioPlayer.currentTime` 을 0 으로 한다.

4 행은 재생 시간도 00:00 로 초기화하기 위해 `convert NSTimeInterval2String(0)`을 활용한다.

6 행은 타이머도 무효화한다.

5. 볼륨 조절하기

볼륨을 조절하기 위해서 `slChangeVolume` 함수를 수정하자. 화면의 슬라이더를 터치해 좌우로 움직이면 볼륨이 조절되도록 할 것이다. 이 동작을 구현하기 위해 슬라이더인 `slVolume` 의 값을 오디오 플레이어(audioPlayer)의 `volume` 값에 대입한다.

```

1  @IBAction func slChangeVolume(_ sender: UISlider) {
2      audioPlayer.volume = slVolume.value
3  }

```

6. 오디오 재생이 끝나면 맨 처음 상태로 돌아가도록 함수를 추가한다. 타이머도 무효화하고 버튼도 다시 정의해야 한다. 재생이 끝났으므로 [Play]버튼은 활성화, 나머지 버튼은 비 활성화한다.

```

1  func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool)
2  {
3      progressTimer.invalidate()
4      setPlayButtons(true, pause: false, stop: false)
5  }

```

3 행은 타이머를 무효화한다.

4 행은 [Play]버튼은 활성화하고 나머지 버튼은 비활성화한다.

7. 결과 확인

이제 재생 시간도 제대로 표시되고 볼륨조절도 가능하다.

오디오가 종료되면 [Play],[Pause],[Stop]버튼이 새로 생긴다.



13-8 녹음을 위한 스토리보드 꾸미기

녹음모드를 위한 작업을 시작 해보자.

1. 먼저 앞에서 만든 스토리보드에 추가 작업을 하자. 왼쪽의 내비게이터 영역에서 스토리보드를 클릭한다. 그리고 오른쪽 아랫부분의 오브젝트 라이브러리에서 검색란 'la'를 입력하여 검색한 후 [레이블(Label)]을 찾아 앞에서 만든 [Volume]아래쪽에 끌어다 놓는다. 내용은 'Recode'로 글씨 크기는 '20'으로 수정한다,



2. 또한 오른쪽 아랫부분의 오브젝트 라이브러리에서 검색란에 'sw'를 입력하여 검색한 후 [스위치(switch)]를 찾아 [Record]의 오른쪽에 끌어다 놓는다. 그리고 스위치의 상태를 [Off]로 설정한다. 앱의 기본화면은 '재생모드',스위치를 켜올 때는 '녹음모드'가 된다.

3. 오브젝트 라이브러리에서 [버튼(Button)]을 찾아 스위치의 아래쪽에 끌어다 놓는다. 그리고 이름을 'Record'로 바꾼다.

4. 오브젝트 라이브러리에서 [레이블(Label)]버튼 오른쪽에 끌어다 놓고 '00:00'으로 수정한다. 그리고 크기를 조금 키워 여유 있게 만든다.

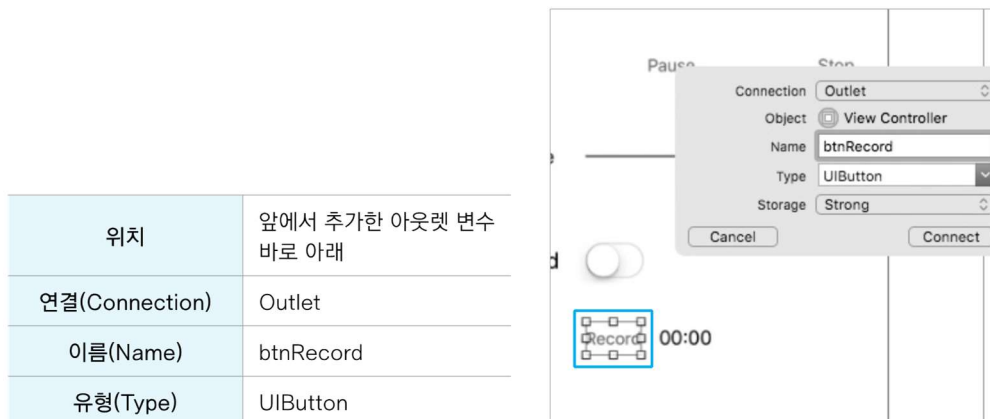
13-9 녹음을 위한 아웃렛 변수와 액션 함수 추가하기

새로 추가한 객체에 아웃렛 변수와 액션 함수를 추가해 보자.

1. 오른쪽 윗부분의 [Show the Assistant editor]버튼을 클릭하여 화면을 두 영역으로 나눈 후 왼쪽에는 'Main.storyboard'가, 오른쪽에는 'ViewController.swift'가 나타나게 한다.

2. [Record]버튼과 [00:00]레이블을 이전에 만들었던 아웃렛 변수 아래쪽에 배치한다.

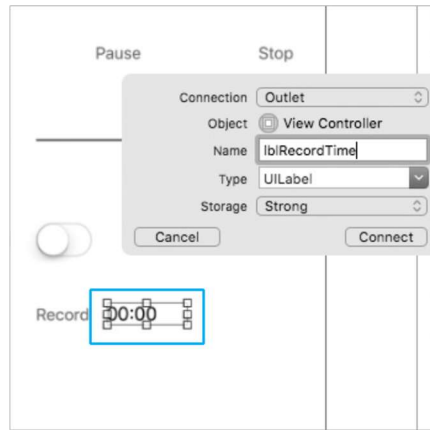
3. 왼쪽창에서 [Record]버튼을 클릭한 후 오른쪽 보조 편집기 영역에서 아웃렛 변수를 생성한 곳 바로 아래로 끌어다 놓는다, 설정창에서 아웃렛 변수의 이름(Name)을 'btnRecord'로 입력한 후 [Connect]버튼을 클릭하여 버튼과 아웃렛 변수를 연결한다.



```
@IBOutlet var btnRecord: UIButton!
```

4. 왼쪽창에서 [00:00]레이블 클릭한 후 아웃렛 변수를 연결한다. 여기서는 이름(Name)을 'lblRecordTime'으로 입력한다.

위치	앞에서 추가한 아웃렛 변수 바로 아래
연결(Connection)	Outlet
이름(Name)	lblRecordTime
유형(Type)	UILabel



```
@IBOutlet var lblRecordTime: UILabel!
```

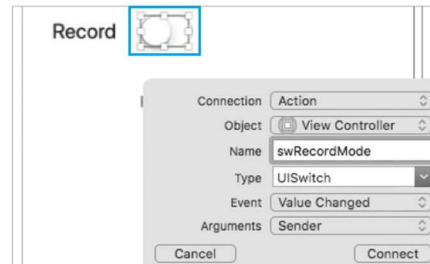
5. [스위치]와 [Record]버튼을 이전에 생성했던 클래스의 마지막 아래쪽에 배치하겠다. 하나씩 따라해 보자.

6. 아웃렛 변수를 추가한 것과 같은 방법으로 액션함수를 추가한다. [스위치]를 마우스 오른쪽 버튼으로 클릭한 후 드래그해서 클래스 아래쪽에 갖다 놓는다.

그리고 연결 설정 창에서 연결(Connection)을 [Action]으로 변경한다, 그리고 이름(Name)을 'swRecordMode'로 입력하고, 유형(Type)은 스위치에 액션을 추가하는 것이므로[UISwitch]를 선택한다.

변경을 완료한 후 [Connect]버튼을 클릭하여 추가한다.

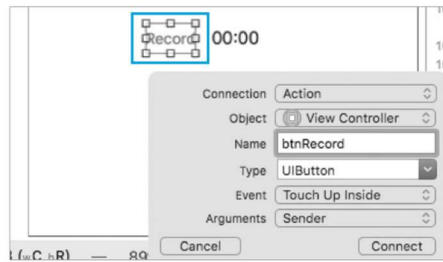
위치	소스의 가장 아래쪽 '}' 바로 위
연결(Connection)	Action
이름(Name)	swRecordMode
유형(Type)	UISwitch



```
@IBAction func swRecordMode(_ sender: UISwitch) {
}
```

7. 마찬가지로 [Record]버튼의 액션 함수를 추가한다.

위치	소스의 가장 아래쪽 '}' 바로 위
연결(Connection)	Action
이름(Name)	btnRecord
유형(Type)	UIButton



```
@IBAction func btnRecord(_ sender: UIButton) {
}
```

13-10 녹음을 위한 초기화하기

아웃렛 변수와 액션 함수를 추가해 보자. 이제는 녹음을 구현해 보자. 녹음할 때 새로운 파일에 녹음이 입혀져야 하고, 녹음시간과 버튼도 새로 설정해야 한다. 앞에서 '재생 모드'를 만들기 위해 초기화를 한 것 처럼 '녹음 모드'를 만들기 위한 초기화도 진행해보자.

1. 녹음을 위한 상수와 변수선언

우선 녹음에 관련된 상수와 변수를 추가하자. 클래스의 위쪽에 있는 기존 코드에 다음 소스를 추가 한다.

```
1 @IBOutlet var lblRecordTime: UILabel!
2
3 var audioRecorder : AVAudioRecorder!
4 var isRecordMode = false
5
6 override func viewDidLoad() {
7     ...
8 }
```

3 행은 audioRecorder 인스턴스를 추가한다.

4 행은 현재 '녹음 모드'라는 것을 나타낼 isRecord Mode 를 추가한다. 기본값을 false 로 하여 처음 앱을 실행했을 때 '녹음 모드'가 아닌 '재생 모드'가 나타나게 한다.

2. 녹음 파일 생성하기

녹음을 했는데 재생 파일에 겹쳐서 저장되면 안 된다. 이렇게 모드에 따라 다른 파일을 선택하기 위해서는 새로운 함수가 필요하다. viewDidLoad 함수 아래에 selectAudioFile 함수를 추가한다.

```
1 func selectAudioFile() {  
2     if !isRecordMode {  
3  
4     }  
5 }
```

3. viewDidLoad 함수에 audioFile 에 관한 코드 한 줄을 [command]키 + [X]키를 눌러 잘라낸다.

4. 방금 생성한 selectAudioFile 함수에 [command]키 + [V]키를 눌러 넣는다.

이 audioFile 은 재생할 때, 즉 녹음 모드가 아닐 때 사용하므로 if 문에 넣는다.

```
1 override func viewDidLoad() {  
2     /* audioFile = Bundle.main.url(forResource:  
3         "Sicilian_Breeze", withExtension: "mp3") */  
4     initPlay()  
5 }  
6 func selectAudioFile() {  
7     if !isRecordMode {  
8         audioFile = Bundle.main.url(forResource:  
9             "Sicilian_Breeze", withExtension: "mp3")  
10  
11     }  
12 }
```

2~3 행은 삭제하고

8~9 행은 추가한다.

5. viewDidLoad 함수에 audioFile 이 위치하던 곳에 방금 생성한 selectAudioFile 함수를 호출하는 코드를 추가한다.

```
1 override func viewDidLoad() {  
2     selectAudioFile()  
3     initPlay()  
4 }  
5 func selectAudioFile() {  
6     if !isRecordMode {  
7         audioFile = Bundle.main.url(forResource:  
8             "Sicilian_Breeze", withExtension: "mp3")  
9  
10    }
```

11	}
12	

2 행을 추가한다.

6. selectAudioFile 함수를 마무리한다.

녹음모드일 때 도큐먼트 디렉토리에 'recordFile.m4a'를 생성하여 사용한다.

이렇게 하면 재생 모드일 때는 오디오 파일인 재생 파일이 재생되고, 녹음 모드일 때는 새 파일이 생성된다.

1	func selectAudioFile() {
2	if !isRecordMode {
3	audioFile = Bundle.main.url(forResource: "Sicilian_Breeze",
4	withExtension: "mp3")
5	} else {
6	let documentDirectory =
7	FileManager.default.urls(for: .documentDirectory, in: .userDomainMask)[0]
8	audioFile =
9	documentDirectory.appendingPathComponent("recordFile.m4a")
10	}
11	}
12	

2~4 행은 재생 모드일 때는 오디오 파일인 'Sicilian_Breeze.mp3'가 선택된다.

5 행~ 10 행은 녹음모드일 때 새 파일인 "recordFile.m4a"가 생성된다.

7. 녹음을 위한 초기화하기

[Record]버튼을 클릭해 녹음을 시작하면 재생 중이던 오디오는 멈추고 재생모드에 관한 모든 것들이 초기화되어야 한다. 이렇게 녹음과 관련하여 오디오의 포맷, 음질, 비트율, 채널 및 샘플률을 초기화하기 위한 함수인 initRecord 를 생성한다,

1	func initRecord() {
2	
3	}

8. viewDidLoad 함수에서 모드에 따라 재생 또는 녹음에 관한 초기화를 따로 해야 하므로 if 문을 사용한다.

1	override func viewDidLoad() {
2	super.viewDidLoad()
3	// Do any additional setup after loading the view, typically from a nib.
4	selectAudioFile()
5	if !isRecordMode {
6	initPlay()
7	btnRecord.isEnabled = false

8	lblRecordTime.isEnabled = false
9	} else {
10	initRecord()
11	}
12	}
13	

6~7 행은 if 문의 조건이 !isRecordMode 이다. 이는 녹음 모드가 아니라는 것이므로 재생모드를 말한다. 따라서 initPlay 함수를 호출한다.

8~9 행은 조건에 해당하는 것이 재생모드이므로 [Record]버튼과 재생 시간은 비 활성화로 설정한다.

11 행은 조건에 해당하지 않는 경우, 이는 녹음 모드라는 것을 의미하므로 initRecord 함수를 호출한다.

9. 이번에는 initRecord 함수에 코드를 추가한다. 이는 녹음에 대한 설정이며 포맷은 'Apple Lossless', 음질은 '최대', 비트율은 '320,000bps(320kbps)', 오디오 채널은 '2'로하고 샘플률은 '44.100Hz'로 설정한다.

1	func initRecord() {
2	let recordSettings = [
3	AVFormatIDKey : NSNumber(value: kAudioFormatAppleLossless as
4	UInt32),
5	AVEncoderAudioQualityKey : AVAudioQuality.max.rawValue,
6	AVEncoderBitRateKey : 320000,
7	AVNumberOfChannelsKey : 2,
8	AVSampleRateKey : 44100.0] as [String : Any]
9	}
10	}

10. initRecord 함수에 코드를 추가한다, selectAudioFile 함수에 정한 audioFile 을 URL 로 하는 audioRecorder 인스턴스를 생성한다. 이때 try~catch 문을 사용한다. 마지막에 audioRecorder 의 델리게이터를 설정하는데 AVAudioRecorderDelegate 를 상속 받아야 한다,

1	class ViewController: UIViewController, AVAudioPlayerDelegate,
2	AVAudioRecorderDelegate {
3	
4	}

1	func initRecord() {
2	...
3	do {
4	audioRecorder = try AVAudioRecorder(url: audioFile, settings:
5	recordSettings)

6	}	catch let error as NSError {
7		print("Error-initRecord : \(error)")
8	}	
9		audioRecorder.delegate = self
10	}	

12. 앞의 initPlay 함수에서와 마찬가지로 initRecord 함수에 코드를 추가한다.

1	func initRecord() {
2	...
3	audioRecorder.delegate = self
4	audioRecorder.isMeteringEnabled = true
5	audioRecorder.prepareToRecord()
6	
7	slVolume.value = 1.0
8	audioPlayer.volume = slVolume.value
9	lblEndTime.text = convertNSTimeInterval2String(0)
10	lblCurrentTime.text = convertNSTimeInterval2String(0)
11	setPlayButtons(false, pause: false, stop: false)
12	}

3 행은 AudioRecorder 의 델리게이트(Delegate)를 self 로 설정한다.

4 행은 박자 관련 isMeteringEnabled 값을 참(True)으로 설정한다.

5 행은 prepareToRecord 함수를 실행한다.

7 행은 볼륨 슬라이더 값을 1.0 으로 설정한다.

8 행은 audioPlayer 의 볼륨도 슬라이더 값과 동일한 1.0 으로 설정한다.

9 행은 총 재생 시간을 0 으로 바꾼다.

10 행은 현재 재생 시간을 0 으로 바꾼다.

11 행은 [Play],[Pause], 및 [Stop]버튼을 비활성화로 설정한다.

13. 마찬가지로 initRecord 함수에 코드를 추가한다. AVAudioSession 의 인스턴스 session 을 생성하고 try~catch 문을 사용해 카테고리를 설정한 다음 액티브를 설정한다,

1	func initRecord() {
2	...
3	let session = AVAudioSession.sharedInstance()
4	do {
5	try session.setCategory(AVAudioSessionCategoryPlayAndRecord)
6	} catch let error as NSError {
7	print(" Error-setCategory : \(error)")
8	}
9	do {
10	try session.setActive(true)
11	} catch let error as NSError {
12	print(" Error-setActive : \(error)")

13	}
14	}
15	

13-11 녹음및재생모드변경하기

1. 스토리보드에서 만든 스위치로 '녹음 모드'와 '재생모드'를 변경할 수 있게 만들어 본다.

1. 스위치를 [On]으로 하면 녹음 모드'가 되고, [Off]로 하면 '재생모드'가 되어야 한다.
이를 구현하기 위해서 swRecordMode 함수에 다음 코드를 추가한다.

1	@IBAction func swRecordMode(_ sender: UISwitch) {
2	if sender.isOn {

3	audioPlayer.stop()
4	audioPlayer.currentTime=0
5	lblRecordTime!.text = convertNSTimeInterval2String(0)
6	isRecordMode = true
7	btnRecord.isEnabled = true
8	lblRecordTime.isEnabled = true
9	} else {
10	isRecordMode = false
11	btnRecord.isEnabled = false
12	lblRecordTime.isEnabled = false
13	lblRecordTime.text = convertNSTimeInterval2String(0)
14	}
15	selectAudioFile()
16	if !isRecordMode {
17	initPlay()
18	} else {
19	initRecord()
20	}
21	}
22	
23	

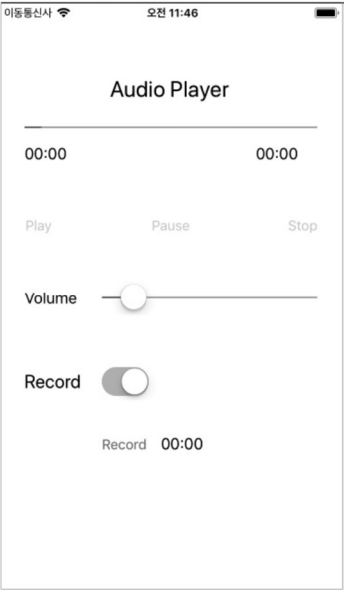
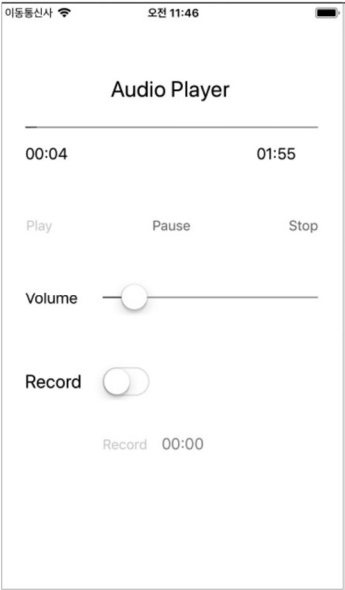
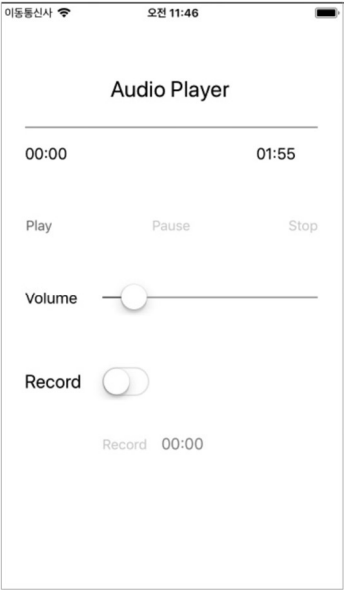
2~9 행은 스위치가 [On]이 되었을 때는 '녹음 모드'이므로 오디오 재생을 중지하고, 현재 재생시간을 00:00으로 만들고 isRecordMode의 값을 참(True)으로 설정하고, [Record]버튼과 녹음시간을 활성화로 설정한다,

10~15 행은 스위치가 [On]이 아닐 때, 즉 '재생 모드'일 때는 isRecordMode의 값을 거짓(false)으로 설정하고, [Record]버튼과 녹음시간을 비활성화하며, 녹음 시간을 0으로 초기화한다.

17~21 행은 selectAudioFile 함수를 호출하여 오디오 파일을 선택하고, 모드에 따라 초기화할 함수를 호출한다,

2. 결과 보기

다시 [실행]버튼을 클릭한다. 재생은 문제 없이 진행되며 스위치가 [On]이 되었을 때 [Play],[Pause] 및 [Stop]버튼을 비활성화되고 현재 재생 시간과 총 재생시간이 모두 "00:00"으로 초기화되어 [Record]버튼과 녹음시간이 활성화되는 것을 볼 수 있다.



13-12 녹음하기

이제 실제로 녹음이 가능하도록 만들어 보자.

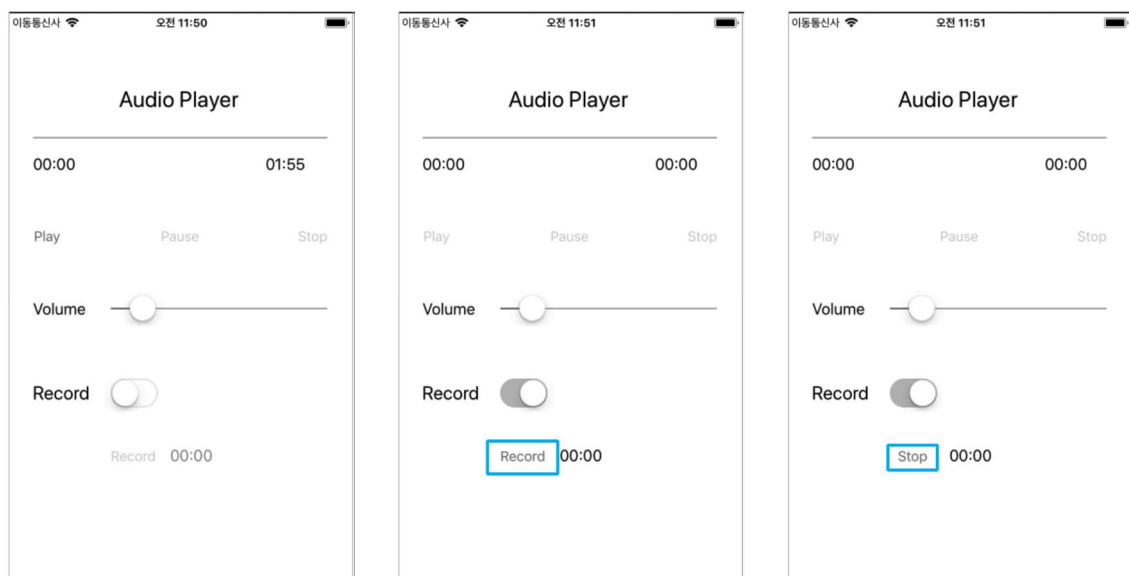
1. 녹음을 위해 btnRecord 함수를 수정한다,
[Record]버튼은 'Record'와 'Stop'으로 버튼 이름이 바뀐다.

```
1  @IBAction func btnRecord(_ sender: UIButton) {  
2      if sender.titleLabel?.text == "Record" {  
3          audioRecorder.record()  
4          sender.setTitle("Stop", for: UIControlState())  
5      } else {  
6          audioRecorder.stop()  
7          sender.setTitle("Record", for: UIControlState())  
8          btnPlay.isEnabled = true  
9          initPlay()  
10     }  
11 }
```

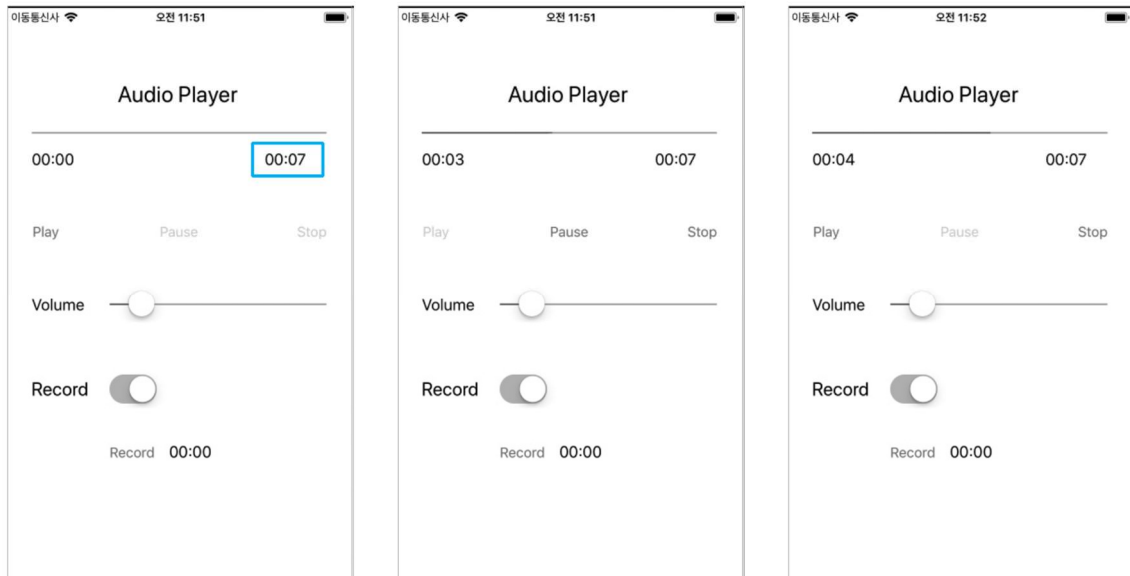
2~4 행은 만약에 버튼이름이 'Record'이면 녹음을 하고 버튼이름을 'Stop'으로 변경한다.
5~9 행은 그렇지 않으면 현재 녹음 중이므로 녹음을 중단하고 버튼이름을 'Stop'으로 변경한다. 그리고 [Play]버튼을 활성화하고 녹음한 파일로 재생을 초기화한다,

2. 결과 보기

다시 [실행] 버튼을 클릭한다. 재생은 문제 없이 진행된다 스위치가 [On]이되면 총 재생 시간이 "00:00"으로 초기화되고 [Record]버튼을 클릭하면 녹음이 시작된다.
하지만 아무런 변화가 없으며 [Record]버튼의 이름이 'Stop'으로 변경된다.



[Stop]버튼을 클릭하면 버튼이름이 'Record'로 변경되고 총 재생 시간도 변경된 것을 확인할 수 있다. 그리고 [Play],[Pause] 및 [Stop]버튼도 제대로 동작한다,



13-13 녹음 시간 표시하기

아래 녹음시간은 변하지 않는다. 녹음할 때 아래에 녹음시간이 표시되도록 만들어 보자. 시간이 표시되도록 하기위해서는 앞에서 '재생 시간'을 표시한 것처럼 타이머를 사용한다.

1. 다시 btnRecord 함수를 수정한다.

```
1  @IBAction func btnRecord(_ sender: UIButton) {
2      if sender.titleLabel?.text == "Record" {
3          audioRecorder.record()
4          sender.setTitle("Stop", for: UIControlState())
5          progressTimer = Timer.scheduledTimer(timeInterval: 0.1, target: self,
6 selector: timeRecordSelector, userInfo: nil, repeats: true)
7      } else {
8          audioRecorder.stop()
9          progressTimer.invalidate()
10         sender.setTitle("Record", for: UIControlState())
11         btnPlay.isEnabled = true
12         initPlay()
13     }
14 }
15
```

5~7 행은 녹음할 때 타이머가 작동하도록 progressTime 에 Timer.scheduledTimer 함수를 사용하는데 0.1 초 간격으로 타이머를 생성한다.

10 행은 녹음이 중지되면 타이머를 무효화한다.

2. 먼저 녹음 타이머를 위한 상수를 추가 한다.

```
1  class ViewController: UIViewController, AVAudioPlayerDelegate,
2  AVAudioRecorderDelegate {
3
4      var audioPlayer : AVAudioPlayer!
5      var audioFile : URL!
6
7      let MAX_VOLUME : Float = 10.0
8
9      var progressTimer : Timer!
10
11     let timePlayerSelector:Selector =
12         #selector(ViewController.updatePlayTime)
13     let timeRecordSelector:Selector =
14         #selector(ViewController.updateRecordTime)
```

13~14 행을 추가한다.

3. updateRecordTime 함수를 생성한다. 타이머에 의해서 0.1 초 간격으로 이 함수를 실행 하는데, 그 때마다 녹음 시간이 표시된다.

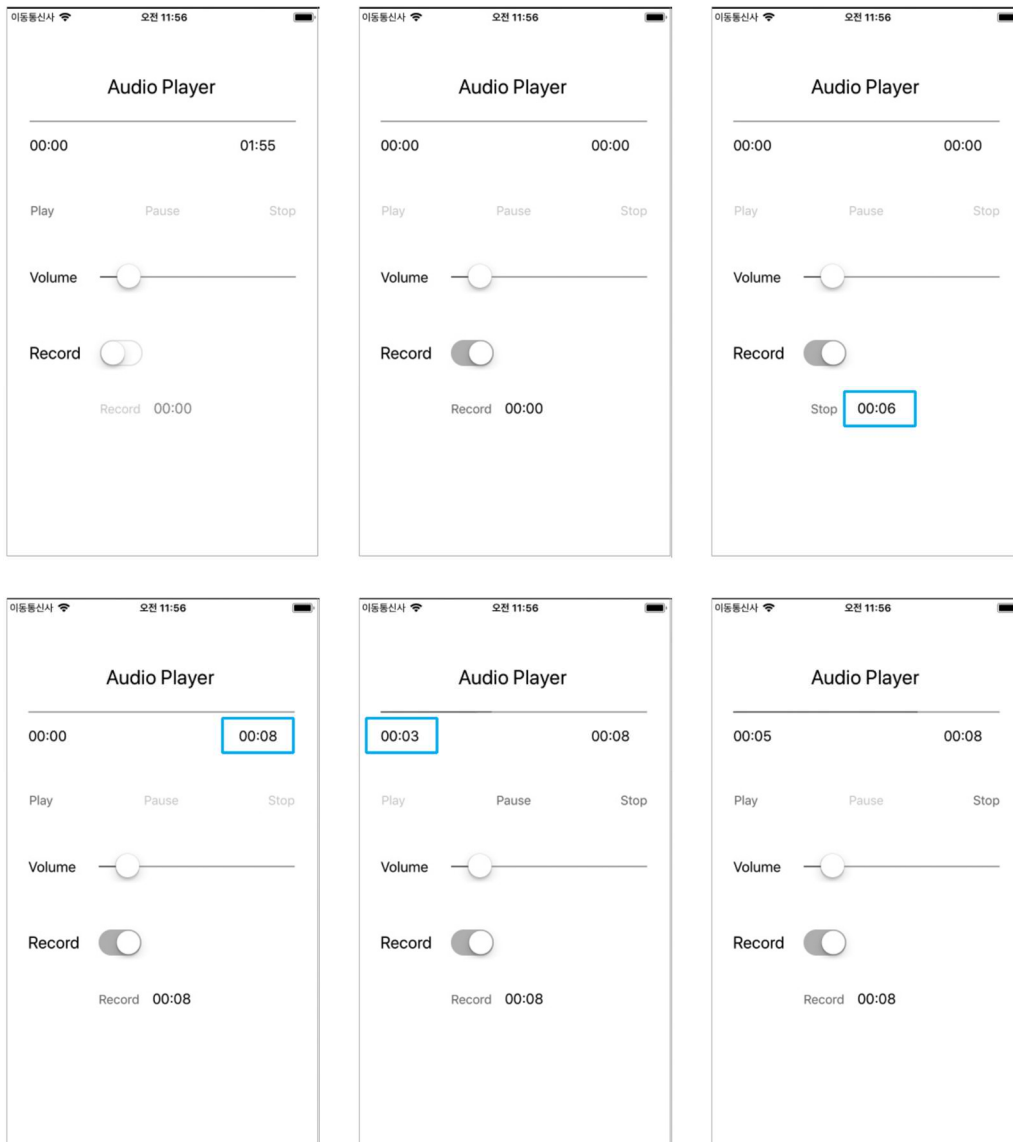
1	@objc func updateRecordTime() {
2	lblRecordTime.text =
3	convertNSTimeInterval2String(audioRecorder.currentTime)
4	}

4. 결과 보기

다시 [실행] 버튼을 클릭한다. 재생은 문제 없이 진행된다 스위치가 [On]이되면 총 재생 시간이 "00:00"으로 초기화되고 [Record]버튼을 클릭하면 녹음이 시작된다

녹음이 진행되는 동안 녹음 시간도 변경된다. [Stop]버튼을 클릭하면 버튼이름이 'Record'로 변경되고 총 재생 시간도 변경된 것을 확인할 수 있다. 그리고 녹음된 파일로 [Play],[Pause] 및 [Stop]버튼도 제대로 동작한다,

스위치가 [Off]가 되면 재생시간이 원래 음악의 전체 시간인 '01:55'로 바뀌면서 원래 음악이 선택된다. 그리고 이 파일로 [Play],[Pause] 및 [Stop]버튼도 제대로 동작한다



전체 소스 보기

```
import UIKit
import AVFoundation

class ViewController: UIViewController, AVAudioPlayerDelegate,
AVAudioRecorderDelegate {

    var audioPlayer : AVAudioPlayer!
    var audioFile : URL!

    let MAX_VOLUME : Float = 10.0

    var progressTimer : Timer!
```

```

let timePlayerSelector:Selector = #selector(ViewController.updatePlayTime)
let timeRecordSelector:Selector = #selector(ViewController.updateRecordTime)

@IBOutlet var pvProgressPlay: UIProgressView!
@IBOutlet var lblCurrentTime: UILabel!
@IBOutlet var lblEndTime: UILabel!
@IBOutlet var btnPlay: UIButton!
@IBOutlet var btnPause: UIButton!
@IBOutlet var btnStop: UIButton!
@IBOutlet var slVolume: UISlider!
@IBOutlet var btnRecord: UIButton!
@IBOutlet var lblRecordTime: UILabel!

var audioRecorder : AVAudioRecorder!
var isRecordMode = false

override func viewDidLoad() {
    super.viewDidLoad()
    // Do any additional setup after loading the view, typically from a nib.
    selectAudioFile()
    if !isRecordMode {
        initPlay()
        btnRecord.isEnabled = false
        lblRecordTime.isEnabled = false
    } else {
        initRecord()
    }
}

func selectAudioFile() {
    if !isRecordMode {
        audioFile = Bundle.main.url(forResource: "Sicilian_Breeze", withExtension:
"mp3")
    } else {
        let documentDirectory = FileManager.default.urls(for: .documentDirectory,
in: .userDomainMask)[0]
        audioFile = documentDirectory.appendingPathComponent("recordFile.m4a")
    }
}

func initRecord() {
    let recordSettings = [
        AVFormatIDKey : NSNumber(value: kAudioFormatAppleLossless as UInt32),
        AVEncoderAudioQualityKey : AVAudioQuality.max.rawValue,
        AVEncoderBitRateKey : 320000,
        AVNumberOfChannelsKey : 2,
        AVSampleRateKey : 44100.0] as [String : Any]

```

```

do {
    audioRecorder = try AVAudioRecorder(url: audioFile, settings:
recordSettings)
    } catch let error as NSError {
        print("Error-initRecord : \(error)")
    }
    audioRecorder.delegate = self
    audioRecorder.isMeteringEnabled = true
    audioRecorder.prepareToRecord()

    slVolume.value = 1.0
    audioPlayer.volume = slVolume.value
    lblEndTime.text = convertNSTimeInterval2String(0)
    lblCurrentTime.text = convertNSTimeInterval2String(0)
    setPlayButtons(false, pause: false, stop: false)
    let session = AVAudioSession.sharedInstance()
    do {
        try session.setCategory(AVAudioSessionCategoryPlayAndRecord)
    } catch let error as NSError {
        print(" Error-setCategory : \(error)")
    }
    do {
        try session.setActive(true)
    } catch let error as NSError {
        print(" Error-setActive : \(error)")
    }
}

func initPlay() {
    do {
        audioPlayer = try AVAudioPlayer(contentsOf: audioFile)
    } catch let error as NSError {
        print("Error-initPlay : \(error)")
    }
    slVolume.maximumValue = MAX_VOLUME
    slVolume.value = 1.0
    pvProgressPlay.progress = 0

    audioPlayer.delegate = self
    audioPlayer.prepareToPlay()
    audioPlayer.volume = slVolume.value
    lblEndTime.text = convertNSTimeInterval2String(audioPlayer.duration)
    lblCurrentTime.text = convertNSTimeInterval2String(0)
    setPlayButtons(true, pause: false, stop: false)
}

func setPlayButtons(_ play:Bool, pause:Bool, stop:Bool) {

```

```

        btnPlay.isEnabled = play
        btnPause.isEnabled = pause
        btnStop.isEnabled = stop
    }

    func convertNSTimeInterval2String(_ time:TimeInterval) -> String {
        let min = Int(time/60)
        let sec = Int(time.truncatingRemainder(dividingBy: 60))
        let strTime = String(format: "%02d:%02d", min, sec)
        return strTime
    }

    override func didReceiveMemoryWarning() {
        super.didReceiveMemoryWarning()
        // Dispose of any resources that can be recreated.
    }

    @IBAction func btnPlayAudio(_ sender: UIButton) {
        audioPlayer.play()
        setPlayButtons(false, pause: true, stop: true)
        progressTimer = Timer.scheduledTimer(timeInterval: 0.1, target: self, selector:
timePlayerSelector, userInfo: nil, repeats: true)
    }

    @objc func updatePlayTime() {
        lblCurrentTime.text = convertNSTimeInterval2String(audioPlayer.currentTime)
        pvProgressPlay.progress = Float(audioPlayer.currentTime/audioPlayer.duration)
    }

    @IBAction func btnPauseAudio(_ sender: UIButton) {
        audioPlayer.pause()
        setPlayButtons(true, pause: false, stop: true)
    }

    @IBAction func btnStopAudio(_ sender: UIButton) {
        audioPlayer.stop()
        audioPlayer.currentTime = 0
        lblCurrentTime.text = convertNSTimeInterval2String(0)
        setPlayButtons(true, pause: false, stop: false)
        progressTimer.invalidate()
    }

    @IBAction func slChangeVolume(_ sender: UISlider) {
        audioPlayer.volume = slVolume.value
    }

    func audioPlayerDidFinishPlaying(_ player: AVAudioPlayer, successfully flag: Bool) {

```

```

        progressTimer.invalidate()
        setPlayButtons(true, pause: false, stop: false)
    }

    @IBAction func swRecordMode(_ sender: UISwitch) {
        if sender.isOn {
            audioPlayer.stop()
            audioPlayer.currentTime=0
            lblRecordTime!.text = convertNSTimeInterval2String(0)
            isRecordMode = true
            btnRecord.isEnabled = true
            lblRecordTime.isEnabled = true
        } else {
            isRecordMode = false
            btnRecord.isEnabled = false
            lblRecordTime.isEnabled = false
            lblRecordTime.text = convertNSTimeInterval2String(0)
        }
        selectAudioFile()
        if !isRecordMode {
            initPlay()
        } else {
            initRecord()
        }
    }

    @IBAction func btnRecord(_ sender: UIButton) {
        if sender.titleLabel?.text == "Record" {
            audioRecorder.record()
            sender.setTitle("Stop", for: UIControlState())
            progressTimer = Timer.scheduledTimer(timeInterval: 0.1, target: self,
selector: timeRecordSelector, userInfo: nil, repeats: true)
        } else {
            audioRecorder.stop()
            progressTimer.invalidate()
            sender.setTitle("Record", for: UIControlState())
            btnPlay.isEnabled = true
            initPlay()
        }
    }

    @objc func updateRecordTime() {
        lblRecordTime.text = convertNSTimeInterval2String(audioRecorder.currentTime)
    }
}

```

