# Lab: create a Watson text-to-speech app with Swift

# Overview

In this lab, you'll create an application that uses IBM Watson's Text to Speech service.

The Text to Speech service can read text and convert it to speech. For example, you can add a voice interface to an application that visually impaired users use to hear (rather than read) their email or text messages.

To create the application in this lab, follow these main steps:

1. Create a typical iOS application in Swift.
2. Install the Watson SDK for iOS.
3. Create the Bluemix Watson service and get the key token to it.
4. Add some code to invoke the cognitive service.

You can see the code the GUI for this lab is in GitHub: https://github.com/blumareks/Swift-Watson-Guis.

## Prerequisites

You need the following software:

- Mac OS X El Capitan
- Xcode 9.0 or later
- Swift 4.0.x
- Carthage (package manager similar to Ant): https://github.com/Carthage/Carthage#installing-carthage

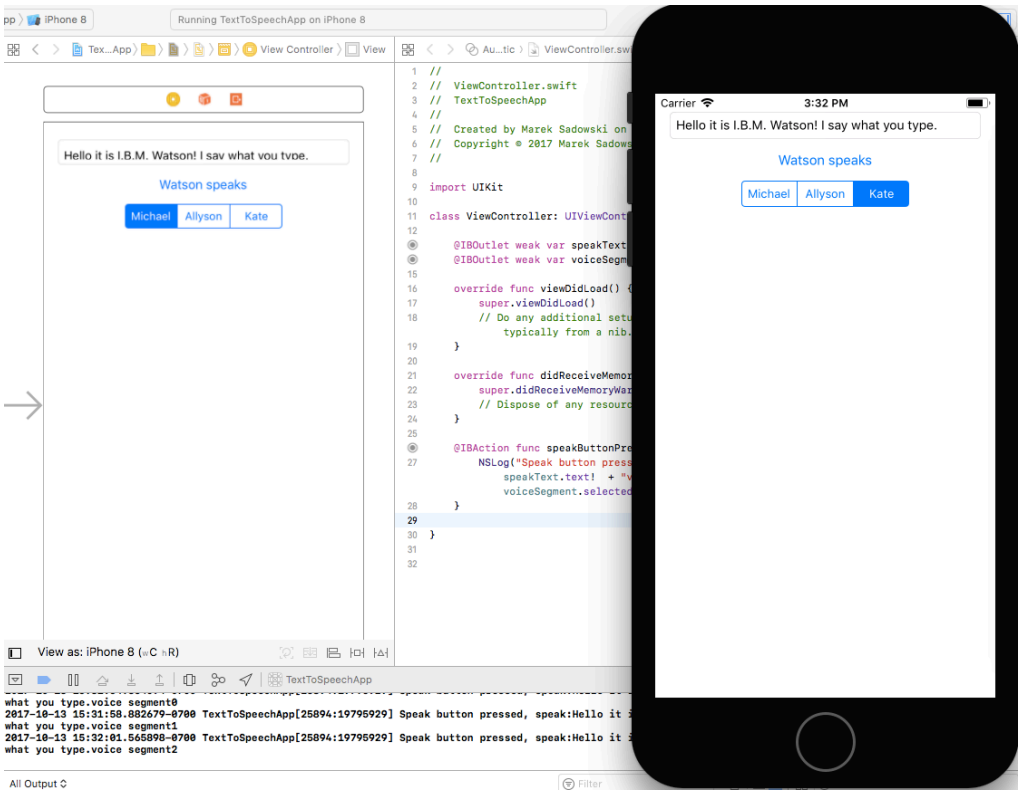
Complete the previous two labs:

- Create a Watson sentiment analysis app with Swift
- Create a Watson photo recognition app with Swift

# Step 1. Create a typical application in Swift

You'll build a simple application with a **Submit** button and an editable text field. There is no output field because the output is going to be delivered through voice.

1. Create a simple single view application with a graphical user interface (GUI) that includes a text field, a segmented control and a button. When a user presses the button, the text in the text field is sent to Watson, which analyzes it and returns a voice file (in the provided voice). This will be a single view application.




You take the text in the text field and simply log it in the console when the button is pressed.

See the Xcode documentation for information about how to create basic GUIs in Xcode.

2. Connect the GUI in Main.storyboard to the code in the ViewController file:
   a. Double-click **TextField** (by pressing the mouse pad with two fingers), which contains the text "Hello it is I.B.M. Watson. I say what you type."
   b. Select **New Reference Outlet** from the list.
   c. In the dialog, enter `speakText`.
   d. Insert the new reference between **class ViewController** and **override func viewDidLoad.**
   e. Click **Connect**.

      The result is `@IBOutlet weak var speakText: UITextField!`

f. Connect the button by entering `speakButtonPressed` in the dialog.

The inserted text is:

```
@IBAction func speakButtonPressed(_ sender: Any) {
}
```

g. Test the code by adding `NSLog(speakText.text!)` to the end of the checkButtonPressed method.

h. Right-click the Segmented Control, which contains three controls on the right pane in the Attributes Inspector, and increase the number of segments to 3. Override the previous titles with Watson and Allyson respectively, and add the title "Kate" to the segment number 2.

i. Double-click to select **New Reference Outlet** from the drop-down list.

j. In the dialog, enter `voiceSegment` and insert it below **@IBOutlet weak var textField: .**

k. Click **Connect.**

The result is: `@IBOutlet weak var voiceSegment: UISegmentedControl!`

NSLog allows you to log your actions on the console.

The code should look like this:

```
Import UIKit
class ViewController: UIViewController {

    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var voiceSegment: UISegmentedControl!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically
from a nib.
    }

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

```
    @IBAction func speakButtonPressed(_ sender: Any) {
        NSLog("Speak button pressed, speak:" + speakText.text!  +
"voice segment" + voiceSegment.selectedSegmentIndex.description)

       //add Watson Service

    }
}
```

I. Build and execute your application.

# Step 2. Install Carthage and add the Watson SDK to your project

As in the previous two labs, you must again install the Carthage dependency manager, the SDK, and then add that SDK to your Xcode project.

1. Install the Carthage dependency manager.

   Use the Carthage dependency manager to install libraries that are used by your application.

   **Important**: If you previously installed the binary version of Carthage, you must delete /Library/Frameworks/CarthageKit.framework.

   To install Carthage by using Homebrew (only on Xcode 7.x), run the commands `brew update` and `brew install Carthage`.

   If want to run the latest development version of Carthage, which might be highly unstable or incompatible, clone the master branch of the repository and then run the command `make install`.

2. Install the Watson SDK:
   a. Open a Bash shell in the root directory of your project.
   b. Enter `cat > cartfile`
   c. Enter `github "watson-developer-cloud/ios-sdk"`
   d. Enter a new line. Then, press control + C to exit Edit mode.
   e. From the command line at the root of the project, run the command `carthage update --platform iOS`. If you receive a compile failure for the frameworks, run the command again.

      The following image shows output from the package manager as it fetches and builds iOS library components:

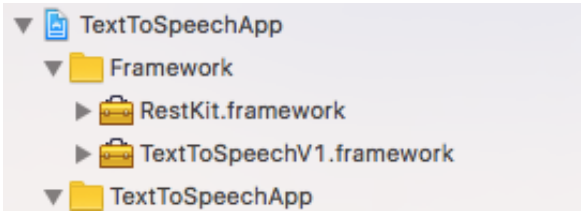
```
Mareks-Air-2:TextToSpeechApp mareksadowski$ carthage update --platform iOS
*** Fetching ios-sdk
*** Downloading ios-sdk.framework binary at "v0.18.0"
*** xcodebuild output can be found in /var/folders/d3/t2klghyd1gj7cfvml8y7zqfr00
00gn/T/carthage-xcodebuild.zL1nSl.log
```

3. Add the SDK to the Xcode project:
   a. Under the project name, create a new group in your Xcode project called `Framework`.
   b. Navigate to the root folder of your project in Xcode and select all the framework files in the <your project>/Carthage/Build/iOS/ directory of your project

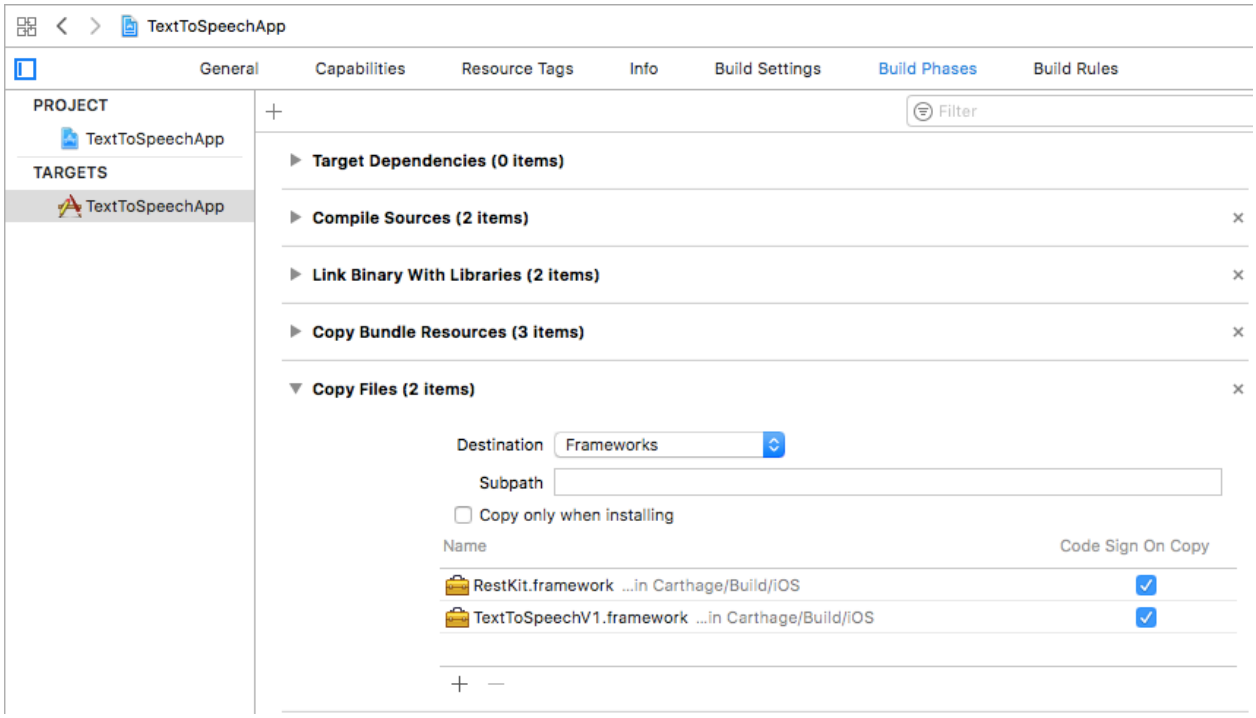
(TextToSpeechV1.framework, RestKit.framework). Drag and drop those files from Finder into the new `Framework` group in your project in Xcode.

**Be sure you clear the option to copy items**. By not copying the items, you create only a reference to those Framework files.

This is the list of the framework libraries:




c. In the **Build Phases** tab, add a new **Copy Files** phase and set its destination to `Frameworks`.

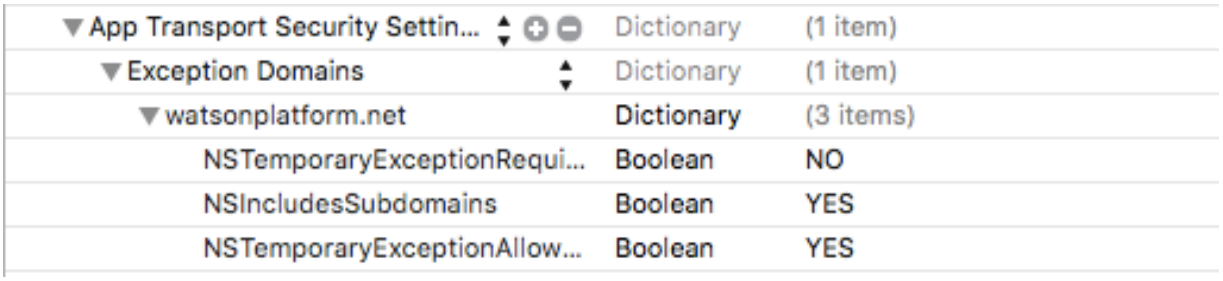



d. For the frameworks that were added by Carthage, copy each framework individually.

e. Whitelist your calls to Watson services and remove the security for watsonplatform.net. Whitelisting allows only specified addresses to get through without the SSL protocol.

To do this, use either Xcode to add exceptions as shown in the following image or add the XML snippets below to your info.plist file:

**Whitelist the calls in Xcode:**




**Whitelist the calls by copying the following XML to the info.plist file:**

```
<key>NSAppTransportSecurity</key>
  <dict>
        <key>NSExceptionDomains</key>
        <dict>
            <key>watsonplatform.net</key>
            <dict>

<key>NSTemporaryExceptionRequiresForwardSecrecy</key>
                <false/>
                <key>NSIncludesSubdomains</key>
                <true/>

<key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
                <true/>
            </dict>
        </dict>
</dict>
```

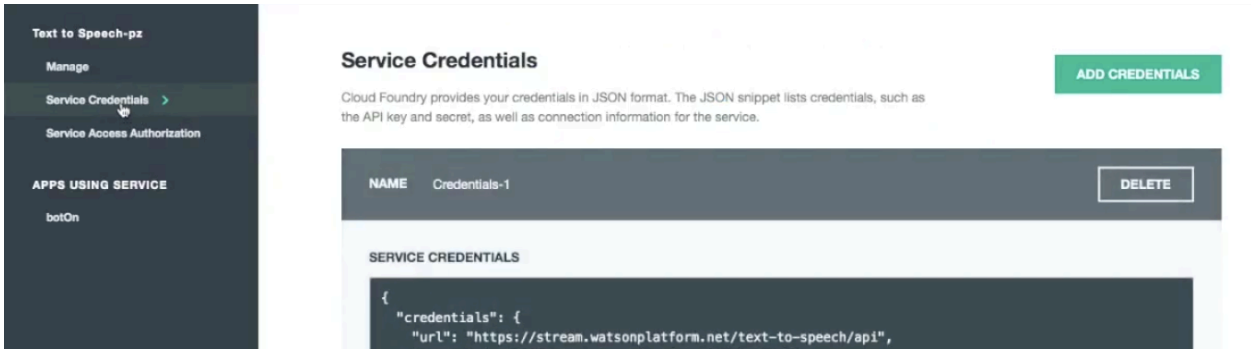
# Step 3. Create the Watson service and get the key token for it

Now, you create the Watson Text to Speech service.

1. Use your existing AlchemyAPI credentials that you created in the previous lab or create a new one by clicking **Watson > Text to Speech** > **Create** from the Bluemix Catalog.

   Be sure to use a static API key as shown in the following image.

2. Find and copy the Service Credentials.




3. Add the following line of code to call the cognitive service. Insert the import field under previously existing import fields.

   Import field:

```
import TextToSpeechV1 //importing Watson TTS service
import AVFoundation   //importing AVFoundation for AVAudioPlayer
```

So far, your code should look like this:

```
    //add Watson Service

let username = "<user from Watson TTS service>" //the user from the
step above
let password = "<password from Watson TTS service>"  //the password
from the step above
let textToSpeech = TextToSpeech(username: username, password:
password)

let text = speakText.text!
let failure = { (error: Error) in print(error) }
textToSpeech.synthesize(text,voice:
    ("0"==voiceSegment.selectedSegmentIndex.description ?
    SynthesisVoice.us_Michael.rawValue
```

```
    : ( "1"==voiceSegment.selectedSegmentIndex.description ?
     SynthesisVoice.us_Allison.rawValue
        : SynthesisVoice.gb_Kate.rawValue)
    ), failure: failure) { data in
    var audioPlayer: AVAudioPlayer // see note below
    audioPlayer = try! AVAudioPlayer(data: data)
    audioPlayer.prepareToPlay()
    audioPlayer.play()
    sleep(10)
  }
```

The full source code for the application should look like this:

```
import UIKit
import TextToSpeechV1 //importing Watson TTS service
import AVFoundation   //importing AVFoundation for AVAudioPlayer



class ViewController: UIViewController {

    @IBOutlet weak var speakText: UITextField!
    @IBOutlet weak var voiceSegment: UISegmentedControl!

    override func viewDidLoad() {
        super.viewDidLoad()
        // Do any additional setup after loading the view, typically from a nib.
    }

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

    @IBAction func speakButtonPressed(_ sender: Any) {
        NSLog("Speak button pressed, speak:" + speakText.text!  + "voice segment" +
voiceSegment.selectedSegmentIndex.description)

        let username = "<user from Watson TTS service>" //the user from the step above
        let password = "<password from Watson TTS service>"  //the password from the step
above
        let textToSpeech = TextToSpeech(username: username, password: password)

        let text = speakText.text!
        let failure = { (error: Error) in print(error) }
        textToSpeech.synthesize(text, voice:
            ("0"==voiceSegment.selectedSegmentIndex.description ?
SynthesisVoice.us_Michael.rawValue : ( "1"==voiceSegment.selectedSegmentIndex.description ?
SynthesisVoice.us_Allison.rawValue : SynthesisVoice.gb_Kate.rawValue)
        ), failure: failure) { data in
            var audioPlayer: AVAudioPlayer // see note below
            audioPlayer = try! AVAudioPlayer(data: data)
            audioPlayer.prepareToPlay()
            audioPlayer.play()
            sleep(10)
        }
    }

}
```

# Step 4. Run the Watson application

1. Run the Watson application by entering text in the text field and clicking the button.

   The text is sent to Watson through the Watson SDK, and Watson returns a sound file in the specified voice, language, and audio file type.

2. Run the application in a simulator, and you should hear speech coming from the speaker.

   You can also use other voices and languages.

## Summary

You now know how to create an application that uses IBM Watson's Text to Speech service in a Swift application.

Use the Text to Speech service to read text and convert it to speech. Use it to read your email, text messages, or other scenarios.