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 blind 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 7.3 or later
- Swift 2.2.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

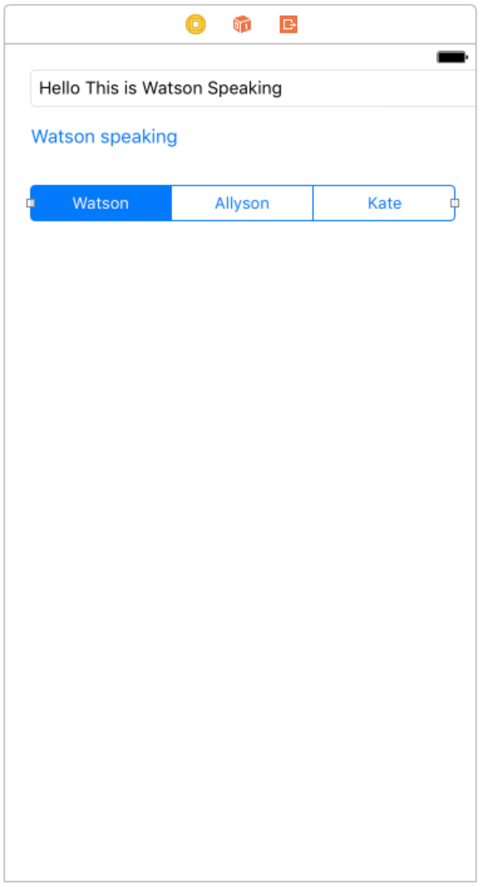
Step 1. Create a typical application in Swift

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.

Step 1a. Create the GUI

First, 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.

Step 1b. Connect the GUI in Main.storyboard to the code in the ViewController file

In Main.storyboard, connect the TextField:

1. Double-click **TextField** (by pressing the mouse pad with two fingers), which contains the text “Hello this is Watson speaking.”

2. Select **New Reference Outlet** from the list.
3. Enter `speakText` in the dialog.
4. Insert the new reference between **class ViewController** and **override func viewDidLoad.**
5. Click **Connect**.

   The result is `@IBOutlet weak var speakText: UITextField!`
6. Connect the button by entering `speakButtonPressed` in the dialog.

   The inserted text is:

   ```
   @IBAction func speakButtonPressed(sender: AnyObject) {
   }
   ```
7. Test the code by adding `NSLog(speakText.text!)` to the end of the checkButtonPressed method.
8. Right-click on the Segmented Control (contains three controls - on the right pane - in the Atributes Inspector - increase the number of segments to 3, override previous titles with Watson and Allyson respectively, and added the title "Kate" to the segment no 2)
9. Double click to select **New Reference Outlet** from drop-down list
10. Enter voiceSegment in the popup dialog.
11. Insert between **Class View Controller** and **override func viewDidLoad.**
12. Click Connect
13. 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: AnyObject) {
            NSLog("Speak button pressed, speak:" + speakText.text!  +
    "voice segment" + voiceSegment.selectedSegmentIndex.description)

           //add Watson Service

        }
    }
```

14. 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.

Step 2a. Install the Carthage dependency manager

Use the Carthage dependency manager to install libraries that are used by your application. You can install Carthage from GitHub or use Homebrew to update it only if you use Xcode 7.x.

> **Important**: If you previously installed the binary version of Carthage, you must delete /Library/Frameworks/CarthageKit.framework.
>
> - To install Carthage from GitHub, download and run the latest release of the Carthage.pkg file: https://github.com/Carthage/Carthage/releases. Then follow the installation prompts.
> - To install Carthage by using Homebrew (only on Xcode 7.x), run the commands `brew update` and `brew install Carthage`.

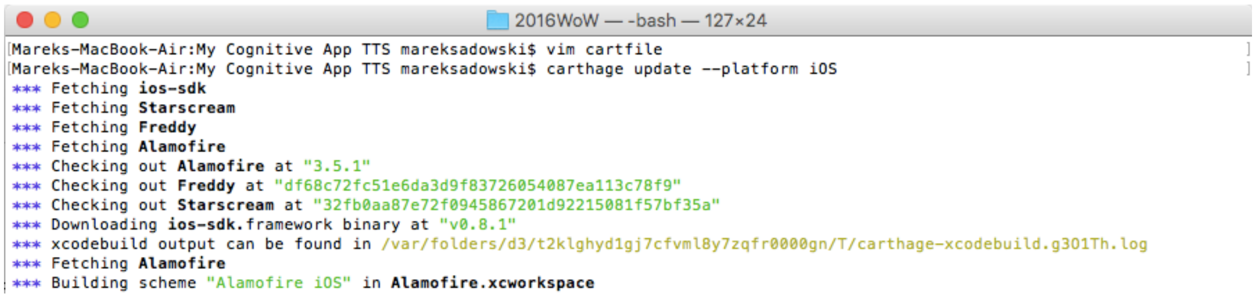
If want to run the latest development version of Carthage, which might be highly unstable or incompatible, clone the master branch of the repository and then run the command `make install`.

Step 2b. Install the Watson SDK

To install the SDK, open a Bash shell in the root directory of your project and follow these steps:

1. Enter `cat > cartfile`
2. Enter `github "watson-developer-cloud/ios-sdk"`
3. Enter a new line.
4. Press control + C to exit Edit mode.
5. 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 framework AlamofireObjectMapper, run the command again.

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



2016WoW — -bash — 127×24

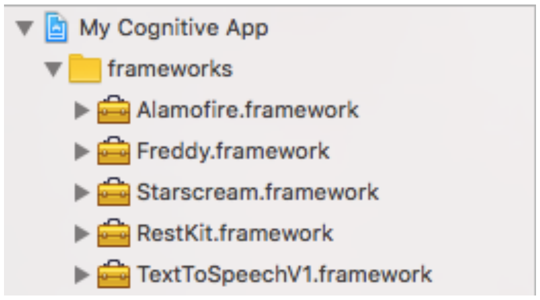
```
Mareks-MacBook-Air:My Cognitive App TTS mareksadowski$ vim cartfile
Mareks-MacBook-Air:My Cognitive App TTS mareksadowski$ carthage update --platform iOS
*** Fetching ios-sdk
*** Fetching Starscream
*** Fetching Freddy
*** Fetching Alamofire
*** Checking out Alamofire at "3.5.1"
*** Checking out Freddy at "df68c72fc51e6da3d9f83726054087ea113c78f9"
*** Checking out Starscream at "32fb0aa87e72f0945867201d92215081f57bf35a"
*** Downloading ios-sdk.framework binary at "v0.8.1"
*** xcodebuild output can be found in /var/folders/d3/t2klghyd1gj7cfvml8y7zqfr0000gn/T/carthage-xcodebuild.g301Th.log
*** Fetching Alamofire
*** Building scheme "Alamofire iOS" in Alamofire.xcworkspace
```

Step 2c. Add the SDK to the Xcode project

1. Under the project name, create a new group in your Xcode project called `Framework`.
2. 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, Alamofire, Freddy, RestKit, Starscream). Drag and drop those files from Finder into the new `Framework` group in your project in Xcode.

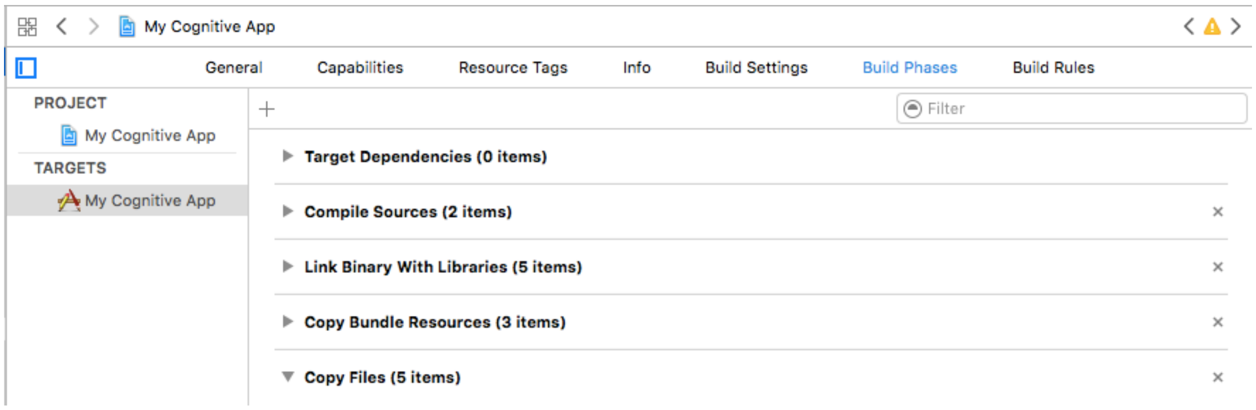
   Be sure you clear the option to copy items. By not copying the items, you create only a reference to those Framework files.

   This is the list of the framework libraries:



My Cognitive App
frameworks
Alamofire.framework
Freddy.framework
Starscream.framework
RestKit.framework
TextToSpeechV1.framework

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



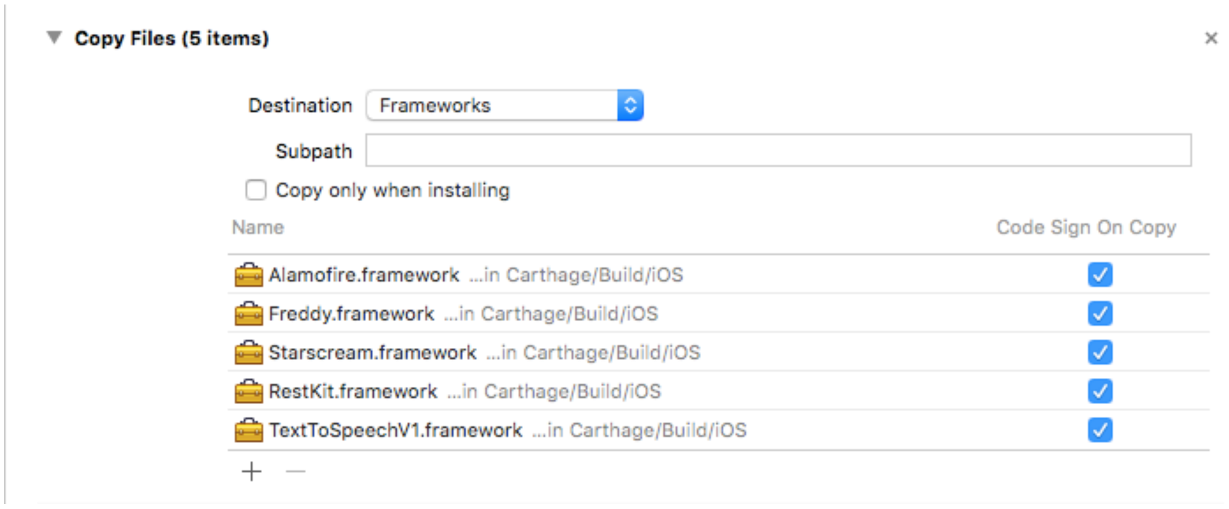
My Cognitive App

General | Capabilities | Resource Tags | Info | Build Settings | Build Phases | Build Rules

PROJECT
My Cognitive App
TARGETS
My Cognitive App

Target Dependencies (0 items)
Compile Sources (2 items)
Link Binary With Libraries (5 items)
Copy Bundle Resources (3 items)
Copy Files (5 items)

4. For the frameworks that were added by Carthage, copy each framework individually. These are the files that you copy:



▼ Copy Files (5 items)

Destination: Frameworks

Subpath:

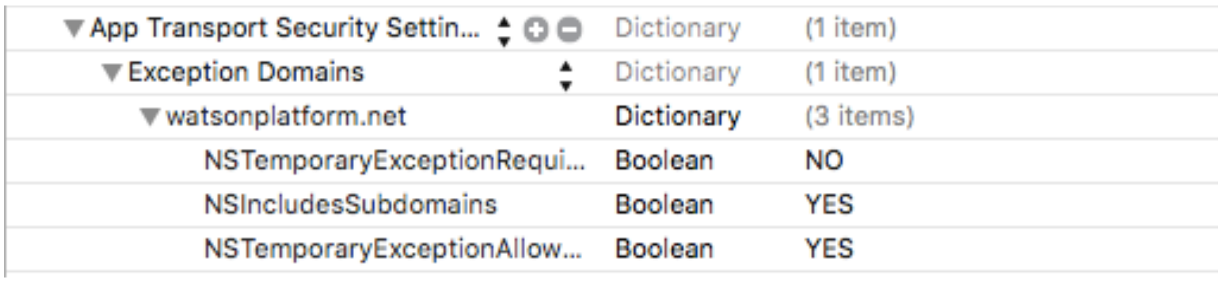
☐ Copy only when installing

Name	Code Sign On Copy
Alamofire.framework ...in Carthage/Build/iOS	✓
Freddy.framework ...in Carthage/Build/iOS	✓
Starscream.framework ...in Carthage/Build/iOS	✓
RestKit.framework ...in Carthage/Build/iOS	✓
TextToSpeechV1.framework ...in Carthage/Build/iOS	✓

5. Whitelist your calls to Watson services and remove the security for watsonplatform.net. Whitelisting is the use of anti-spam filtering software to allow only specified addresses to get through.

   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:



▼ App Transport Security Settin...	Dictionary	(1 item)
▼ Exception Domains	Dictionary	(1 item)
▼ watsonplatform.net	Dictionary	(3 items)
NSTemporaryExceptionRequi...	Boolean	NO
NSIncludesSubdomains	Boolean	YES
NSTemporaryExceptionAllow...	Boolean	YES

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>
```

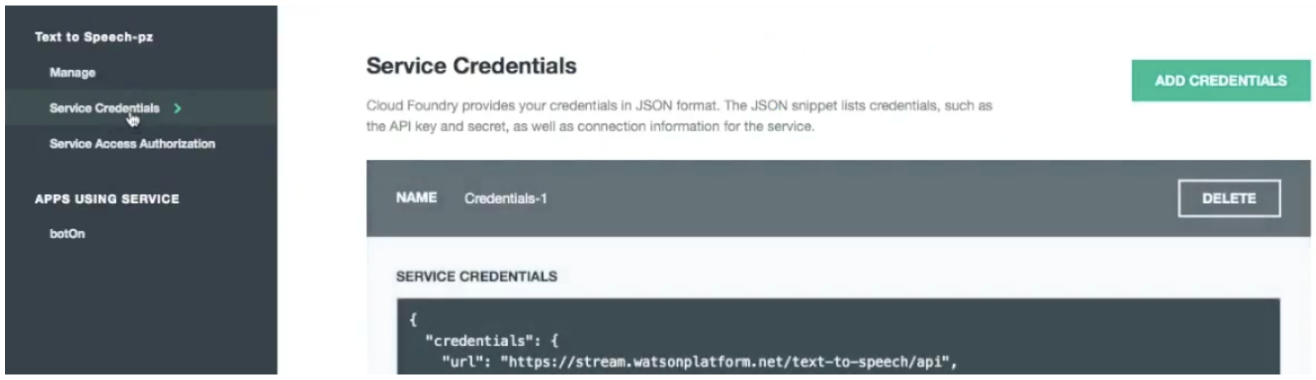
Step 2d. Create a Watson service and get the key token for it

Now, you create the Watson AlchemyAPI service.

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

   You can create only one AlchemyAPI service (and credentials) in a 24-hour period.

   Be sure to use a static API key as shown in the following image.
2. Find the Service Credentials.

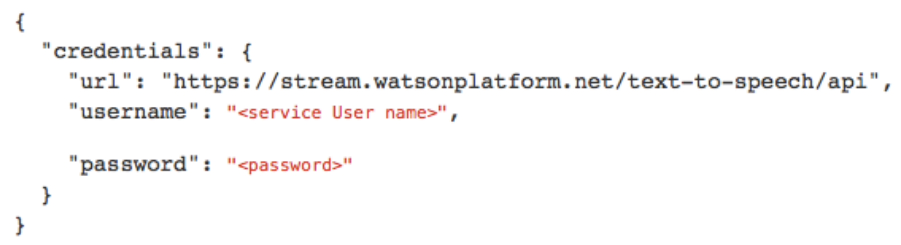


Service Credentials

Cloud Foundry provides your credentials in JSON format. The JSON snippet lists credentials, such as the API key and secret, as well as connection information for the service.

Credentials-1

```
{
  "credentials": {
    "url": "https://stream.watsonplatform.net/text-to-speech/api",
```



```
{
  "credentials": {
    "url": "https://stream.watsonplatform.net/text-to-speech/api",
    "username": "<service User name>",

    "password": "<password>"
  }
}
```

Bluemix provides your credentials in JSON format. The JSON snippet lists credentials, such as the API user name and password, and connection information for the service.

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: NSError) in print(error) }
textToSpeech.synthesize(text, voice:
("0"==voiceSegment.selectedSegmentIndex.description ?
SynthesisVoice.US_Michael :
( "1"==voiceSegment.selectedSegmentIndex.description ?
SynthesisVoice.US_Allison : SynthesisVoice.GB_Kate)), failure:
failure) { data in
    do {
        var audioPlayer: AVAudioPlayer // see note below
        audioPlayer = try! AVAudioPlayer(data: data)
        audioPlayer.prepareToPlay()
        audioPlayer.play()
        sleep(10)
    } catch {
        NSLog("something went terribly wrong")
    }
}
```

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: AnyObject) {
       NSLog("Speak button pressed, speak:" + speakText.text! + "voice
segment" + voiceSegment.selectedSegmentIndex.description)

       //add Watson Service
       let username = "<user from Watson TTS service>"
       let password = "<password from Watson TTS service>"
       let textToSpeech = TextToSpeech(username: username, password:
password)
       let text = speakText.text!
       let failure = { (error: NSError) in print(error) }
       textToSpeech.synthesize(text, voice:
            ("0"==voiceSegment.selectedSegmentIndex.description ?
                  SynthesisVoice.US_Michael :
            ( "1"==voiceSegment.selectedSegmentIndex.description ?
            SynthesisVoice.US_Allison : SynthesisVoice.GB_Kate)), failure:
            failure) { data in
           do {
               var audioPlayer: AVAudioPlayer // see note below
               audioPlayer = try! AVAudioPlayer(data: data)
               audioPlayer.prepareToPlay()
               audioPlayer.play()
               sleep(10)
           } catch {
               NSLog("something went terribly wrong")
           }
       }
   }
}
```

Step 3. Run the Watson application

You can now 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.

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

You can also use other voices and languages.