



Lab: create a Watson sentiment analysis app with Swift

Overview

This lab shows you how to build a mobile application that will analyze tone, or sentiment, in text. You'll build the application in Swift and use the IBM Watson AlchemyAPI service to analyze sentiment conveyed in text.

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

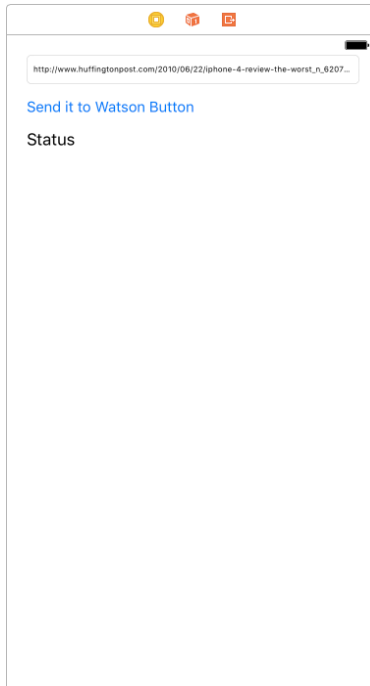
You need the following software:

- 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>

Step 1. Create a typical application in Swift

You'll build a simple application with a submit button, an editable text field, and an output field.

Step 1a. Create the GUI



First, create a simple single view application with a graphical user interface (GUI) that includes a text field, a label, and a button. When a user presses the button, the URL in the text field is sent to Watson, which analyzes it and returns an opinion that is shown in the output field.

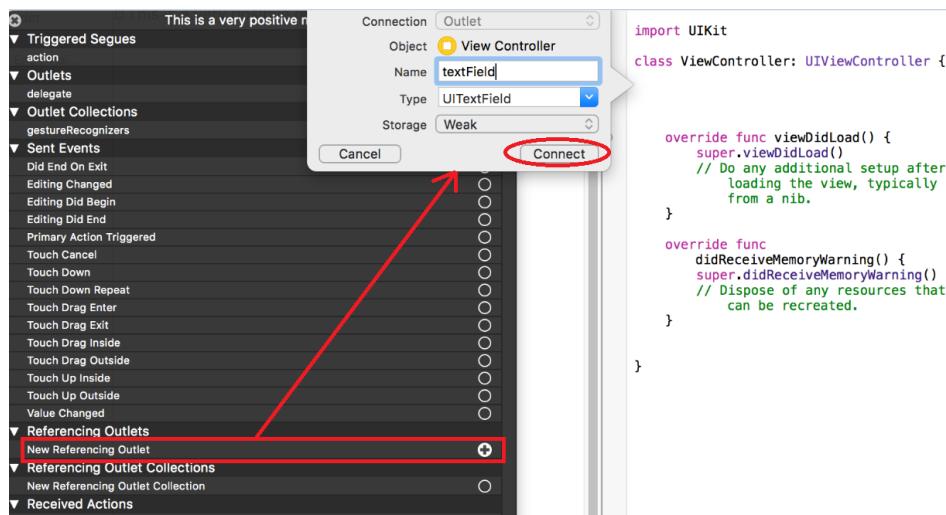
You take the text in the text field and simply echo it in the label field when the button is pressed.

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

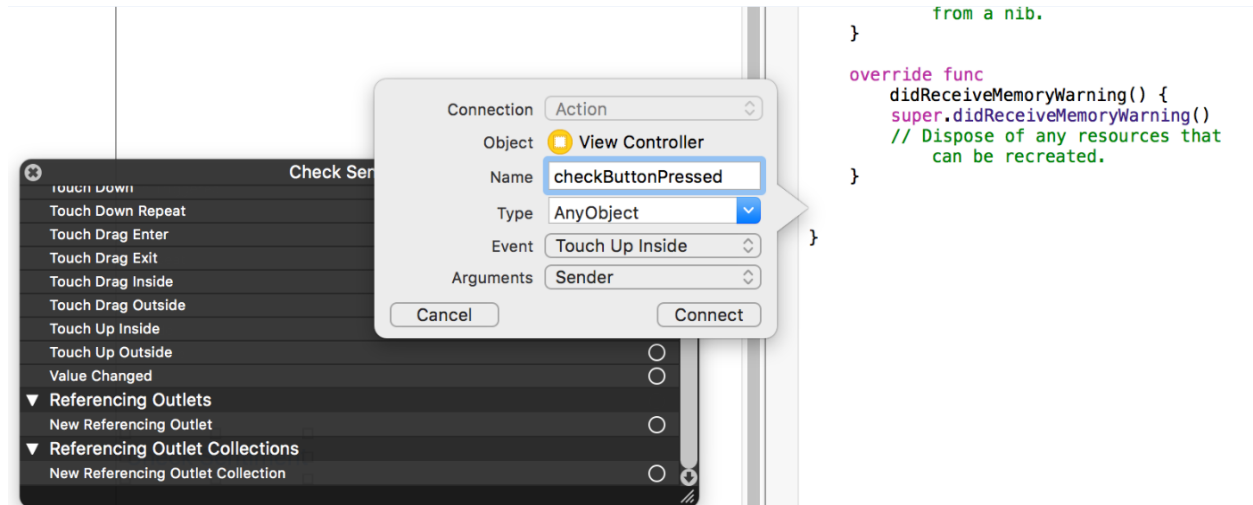
In Main.storyboard, connect the TextField:

1. Double-click **TextField** (by pressing the mouse pad with two fingers), which contains the URL to the document “http://www.huffingtonpost.com/2010/06/22/iphone-4-review-the-worst_n_620714.html”
2. Select **New Referencing Outlet** from the list.
3. Enter `textField` in the dialog.
4. Insert the new reference between **class ViewController** and **override func viewDidLoad**.
5. Click **Connect**.



The inserted text is `@IBOutlet weak var textField: UITextField!`

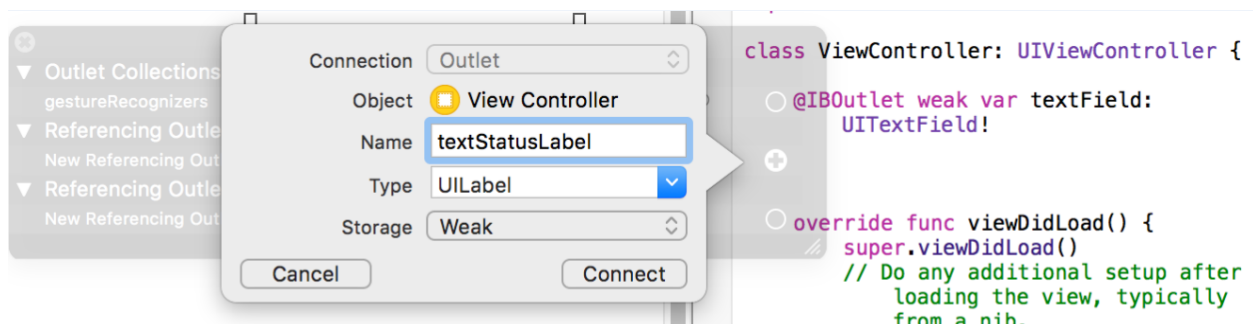
6. Connect the button. Insert the sent event named **Touch up Inside** before the last curly bracket.



The inserted text is:

```
@IBAction func checkButtonPressed(sender: AnyObject) {
    }
}
```

7. Connect the text label. Insert the parameter **UILabel** between `@IBOutlet weak var textField: UITextField!` and `override func...`



The inserted text is `@IBOutlet weak var textStatusLabel: UILabel!`

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

`NSLog` allows you to log your actions. Setting `textStatusLabel.text` with `textField.text` allows you to show the entered text of the Text field in the Label field.

```
import UIKit
class ViewController: UIViewController {
```

```
@IBOutlet weak var textField: UITextField!
@IBOutlet weak var textStatusLabel: UILabel!

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 checkButtonPressed(sender: AnyObject) {
    NSLog(textField.text!)

    //checking sentiment

    //setting feedback on sentiment
    textStatusLabel.text = textField.text
}
}
```

9. Build and execute your application.

If you see the text “http://www.huffingtonpost.com/2010/06/22/iphone-4-review-the-worst_n_620714.html” the GUI is working.

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

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`.

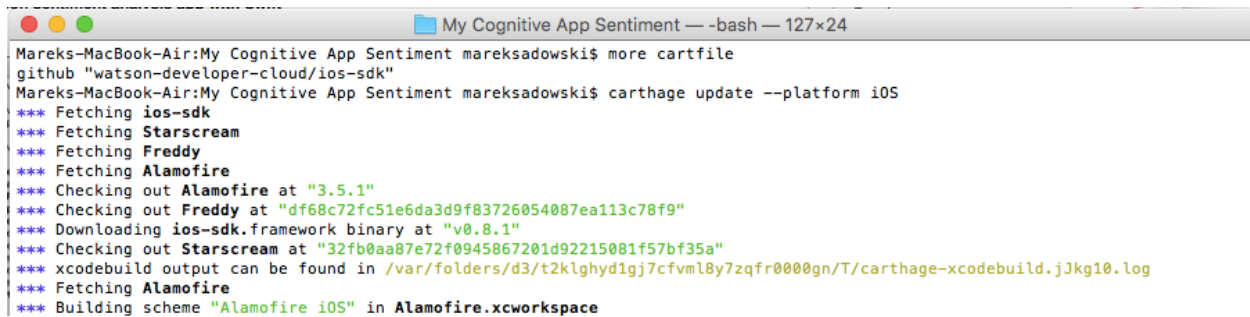
If you want to run the latest development version of Carthage, which might be highly unstable or incompatible, clone the master branch of the repository and then run the command `make install`.

Important: If you have two-factor authentication, Carthage requires an OTP header.

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`.



```

My Cognitive App Sentiment — -bash — 127x24
Mareks-MacBook-Air:My Cognitive App Sentiment mareksadowski$ more cartfile
github "watson-developer-cloud/ios-sdk"
Mareks-MacBook-Air:My Cognitive App Sentiment 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"
*** Downloading ios-sdk.framework binary at "v0.8.1"
*** Checking out Starscream at "32fb0aa87e72f0945867201d92215081f57bf35a"
*** xcodebuild output can be found in /var/folders/d3/t2klghyd1gj7cfvml8y7zqfr0000gn/T/carthage-xcodebuild.jJkg10.log
*** Fetching Alamofire
*** Building scheme "Alamofire iOS" in Alamofire.xcworkspace

```

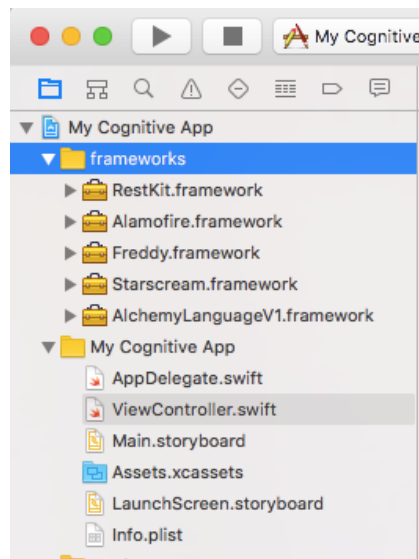
The following image shows output from the package manager as it fetches and builds iOS library components.

Step 2c. Add the SDK to the Xcode project

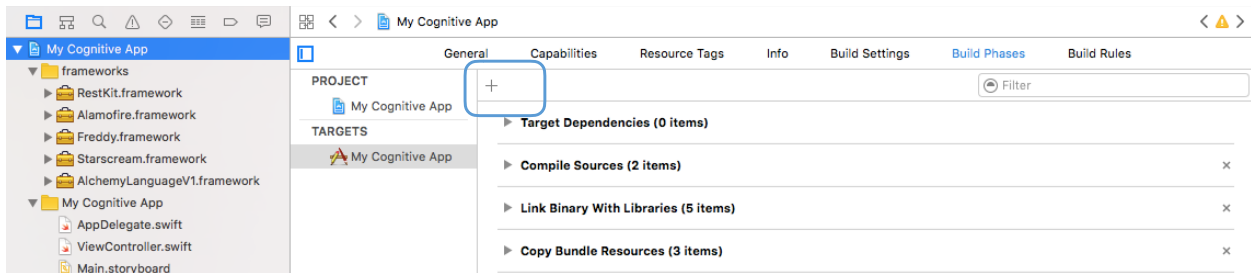
1. Under the project name, create a new group in your Xcode project called `Framework`.
2. Navigate to the root folder of your project in Xcode and select all the framework files in the `<your project>/Carthage/Build/iOS/` directory of your project (AlchemyLanguageV1.framework, Alamofire, Freddy, RestKit, 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.

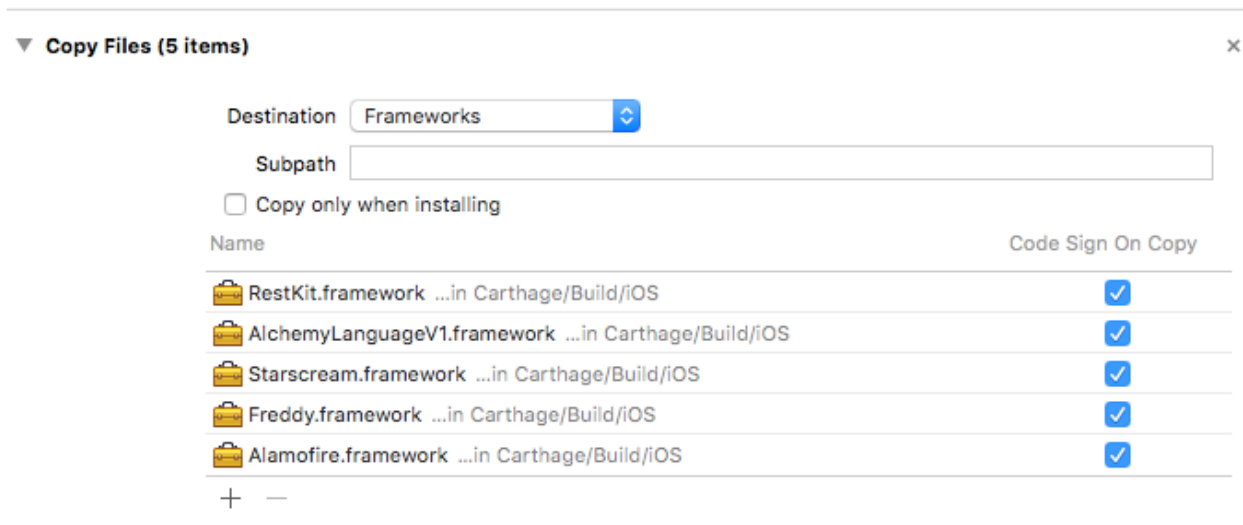
This is the list of the framework libraries:



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



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



5. Whitelist your calls to Watson services and remove the security for watsonplatform.net. Whitelisting is to allow only specified addresses to get through without 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:

▼ 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/>

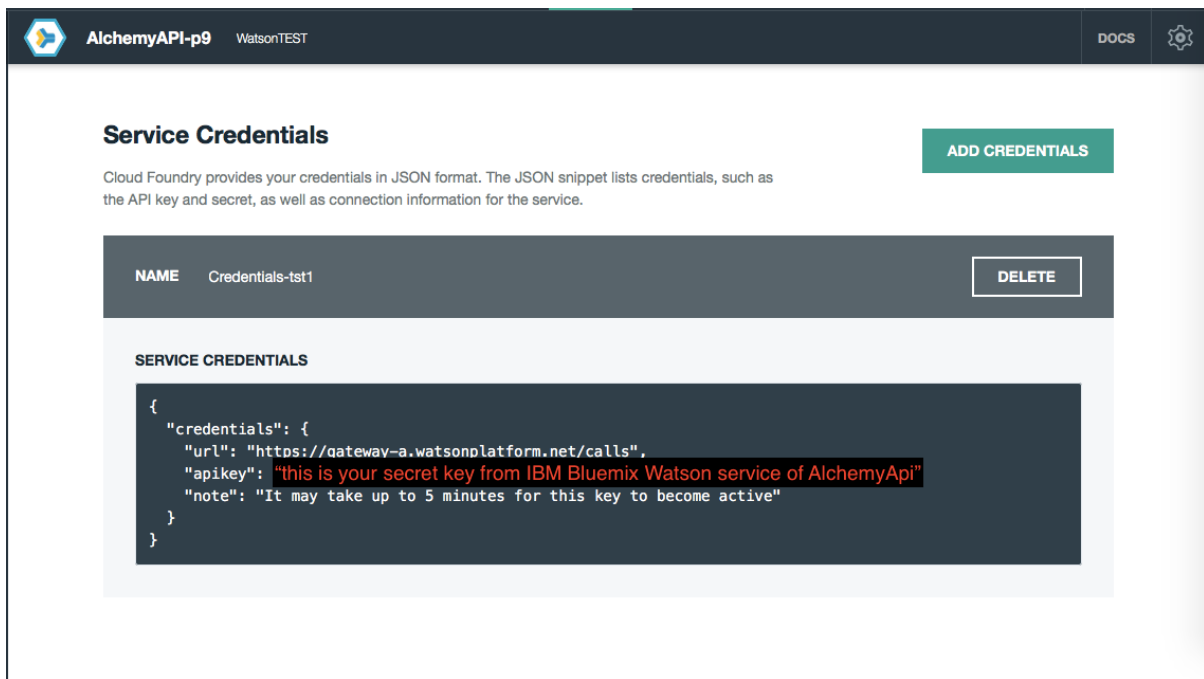
<key>NSTemporaryExceptionAllowsInsecureHTTPLoads</key>
  <true/>
    </dict>
  </dict>
</dict>

```

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

Now, you create the Watson AlchemyAPI service.

1. From the IBM Bluemix catalog, click **Watson > AlchemyAPI > Create**. Be sure to use a static API key as shown in the following image.
2. Find the Service Credentials.



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

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

Import field:

```
import AlchemyLanguageV1
```

So far, your code should look like this:

```
//checking sentiment
let apiKey = "<replace it with the AlchemyAPI key>"
let alchemyLanguage = AlchemyLanguage(apiKey: apiKey)

//alchemyLanguage.
```

```

        let url = textField.text! //"http://www.huffingtonpost.com/
2010/06/22/iphone-4-review-the-worst_n_620714.html"
        let failure = { (error: NSError) in print(error) }

        NSLog("calling GetTextsentiment url          :::::::::::::::")
        alchemyLanguage.getTextSentiment(forURL: url, failure:
failure) { sentiment in
            print(sentiment)
            NSLog((sentiment.docSentiment?.type)!)

            //setting feedback on sentiment
            self.textStatusLabel.text = "text status
sentiment ::::::::::::::: " + (sentiment.docSentiment?.type)!

        }

```

You can ignore any warning messages including the message about “var textData was never mutated.”

When you run the application, you'll see the following results after the application checks the sentiment of the text **at the given URL**:



This is the complete code for the application:

```
//
// ViewController.swift
// My Cognitive App
//
// Created by Marek Sadowski on 5/10/16.
// Copyright © 2016 Marek Sadowski. All rights reserved.
//

import UIKit
import AlchemyLanguageV1

class ViewController: UIViewController {

    @IBOutlet weak var textField: UITextField!
    @IBOutlet weak var textStatusLabel: UILabel!

    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 checkButtonPressed(sender: AnyObject) {
        NSLog(textField.text!)

        //checking sentiment
        let apiKey = "<replace it with the AlchemyAPI key>"
        let alchemyLanguage = AlchemyLanguage(apiKey: apiKey)

        //alchemyLanguage.
    }
}
```

```
        let url = textField.text! // "http://www.huffingtonpost.com/
2010/06/22/iphone-4-review-the-worst_n_620714.html"
        let failure = { (error: NSError) in print(error) }

        NSLog("calling GetTextsentiment url :")

        alchemyLanguage.getTextSentiment(forURL: url, failure: failure)
    { sentiment in
        print(sentiment)
        NSLog((sentiment.docSentiment?.type)!)

        //setting feedback on sentiment
        self.textStatusLabel.text = "text status sentiment :
" + (sentiment.docSentiment?.type)!

    }
}
}
```

Step 3. Run the Watson application

You can now run the Watson application in Xcode by entering various URLs in the text field and clicking the button.

The URL is sent to Watson through the Watson SDK, and Watson returns a sentiment type that is displayed in the status field.