Testmunk Documentation

Release 0.1

Testmunk team

February 29, 2016

1 Android **3**
1.1 Installation . . . 3
1.1.1 Install Calabash gem . . . 3
1.1.2 Download and Install Android SDK . . . 4
1.1.3 Configure Bash profile . . . 4
1.1.4 Download the Testmunk sample application . . . 5
1.2 Preparing testcases . . . 5
1.2.1 Inspect app for elements . . . 5
1.2.2 Writing testcases . . . 7
1.3 Calabash Ruby API . . . 7
1.3.1 Useful methods . . . 8
1.4 Running testruns . . . 9
1.4.1 General . . . 9
1.4.2 Running locally on the emulator . . . 9
1.4.3 Running on your local device . . . 9
1.4.4 Running on multiple Android devices . . . 9
1.5 Image Comparison in Calabash . . . 9

2 iOS **13**
2.1 Installation . . . 13
2.1.1 Prerequisites . . . 13
2.1.2 Install Calabash gem . . . 14
2.1.3 Installing framework in Xcode project . . . 14
2.2 Preparing testcases . . . 21
2.2.1 Inspect app for elements . . . 21
2.2.2 Writing testcases . . . 22
2.3 Calabash Ruby API . . . 23
2.3.1 Useful methods . . . 23
2.4 Running testruns . . . 25
2.4.1 General . . . 25
2.4.2 Running locally on the simulator . . . 25
2.4.3 Running locally on a device . . . 25
2.4.4 Running on multiple iOS devices . . . 26
2.5 Updating Calabash . . . 26
2.5.1 Check the version of Calabash you have installed . . . 27
2.5.2 Update process . . . 27

3 Teststep library **29**

3.1 Touching . . . 29
3.2 Assertions . . . 36
3.3 Input . . . 42
3.4 Waiting . . . 47
3.5 Buttons . . . 54
3.6 Gestures . . . 55

4 REST API **61**
4.1 API Overview . . . 61
4.1.1 Introduction . . . 61
4.1.2 Schema . . . 61
4.1.3 Authentication . . . 62
4.1.4 Versioning . . . 62
4.1.5 Errors . . . 62
4.1.6 Error codes . . . 63
4.1.7 Email notifications . . . 63
4.2 App API . . . 63
4.2.1 List current apps for your organisation . . . 63
4.2.2 Create a new App . . . 64
4.3 Devices API . . . 64
4.3.1 Get available devices . . . 64
4.4 Testruns API . . . 65
4.4.1 Create a new testrun . . . 65
4.4.2 Selecting Devices to Test On . . . 66
4.4.3 Start an existing testrun . . . 66
4.4.4 Get testrun status . . . 67
4.4.5 Get list of testruns . . . 68
4.5 Continuous Integration . . . 69
4.5.1 Jenkins . . . 69

5 Slack Integration **75**
5.1 Setup Guide . . . 75

Contents:

CHAPTER 1

Android

Testmunk Android enable you to write automated functional testcases that you can run on various Android devices with different OS versions. Our goal is that you are able to reduce your manual testing time tremendeously. Following the installation you will be able to write testcases and let them run locally on your emulator and Android device as well as on a variety of Android devices over the cloud in the Testmunk device lab.

1.1 Installation

1.1.1 Install Calabash gem

In order to get started with Testmunk, you need to install the Calabash gem. Simply open your terminal window and execute:

```
$ gem install calabash-android
```

In case you don't have the right permissions, please execute:

```
$ sudo gem install calabash-android
```

Hint: Calabash is a framework that allows you to write automated mobile application tests for iOS and Android. It provides APIs for mimicking input to the devices, and reading its output.

What is Calabash?

Danger: If you get an error that reads `...can't find header files for ruby at /System/...`, that means you do not have the Xcode command-line tools correctly installed. Make sure you have Xcode installed (or download it here) and then run this command in Terminal:

```
$ xcode-select --install
```

Header files can't be found

Danger: If you are getting an error that says "clang: error: unknown argument: '-multiply_definedsuppress"', you must run these 2 commands instead:

```
$ sudo -i

$ ARCHFLAGS=-Wno-error=unused-command-line-argument-hard-error-in-future gem install calabash-andro
```

This error is due to deprecated arguments for the `clang` executable that `gem` calls when installing certain extensions.
'clang error'

1.1.2 Download and Install Android SDK

The Android SDK is the essential tool to build Android apps; by downloading it you will have access to a few tools that you need to test the app.

Download the latest Android SDK

After your download please copy paste both folders (`sdk` and `eclipse`) into your Applications folder.

1.1.3 Configure Bash profile

The Bash profile .bash_profile is a hidden file in your personal folder that you will need to configure for your Android SDK. After having moved the sdk and eclipse folder in your Applications folder, you can copy paste the following 2 lines into your .bash_profile:

```
export ANDROID_HOME=/Applications/sdk
export PATH=$PATH:$ANDROID_HOME:$ANDROID_HOME/tools:$ANDROID_HOME/platform-tools:$ANDROID_HOME/add-or
```

After you have configured your bash profile, please close all terminal windows to affect your changes.

Hint: .bash_profile is a shell script that gets executed every time you open a new Terminal window. It deals with configuration for all of your Terminal commands. It can be found in your user folder, `~/`.

Take into account that this file might not exist; in that case, create a new empty one.

Also consider that, since the filename begins with a period, this file is hidden. In order to see hidden files in a Finder window, run these commands in OS X Mavericks:

```
$ defaults write com.apple.finder AppleShowAllFiles TRUE
$ killall Finder
```

or these commands in OS X versions prior to Mavericks:

```
$ defaults write com.apple.Finder AppleShowAllFiles TRUE
$ killall Finder
```

What is a .bash_profile? Where can I find it?

Plug in your Android device

After you have plugged in your android device into your computer, open a new terminal window and execute:

```
$ adb devices
```

You should see output similar to the following, which confirmes that your device was recognized:

```
List of devices attached
605A000600000001015F3E001200C00B     device
```

In case you don't see any output please confirm that you activated the "USB debugging" mode for your device. You can activate it by going to "Settings".

1.1.4 Download the Testmunk sample application

We provide a simple Android app template for the purposes of this tutorial. To get this app, download or clone this GitHub repository.

In this tutorial, we will only be using the APK file, but the source code is included so you can take a look if you wish.

1.2 Preparing testcases

After you have downloaded the files open a new terminal window and navigate via cd to the TMSampleAndroid folder that you just downloaded. Then execute:

```
$ calabash-android run TestmunkTest_debug.apk --verbose
```

Danger: If this command returns an error that says "No keystores found. Please create one or run calabash-android setup to configure calabash-android to use an existing keystore," then, run the following commands before continuing:

```
$ keytool -genkey -v -keystore ~/.android/debug.keystore -alias androiddebugkey -storepass android

$ calabash-android run TestmunkTest_debug.apk
```

No keystores found

You should get prompted to resign the app. Follow the terminal instructions to resign the app. After resigning please again execute:

```
$ calabash-android run TestmunkTest_debug.apk --verbose
```

This should install the app on your device, and after a minute or two it should get launched and our sample testcases should get executed. It will ca. 3-5 min for the testcases to get completed, you can also exit the terminal to stop the testrun. Simply enter `exit` into the terminal.

In the following we'd like to show you how you can easily write your own testcase.

1.2.1 Inspect app for elements

UI Automator Viewer

Please tap on the app on your device so that it is launched. Open a new terminal window and execute:

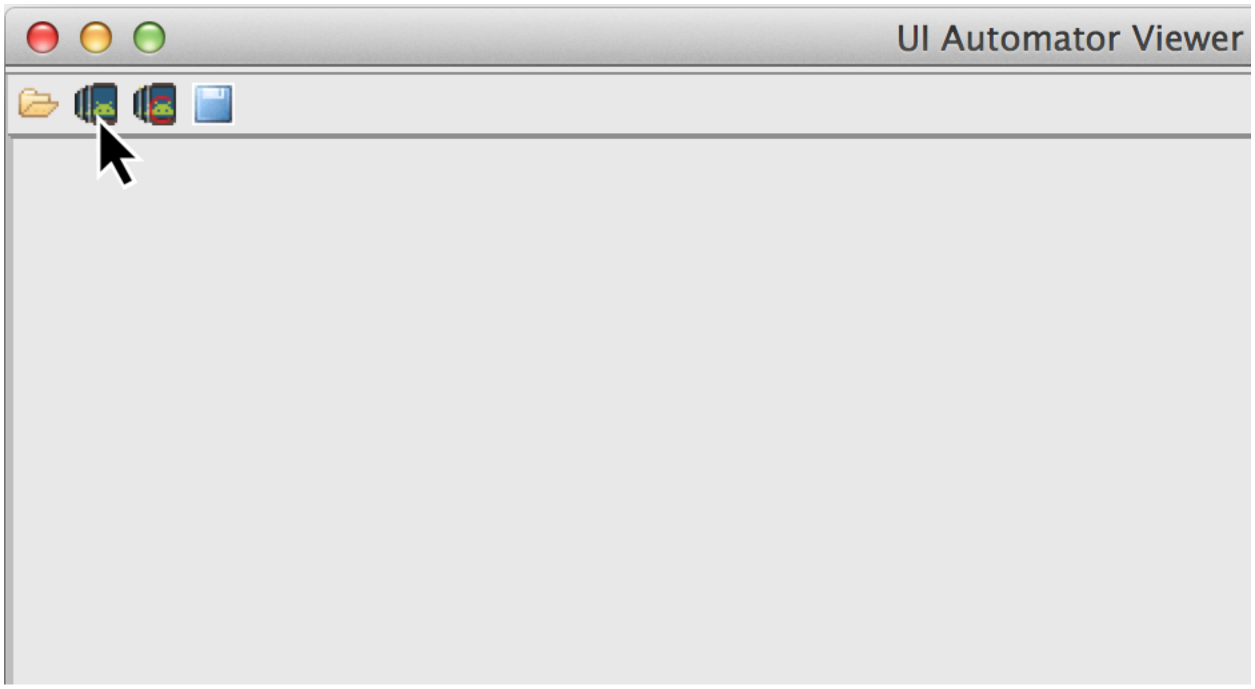
```
$ uiautomatorviewer
```

Hint: In case you need to install an `APK` file on your device, you can `cd` into the folder that contains it and use this command:

```
$ adb install NameofTheFile.apk
```

Installing APKs

On the newly opened window, please click on the device icon on the upper left corner to get an actual screenshot from the device.



This inspection is important to identify the right elements that you later will need for your testcases. For example the `resource_id` is needed when you use the teststep `Then I touch view with id....`

Important: If you use an Android version lower than API level 18 / Jelly Bean you will not be able to interact with the `resource-id` of the element.

Early versions of Android

Calabash console

A more advanced way of inspecting elements on the view is using the console. Open a new terminal window, `cd` into the folder that contains your `APK` file, and enter:

```
$ calabash-android console TestmunkTest_debug.apk
```

and then enter these commands:

```
> start_test_server_in_background

> query("*")
```

You should see a list of all visible elements.

1.2.2 Writing testcases

Within the sample app that you just downloaded, please open the `my_first.feature` file within the `feature` folder. These are some sample testcases that we scripted for a demo application. One testcase looks like this:

```
Feature: Testapp V.1.2

Scenario: 1) Going to next screen and back
        When I enter "Something" into input field number 1
        Then I press the "See details" button
        Then I wait
        Then I should see text containing "Something"
        Then I go back
        Then I should see text containing "test app"
```

In order to write a second testcase write a new testscenario. For example:

```
Scenario: 1) Going to next screen and back
        When I enter "Something" into input field number 1
        Then I press the "See details" button
        Then I wait
        Then I should see text containing "Something"
        Then I go back
        Then I should see text containing "test app"

Scenario: 2) Clear the input field
        When I enter "Something" into input field number 1
        Then I clear input field number 1
        Then I press the "See details" button
        Then I should not see "Something"
```

Once you have writen your testcases, it is necessary to save them. Running a testrun requires the `feature` folder (that `my_first.feature` is in) to be compressed to a `.zip` file. If you are using the testmunk website to run testcases, then the `.apk` file chosen should be your app, and the `.zip` file chosen should be the `feature` folder you just created.

Hint: For writing testcases, we recommend using Sublime Text 2 with the Cucumber syntax highlighting plugin.

Text editor suggestion

In case you were wondering where these steps come from, have a look at the Teststep library. These are all steps that you can be using right away. In case you'd like to extend and write your own steps, have a look into the .rb file in the `step_definitions` folder and the Calabash Ruby API.

1.3 Calabash Ruby API

Calabash offers a Ruby API that we support for defining special teststeps.

A new teststep is defined in the following way:

```
# Define a regular expression to catch the step
Then(/^"(.*?)" radio button should be selected$/) do |arg1|
  # Use calls to the Calabash API to get information
  if(!query("RadioButton text:'#{arg1}'", :checked).first())
    # Act on that information
    fail("The radio button with text #{arg1} should be selected")
  end
end
```

A teststep is considered succesful if the execution of its codeblock runs with neither explicit fails nor uncaught errors.

A nice way to try the different commands on this API is to run the Calabash console and test them.

1.3.1 Useful methods

This are some useful functions that the Calabash API provides. You can see more about them on the Calabash GitHub documentation.

query(uiquery, *args)

Query returns an array with the views on the screen that match it.

```
> query("FrameLayout index:0")

[
    [0] {
                        "id" => "content",
                   "enabled" => true,
        "contentDescription" => nil,
                     "class" => "android.widget.FrameLayout",
                      "rect" => {
            "center_y" => 617.0,
            "center_x" => 384.0,
              "height" => 1134,
                   "y" => 50,
               "width" => 768,
                   "x" => 0
        },
               "description" => "android.widget.FrameLayout{41f40dc0 V.E..... ........ 0,50-768,1184
    }
]
```

Each result is a Ruby hash map object.

```
> query("FrameLayout index:0").first.keys

[
    [0] "id",
    [1] "enabled",
    [2] "contentDescription",
    [3] "class",
    [4] "rect",
    [5] "description"
]

> query("FrameLayout index:0")[0]["id"]

"content"
```

wait_for_elements_exist(elements_arr, options={})

Waits for all queries in the `elements_arr` array to return results before continuing the test.

```
wait_for_elements_exist( ["button marked:'OK'", "* marked:'Cancel'"], :timeout => 2)
```

touch(uiquery, options={})

Touches the first result of the query `uiquery`.

```
touch("FrameLayout index:0")
touch(query("FrameLayout"))
```

1.4 Running testruns

1.4.1 General

Testmunk Android enables you to run your testcases on:

1. the virtual emulator
2. on your plugged in Android device
3. on a variety of Android devices with different OS versions in the Testmunk device lab.

1.4.2 Running locally on the emulator

Go to applications and start “Eclipse”. In the menu bar click click on *Window* > *Android Virtual Device Manager* and create an emulator you want to test on.

1.4.3 Running on your local device

Ensure that your device is being recognised by starting a terminal window and executing adb devices.

In order to run your tests on your device, please navigate via `cd` to your project folder and execute:

```
$ calabash-android run sample.apk --verbose
```

Your testrun should get executed on your device. It’s important that you use an apk file that is in debug mode.

1.4.4 Running on multiple Android devices

In order to run your testcases on Testmunk’s devices and see a report with your test results and screenshots, simply create an account, upload your apk file and testcases.

1.5 Image Comparison in Calabash

The goal of this post is to show how we can do basic image recognition using Calabash Android library.

Image comparison is another way that allows you to assert your tests using Calabash. However, Calabash does not support it by default. So, we have created some custom steps that you can include in your features folder, and you’ll have image comparison working in a short time.

Image comparison is a tricky topic. Some comparisons are as simple as pixel by pixel checking; very advanced scenarios may compare a small image within a bigger image, or even images which are slightly shifted or compressed.

We've chosen the simple approach for now, which means a pixel by pixel check. This check uses a difference blend, which is the same thing Github uses to diff images.

If we have pixelation, or an image that is slightly lighter or darker, the steps will still be able to make the comparison. Another benefit is that it returns a more realistic readout of percentage changed, and allows us to set maximum thresholds while testing.

If you want to compare an image (local or remote) with the current screen shot, it needs to match the resolution in order to be effective. The best use case is testing the app on a device that you already have the screenshots for.

To get up and running, we will need to install an extra gem to handle the image manipulation. We can do that using:

```
$ gem install oily_png
```

This is in addition to the calabash-android gem, which should already be installed and configured.

Once you have the gem installed, create a new file under features/step_definitions folder (with any name). Paste in the following code:

```
require 'oily_png'
require 'open-uri'
include ChunkyPNG::Color

def starts_with(item, prefix)
  prefix = prefix.to_s
  item[0, prefix.length] == prefix
end

# compares two images on disk, returns the % difference
def compare_image(image1, image2)
  images = [
    ChunkyPNG::Image.from_file("screens/#{image1}"),
    ChunkyPNG::Image.from_file("screens/#{image2}")
  ]
  count=0
  images.first.height.times do |y|
    images.first.row(y).each_with_index do |pixel, x|

      images.last[x,y] = rgb(
        r(pixel) + r(images.last[x,y]) - 2 * [r(pixel), r(images.last[x,y])].min,
        g(pixel) + g(images.last[x,y]) - 2 * [g(pixel), g(images.last[x,y])].min,
        b(pixel) + b(images.last[x,y]) - 2 * [b(pixel), b(images.last[x,y])].min
      )
      if images.last[x,y] == 255
        count = count + 1
      end
    end
  end

  100 - ((count.to_f / images.last.pixels.length.to_f) * 100);
end

# find the file
def get_screenshot_name(folder, fileName)
  foundName = fileName
  Dir.foreach('screens/') do |item|
  next if item == '.' or item == '..'
```

```
    if item.start_with? fileName.split('.')[0]
      foundName = item
    end
  end

  foundName
end

def setup_comparison(fileName, percentageVariance, forNotCase = false)
  screenshotFileName = "compare_#{fileName}"
  screenshot({ :prefix => "screens/", :name => screenshotFileName })

  screenshotFileName = get_screenshot_name("screens/", screenshotFileName)
  changed = compare_image(fileName, screenshotFileName)
  FileUtils.rm("screens/#{screenshotFileName}")

  assert = true
  if forNotCase
    assert = changed.to_i < percentageVariance
  else
    assert = changed.to_i > percentageVariance
  end

  if assert
    fail(msg="Error. The screen shot was different from the source file. Difference: #{changed.to_i}
  end

end

def setup_comparison_url(url, percentageVariance)
  fileName = "tester.png"
  open("screens/#{fileName}", 'wb') do |file|
    file << open(url).read
  end

  setup_comparison(fileName, percentageVariance)
  FileUtils.rm("screens/#{fileName}")
end

Then(/^I compare the screen with "(.*?)"$/) do |fileName|
  setup_comparison(fileName, 0)
end

Then(/^I compare the screen with url "(.*?)"$/) do |url|
  setup_comparison_url(url, 0)
end

Then(/^the screen should not match with "(.*?)"$/) do |fileName|
  setup_comparison(fileName, 0, true)
end

Then(/^I expect atmost "(.*?)" difference when comparing with "(.*?)"$/) do |percentageVariance, file
  setup_comparison(fileName, percentageVariance.to_i)
end

Then(/^I expect atmost "(.*?)" difference when comparing with url "(.*?)"$/) do |percentageVariance,
  setup_comparison_url(url, percentageVariance.to_i)
end
```

If you are using local screen shots, add the source images to a "screens" folder at the same level as the features folder. You will use the name of these images in your test steps.

The following steps are available after injecting the library:

```
Then I compare the screen with "login_screen.png"
Then I expect atmost "2%" difference when comparing with "login_screen_fail.png"

Then I compare the screen with url "http://testmunk.com/login_screen.png"
Then I expect atmost "2%" difference when comparing with url "http://testmunk.com/login_screen_fail.p

Then the screen should not match with "screen2.png"
```

You have three different types of steps. One asserts an exact match, another asserts an approximate match (i.e. up to 2%), and the final one reads if the image does not match (asserting if a particular view-changing action has happened or not). You can also use local files (which should be present in the /screens folder) or remotely uploaded files.

If there is a match failure, you will get the percentage difference in the output so you know how much of the screenshot was to the source.

Sources:

- http://jeffkreeftmeijer.com/2011/comparing-images-and-creating-image-diffs/

Note:

- This will work with Calabash iOS as well. However, for games using OpenGL, the screenshot utility of Calabash does not work.

CHAPTER 2

iOS

Testmunk iOS enables you to write automated functional testcases that you can run on various iOS devices with different OS versions. Our goal is that you are able to reduce your manual testing time tremendeously. Following the installation you will be able to write testcases and let them run locally on your simulator and iOS device as well as on a variety of iOS devices over the cloud in the testmunk device lab.

2.1 Installation

This section will guide you on how to prepare your environment and Xcode project for running Calabash tests on iOS apps locally, and deploy them to testmunk for testing on multiple devices. For this tutorial, we recommend you use our sample app project (just clone the repository or download the ZIP file).

Hint: Calabash is a framework that allows you to write automated mobile application tests for iOS and Android. It provides APIs for mimicking input to the devices, and reading its output.

What is calabash?

2.1.1 Prerequisites

You must be using a machine with Mac OS X. This machine must also have Xcode with its command line tools, and Ruby installed.

Hint: Download Xcode from here. Once you have Xcode installed, run this command in Terminal to install the command-line tools:

```
$ xcode-select --install
```

How do I install Xcode and its command line tools?

Hint: Your machine must have at least version 1.8.7 for Ruby to ensure compatibility for the gems we will install in the next section. By default, your Mac OS X 10.8 installation comes with Ruby 1.8.7 installed. You can check your version by running `ruby -v` in Terminal. If you still need to upgrade, you can try the following steps:

1. Start Terminal. This can be found in the Applications folder -> Utilities folder.
2. Check if you have RVM. This can be done by typing `rvm` into the Terminal and pressing Enter.

3. If you do not have RVM, install RVM using the following command in Terminal: `curl -L get.rvm.io | bash -s`
4. If you have RVM, install a newer version of Ruby using the following command: `rvm install 1.9.3`.
5. Once Ruby is installed, you can verify the update using `ruby -v` in Terminal.

How do I install Ruby?

2.1.2 Install Calabash gem

In order to get started with testmunk, you need to install the calabash gem. Simply open your terminal window and execute:

```
$ gem install calabash-cucumber
```

In case you don't have the right permissions, please execute:

```
$ sudo gem install calabash-cucumber
```

Danger: If you get an error that reads `...can't find header files for ruby at /System/...`, that means you do not have the Xcode command-line tools correctly installed. Make sure you have Xcode installed (or download it here) and then run this command in Terminal:

```
$ xcode-select --install
```

Header files can't be found

Danger: If you are getting an error that says "clang: error: unknown argument: '-multiply_definedsuppress"', you must run these 2 commands instead:

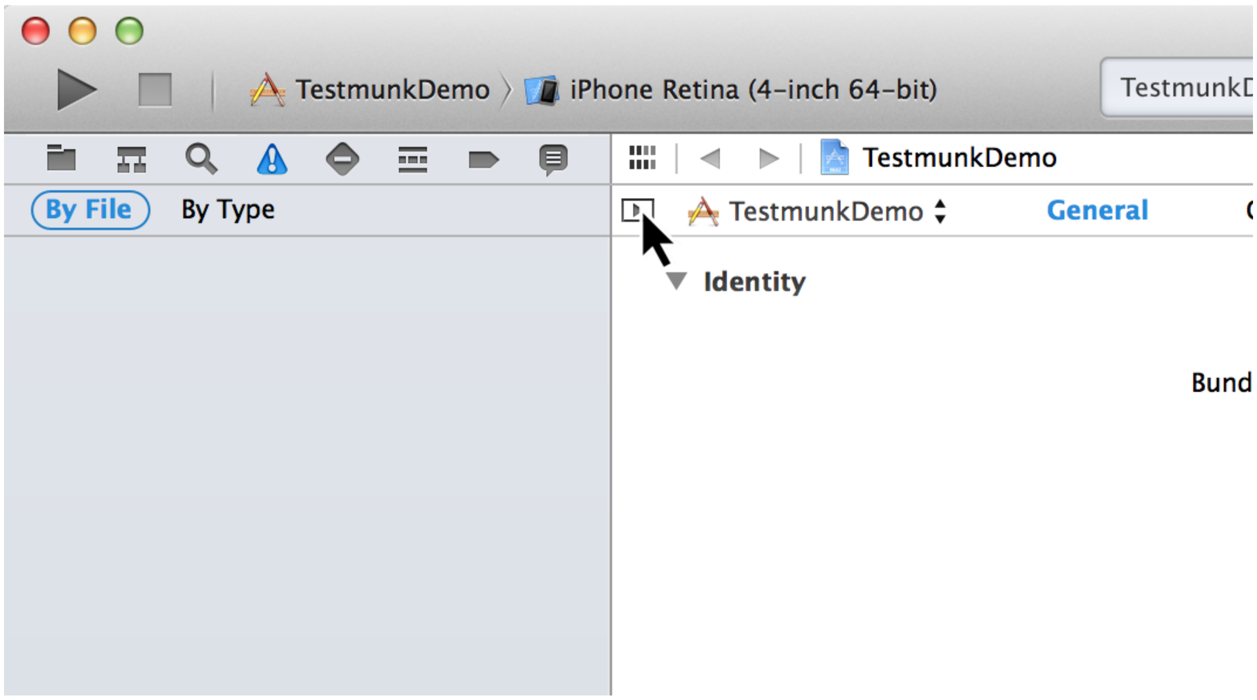
```
$ sudo -i

$ ARCHFLAGS=-Wno-error=unused-command-line-argument-hard-error-in-future gem install calabash-cucum
```

This error is due to deprecated arguments for the `clang` executable that `gem` calls when installing certain extensions.
'clang error'

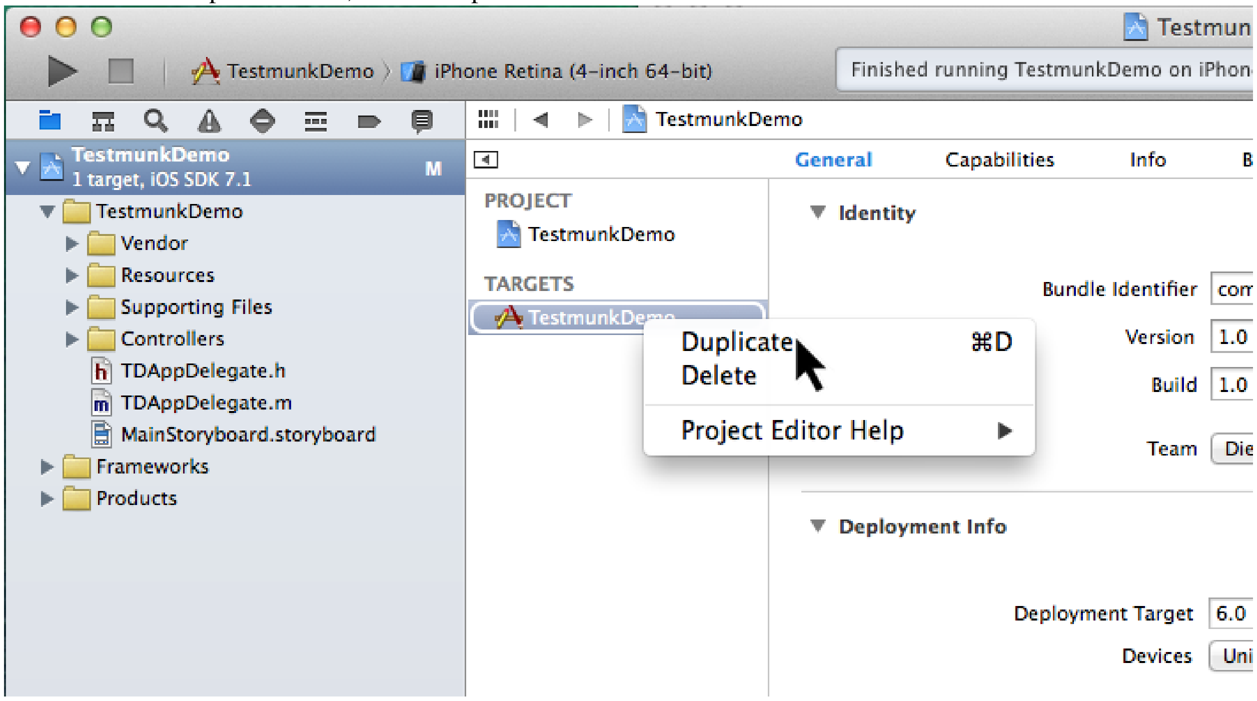
2.1.3 Installing framework in Xcode project

Creating a new build target

1. Open the testmunk sample project in Xcode.
2. Select your project (from the File Navigator).
3. Right click your target in the list of targets. If you do not see the list of targets, you need to press this button:

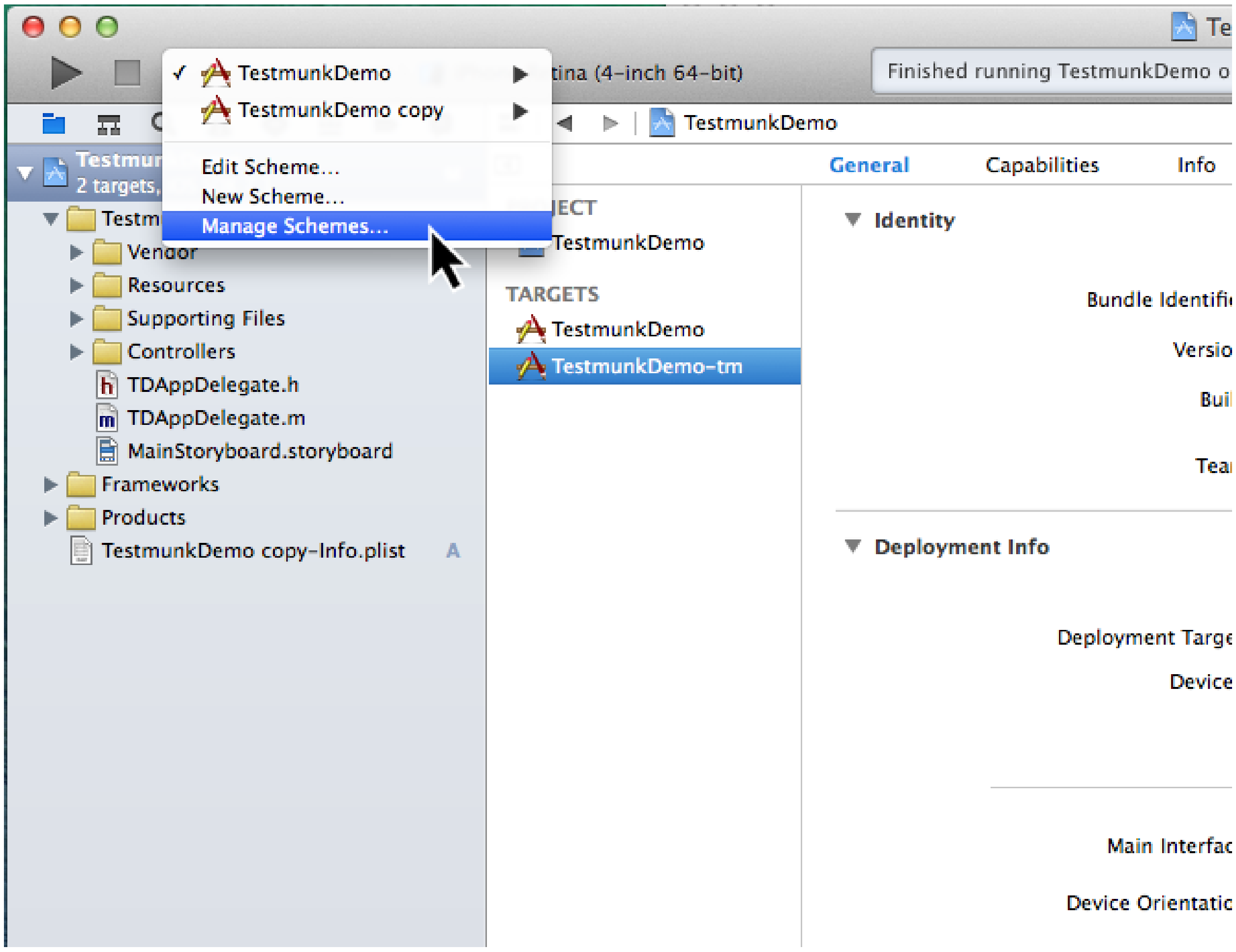


4. From the dropdown menu, select “Duplicate”.

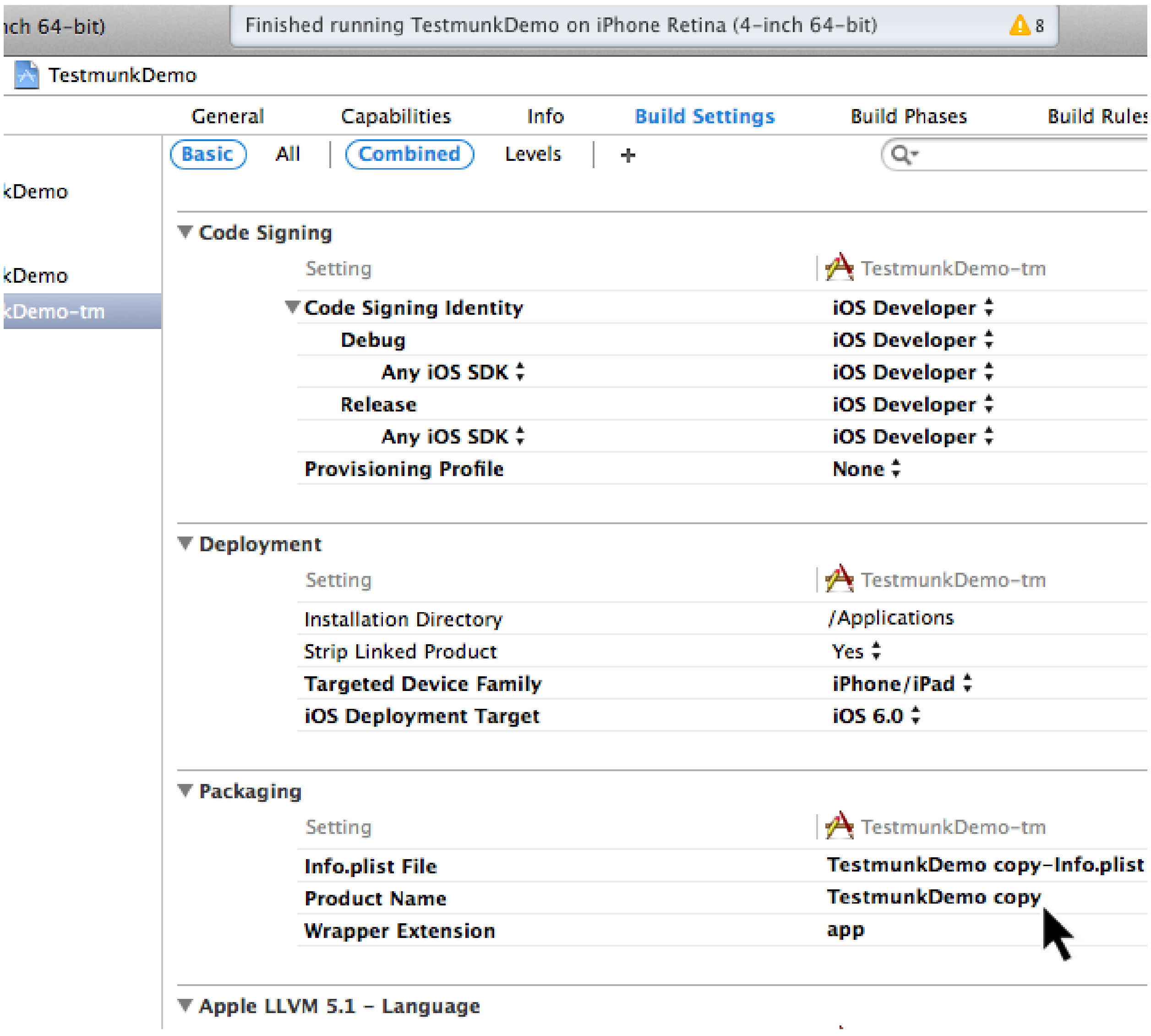


5. Rename the new target from “TestmunkDemo copy” to “TestmunkDemo-tm” by clicking on it and pressing Enter.

6. Click in the toolbar where it says TestmunkDemo, and from the dropdown menu, select “Manage Schemes”.



7. Rename the new scheme from “TestmunkDemo copy” to “TestmunkDemo-tm” by clicking on it and pressing Enter. Then press OK.
8. Click on “Build Settings” and under *Packaging* set the “Product Name” to “TestmunkDemo-tm”


ich 64-bit)

Finished running TestmunkDemo on iPhone Retina (4-inch 64-bit) ⚠ 8

TestmunkDemo

▼ Code Signing

Setting	TestmunkDemo-tm
▼Code Signing Identity	iOS Developer ‡
Debug	iOS Developer ‡
Any iOS SDK ‡	iOS Developer ‡
Release	iOS Developer ‡
Any iOS SDK ‡	iOS Developer ‡
Provisioning Profile	None ‡

▼ Deployment

Setting	TestmunkDemo-tm
Installation Directory	/Applications
Strip Linked Product	Yes ‡
Targeted Device Family	iPhone/iPad ‡
iOS Deployment Target	iOS 6.0 ‡

▼ Packaging

Setting	TestmunkDemo-tm
Info.plist File	TestmunkDemo copy-Info.plist
Product Name	TestmunkDemo copy
Wrapper Extension	app

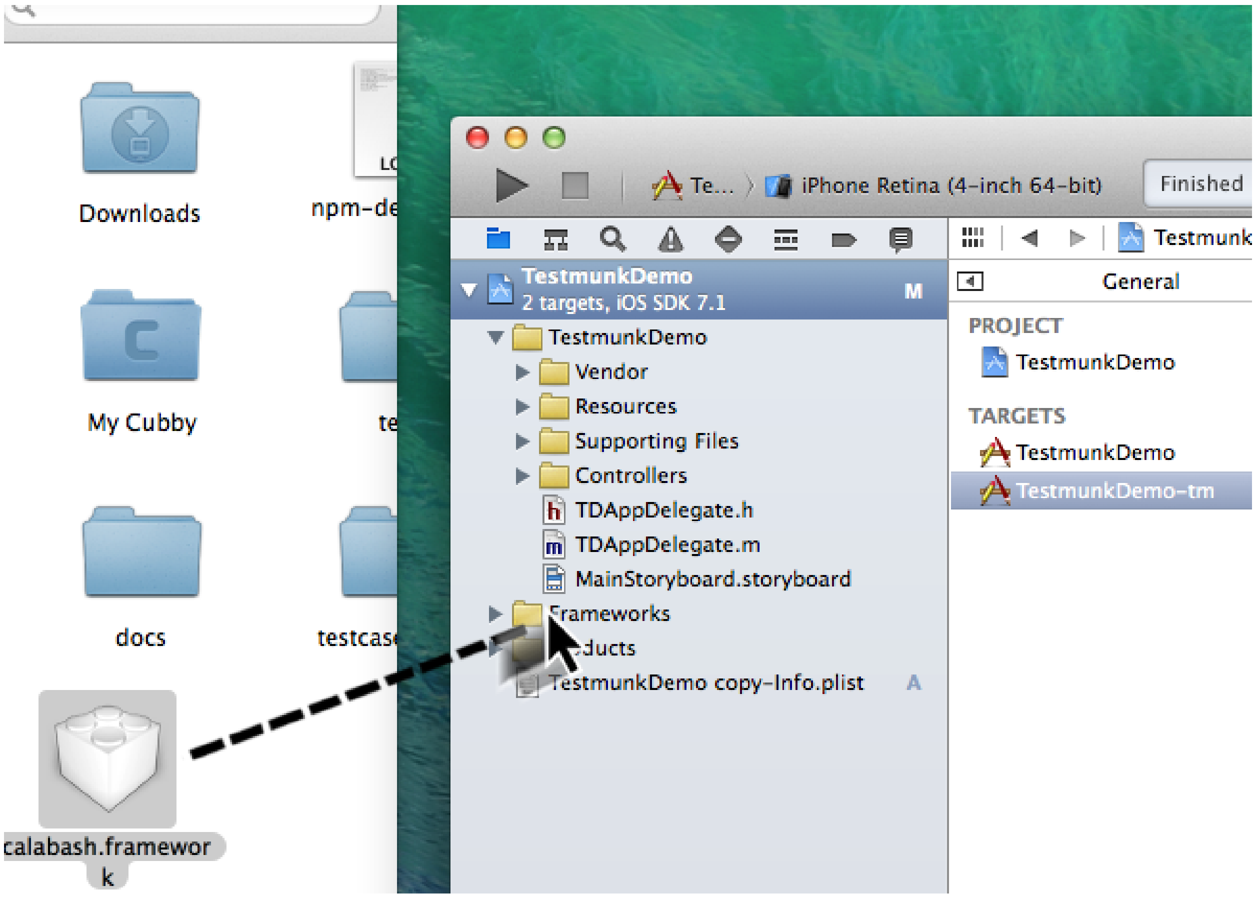
▼ Apple LLVM 5.1 - Language

Hint: You want to build your app with the Calabash framework only if you are building your app for testing purposes. That is why we are setting up a target specifically for running tests.

Why are we creating a new build target?

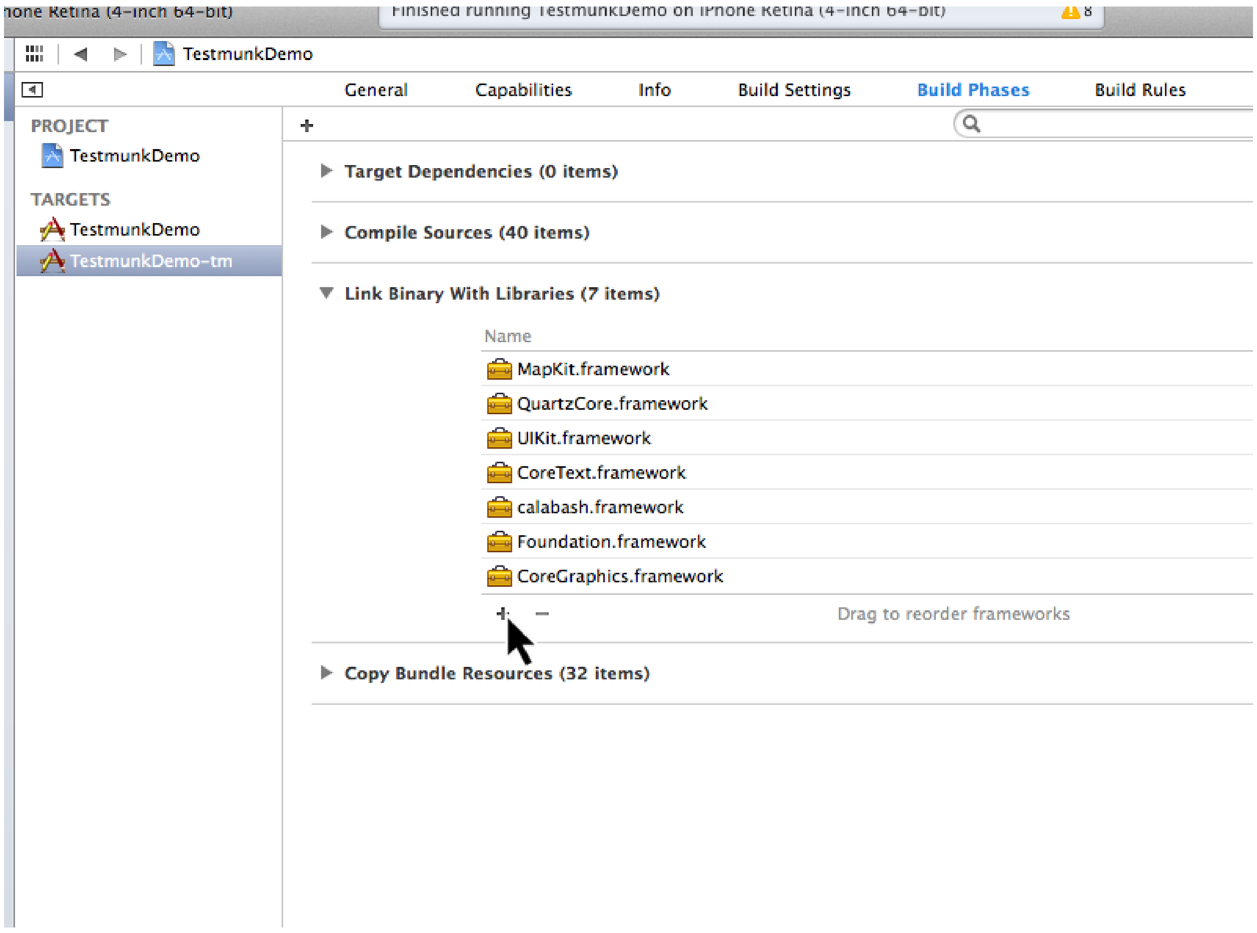
Link the Calabash framework

1. Open terminal and run `calabash-ios download` to download the latest `calabash.framework` file.
2. Run the command `open .` in Terminal.
3. Drag `calabash.framework` from its current location to the project's Frameworks folder in Xcode.



4. In the pop up window that appears, select *Copy items into destination group's folder (if needed)* and make sure "TestmunkDemo-tm" is the only selected target.

5. Select the "TestmunkDemo-tm" target, go to "Build Phases", and in the "Link Binary With Libraries" section, make sure that `calabash.framework` is present. Link the CFNetwork framework so that Calabash can communicate with your app, by clicking '+', and selecting `CFNetwork.framework`.



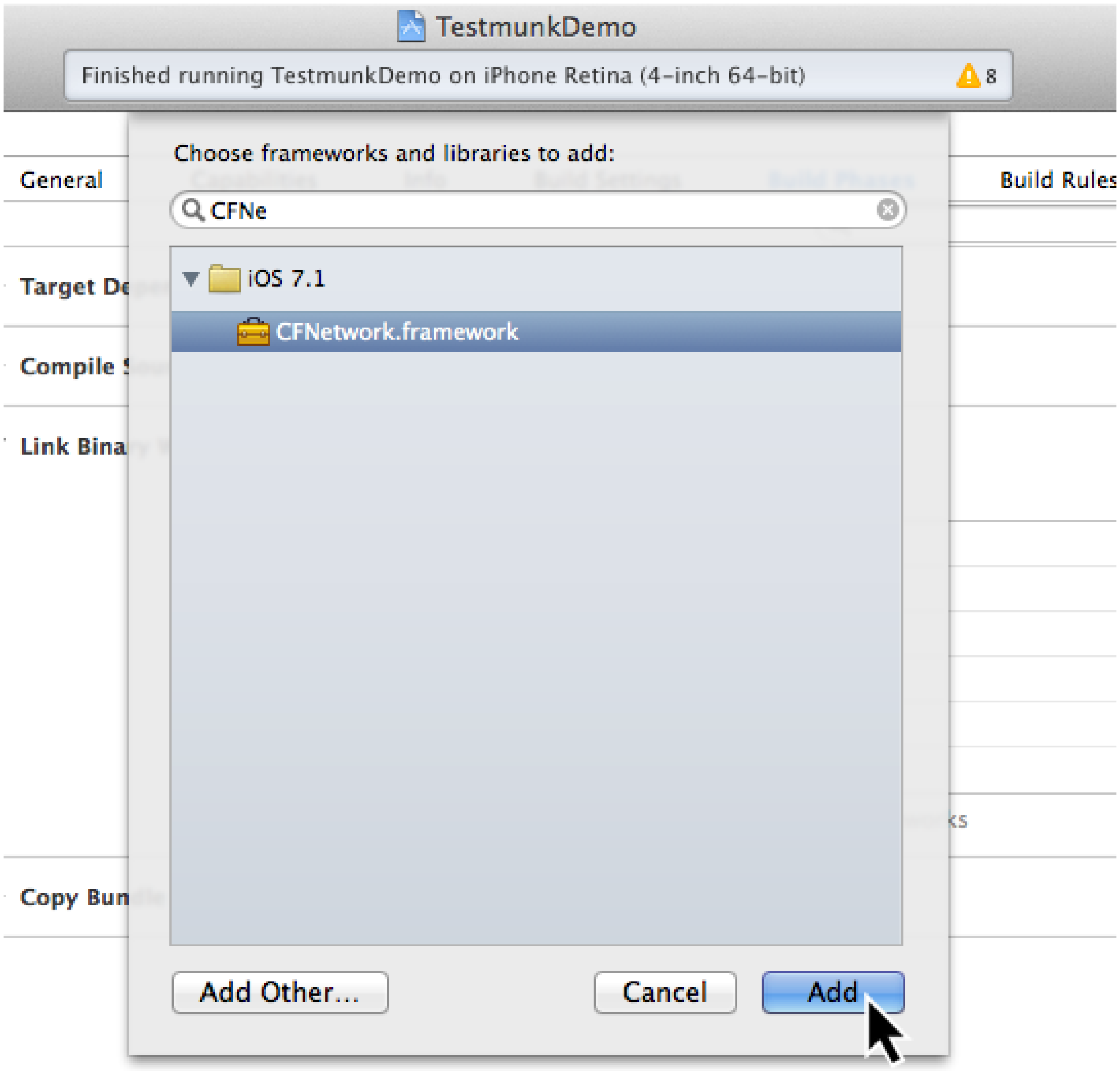
Target Dependencies (0 items)

Compile Sources (40 items)

Link Binary With Libraries (7 items)

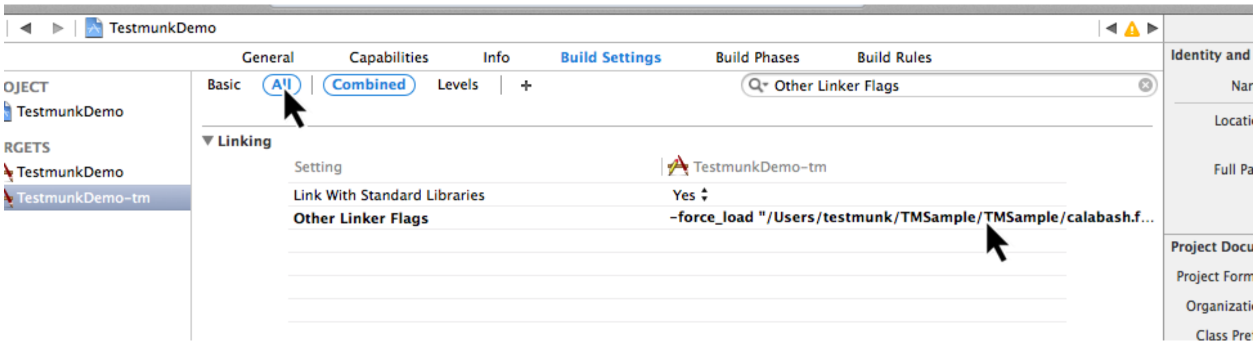
Name
MapKit.framework
QuartzCore.framework
UIKit.framework
CoreText.framework
calabash.framework
Foundation.framework
CoreGraphics.framework

Copy Bundle Resources (32 items)



Configure the build target

1. Select “Build Settings”
2. Change the filter from “Basic” to “All”
3. Make sure that “Other Linker Flags” contains: `-force_load "$(SRCROOT)/calabash.framework/calabash" -lstdc++`



▼ Linking

Setting	TestmunkDemo-tm
Link With Standard Libraries	Yes
Other Linker Flags	-force_load "/Users/testmunk/TMSample/TMSample/calabash.f...

Test the configuration

Build and run your application on the simulator. You should be getting console output similar to this:

```
2014-05-30 16:08:07.882 TestmunkDemo-tm[3088:60b] Creating the server: <LPHTTPServer: 0xa0970d0>
2014-05-30 16:08:07.883 TestmunkDemo-tm[3088:60b] Calabash iOS server version: CALABASH VERSION: 0.9
2014-05-30 16:08:07.889 TestmunkDemo-tm[3088:60b] Started LPHTTP server on port 37265
2014-05-30 16:08:08.810 TestmunkDemo-tm[3088:1903] Bonjour Service Published: domain(local.) type(_ht
```

2.2 Preparing testcases

After your Xcode project has been set up, and you have ran your app on the simulator for the first time, you are ready to make and run your own testcases.

2.2.1 Inspect app for elements

Accessibility Inspector

To be able to test, you need to have a way to reference different elements in your application. iOS devices have been setting new, improved usability standards for impaired users, since Accessibility functions help them navigate through the app. These Accessibility functions come in very handy for test automation. Test cases become more robust and easier to maintain. You can enable the Accessibility Inspector by starting the Simulator, then going `Settings -> General` and setting Accessibility Inspector to ON.

Once the Accessibility Inspector is enabled, you can switch between two modes, inspection and navigation. If the Accessibility Inspector is minimized, then the app is in navigation mode. This means that you can tap on buttons to perform actions.

However, once you click on the X button, the window enlarges – then you are in inspection mode. Now clicking on the button will show its accessibility details.

To go back to the navigation mode, simply click the X again to minimize the Accessibility Inspector.

Calabash console

A more advanced way of inspecting elements on the view is using the Calabash console to see a list of the app's visible elements. Inside the folder you downloaded, go to `TMSample/` and run this Terminal command:

```
$ calabash-ios console
> start_test_server_in_background
```

This will start our test application in the iOS simulator. then enter:

```
> query("*")
```

You should see a list of all visible elements.

```
[
    [ 0] {
              "class" => "UIWindow",
                 "id" => nil,
               "rect" => {
            "center_x" => 160,
                   "y" => 0,
               "width" => 320,
                   "x" => 0,
            "center_y" => 284,
              "height" => 568
        },
              "frame" => {
                 "y" => 0,
             "width" => 320,
                 "x" => 0,
            "height" => 568
        },
              "label" => nil,
        "description" => "<UIWindow: 0xb26fc60; frame = (0 0; 320 568); gestureRecognizers = <NSArray
    },
    [ 1] {
              "class" => "UIView",
                 "id" => nil,
               "rect" => {
            "center_x" => 160,
                   "y" => 0,
               "width" => 320,
                   "x" => 0,
            "center_y" => 284,
              "height" => 568
        },
              "frame" => {
                 "y" => 0,
             "width" => 320,
                 "x" => 0,
            "height" => 568
        },
              "label" => nil,
        "description" => "<UIView: 0x9eb3610; frame = (0 0; 320 568); autoresize = RM+BM; layer = <CA
    },
    ...

]
```

2.2.2 Writing testcases

We have installed a feature folder in your project folder. Inside the folder you downloaded, go to `TMSample/features/`, and open the `myfirst.feature` file to write your testcases. You can ignore the folders step_definitions and support at this point.

The my_first.feature file is structured in the following way:

```
# Scenario name
Scenario: Login
  Given I am on the Welcome Screen # Teststeps
  Then I touch the "Email" input field
  Then I use the keyboard and type "test@testname.com"
  Then I touch the "Password" input field
  Then I use the keyboard and type "testmunk"
  Then I touch "SIGN IN"
  Then I wait
  Then I should see "Hello world"
```

Hint: For writing testcases, we recommend using Sublime Text 2 with the Cucumber syntax highlighting plugin.

Text editor suggestion

You can write as many testcases as you want in your `myfirst.feature` file. Don't delete the feature title line, since it is needed for a successful execution of your testcase.

Good to know: Ensure that each testcase starts from the beginning. We call it testcase independency. When you run your app on our devices, we clear the app data before each testcase. So, if your app always starts with the login process, you will need to have teststeps that cover the login process at the beginning of each testcase. Testcase independency makes your testcases more robust, and it means every testcase can be tested independently.

Our teststep library can come in handy as a reference as you're writing your testcases.

If you run your app over our device lab, we automatically take screenshots after each teststep – you don't need to worry about it at all.

2.3 Calabash Ruby API

Calabash offers a Ruby API that we support for defining special teststeps.

A new teststep is defined in the following way:

```
# Define a regular expression to catch the step
Then(/^"(.*?)" radio button should be selected$/) do |arg1|
  # Use calls to the Calabash API to get information
  if(!query("RadioButton text:'#{arg1}'", :checked).first())
    # Act on that information
    fail("The radio button with text #{arg1} should be selected")
  end
end
```

A teststep is considered succesful if the execution of its codeblock runs with neither explicit fails nor uncaught errors.

A nice way to try the different commands on this API is to run the Calabash console and test them.

2.3.1 Useful methods

This are some useful functions that the Calabash API provides. You can see more about them on the Calabash GitHub documentation.

query(uiquery, *args)

Query returns an array with the views on the screen that match it.

```
> query("UIButton")

[
    [0] {
                  "class" => "FUIButton",
                     "id" => nil,
                   "rect" => {
                "center_x" => 160,
                       "y" => 194,
                   "width" => 300,
                       "x" => 10,
                "center_y" => 216,
                  "height" => 44
        },
                  "frame" => {
                     "y" => 194,
                 "width" => 300,
                     "x" => 10,
                "height" => 44
        },
                  "label" => "SIGN IN",
        "description" => "<FUIButton: 0x9f909e0; baseClass = UIButton; frame = (10 194; 300 44); opac
    }
]
```

Each result is a Ruby hash map object.

```
> query("UIButton").first.keys

[
    [0] "class",
    [1] "id",
    [2] "rect",
    [3] "frame",
    [4] "label",
    [5] "description"
]

> query("UIButton")[0]["label"]

"SIGN IN"
```

wait_for_elements_exist(elements_arr, options={})

Waits for all queries in the `elements_arr` array to return results before continuing the test.

```
wait_for_elements_exist( ["button marked:'OK'", "* marked:'Cancel'"], :timeout => 2)
```

touch(uiquery, options={})

Touches the first result of the query `uiquery`.

```
touch("UIButton index:0")
touch(query("UIButton"))
```

2.4 Running testruns

2.4.1 General

Testmunk iOS enables you to run your testcases on:

1. the virtual simulator
2. on a variety of iOS devices with different OS versions in the testmunk device lab.

2.4.2 Running locally on the simulator

Inside the folder you downloaded, go to `TMSample/`, where the Xcode project resides, and run the following command:

```
$ cucumber -v
```

That will initiate the testruns on your simulator.

Hint: The `-v` argument launches Cucumber in *verbose mode*, which means that it will print more detailed information to the console while running. We run it this way to know exactly what went wrong or right with the testing.

Why '-v'?

Danger: If you are getting an error that reads `tool 'xcodebuild' requires Xcode, but active developer directory ...`, then open Xcode, and go to *Xcode* > *Preferences* > *Locations* and in the *Command Line Tools* dropdown menu select *Xcode*.
'xcodebuild' error

Danger: If `cucumber -v` dosen't work, try this fixes one at a time:
- Manually run your app on the simulator through Xcode, and then close and stop it.
- Make sure Xcode is not executing any projects at the time.
- Have only one instance of Xcode open, with the project you are trying to run, and hte scheme you want to build selected.
- Make sure you have the same version of Calabash in your Terminal tool and the framework you are linking in your project.
- Choose the *Reset content and settings* option in your iOS simulator.

Other errors

2.4.3 Running locally on a device

If you want to run a test locally in your device, you need to:

- Connect your device to your computer with a USB cable
- Connect your device and computer to the same LAN

- Deploy the app to your device using Xcode, making sure to deploy the target that has the Calabash framework linked to it (a.k.a. the "...-tm" target).
- `cd` to the directory that contains your `.app` file and your `features` folder
- Use this command:

```
$ RESET_BETWEEN_SCENARIOS=1 DEVICE_ENDPOINT=http://192.168.1.43:37265 DEVICE_TARGET=97da4f58
```

  Make sure to replace `192.168.1.43` with your device's LAN IP address, `97da4f58c9a95b7286c760372fd3d27be85a17cf` with your device's UDID, `com.sample.TestmunkDemo-copy` with your application's Bundle Identifier, and `TestmunkDemo.app` with your `.app` filename.

Hint: The RESET_BETWEEN_SCENARIOS=1 variable will make your tests start with a fresh install of your application. We recommend this method to keep tests independant, as opposed to relying on each other, to make them easier to debug. Testmunk servers will always reinstall your app after every testcase.

What is 'RESET_BETWEEN_SCENARIOS=1'?

Danger: If you are getting an error that reads `tool 'xcodebuild' requires Xcode, but active developer directory ...`, then open Xcode, and go to *Xcode* > *Preferences* > *Locations* and in the *Command Line Tools* dropdown menu select *Xcode*.
'xcodebuild' error

Danger: If the command dosen't work, try this fixes one at a time:
- Manually run your app on the simulator through Xcode, and then close and stop it.
- Make sure Xcode is not executing any projects at the time.
- Have only one instance of Xcode open, with the project you are trying to run, and the scheme you want to build selected.
- Make sure you have the same version of Calabash in your Terminal tool and the framework you are linking in your project.

Other errors

2.4.4 Running on multiple iOS devices

In order to run your testcases on testmunk's devices and see a report with your test results and screenshots, simply create an account, upload your IPA file and testcases.

Hint: To export the IPA file for your app, open your Xcode project, make sure to select the "...-tm" scheme and "iOS Device" as your target device. Then, in the title bar and go to *Product* > *Archive*. In the *Archives* window that pops up, press the *Distribute...* button, select *Save for Enterprise or Ad Hoc Deployment*, choose the Provisioning Profile you sign your app with, and export the file. Leave the *Save for Enterprise Distribution* checkbox unchecked.

How do I export my IPA file?

2.5 Updating Calabash

For your tests to run in both your machine and the Testmunk servers, you need to have the same version of Calabash as we do (currently it is `0.9.169`). You can achieve this by updating your Calabash installation.

In order for your tests to run, you need to have the same version of Calabash in both the Calabash Ruby gem and the Objective-C framework you link with the ”...-tm” build of your iOS application.

2.5.1 Check the version of Calabash you have installed

Ruby gem version

Run this command on your Terminal:

```
$ calabash-ios version
```

Framework version

Run in the simulator the app for which you want to check the version, and then run this commands in Terminal:

```
$ calabash-ios console # this opens the Calabash console
> server_version['version'] # this queries the Calabash server running in your application for its ve
```

2.5.2 Update process

First, download the updated Ruby gem by running:

```
$ gem install calabash-cucumber
```

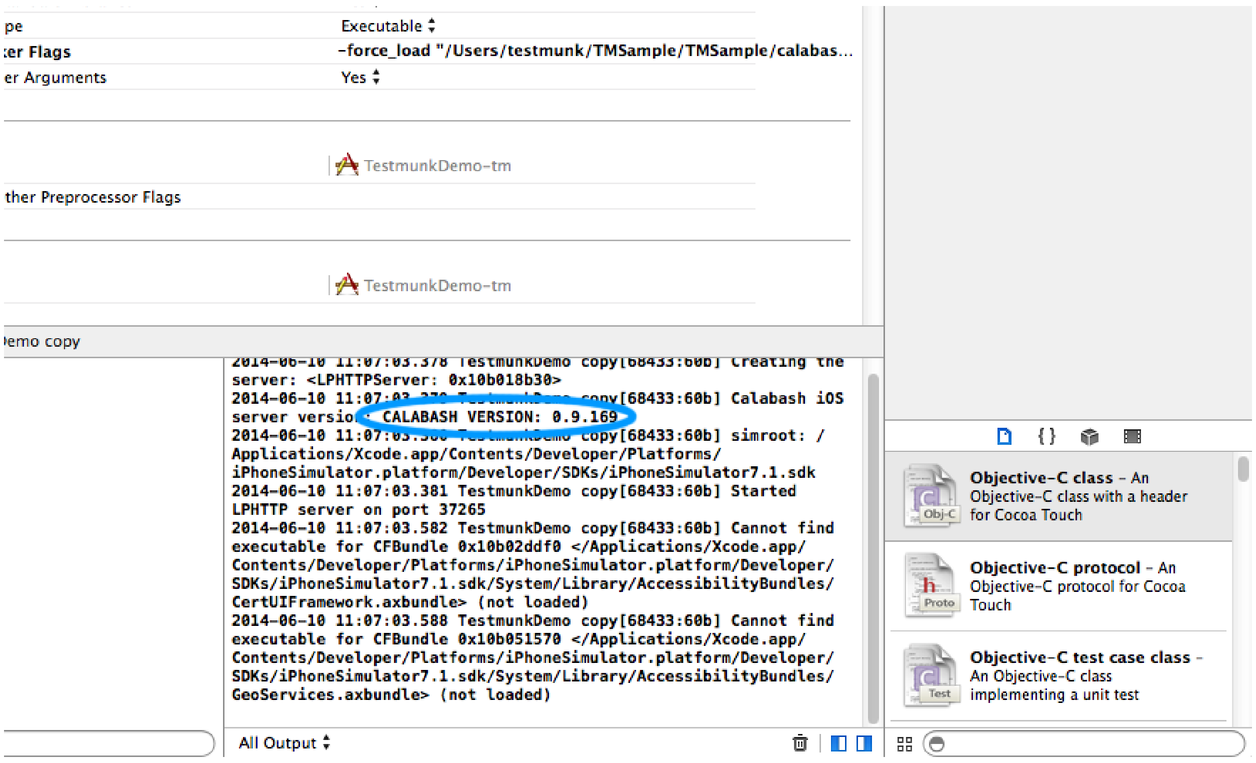
Then, for each Xcode project containing a build target with the Calabash framework, do this:

1. In Terminal, `cd` into the folder that contains the `calabash.framework` file, and run this command:

```
$ calabash-ios download
```

2. Open the project in Xcode.
3. Press Shift + Option + Command + K to clean your Xcode project.
4. Delete the app from any iOS devices or simulators that have it.
5. Rebuild your app.
6. Go through the commands in the “Check the version of Calabash you have installed” section above to make sure your framework was properly updated.

Hint: You can also check the verison of your app’s Calabash framework by looking at the console output in Xcode.



pe	Executable
:er Flags	-force_load "/Users/testmunk/TMSample/TMSample/calabas…
er Arguments	Yes
	TestmunkDemo-tm
ther Preprocessor Flags	
	TestmunkDemo-tm

```
2014-06-10 11:07:03.378 TestmunkDemo copy[68433:60b] Creating the
server: <LPHTTPServer: 0x10b018b30>
2014-06-10 11:07:03.379 TestmunkDemo copy[68433:60b] Calabash iOS
server version: CALABASH VERSION: 0.9.169
2014-06-10 11:07:03.380 TestmunkDemo copy[68433:60b] simroot: /
Applications/Xcode.app/Contents/Developer/Platforms/
iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator7.1.sdk
2014-06-10 11:07:03.381 TestmunkDemo copy[68433:60b] Started
LPHTTP server on port 37265
2014-06-10 11:07:03.582 TestmunkDemo copy[68433:60b] Cannot find
executable for CFBundle 0x10b02ddf0 </Applications/Xcode.app/
Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/
SDKs/iPhoneSimulator7.1.sdk/System/Library/AccessibilityBundles/
CertUIFramework.axbundle> (not loaded)
2014-06-10 11:07:03.588 TestmunkDemo copy[68433:60b] Cannot find
executable for CFBundle 0x10b051570 </Applications/Xcode.app/
Contents/Developer/Platforms/iPhoneSimulator.platform/Developer/
SDKs/iPhoneSimulator7.1.sdk/System/Library/AccessibilityBundles/
GeoServices.axbundle> (not loaded)
```

emo copy
All Output
Objective-C class - An Objective-C class with a header for Cocoa Touch
Objective-C protocol - An Objective-C protocol for Cocoa Touch
Objective-C test case class - An Objective-C class implementing a unit test

Alternative method for checking framework version

CHAPTER 3

Teststep library

3.1 Touching

teststep ios

Touches any element with the accessibility label "label". This is usually the title of the element, or can be set manually in Xcode.

Examples:

```
Then I touch "login"
Then I touch "settings_button"
```

Implementation:

```
Then /^I (?:press|touch) "([^\"]*)"$/ do |name|
  touch("view marked:'#{name}'")
  sleep(STEP_PAUSE)
end
```

Then I touch "accLabel"

teststep ios

Touches the button with the accessibility label "label". This is usually the button [UIButton] title, or can be set manually in Xcode.

Examples:

```
Then I touch the "login" button
```

Related Teststeps:

- Then I touch button number 1
- Then I touch "label"

Implementation:

```
Then /^I (?:press|touch) the "([^\"]*)" button$/ do |name|
  touch("button marked:'#{name}'")
  sleep(STEP_PAUSE)
end
```

Then I touch the “login” button

teststep ios

Touches the first button [UIButton] it can find. If there is no button on that index, it will return an error.

Examples:

```
Then I touch button number 1
```

Implementation:

```
Then /^I (?:press|touch) button number (\d+)$/ do |index|
  index = index.to_i
  screenshot_and_raise "Index should be positive (was: #{index})" if (index<=0)
  touch("button index:#{index-1}")
  sleep(STEP_PAUSE)
end
```

Then I touch button number 1

teststep ios

Touches (and activates) the input field [UITextField] with the label string passed.

Examples:

```
Then I touch the "Email Address" input field
```

Implementation:

```
Then /^I (?:press|touch) the "([^\"]*)" (?:input|text) field$/ do |name|
  placeholder_query = "textField placeholder:'#{name}'"
  marked_query = "textField marked:'#{name}'"
  if !query(placeholder_query).empty?
    touch(placeholder_query)
  elsif !query(marked_query).empty?
    touch(marked_query)
  else
    screenshot_and_raise "could not find text field with placeholder '#{name}' or marked as '#{name}
  end
  sleep(STEP_PAUSE)
end
```

Then I touch the “placeholder” input field

teststep ios

Touches the table cell [UITableViewCell] by number. It only works on visible cells.

Examples:

```
Then I touch list item number 1
```

Implementation:

```
Then /^I (?:press|touch) list item number (\d+)$/ do |index|
   index = index.to_i
   screenshot_and_raise "Index should be positive (was: #{index})" if (index<=0)
   touch("tableViewCell index:#{index-1}")
   sleep(STEP_PAUSE)
end
```

Then I touch list item number 1

---

teststep ios

Toggles the switch (UISwitch) available in the current view. This macro only works if there is one switch in view. [See related for multiple switches]

Examples:

```
Then I touch the switch
```

Related Teststeps:

- Then I toggle the “label” switch

Implementation:

```
Then /^I toggle the switch$/ do
  touch("switch")
  sleep(STEP_PAUSE)
end
```

Then I toggle the switch

---

teststep ios

Toggles the switch which is tagged by the label provided.

Examples:

```
Then I toggle the "Weekly Reminder" switch
```

Implementation:

```
Then /^I toggle the "([^\"]*)" switch$/ do |name|
  touch("switch marked:'#{name}'")
  sleep(STEP_PAUSE)
end
```

Then I toggle the “accLabel” switch

---

teststep ios

Touches the done button on the keyboard.

Examples:

```
Then I touch done
```

Implementation:

```
Then /^I (?:touch|press) (?:done|search)$/ do
  done
  sleep(STEP_PAUSE)
end
```

Then I touch done

---

teststep ios

Touches the user's pin – the blue dot [MKUserLocation].

Examples:

```
Then I touch the user location
```

Implementation:

```
Then /^I touch (?:the)? user location$/ do
  touch("view:'MKUserLocationView'")
  sleep(STEP_PAUSE)
end
```

Then I touch the user location

---

teststep ios

This macro will attempt to touch the screen on the points provided. Please be careful when using this as elements' positions may change on different devices.

Examples:

```
Then I touch on screen 200 from the left and 100 from the top
```

Implementation:

```
Then /^I (?:press|touch) on screen (\d+) from the left and (\d+) from the top$/ do |x, y|
  touch(nil, {:offset => {:x => x.to_i, :y => y.to_i}})
  sleep(STEP_PAUSE)
end
```

Then I touch on screen 100 from the left and 250 from the top

---

teststep android

Taps the button containing the specified text.

Examples:

```
Given I press the "login" button
```

Implementation:

```
Given /^I press the "([^\"]*)" button$/ do |text|
  tap_when_element_exists("android.widget.Button {text CONTAINS[c] '#{text}'}")
end
```

Given I press the “login” button

---

teststep android

Taps the button with the specified index.

Examples:

```
Then I press button number 1
```

Implementation:

```
Then /^I press button number (\d+)$/ do |index|
  tap_when_element_exists("android.widget.Button index:#{index.to_i-1}")
end
```

Then I press button number 1

---

teststep android

Taps the image button with the specified index.

Examples:

```
Then I press image button number 2
```

Implementation:

```
Then /^I press image button number (\d+)$/ do |index|
  tap_when_element_exists("android.widget.ImageButton index:#{index.to_i-1}")
end
```

Then I press image button number 2

---

teststep android

Taps the view with the given ID.

Examples:

```
Then I press view with id "home_button"
```

Implementation:

```
Then /^I press view with id "([^\"]*)"$/ do |id|
  tap_when_element_exists("* id:'#{id}'")
end
```

Then I press view with id “home_button”

---

teststep android

Taps the view marked by the specified identifier.

Examples:

```
Then I press "signup"
```

Implementation:

```
Then /^I press "([^\"]*)"$/ do |identifier|
  tap_when_element_exists("* marked:'#{identifier}'")
end
```

Then I press “signup”

---

teststep android

Taps the specified text.

Examples:

```
Then I touch the "welcome" text
```

Implementation:

```
Then /^I touch the "([^\"]*)" text$/ do |text|
  tap_when_element_exists("* {text CONTAINS[c] '#{text}'}")
end
```

Then I touch the “welcome” text

---

teststep android

Taps the list item with the specified index in the first visible list.

Examples:

```
Then I press list item number 1
```

Implementation:

```
Then /^I press list item number (\d+)$/ do |index|
  tap_when_element_exists("android.widget.ListView index:0 android.widget.TextView index:#{index.to_
end
```

Then I press list item number 1

---

teststep android

Long presses the list item with the specified index in the first visible list.

Examples:

```
Then I long press list item number 1
```

Implementation:

```
Then /^I long press list item number (\d+)$/ do |index|
  long_press_when_element_exists("android.widget.ListView index:0 android.widget.TextView index:#{inc
end
```

Then I long press list item number 1

---

teststep android

Taps the screen at the specified location.

Examples:

```
Then I click on screen 20% from the left and 30% from the top
```

Implementation:

```
Then /^I click on screen (\d+)% from the left and (\d+)% from the top$/ do |x, y|
  perform_action('click_on_screen', x, y)
end
```

Then I click on screen 20% from the left and 30% from the top

---

teststep android

Toggles the checkbox with the specified index.

Examples:

```
Then I toggle checkbox number 1
```

Implementation:

```
Then /^I toggle checkbox number (\d+)$/ do |index|
  tap_when_element_exists("android.widget.CheckBox index:#{index.to_i-1}")
end
```

Then I toggle checkbox number 1

---

teststep android

Long presses the view containing the specified text.

Examples:

```
Then I long press "login"
```

Implementation:

```
Then /^I long press "([^\"]*)"$/ do |text|
  long_press_when_element_exists("* {text CONTAINS[c] '#{text}'}")
end
```

Then I long press “login”

---

teststep android

Long presses the view containing the specified text and then selects an item from the menu that appears.

Examples:

```
Then I long press "signup"
Then I select "item number 1" from the menu
```

Implementation:

```
Then /^I long press "([^\"]*)"$/ do |text|
  long_press_when_element_exists("* {text CONTAINS[c] '#{text}'}")
end

Then /^I select "([^\"]*)" from the menu$/ do |identifier|
  select_options_menu_item(identifier)
end
```

Then I long press "signup" and select item number 1

---

teststep android

Long presses the view containing the specified text and then selects an item from the menu that appears.

Examples:

```
Then I long press "login"
Then I select "welcome" from the menu
```

Implementation:

```
Then /^I long press "([^\"]*)"$/ do |text|
  long_press_when_element_exists("* {text CONTAINS[c] '#{text}'}")
end

Then /^I select "([^\"]*)" from the menu$/ do |identifier|
  select_options_menu_item(identifier)
end
```

Then I long press "login" and select "welcome"

3.2 Assertions

---

teststep ios android

This teststep will check the view for the provided parameter as an accessibility label or text in a UILabel. If calabash is unable to find the label or text, then this teststep fails.

Examples:

```
Then I should see "Welcome"
```

Implementation iOS:

```
Then /^I should see "([^\"]*)"$/ do |expected_mark|
  res = (element_exists( "view marked:'#{expected_mark}'" ) or
 element_exists( "view text:'#{expected_mark}'"))
 if not res
screenshot_and_raise "No element found with mark or text: #{expected_mark}"
  end
end
```

Implementation Android:

```
Then /^I should see "([^\"]*)"$/ do |text|
  wait_for_text(text, timeout: 10)
end
```

Then I should see "text or label"

---

teststep ios android

This is the inverse of the "Then I should see text", this will check all the views to make sure that this particular label is not in the view. If it is, this teststep will fail. It is useful if you want to make sure you have left a certain screen.

Examples:

```
Then I should not see "Logout"
```

Implementation iOS:

```
Then /^I should not see "([^\"]*)"$/ do |expected_mark|
  res = query("view marked:'#{expected_mark}'")
  res.concat query("view text:'#{expected_mark}'")
   unless res.empty?
   screenshot_and_raise "Expected no element with text nor accessibilityLabel: #{expected_mark}, foun
  end
end
```

Implementation Android:

```
Then /^I should not see "([^\"]*)"$/ do |text|
  wait_for_text_to_disappear(text, timeout: 10)
end
```

Related Teststeps:

- Then I should see "text or label"

Then I should not see "text or label"

---

teststep ios

Checks all the views to make sure that the view with the provided accessibility label "view" is available. It will fail if it does not find such a view.

Examples:

```
Then I see the "Logout"
```

Implementation:

```
Then /^I see the "([^\"]*)"$/ do |text|
  macro %Q|I should see "#{text}"|
end
```

Related Teststeps:

- Then I should not see "text or label"
- Then I don't see the "someview"

Then I see the "someview"

---

teststep ios

Checks all the views to make sure that the view with the provided accessibility label "view" is not available. It will fail if it finds such a view.

Examples:

```
Then I don't see the "Logout"
```

Implementation:

```
Then /^I don't see the "([^\"]*)"$/ do |text|
  macro %Q|I should not see "#{text}"|
end
```

Related Teststeps:

- Then I should not see "text or label"

Then I don't see the "someview"

---

teststep ios

Checks the view for the existence of a specified button. If Calabash is unable to find the button, then the teststep fails.

Examples:

```
Then I should see a "login" button
```

Implementation:

```
Then /^I should see a "([^\"]*)" button$/ do |expected_mark|
  check_element_exists("button marked:'#{expected_mark}'")
end
```

Then I should see a "login" button

---

teststep ios

Checks the view for the existence of a specified button. If Calabash is able to find the button, then the teststep fails.

Examples:

```
Then I should not see a "login" button
```

Implementation:

```
Then /^I should not see a "([^\"]*)" button$/ do |expected_mark|
  check_element_does_not_exist("button marked:'#{expected_mark}'")
end
```

Related teststeps:

- Then I should see a "login" button

Then I should not see a “login” button

teststep ios android

Asserts that specified text can be found. If Calabash is not able to find the text, then the teststep fails.

Examples:

```
Then I see the text "Hello"
```

Implementation:

```
Then /^I see the text "([^\"]*)"$/ do |text|
  macro %Q|I should see "#{text}"|
end
```

Then I see the text “some text”

teststep ios android

Asserts that specified text can not be found. If Calabash is able to find the text, then the teststep fails.

Examples:

```
Then I don't see the text "Hello"
```

Implementation:

```
Then /^I don't see the text "([^\"]*)"$/ do |text|
  macro %Q|I should not see "#{text}"|
end
```

Then I don’t see the text “some text”

teststep ios

Looks for a label [UILabel] with text which contains the provided prefix

Examples:

```
Then I should see text starting with "Welcome"
```

Implementation:

```
Then /^I (?:should)? see text starting with "([^\"]*)"$/ do |text|
  res = query("view {text BEGINSWITH '#{text}'}").empty?
  if res
        screenshot_and_raise "No text found starting with: #{text}"
  end
end
```

Related Teststeps:

- Then I should see text ending with “suffix”
- Then I should see text containing “sub text”

Then I should see text starting with "prefix"

teststep ios android

Will look for a label [UILabel] which contains the text provided

Examples:

```
Then I should see text containing "available"
```

Implementation:

```
Then /^I (?:should)? see text containing "([^\"]*)"$/ do |text|
  res = query("view {text LIKE '*#{text}*'}").empty?
  if res
screenshot_and_raise "No text found containing: #{text}"
  end
end
```

Related Teststeps:

- Then I should see text ending with "suffix"
- Then I should see text starting with "prefix"

Then I should see text containing "sub text"

teststep ios

Checks all labels [UILabel] for text that ends with the provided suffix

Examples:

```
Then I should see text ending with "suffix"
```

Implementation:

```
Then /^I (?:should)? see text ending with "([^\"]*)"$/ do |text|
  res = query("view {text ENDSWITH '#{text}'}").empty?
  if res
screenshot_and_raise "No text found ending with: #{text}"
  end
end
```

Related Teststeps:

- Then I should see text containing "sub text"
- Then I should see text starting with "prefix"

Then I should see text ending with "suffix"

teststep ios

Checks to see if the view contains 2 input fields, the input fields can be replaced with buttons, or other elements.

Examples:

```
Then I see 2 buttons
```

Implementation:

```
Then /^I see (\d+) (?:input|text) field(?:s)?$/ do |count|
  count = count.to_i
  cnt = query(:textField).count
  if cnt < count
screenshot_and_raise "Expected at least #{count} text/input fields, found #{cnt}"
  end
end
```

Then I see 2 input fields

---

teststep ios

Checks to see if the view contains an input field with the specified placeholder or accessibilityLabel.

Examples:

```
Then I should see a "Username" input field
```

Implementation:

```
Then /^I should see a "([^\"]*)" (?:input|text) field$/ do |expected_mark|
  res = element_exists("textField placeholder:'#{expected_mark}'") ||
        element_exists("textField marked:'#{expected_mark}'")
  unless res
        screenshot_and_raise "Expected textfield with placeholder or accessibilityLabel: #{expected_m
  end
end
```

Related Teststeps:

- Then I should not see a “Username” input field

Then I should see a “Username” input field

---

teststep ios

Checks to see if the view does not contain an input field with the specified placeholder or accessibilityLabel.

Examples:

```
Then I should not see a "Username" input field
```

Implementation:

```
Then /^I should not see a "([^\"]*)" (?:input|text) field$/ do |expected_mark|
  res = query("textField placeholder:'#{expected_mark}'")
  res.concat query("textField marked:'#{expected_mark}'")
  unless res.empty?
screenshot_and_raise "Expected no textfield with placeholder nor accessibilityLabel: #{expected_mark
  end
 end
```

Related Teststeps:

- Then I should see a “Username” input field

Then I should not see a "Username" input field

---

teststep ios

Checks the views to see if there is a user location (blue dot) [MKUserLocation] inside a map [UIMapView].

Examples:

```
Then I should see the user location
```

Implementation:

```
Then /^I should see (?:the)? user location$/ do
  check_element_exists("view:'MKUserLocationView'")
end
```

Then I should see the user location

---

teststep ios

This step checks if a mapview is on the screen.

Examples:

```
Then I should see a map
```

Implementation

```
Then /^I should see a map$/ do
  check_element_exists("view:'MKMapView'")
end
```

Then I should see a map

---

teststep android

Asserts that specified text can not be found. If Calabash is able to find the text, then the teststep fails.

Examples:

```
Then I don't see "Hello"
```

Implementation:

```
Then /^I don't see "([^\"]*)"$/ do |text|
  wait_for_text_to_disappear(text, timeout: 10)
end
```

Then I don't see "text"

3.3 Input

---

teststep ios

Enters “text” into the input / text field [UITextField] which has the placeholder text set as “label”

Examples:

```
Then I enter "user@testmunk.com" into the "Email Address" input field
```

Implementation:

```
Then /^I enter "([^\"]*)" into the "([^\"]*)" field$/ do |text_to_type, field_name|
  touch("textField marked:'#{field_name}'")
  wait_for_keyboard()
  keyboard_enter_text text_to_type
  sleep(STEP_PAUSE)
end
```

Then I enter “text to write” into the “placeholder” input field

---

teststep ios android

Enters “text” into the relevant input / text field [UITextField]. If there are several input fields you will need to check which input field number is correct.

Examples:

```
Then I enter "First name" into input field number 1
```

Implementation iOS:

```
Then /^I enter "([^\"]*)" into (?:input|text) field number (\d+)$/ do |text, index|
  index = index.to_i
  screenshot_and_raise "Index should be positive (was: #{index})" if (index<=0)
  touch("textField index:#{index-1}")
  wait_for_keyboard()
  keyboard_enter_text text
  sleep(STEP_PAUSE)
end
```

Implementation Android:

```
Then /^I enter "([^\"]*)" into input field number (\d+)$/ do |text, index|
  enter_text("android.widget.EditText index:#{index.to_i-1}", text)
end
```

Then I enter “text” into input field number 1

---

teststep ios android

Clears the text field [UITextField][UITextView].

Examples:

```
Then I clear "Email Address"
```

Implementation iOS:

```
When /^I clear "([^\"]*)"$/ do |name|
  msg = "When I clear <name>' will be deprecated because it is ambiguous - what should be cleared?"
  _deprecated('0.9.151', msg, :warn)
  clear_text("textField marked:'#{name}'")
end
```

Implementation Android:

```
Then /^I clear input field with id "([^\"]*)"$/ do |id|
  clear_text("android.widget.EditText id:'#{id}'")
end
```

Then I clear “placeholder”

---

teststep ios android

Clears the text from the specified input / text field [UITextField]. If there are several input fields you will need to check which input field number is correct.

Examples:

```
Then I clear input field number 1
```

Implementation iOS:

```
Then /^I clear (?:input|text) field number (\d+)$/ do |index|
  index = index.to_i
  screenshot_and_raise "Index should be positive (was: #{index})" if (index<=0)
  clear_text("textField index:#{index-1}")
end
```

Implementation Android:

```
Then /^I clear input field with id "([^\"]*)"$/ do |id|
  clear_text("android.widget.EditText id:'#{id}'")
end
```

Then I clear input field number 1

---

teststep android

Finds the timepicker with the specified index and changes the time.

Examples:

```
Given I set the time to "14:00" on TimePicker with index "5"
```

Implementation:

```
Given /^I set the time to "(\d\d:\d\d)" on TimePicker with index ([^\"]*)$/ do |time, index|
  set_time("android.widget.TimePicker index:#{index.to_i-1}", time)
end
```

Given I set the time to “14:00” on TimePicker with index “5”

---

teststep android

Finds the timepicker with the specified index and changes the time.

Examples:

```
Given I set the "timePickerLabel" time to "14:00"
```

Implementation:

```
Given /^I set the "([^\"]*)" time to "(\d\d:\d\d)"$/ do |content_description, time|
  set_time("android.widget.TimePicker {contentDescription LIKE[c] '#{content_description}'}", time)
end
```

Given I set the “timePickerLabel” time to “14:00”

---

teststep android

Finds the datepicker by content description and changes the date.

Examples:

```
Given I set the "what_is_the_date" date to "31-12-1999"
```

Implementation:

```
Given /^I set the "([^\"]*)" date to "(\d\d-\d\d-\d\d\d\d)"$/ do |content_description, date|
  set_date("android.widget.DatePicker {contentDescription LIKE[c] '#{content_description}'}", date)
end
```

Given I set the “datePickerLabel” date to “31-12-1999”

---

teststep android

Enters the specified text into the input field with the specified id.

Examples:

```
Then I enter text "Hello" into field with id "type_here"
```

Implementation:

```
Then /^I enter text "([^\"]*)" into field with id "([^\"]*)"$/ do |text, id|
  enter_text("android.widget.EditText id:'#{id}'", text)
end
```

Then I enter text “text” into field with id “fieldId”

---

teststep android

Enters the specified text into the input field that has the specified content description.

Examples:

```
Then I enter "Hello" as "text"
```

Implementation:

```
Then /^I enter "([^\"]*)" as "([^\"]*)"$/ do |text, content_description|
  enter_text("android.widget.EditText {contentDescription LIKE[c] '#{content_description}'}", text)
end
```

Then I enter “text” as “fieldId”

---

teststep android

Enters the specified text into the input field that has the specified content description.

Examples:

```
Then I enter "Hello" into "type_here"
```

Implementation:

```
Then /^I enter "([^\"]*)" into "([^\"]*)"$/ do |text, content_description|
  enter_text("android.widget.EditText {contentDescription LIKE[c] '#{content_description}'}", text)
end
```

Then I enter “text” into “fieldId”

---

teststep android

Clears the text of the input field with the specified id.

Examples:

```
Then I clear input field with id "type_here"
```

Implementation:

```
Then /^I clear input field with id "([^\"]*)"$/ do |id|
  clear_text("android.widget.EditText id:'#{id}'")
end
```

Then I clear input field with id “fieldId”

---

teststep android

Finds the spinner marked by the specified ‘spinner_identifier’ or has a childview marked by the specified ‘spinner_identifier’. It then selects the menu item marked by the specified ‘item_identifier’.

Examples:

```
Then I select "Hello" from "spinner"
```

Implementation:

```
Then /^I select "([^\"]*)" from "([^\"]*)"$/ do |item_identifier, spinner_identifier|
  spinner = query("android.widget.Spinner marked:'#{spinner_identifier}'")

  if spinner.empty?
    tap_when_element_exists("android.widget.Spinner * marked:'#{spinner_identifier}'")
  else
    touch(spinner)
  end

  tap_when_element_exists("android.widget.PopupWindow$PopupViewContainer * marked:'#{item_identifier
end
```

Then I select “item text” from “spinnerLabel”

3.4 Waiting

teststep ios

Makes the testrun wait until the label [UILabel] with the text appears, or any other element eg. button [UIButton] appears.

Examples:

```
Then I wait to see "Welcome"
Then I wait to see "Please log in:"
```

Implementation:

```
Then /^I wait to see "([^\"]*)"$/ do |expected_mark|
  wait_for(WAIT_TIMEOUT) { view_with_mark_exists( expected_mark ) }
end
```

Related Teststeps:

- Then I wait for “text or label” to appear

Then I wait to see “text or label”

teststep ios

Waits until the label [UILabel] with the text appears, or any other element eg. button [UIButton] appears.

Examples:

```
Then I wait for "Welcome" to appear
```

Implementation:

```
Then /^I wait for "([^\"]*)" to appear$/ do |name|
  macro %Q|I wait to see "#{name}"|
end
```

Related Teststeps:

- Then I wait to see “text or label”

Then I wait for “text or label” to appear

teststep ios

This will wait until an element with the label or text provided has disappeared.

Examples:

```
Then I wait until I don't see "loading..."
```

Implementation:

```
Then /^I wait until I don't see "([^\"]*)"$/ do |expected_mark|
  sleep 1## wait for previous screen to disappear
  wait_for(WAIT_TIMEOUT) { not element_exists( "view marked:'#{expected_mark}'" )}
end
```

Related Teststeps:

- Then I wait to not see "text or label"

Then I wait until I don't see "text or label"

---

teststep ios

Waits until an element with the label or text provided has disappeared.

Examples:

```
Then I wait to not see "loading..."
```

Implementation:

```
Then /^I wait to not see "([^\"]*)"$/ do |expected_mark|
  macro %Q|I wait until I don't see "#{expected_mark}"|
end
```

Related Teststeps:

- Then I wait until I don't see "text or label"

Then I wait to not see "text or label"

---

teststep ios

Waits for a button with the specified accessibility label to apprear.

Examples:

```
Then I wait for the "login" button to appear
```

Implementation:

```
Then /^I wait for the "([^\"]*)" button to appear$/ do |name|
  wait_for(WAIT_TIMEOUT) { element_exists( "button marked:'#{name}'" ) }
end
```

Then I wait for the "login" button to appear

---

teststep ios

Waits until the title in the navgation bar [UINavigationBar] changes to the provided text (ie. when the view changes), or the timeout occurs.

Examples:

```
Then I wait to see a navigation bar titled "Welcome"
Then I wait to see a navigation bar titled "Login"
```

Implementation:

```
Then /^I wait to see a navigation bar titled "([^\"]*)"$/ do |expected_mark|
  msg = "waited for '#{WAIT_TIMEOUT}' seconds but did not see the navbar with title '#{expected_mark
  wait_for(:timeout => WAIT_TIMEOUT,
                   :timeout_message => msg ) do
        all_items = query("navigationItemView marked:'#{expected_mark}'")
        button_items = query("navigationItemButtonView")
        non_button_items = all_items.delete_if { |item| button_items.include?(item) }
        !non_button_items.empty?
  end
end
```

Then I wait to see a navigation bar titled "title"

---

teststep ios

Waits until the specified input field appears.

Examples:

```
Then I wait for the "Username" input field
```

Implementation:

```
Then /^I wait for the "([^\"]*)" (?:input|text) field$/ do |placeholder_or_view_mark|
  wait_for(WAIT_TIMEOUT) {
    element_exists( "textField placeholder:'#{placeholder_or_view_mark}'") ||
        element_exists( "textField marked:'#{placeholder_or_view_mark}'")
  }
end
```

Then I wait for the "label" input field

---

teststep ios

Waits until the relevant number of textfields are loaded.

Examples:

```
Then I wait for 2 input fields
```

Implementation:

```
Then /^I wait for (\d+) (?:input|text) field(?:s)?$/ do |count|
  count = count.to_i
  wait_for(WAIT_TIMEOUT) { query(:textField).count >= count   }
end
```

Then I wait for 2 input fields

---

teststep ios

Waits for X seconds

Examples:

```
Then I wait for 1 second
Then I wait for 2 seconds
Then I wait for 2.4 seconds
```

Implementation:

```
Then /^I wait for ([\d\.]+) second(?:s)?$/ do |num_seconds|
  num_seconds = num_seconds.to_f
  sleep num_seconds
end
```

Then I wait for X seconds

---

teststep ios android

Waits for 2 seconds.

Examples:

```
Then I wait
```

Implementation:

```
Then /^I wait$/ do
  sleep 2
end
```

Then I wait

---

teststep android

Waits until there are no more progress bars.

Examples:

```
Then I wait for progress
```

Implementation:

```
Then /^I wait for progress$/ do
  wait_for_element_does_not_exist("android.widget.ProgressBar")
end
```

Then I wait for progress

---

teststep android

Description coming soon!

Then I wait for dialog to close

---

teststep android

This teststep will make the testrun wait until the label [UILabel] with the text appears, or any other element eg. button [UIButton] appears.

Examples:

```
Then I wait to see "Welcome"
Then I wait to see "Please log in:"
```

Implementation:

```
Then /^I wait to see "([^\"]*)"$/ do |text|
  wait_for_text(text)
end
```

Then I wait to see "text or label"

---

teststep android

Waits until the label [UILabel] with the text appears, or any other element eg. button [UIButton] appears.

Examples:

```
Then I wait for "Hello" to appear
```

Implementation:

```
Then /^I wait for "([^\"]*)" to appear$/ do |text|
  wait_for_text(text)
end
```

Then I wait for "text or label" to appear

---

teststep android

Waits up to 5 seconds for the specified text, or any other element e.g. [UIButton], to appear.

Examples:

```
Then I wait up to 5 seconds for "Click me" to appear
```

Implementation:

```
Then /^I wait up to (\d+) seconds for "([^\"]*)" to appear$/ do |timeout, text|
  wait_for_text(text, timeout: timeout.to_i)
end
```

Related Teststeps:

- Then I wait up to 5 seconds to see "text or label"

Then I wait up to 5 seconds for "text or label" to appear

---

teststep android

Waits up to 5 seconds for the specified text, or any other element e.g. [UIButton], to appear.

Examples:

```
Then I wait up to 5 seconds to see "Click me"
```

Implementation:

```
Then /^I wait up to (\d+) seconds to see "([^\"]*)"$/ do |timeout, text|
  wait_for_text(text, timeout: timeout.to_i)
end
```

Related Teststeps:

- Then I wait up to 5 seconds for “text or label” to appear

Then I wait up to 5 seconds to see “text or label”

---

teststep android

Waits for a button with the specified accessibility label to apprear.

Examples:

```
Then I wait for the "login" button to appear
```

Implementation:

```
Then /^I wait for the "([^\"]*)" button to appear$/ do |identifier|
  wait_for_element_exists("android.widget.Button marked:'#{identifier}'");
end
```

Then I wait for the “id” button to appear

---

teststep android

Waits for a screen with the specified id to apprear.

Examples:

```
Then I wait for the "home" screen to appear
```

Implementation:

```
Then /^I wait for the "([^\"]*)" screen to appear$/ do |activity_name|
  wait_for_activity(activity_name)
end
```

Then I wait for the “id” screen to appear

---

teststep android

Waits for the view with the specified viewID to appear

Examples:

```
Then I wait for the view with id "checkout" to appear
```

Implementation:

```
Then /^I wait for the view with id "([^\"]*)" to appear$/ do |id|
  wait_for_element_exists("* id:'#{id}'")
end
```

Then I wait for the view with id "viewId" to appear

---

teststep android

Waits up to 5 seconds for the screen with the specified id to appear (the test will move on if id is found before 5 seconds is up).

Examples:

```
Then I wait up to 5 seconds for the "checkout" screen to appear
```

Implementation:

```
Then /^I wait upto (\d+) seconds for the "([^\"]*)" screen to appear$/ do |timeout, activity_name|
  wait_for_activity(activity_name, timeout: timeout.to_i)
end
```

Related Teststeps:

- Then I wait for the view with id "viewId" to appear

Then I wait up to 5 seconds for the "id" screen to appear

---

teststep android

Waits for 1 second.

Examples:

```
Then I wait for a second
```

Implementation:

```
Then /^I wait for 1 second$/ do
  sleep 1
end
```

Related Teststeps:

- Then I wait for 5 seconds

Then I wait for a second

---

teststep android

Waits for X seconds.

Examples:

```
Then I wait for 1 second
Then I wait for 2 seconds
Then I wait for 2.4 seconds
```

Implementation:

```
Then /^I wait for (\d+) seconds$/ do |seconds|
  sleep(seconds.to_i)
end
```

Related Teststeps:

- Then I wait for a second

Then I wait for X seconds

3.5 Buttons

teststep ios android

Simulates the user pressing the back button

Examples:

```
Then I go back
```

Implementation iOS:

```
Then /^I go back$/ do
  touch("navigationItemButtonView first")
  sleep(STEP_PAUSE)
end
```

Implementation Android:

```
Then /^I go back$/ do
  press_back_button
end
```

Then I go back

teststep android

Simulates the user pressing the menu key

Examples:

```
Then I press the menu key
```

Implementation:

```
Then /^I press the menu key$/ do
  press_menu_button
end
```

Then I press the menu key

teststep android

Simulates the user pressing the enter button

Examples:

```
Then I press the enter button
```

Implementation:

```
Then /^I press the enter button$/ do
  perform_action('send_key_enter')
end
```

Then I press the enter button

3.6 Gestures

teststep ios android

Performs a swipe gesture arbitrarily on the screen.

Examples:

```
Then I swipe left
```

Implementation iOS:

```
Then /^I swipe (left|right|up|down)$/ do |dir|
  swipe(dir)
  sleep(STEP_PAUSE)
end
```

Implementation Android:

```
Then /^I swipe left$/ do
  perform_action('swipe', 'left')
end
```

Options:

You can use left, right up or down as parameters.

Then I swipe left/right/up/down

teststep ios

Swipes a scroll view by index/number (and offset), or accessibilityLabel.

Examples:

```
Then I swipe left on number 2
```

Implementation:

```
Then /^I swipe (left|right|up|down) on number (\d+)$/ do |dir, index|
  index = index.to_i
  screenshot_and_raise "Index should be positive (was: #{index})" if (index<=0)
  swipe(dir, {:query => "scrollView index:#{index-1}"})
  sleep(STEP_PAUSE)
end
```

Then I swipe left on number 2

teststep ios

Swipes a scroll view by index/number (and offset), or accessibilityLabel at a specified set of coordinates. Note that the coordinate system is for the element.

Examples:

```
Then I swipe left on number 2 at x 20 and y 10
```

Implementation:

```
Then /^I swipe (left|right|up|down) on number (\d+) at x (\d+) and y (\d+)$/ do |dir, index, x, y|
  index = index.to_i
  screenshot_and_raise "Index should be positive (was: #{index})" if (index<=0)
  swipe(dir, {:offset => {:x => x.to_i, :y => y.to_i}, :query => "scrollView index:#{index-1}"})
  sleep(STEP_PAUSE)
end
```

Then I swipe left on number 2 at x 20 and y 10

teststep ios

Swipes in the direction given, on the object which contains the mentioned accessibility label.

Examples:

```
Then I swipe right on "Morocco"
```

Implementation:

```
      Then /^I swipe (left|right|up|down) on "([^\"]*)"$/ do |dir, mark|
swipe(dir, {:query => "view marked:'#{mark}'"})
sleep(STEP_PAUSE)
      end
```

Options:

Direction can be left, right, up and down

Related Teststeps:

- Then I swipe left/right

Then I swipe left/right/up/down on “accLabel”

teststep ios

Swipes a specified table cell by number

Examples:

```
Then I swipe on cell number 2
```

Implementation:

```
Then /^I swipe on cell number (\d+)$/ do |index|
  index = index.to_i
  screenshot_and_raise "Index should be positive (was: #{index})" if (index<=0)
  cell_swipe({:query => "tableViewCell index:#{index-1}"})
  sleep(STEP_PAUSE)
end
```

Then I swipe on cell number 2

---

teststep ios

Performs a pinch gesture on the screen.

Examples:

```
Then I pinch to zoom in
Then I pinch to zoom out
```

Implementation:

```
Then /^I pinch to zoom (in|out)$/ do |in_out|
  pinch(in_out)
  sleep(STEP_PAUSE)
end
```

Options:

Parameter (zoom in) can also be zoom out

Then I pinch to zoom in

---

teststep ios

Performs a pinch gesture on the specified element.

Examples:

```
Then I pinch to zoom in on "image"
```

Implementation:

```
Then /^I pinch to zoom (in|out) on "([^\"]*)"$/ do |in_out, name|
  pinch(in_out,{:query => "view marked:'#{name}'"})
  sleep(STEP_PAUSE)
end
```

Options:

Parameter (zoom in) can also be zoom out

Then I pinch to zoom in on “accLabel”

---

teststep ios

Attempts to scroll on the specified accessibility label.

Examples:

```
Then I scroll down
Then I scroll up
```

Implementation:

```
Then /^I scroll (left|right|up|down) on "([^\"]*)"$/ do |dir,name|
  scroll("view marked:'#{name}'", dir)
  sleep(STEP_PAUSE)
end
```

Options:

The last parameter (down) can also be up, left and right.

Then I scroll down on "accLabel"

---

teststep ios android

Attempts to arbitrarily scroll down on the view.

Examples:

```
Then I scroll down
Then I scroll up
```

Implementation iOS:

```
Then /^I scroll (left|right|up|down)$/ do |dir|
  scroll("scrollView index:0", dir)
  sleep(STEP_PAUSE)
end
```

Implementation Android:

```
Then /^I scroll down$/ do
  scroll_down
end
```

Options:

The parameter (down) can also be up, left or right.

Then I scroll down

---

teststep android

Selects the option with the specified id from the menu.

Examples:

```
Then I select "green" from the menu
```

Implementation:

```
Then /^I select "([^\"]*)" from the menu$/ do |identifier|
  select_options_menu_item(identifier)
end
```

Then I select "id" from the menu

---

teststep android

Drags from one point on the screen to another. Note the number of steps is a parameter that defines how many steps are in the swipe between the specified coordinates.

Examples:

```
Then I drag from 50:100 to 50:250 moving with 20 steps
```

Implementation:

```
Then /^I drag from (\d+):(\d+) to (\d+):(\d+) moving with (\d+) steps$/ do |from_x, from_y, to_x, to_
  perform_action('drag', from_x, to_x, from_y, to_y, steps)
end
```

Then I drag from 50:100 to 50:250 moving with 20 steps

REST API

4.1 API Overview

4.1.1 Introduction

The testmunk API provides a RESTful interface (adhering to REST architectural constraints) for your data on testmunk. It is the starting point for anyone who would like to integrate testmunk into another service, for example Continuous Integration Servers such as *Jenkins* or Travis.

You can use the API to interact with:

- Apps
- *List current apps for your organisation*
- *Create a new App*
- Devices
- *Get available devices*
- Testruns
- *Create a new testrun*
- *Start an existing testrun*
- *Get testrun status*
- *Get list of testruns*

4.1.2 Schema

All API access is over HTTPS and accessed from the *api.testmunk.com* domain. All API responses are provided in JSON format. API requests need to be provided with the Content-Type *application/json*. The only exception is when you upload an app.

Here the Content-Type is *multipart/form-data* due to the fact that we receive large files.

Timestamps are returned in ISO 8601 format:

```
YYYY-MM-DDTHH:MM:SSZ
```

4.1.3 Authentication

All API requests are authenticated using your Testmunk API key. You can get the key of your organisation by:

1. Going to your Testmunk Dashboard, and
2. Clicking on the user icon in the navigation bar on the top right > `Account Settings` > `Apps`.

We follow the typical HTTP Basic authentication scheme of providing a Base64 encoded hash of your authentication credentials in the HTTP request header. The credentials are simply your API key.

Apply Base64 strict encoding to your API key and include an HTTP header in your request using the following format:

```
Authorization: Basic YOUR_BASE64_ENCODED_API_KEY.
```

If you are using `curl` you can rest easy – it takes care of the right encoding for you.

```
$ curl https://AQS0LCTvCv6mTwod5PwtU2i1JVY2J6rW@api.testmunk.com/apps/Testmunk/testruns
```

Hint: In this example, the key is AQS0LCTvCv6mTwod5PwtU2i1JVY2J6rW and is specified between the http protocol and the domain.

API key

4.1.4 Versioning

All API requests are subject to versioning. It is strongly advised that you specify a version when requesting resources through the API. This is done by supplying an HTTP `Accept` header with the appropriate version. In case you do not specify a version or you specify an invalid version, we will use the latest API version (currently `v1`).

The versioning format is `application/vnd.testmunk.v1+json` where `v1` is the version identifier.

```
$ curl -H 'Accept: application/vnd.testmunk.v1+json' \
       'https://AQS0LCTvCv6mTwod5PwtU2i1JVY2J6rW@api.testmunk.com/apps/Testmunk/testruns'
```

4.1.5 Errors

If your request results in an error, the API will respond with an HTTP status code in the 4xx class or 5xx class, depending on the cause. The body of the response will contain a JSON formatted error message using the following schema:

```
{
  "message": "A human readable explanation of the problem",
  "code": "A unique string identifying the problem e.g. ValidationFailed",
  "errors": [ // Optional - only on ValidationFailed
    {
      "field" : "The property which was invalid",
      "resource" : "The name of the invalid resource",
      "code" : "A unique string identifying what is wrong with the field"
    }
  ]
}
```

4.1.6 Error codes

You can decide how to handle errors in your code based on the HTTP status code. The status codes we respond in case of an error and what they mean:

- `400 Bad Request`: One of your inputs was incorrectly encoded and could not be processed.
- `401 Unauthorized`: You need to provide authentication credentials, or your credentials were rejected.
- `422 Record is invalid`: One of the values you supplied for an attribute did not pass validation. The error object tells you more details about it. These are the possible validation error codes:
  - `MissingField`: The required field on a resource has not been set.
  - `Invalid`: The formatting of a field is invalid. The documentation for that resource should be able to give you more specific information.
  - `NotExist`: The resource does not exist.
  - `AlreadyExist`: Another resource has the same value as this field. This can happen in resources that must have some unique key (such as App names).
- `500 Internal Server Error`: We messed up somewhere. We've been notified of the issue, and our engineering team will look into it.

4.1.7 Email notifications

Results of your testruns will be sent as email notifications. You can specify the recipients within the notifications tab under your Account Settings on the Testmunk Dashboard.

4.2 App API

4.2.1 List current apps for your organisation

```
GET /apps
```

Curl example

```
curl -X GET \
  -H 'Accept: application/vnd.testmunk.v1+json' \
  'https://AQS0LCTvCv6mTwod5PwtU2i1JVY2J6rW@api.testmunk.com/apps'
```

Output

```
[
    {
        "id": "547f90d9a0eed17d87987355",
        "createdAt": "2014-12-03T22:38:17Z",
        "organisationId": "531df352a4b0c9d6f7b7bdfa",
        "name": "IOS-project"
    },
    {
        "id": "54b5a1d4e4b0ed04cd79f654",
        "createdAt": "2015-01-13T22:53:08Z",
        "organisationId": "531df352a4b0c9d6f7b7bdfa",
        "name": "Android-project"
    }
]
```

4.2.2 Create a new App

Creates a new app based on the provided name.

```
POST /apps
```

Curl example

```
curl -X POST \
    -H 'Accept: application/vnd.testmunk.v1+json' \
    -H 'Content-Type: application/json' \
    -d '{"appName":"My-new-project"}' \
    "http://AQS0LCTvCv6mTwod5PwtU2i1JVY2J6rW@api.testmunk.com/api/apps"
```

Input

- `appName` (Required): The new name for your app, has to be unique.

Output

The results come in pairs of `[device name, OS version]`:

```
{
    "id":"54c427a8e4b0dee6ac5d89r4",
    "createdAt":"2015-01-24T23:15:52Z",
    "organisationId":"531pf381e7b0z9d6f7b7bdfb",
    "name":"My-new-project"
}
```

4.3 Devices API

4.3.1 Get available devices

Will return all available devices for your organisation, in JSON format.

```
GET /devices
```

Curl example

```
curl -X GET \
  -H 'Accept: application/vnd.testmunk.v1+json' \
  'https://AQS0LCTvCv6mTwod5PwtU2i1JVY2J6rW@api.testmunk.com/devices?platform=ios'
```

Input

- `platform` (Optional): Either `ios` or `android`.

Output

The results come in pairs of `[device name, OS version]`:

```
[["ipod-5-A","7.1"],["iphone-4s-A","7.1"],["ipad-3-B","8.1"],["iphone-6-A","8.1"]]
```

4.4 Testruns API

4.4.1 Create a new testrun

Creates a new testrun based on an `.ipa` or `.apk` file. The testrun is automatically started if you set the parameter `autoStart=true`. Request data needs to be sent as `multipart/form-data`.

```
POST /apps/:appName/testruns
```

Curl example

```
  $ curl \
    -H 'Accept: application/vnd.testmunk.v1+json' \
    -F 'file=@iphone.ipa' \
-F 'testcases=@features.zip' \
    -F 'email=hello@testmunk.com' \
    -F 'autoStart=true' \
    -F 'public=true' \
    -F 'devices=ipod-5-A,iphone-4s-A,iphone-6-A' \
    'https://AQS0LCTvCv6mTwod5PwtU2i1JVY2J6rW@api.testmunk.com/apps/Testmunk/testruns'
```

Input

- `appName` (Required): Name of your Testmunk app.
- `file` (Required): iOS or apk app file. Only the format .ipa and .apk allowed.
- `testcases` (Required): Zip file containing the features folder. Zip file should contain the zipped features folder, as you would upload to our website.
- `email` (Required): An email address that is associated with your testmunk account and API key. This can either be your primary email that you registered on testmunk or a team member you invited to the account.
- `testrunName` (Optional): Name of the new testrun. If not specified, the name will get auto-generated, e.g. ‘Testrun 10’
- `autoStart` (Optional): true starts the testrun after upload.
- `public` (Optional): All testruns URLs will automatically be public and can be shared with non testmunk users. Email notifications will also include the public link.
- `devices` (Optional): A comma separated list of `device names`. You can get the device names from the *Devices API* endpoint. **You only need to set the device names, not the OS version**.

Response

```
Status: 201 created
```

```
{
    "id":"52299330e4b07118a7c2cad8",
    "name":"Testrun 10",
    "app":"Testmunk",
    "status":"NotStarted",
    "counts":{
        "numSuccess":0,
        "numFailed":0,
        "numSkipped":0
    },
    "createdAt":"2015-02-07T00:43:17Z",
    "platform":"iOS",
    "devices":[
        "ipod-5-A,iphone-4s-A,iphone-6-A"
    ],
    "testcases":1,
    "stoppedByUser":false
}
```

4.4.2 Selecting Devices to Test On

To select devices to test on, go to testmunk.com and navigate to `Account Settings` > `REST API`. Or you can also set the devices the moment you create a testrun using the *Create a new testrun* endpoint.

4.4.3 Start an existing testrun

Starts an existing testrun based on the testrunId. The testrun need to have the status `NotStarted` (setting autoStart=false when creating the testrun).

```
POST /testruns/:testrunId/run
```

Curl example

```
$ curl \
  -X POST \
  -H 'Accept: application/vnd.testmunk.v1+json' \
  -H 'Content-Type: application/json' \
  -d '{"email": "hello@testmunk.com"}' \
  'https://AQS0LCTvCv6mTwod5PwtU2i1JVY2J6rW@api.testmunk.com/testruns/52299330e4b07118a7c2cad8/run'
```

Input

- `testrunId` (Required).
- `email` (Required): An email address that is associated with your testmunk account and API key. This can either be your primary email that you registered on testmunk or a team member you invited to the account.

```
{
  "email": "markus@testmunk.com"
}
```

Response

```
Status: 200 Ok
```

```
{
    "id":"52299330e4b07118a7c2cad8",
    "name":"Testrun 10",
    "app":"Testmunk",
    "status":"Waiting",
    "counts":{
        "numSuccess":0,
        "numFailed":0,
        "numSkipped":0
    },
    "createdAt":"2015-02-07T00:43:17Z",
    "platform":"iOS",
    "devices":[
        "ipod-5-A,iphone-4s-A,iphone-6-A"
    ],
    "testcases":1,
    "stoppedByUser":false
}
```

4.4.4 Get testrun status

Returns information about a testrun with the specified ID, if it exists. Useful to get the status of your testrun (failed, success)

```
GET /apps/:appName/testruns/:testrunId
```

Curl example

```
$ curl \
  -X GET \
  -H 'Accept: application/vnd.testmunk.v1+json' \
  'https://AQS0LCTvCv6mTwod5PwtU2i1JVY2J6rW@api.testmunk.com/apps/Testmunk/testruns/54d54fe03004286c
```

Input

- `testrunId` (Required).
- `appName` (Required): Name of your Testmunk app.

Response

```
Status: 200 Ok
```

```
{
    "id":"54d54fe03004286c71cb99e0",
    "name":"Testrun 100",
    "app":"Testmunk",
    "status":"Success",
    "counts":{
        "numSuccess":1,
        "numFailed":0,
        "numSkipped":0
    },
    "createdAt":"2015-02-06T23:36:06Z",
    "startUserTime":"2015-02-06T23:40:19Z",
    "startExecutionTime":"2015-02-06T23:41:09Z",
    "endTime":"2015-02-06T23:41:52Z",
    "platform":"Android",
    "devices":[
        "lg-nexus-5-A"
    ],
    "testcases":1,
    "stoppedByUser":false
}
```

4.4.5 Get list of testruns

Returns a list of all the testruns for the given App, if it exists.

```
GET /apps/:appName/testruns
```

Curl example

```
$ curl \
  -X GET \
  -H 'Accept: application/vnd.testmunk.v1+json' \
  'https://AQS0LCTvCv6mTwod5PwtU2i1JVY2J6rW@api.testmunk.com/apps/AppName/testruns'
```

Input

- `appName` (Required): Name of your Testmunk app.

Response

```
Status: 200 Ok
```

```
[
    {
        "id":"54d54fe03004286c71cb99e0",
        "name":"Testrun 100",
        "app":"Testmunk",
        "status":"Success",
        "counts":{
            "numSuccess":1,
            "numFailed":0,
            "numSkipped":0
        },
        "createdAt":"2015-02-06T23:36:06Z",
        "startUserTime":"2015-02-06T23:40:19Z",
        "startExecutionTime":"2015-02-06T23:41:09Z",
        "endTime":"2015-02-06T23:41:52Z",
        "platform":"Android",
        "devices":[
            "lg-nexus-5-A"
        ],
        "testcases":1,
        "stoppedByUser":false
    },
    {
        "id":"34d54fe04904286c71cb87a1",
        "name":"Testrun 99",
        "app":"Testmunk",
        "status":"Success",
        "counts":{
            "numSuccess":2,
            "numFailed":0,
            "numSkipped":0
        },
        "createdAt":"2015-01-06T23:36:06Z",
        "startUserTime":"2015-01-06T23:40:19Z",
        "startExecutionTime":"2015-01-06T23:41:09Z",
        "endTime":"2015-01-06T23:41:52Z",
        "platform":"Android",
        "devices":[
            "lg-nexus-5-A"
        ],
        "testcases":2,
        "stoppedByUser":false
    }
]
```

4.5 Continuous Integration

Testmunk can easily be integrated into your development process. An example of how to integrate testmunk with Jenkins is provided below.

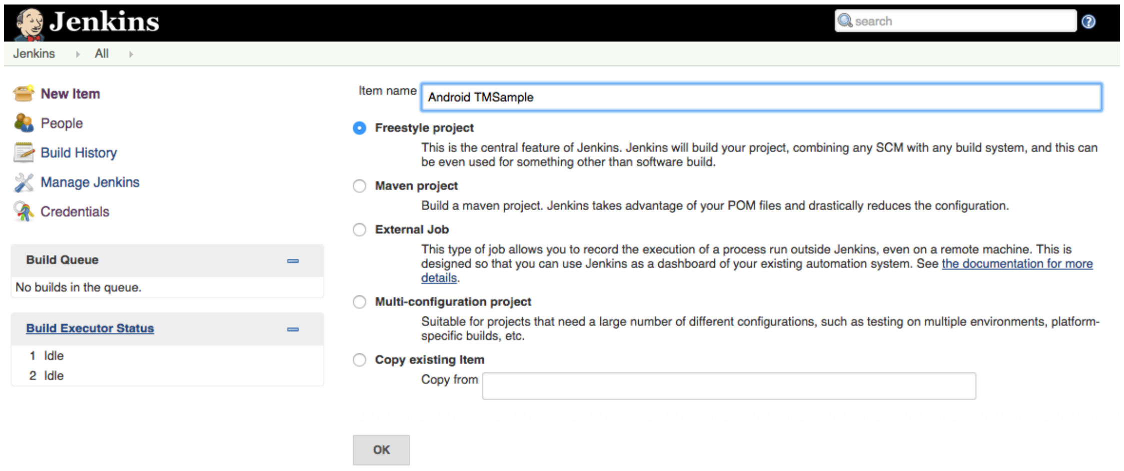
4.5.1 Jenkins

Jenkins is a widely used, extensible open source continuous integration server.

Configuration

1. Create new Item

   First, we will select Freestyle project as the project template for the Android app that we will checkout from a GitHub repository.



Item name: Android TMSample

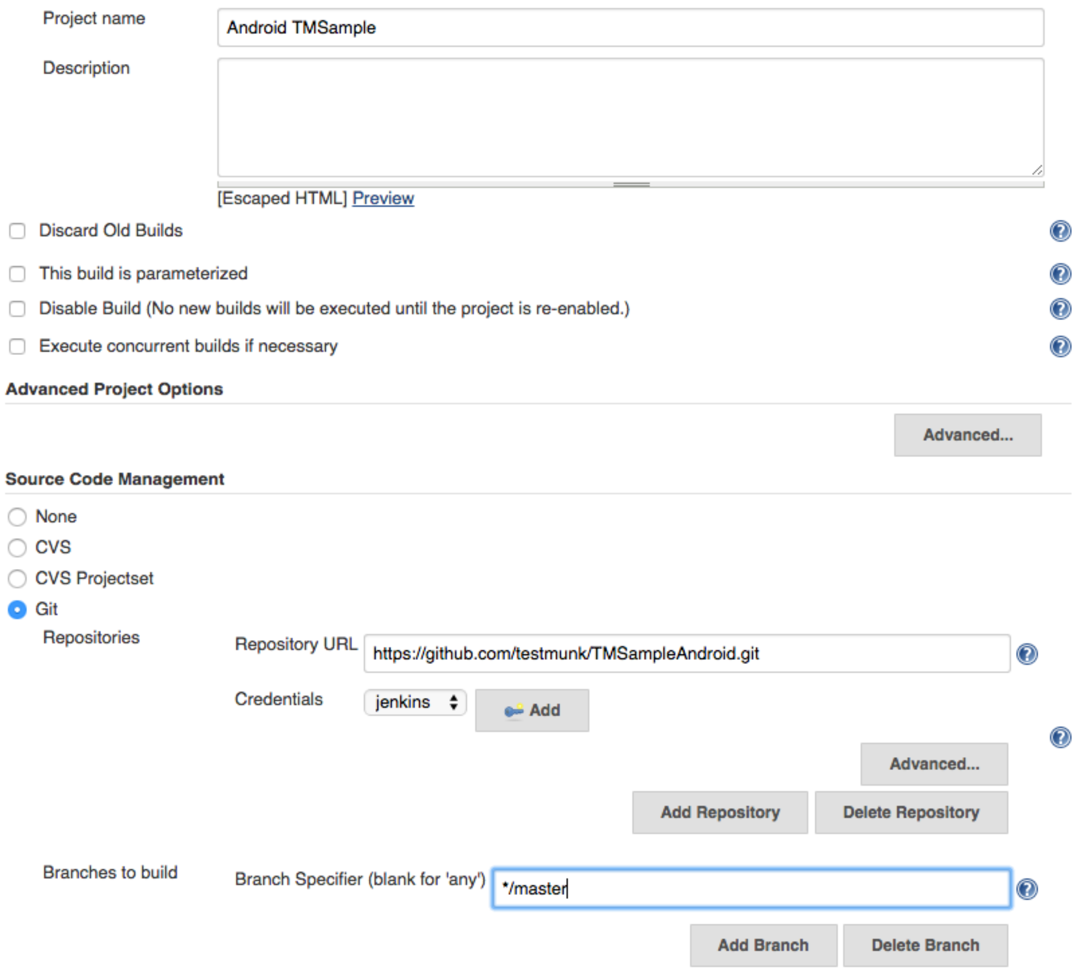
- [✓] **Freestyle project** — This is the central feature of Jenkins. Jenkins will build your project, combining any SCM with any build system, and this can be even used for something other than software build.
- **Maven project** — Build a maven project. Jenkins takes advantage of your POM files and drastically reduces the configuration.
- **External Job** — This type of job allows you to record the execution of a process run outside Jenkins, even on a remote machine. This is designed so that you can use Jenkins as a dashboard of your existing automation system. See the documentation for more details.
- **Multi-configuration project** — Suitable for projects that need a large number of different configurations, such as testing on multiple environments, platform-specific builds, etc.
- **Copy existing Item** — Copy from

No builds in the queue.

1 Idle
2 Idle

2. Configure a repository to get the latest source code of your app

   Jenkins can integrate with many different types of source control management systems, such as CVS, SVN, and Git. For our purpose we will use Git as an example. The TMSample app that we will test is available in GitHub repository. In the image below, you can see us linking the app. The up to date code will be taken from the master branch. You can then change it depending on your needs and testing cycle.



Project name: Android TMSample

Description:

[Escaped HTML] Preview

- Discard Old Builds
- This build is parameterized
- Disable Build (No new builds will be executed until the project is re-enabled.)
- Execute concurrent builds if necessary

Advanced Project Options

Source Code Management

- None
- CVS
- CVS Projectset
- [✓] Git

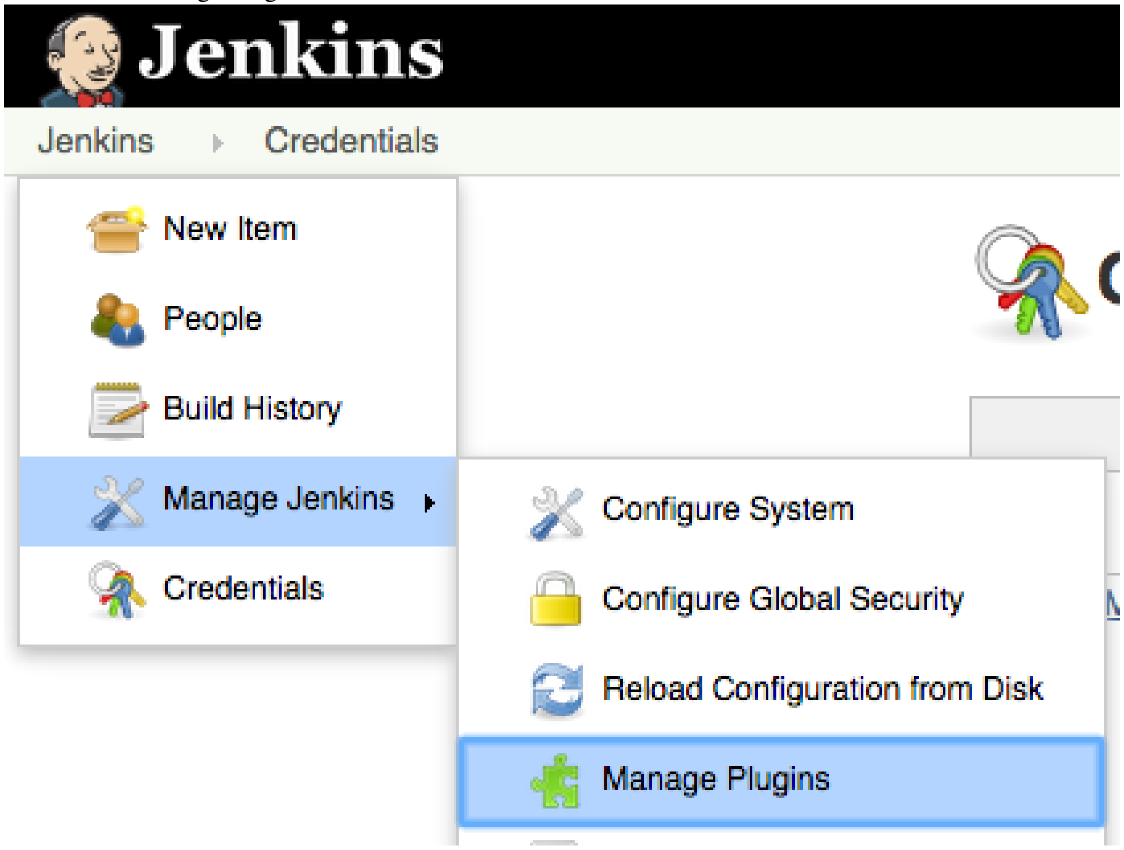
Repositories — Repository URL: https://github.com/testmunk/TMSampleAndroid.git

Credentials: jenkins

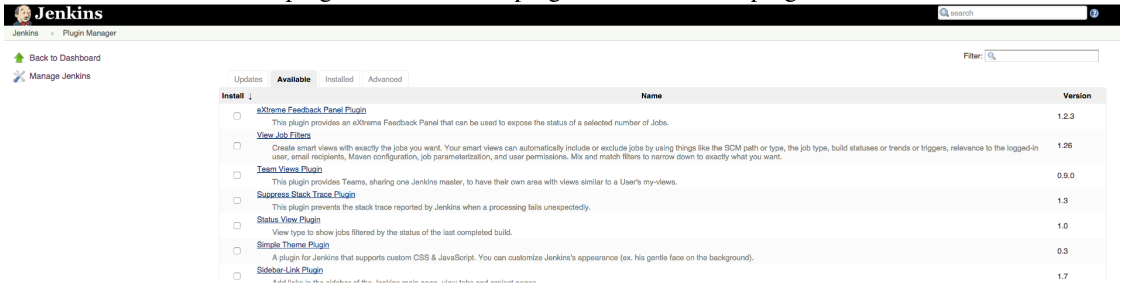
Branches to build — Branch Specifier (blank for 'any'): */master

Hint:

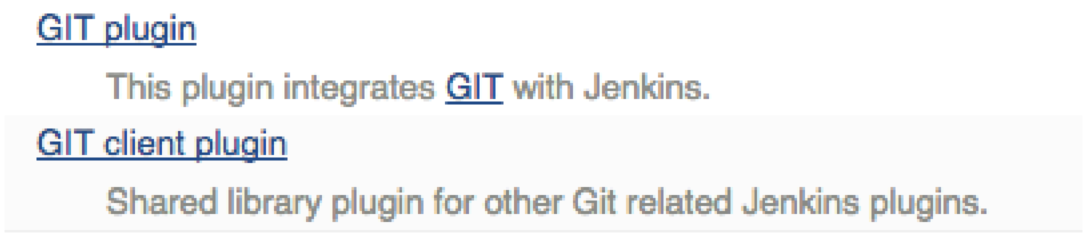
1. Go to Manage Plugins



2. Switch to Available plugins and find GIT plugin and GIT client plugin



Install ↓	Name	Version
☐	eXtreme Feedback Panel Plugin This plugin provides an eXtreme Feedback Panel that can be used to expose the status of a selected number of Jobs.	1.2.3
☐	View Job Filters Create smart views with exactly the jobs you want. Your smart views can automatically include or exclude jobs by using things like the SCM path or type, the job type, build statuses or trends or triggers, relevance to the logged-in user, email recipients, Maven configuration, job parameterization, and user permissions. Mix and match filters to narrow down to exactly what you want.	1.26
☐	Team Views Plugin This plugin provides Teams, sharing one Jenkins master, to have their own area with views similar to a User's my-views.	0.9.0
☐	Suppress Stack Trace Plugin This plugin prevents the stack trace reported by Jenkins when a processing fails unexpectedly.	1.3
☐	Status View Plugin View type to show jobs filtered by the status of the last completed build.	1.0
☐	Simple Theme Plugin A plugin for Jenkins that supports custom CSS & JavaScript. You can customize Jenkins's appearance (ex. his gentle face on the background).	0.3
☐	Sidebar-Link Plugin	1.7



GIT plugin

This plugin integrates GIT with Jenkins.

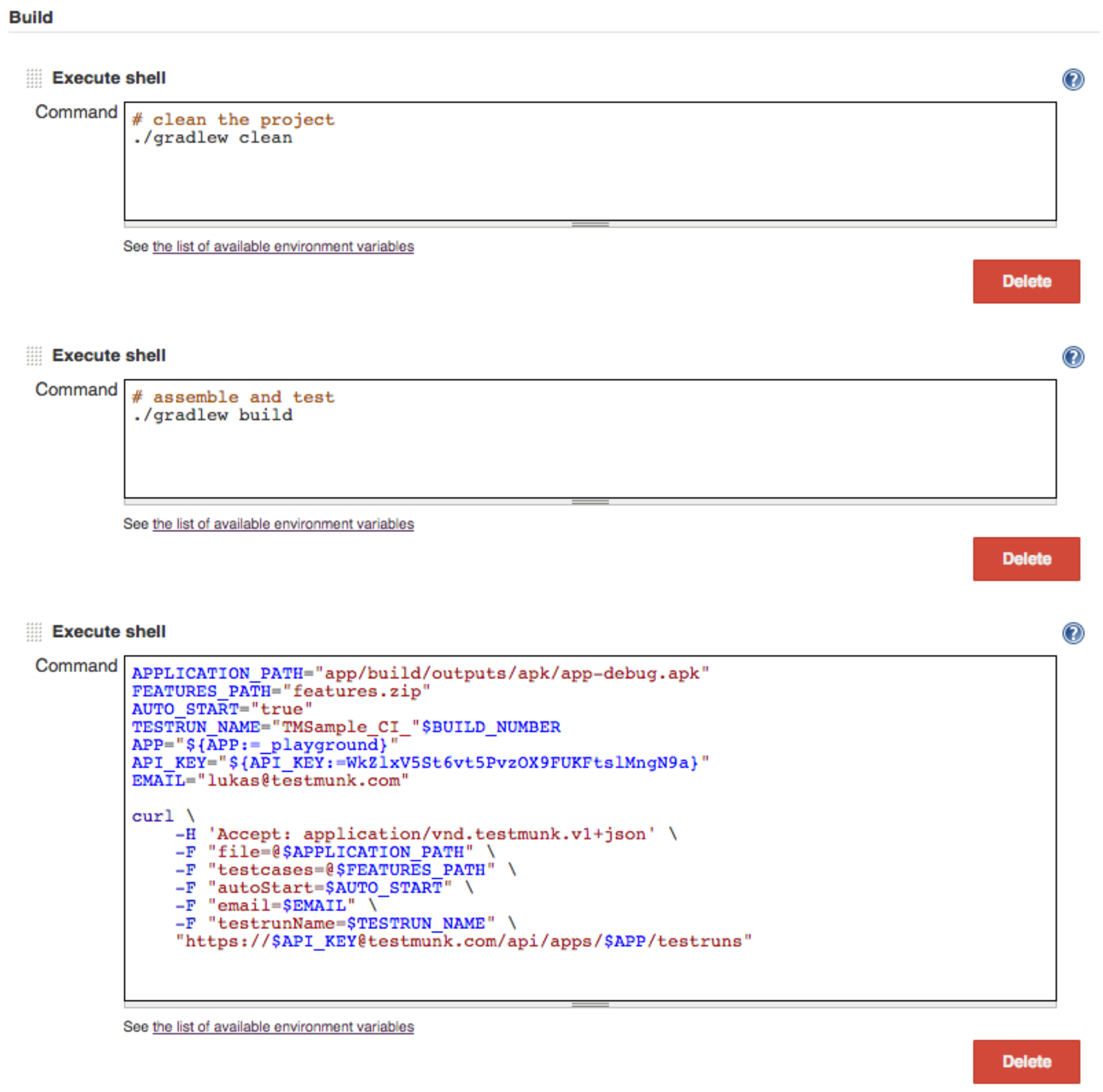
GIT client plugin

Shared library plugin for other Git related Jenkins plugins.

3. Select and install it. Afterwards restart Jenkins (go to: [jenkins_url]/restart)

How to install your Git plugin

3. Setup build steps



```
# clean the project
./gradlew clean
```

See the list of available environment variables

```
# assemble and test
./gradlew build
```

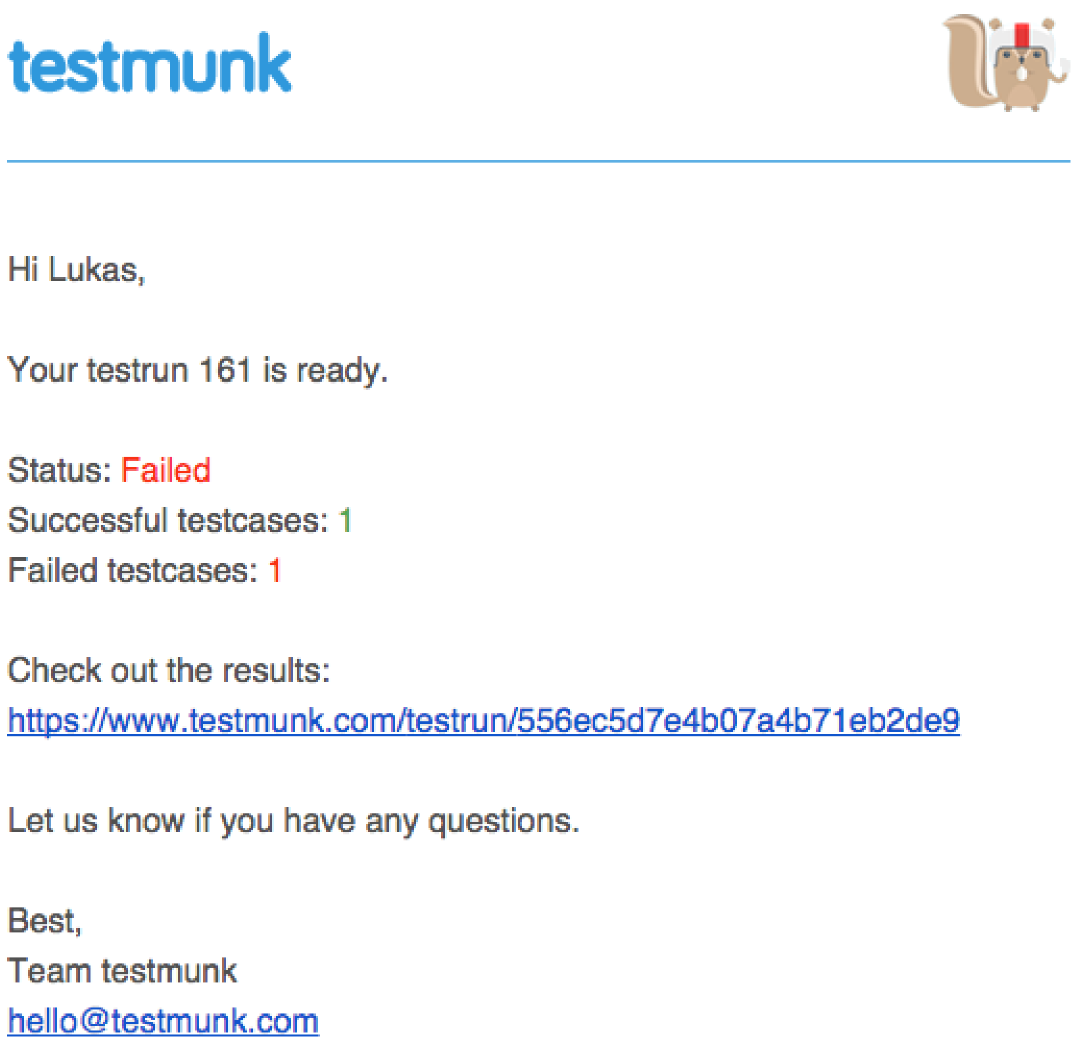
See the list of available environment variables

```
APPLICATION_PATH="app/build/outputs/apk/app-debug.apk"
FEATURES_PATH="features.zip"
AUTO_START="true"
TESTRUN_NAME="TMSample_CI_"$BUILD_NUMBER
APP="${APP:=_playground}"
API_KEY="${API_KEY:=WkZlxV5St6vt5PvzOX9FUKFtslMngN9a}"
EMAIL="lukas@testmunk.com"

curl \
    -H 'Accept: application/vnd.testmunk.v1+json' \
    -F "file=@$APPLICATION_PATH" \
    -F "testcases=@$FEATURES_PATH" \
    -F "autoStart=$AUTO_START" \
    -F "email=$EMAIL" \
    -F "testrunName=$TESTRUN_NAME" \
    "https://$API_KEY@testmunk.com/api/apps/$APP/testruns"
```

Gradle is the build tool that is suggested by Google. It is used in Android Studio IDE. Here, we use clean and build tasks to create an .apk that will be later tested on Testmunk.

Hint: You can use xcodebuild to build your iOS app the same way we use gradlew here

4. The last build step calls *Testruns API* and creates a new testrun with the .apk and features.zip that are in the folder. Test results will be sent to all team members including lukas@testmunk.com



Hi Lukas,

Your testrun 161 is ready.

Status: Failed

Successful testcases: 1

Failed testcases: 1

Check out the results:

https://www.testmunk.com/testrun/556ec5d7e4b07a4b71eb2de9

Let us know if you have any questions.

Best,

Team testmunk

hello@testmunk.com

Hint: It's common to keep tests in a repository. At Testmunk, we work with GitHub Pull Requests to ensure our test code retains the best possible quality. All software development best practises apply here as well.

Here is a script (bash) you can use as a build task to get the latest features from your repository, then zip them to features.zip. Add this build task before starting a new testrun.

```
fetch_tests()
{
   printf "\n## Fetching tests from $GITHUB_URL ##\n"


   curl -sL --user "$GITHUB_USERNAME:$GITHUB_PASSWORD" "$GITHUB_URL" > "$TESTS_PATH"
}


prepare_tests()
{
   printf "\n## Preparing features.zip ##\n"
```

```
    unzip "$TESTS_PATH"
    mv "$GITHUB_REPO_NAME/features" "features"
    zip -r "$FEATURES_PATH" "features/"
}

fetch_tests
prepare_tests
```

where the example environment is as follows:

```
GITHUB_USERNAME="lukas"
GITHUB_PASSWORD="xxx"
GITHUB_URL="https://github.com/testmunk/tm_tests/archive/master.zip"
GITHUB_REPO_NAME="testcases_tm"
TESTS_PATH="tests.zip"
FEATURES_PATH="features.zip"
```

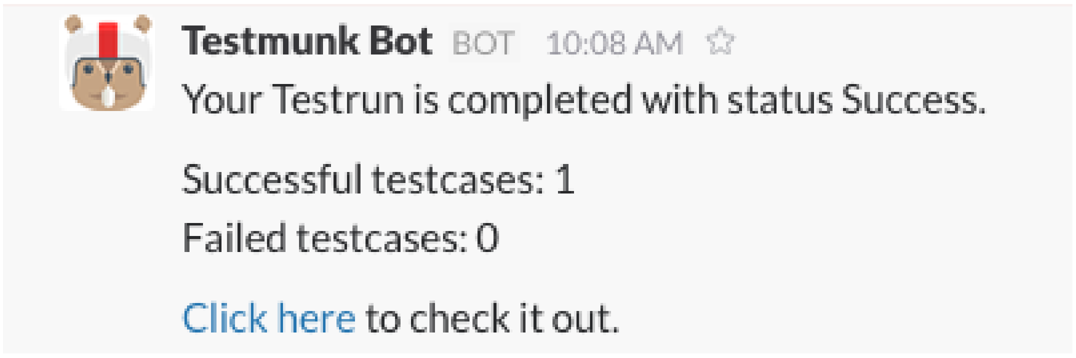
Getting testcases from GitHub repository

Slack Integration

Here at testmunk we are big fans of Slack. We use it daily; it makes collaboration easier and work more productive. We also, of course, spend a lot of time thinking about how we can make the lives of developers and QA engineers easier, especially when it comes to their mobile app testing.

We are therefore excited to share that, starting today, in addition to our REST API and email notifications, our clients can report Testmunk test results to a Slack channel as they are executed.

Test results from your test runs can now be directly posted to your specific Slack channel. This means your mobile development or QA team can stay instantly up to date on the latest test results:



Testmunk Bot BOT 10:08 AM

Your Testrun is completed with status Success.

Successful testcases: 1

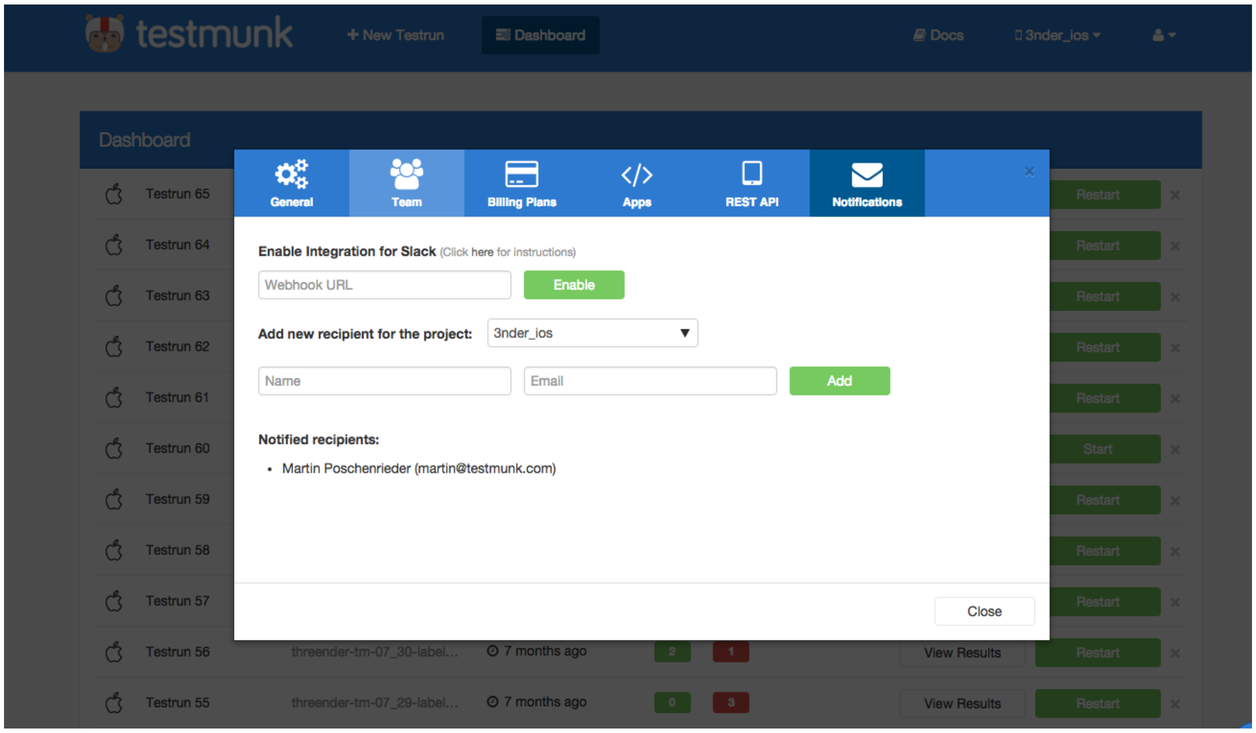
Failed testcases: 0

Click here to check it out.

5.1 Setup Guide

Testmunk - Slack integration allows you to post test results from your test runs directly to a specific slack channel.

Slack integration can be set up by accessing Account Settings (top right dropdown) -> Notification Tab.



Enable Integration for Slack (Click here for instructions)

Webhook URL [Enable]

Add new recipient for the project: 3nder_ios

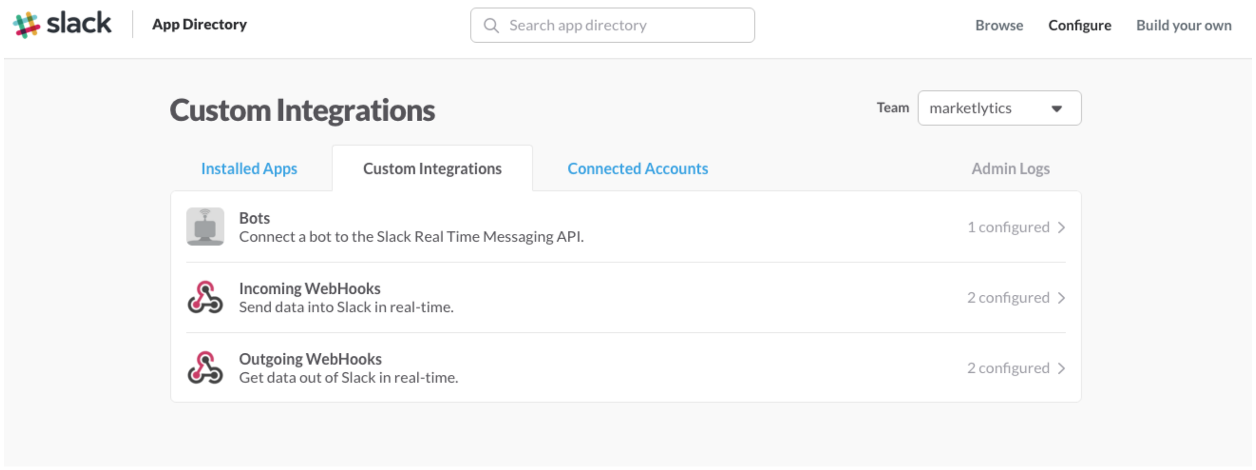
Name | Email [Add]

Notified recipients:

- Martin Poschenrieder (martin@testmunk.com)

The Webhook URL needs to be taken from your Slack account. You need to follow the steps below:

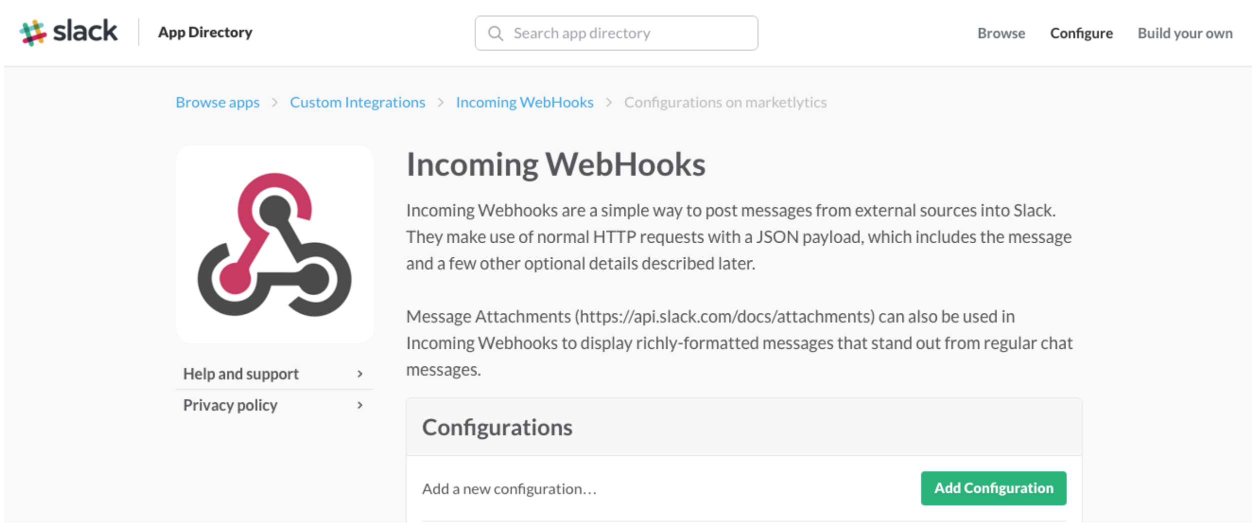
1. Go to the Slack integration settings page: https://<TEAM_ID>.slack.com/apps/manage/custom-integrations



Custom Integrations

Bots Connect a bot to the Slack Real Time Messaging API.	1 configured
Incoming WebHooks Send data into Slack in real-time.	2 configured
Outgoing WebHooks Get data out of Slack in real-time.	2 configured

2. Click on Incoming Webhooks



slack | App Directory

Search app directory

Browse Configure Build your own

Browse apps > Custom Integrations > Incoming WebHooks > Configurations on marketlytics

Incoming WebHooks

Incoming Webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details described later.

Message Attachments (https://api.slack.com/docs/attachments) can also be used in Incoming Webhooks to display richly-formatted messages that stand out from regular chat messages.

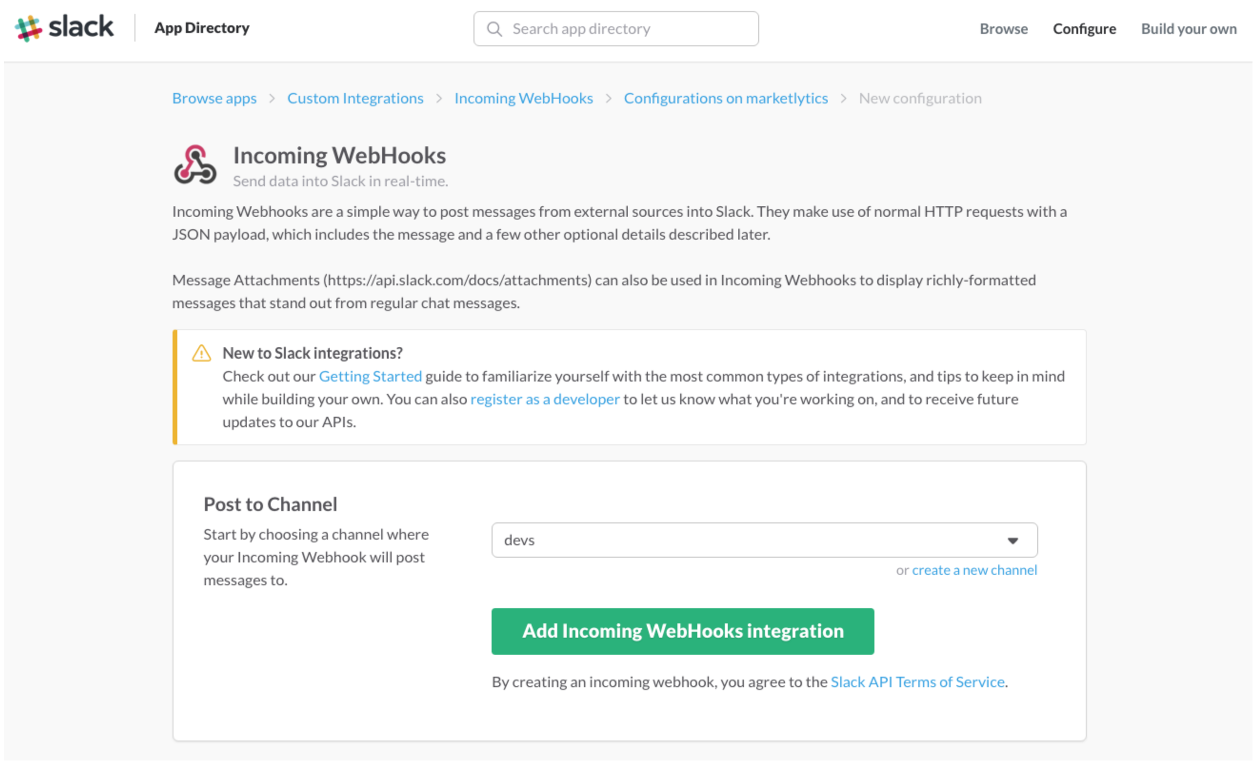
Help and support >

Privacy policy >

Configurations

Add a new configuration… Add Configuration

3. Click on Add Configuration



slack | App Directory

Search app directory

Browse Configure Build your own

Browse apps > Custom Integrations > Incoming WebHooks > Configurations on marketlytics > New configuration

Incoming WebHooks

Send data into Slack in real-time.

Incoming Webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details described later.

Message Attachments (https://api.slack.com/docs/attachments) can also be used in Incoming Webhooks to display richly-formatted messages that stand out from regular chat messages.

New to Slack integrations?

Check out our Getting Started guide to familiarize yourself with the most common types of integrations, and tips to keep in mind while building your own. You can also register as a developer to let us know what you're working on, and to receive future updates to our APIs.

Post to Channel

Start by choosing a channel where your Incoming Webhook will post messages to.

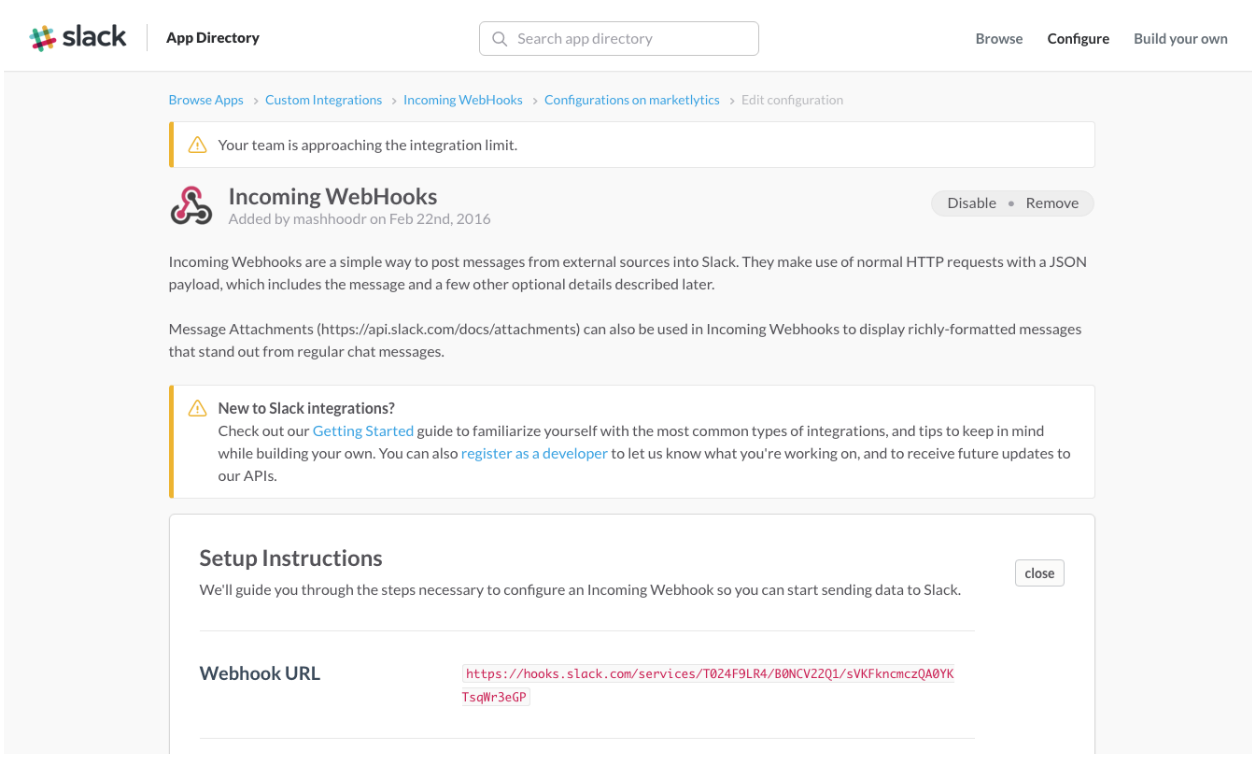
devs

or create a new channel

Add Incoming WebHooks integration

By creating an incoming webhook, you agree to the Slack API Terms of Service.

4. Choose the channel where the notification should be posted, and click Add Incoming WebHooks integration.



Browse Apps > Custom Integrations > Incoming WebHooks > Configurations on marketlytics > Edit configuration

Your team is approaching the integration limit.

Incoming WebHooks
Added by mashhoodr on Feb 22nd, 2016

Incoming Webhooks are a simple way to post messages from external sources into Slack. They make use of normal HTTP requests with a JSON payload, which includes the message and a few other optional details described later.

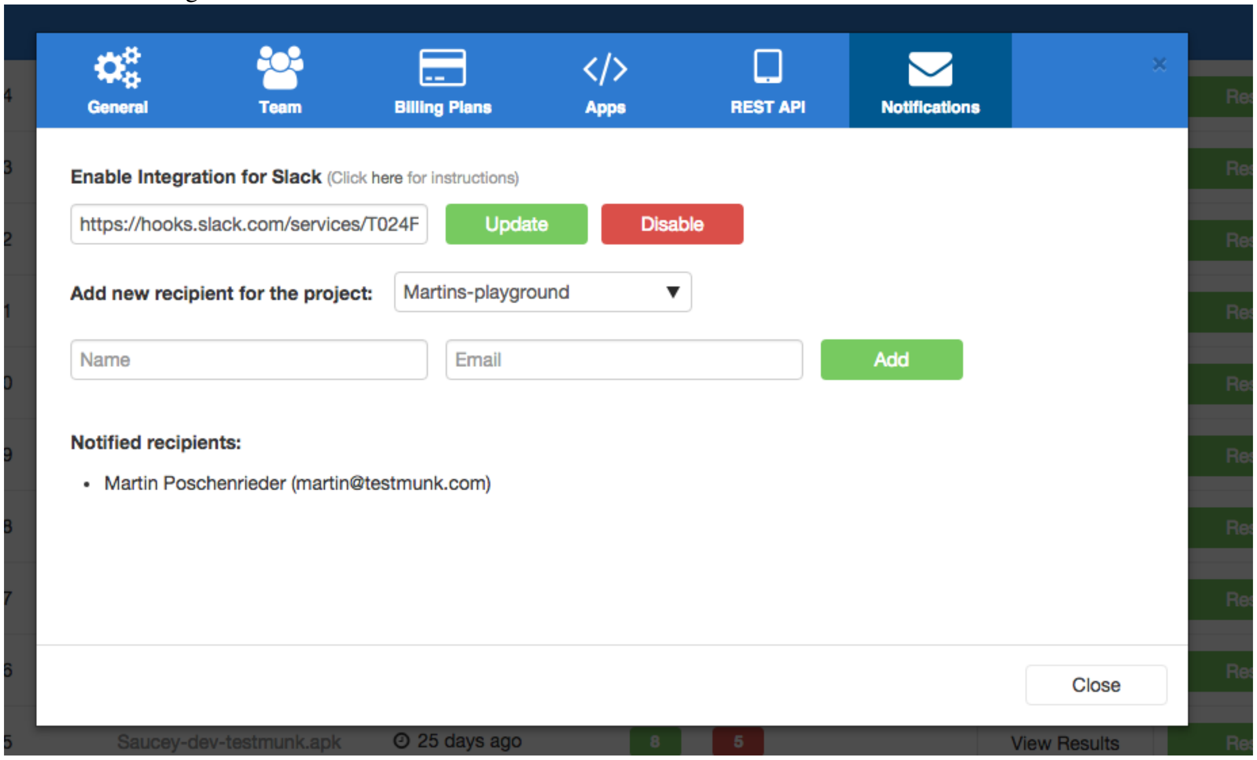
Message Attachments (https://api.slack.com/docs/attachments) can also be used in Incoming Webhooks to display richly-formatted messages that stand out from regular chat messages.

New to Slack integrations?
Check out our Getting Started guide to familiarize yourself with the most common types of integrations, and tips to keep in mind while building your own. You can also register as a developer to let us know what you're working on, and to receive future updates to our APIs.

Setup Instructions
We'll guide you through the steps necessary to configure an Incoming Webhook so you can start sending data to Slack.

Webhook URL `https://hooks.slack.com/services/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`

5. You can now see the Webbook URL, copy this; then scroll down and click on save settings:
6. Go back to the Testmunk settings page and paste the Webhook URL; then click Enable.
7. Start a test run and get the notification in Slack! It'll be like the example notification at the top of this post.
8. If you want to disable the notification, simply access the notification tab in settings again and click Disable. You will no longer be notified of the test runs in slack.



Enable Integration for Slack (Click here for instructions)

https://hooks.slack.com/services/T024F

Add new recipient for the project: Martins-playground

Notified recipients:

- Martin Poschenrieder (martin@testmunk.com)