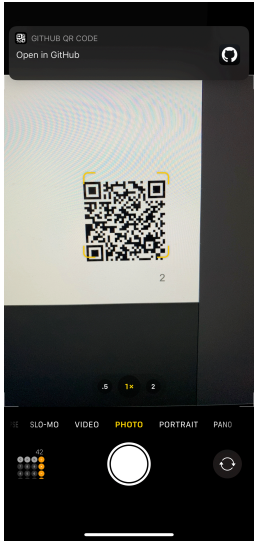
Automating Dark Mode Screenshots with Xcode 11

Alexander Botkin, Principal Engineer

23 April 2020

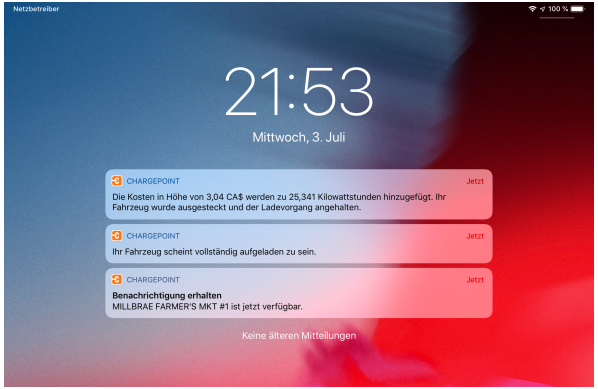
Watch that bottom right corner!

Open the stock iOS Camera app, point at the QR code, & tap the banner if you want to learn more!



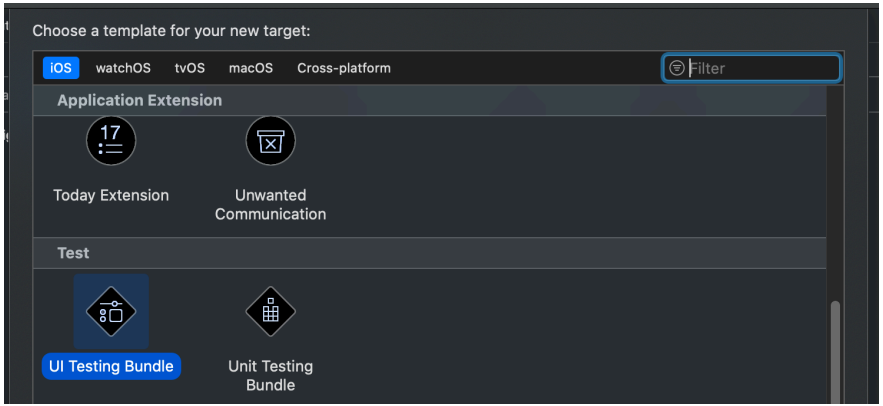
Why Automate Screenshots?

- Simplifies upkeep on your App Store Connect page
- Provides a visual baseline you can review & compare
- Can guide development of massive UI changes in iOS
  - Safe Area layout, Dark Mode
- Empowers small teams like ChargePoint's iOS team



How do we create screenshots?

- UI Testing Bundle target
- Xcode generates a <YourAppName>UITests.swift file with an XCTestCase subclass
- We need to add test functions that save attachments onto the XCTestCase



How do we create screenshots?

```
func testMainViewScreenshot() throws {
    // UI tests must launch the application that they test.
    let app = XCUIApplication()
    app.launch()

    // Let's ensure the view has appeared by using the accessibility identifier
    // we set up in the storyboard
    let darkMapVCView = app.otherElements["Dark Map View"];
    XCTAssertTrue(darkMapVCView.waitForExistence(timeout: 3))

    // Now let's get a screenshot & save it to the xcresult as an attachment
    let screenshot = XCUIScreen.main.screenshot()

    let attachment = XCTAttachment(screenshot: screenshot)
    attachment.name = "MyAutomation_darkMapView"
    attachment.lifetime = .keepAlways

    self.add(attachment)
}
```

How do we create screenshots?

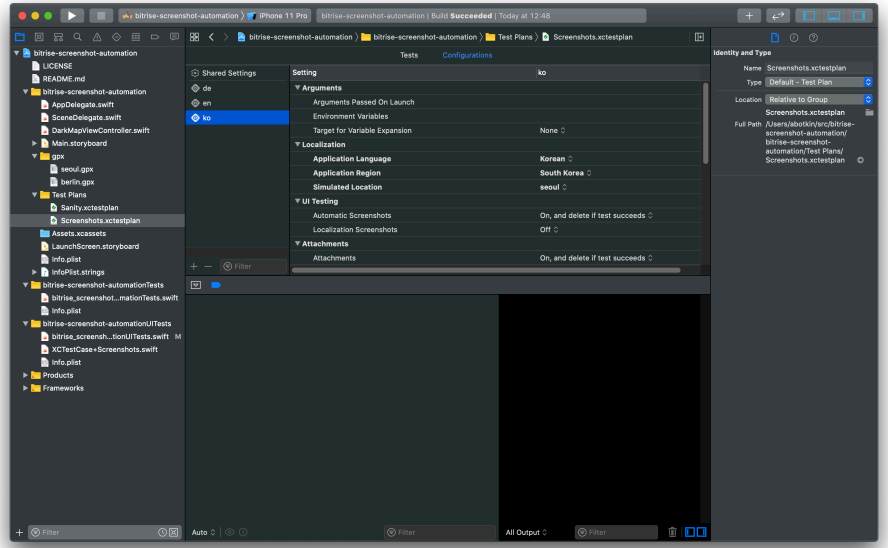
```
//
//  XCTestCase+Screenshots.swift
//  bitrise-screenshot-automationUITests
//
//  Created by Alexander Botkin on 4/20/20.
//  Copyright 2020 ChargePoint, Inc. All rights reserved.
//

import XCTest

extension XCTestCase {
    func saveScreenshot(_ name: String) {
        let screenshot = XCUIScreen.main.screenshot()

        let attachment = XCTAttachment(screenshot: screenshot)
        attachment.name = name
        attachment.lifetime = .keepAlways

        self.add(attachment)
    }
}
```

How do we create screenshots?

```
func testMainViewScreenshot() throws {
    // UI tests must launch the application that they test.
    let app = XCUIApplication()
    app.launch()

    // Let's ensure the view has appeared by using the accessibility identifier
    // we set up in the storyboard
    let darkMapVCView = app.otherElements["Dark Map View"];
    XCTAssertTrue(darkMapVCView.waitForExistence(timeout: 3))

    // Now let's get a screenshot & save it to the xcresult as an attachment
    self.saveScreenshot("MyAutomation_darkMapView")
}
```

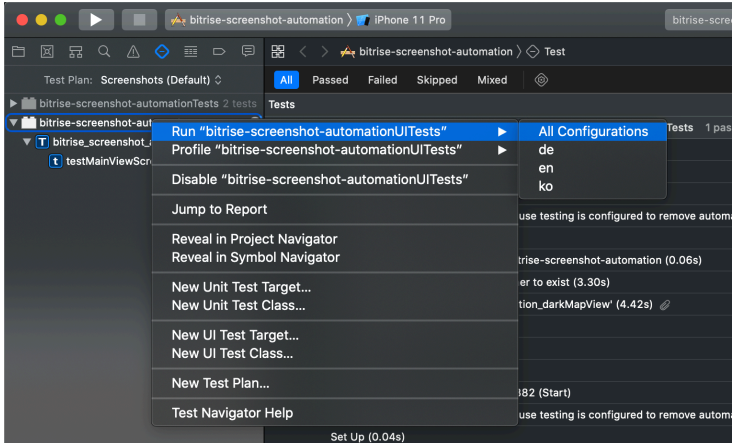
How do we create screenshots?

- How do we run the app in different languages & regions?
  - Xcode 10 & below: make multiple schemes
  - Xcode 11: make one test plan & add various configurations
- Create run configurations specific to a app language/region pairing
  - Does not affect device language & thus some system prompts
- Add GPX file in Xcode to simulate a custom GPS coordinate or route



How do we create screenshots?

- With test plans, can run all configurations & create one report (xcresult)
  - xcodebuild option: *-testPlan 'Screenshots'*
- Can also run one specific run configuration
  - xcodebuild option: *-only-test-configuration "de"*
- On Bitrise's Xcode Test step, these options are have to be added in the Debug menu
  - Additional options for `xcodebuild build test`



Additional options for `xcodebuild build test` call

Options added to the end of the `xcodebuild build test` call.
If you leave empty this input, xcodebuild will be called as:
`xcodebuild -project\-workspace PROJECT.xcodeproj\WORKSPACE.xcworkspace -scheme SCHEME build`
In case of `generate_code_coverage_files: "yes"` `xcodebuild` gets two additional flags:

- GCC*INSTRUMENT*PROGRAM*FLOW*ARCS=YES
- GCC*GENERATE*TEST*COVERAGE*FILES=YES

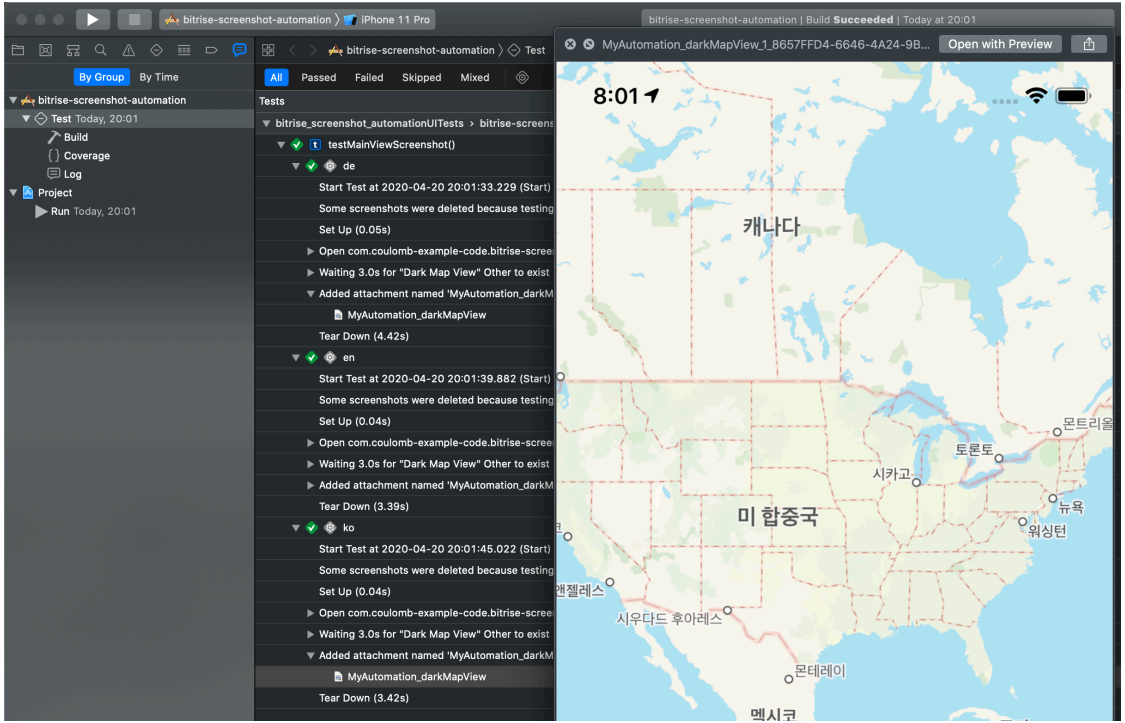
If you want to add more options, list that separated by space character. Example: `-xcconfig PATH -verbose`

```
-derivedDataPath derivedData -testPlan 'Screenshots'
```

Environment Variables will be replaced in input by the Bitrise CLI before starting the Step. ?

How do we create screenshots?

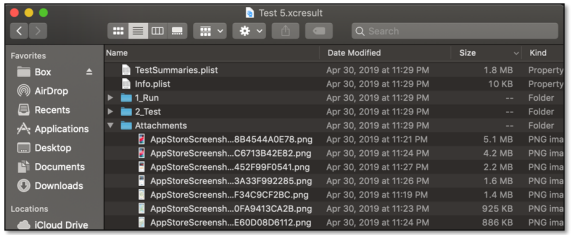
+ We have the report (xcresult), but how do we get all the screenshots out at once?

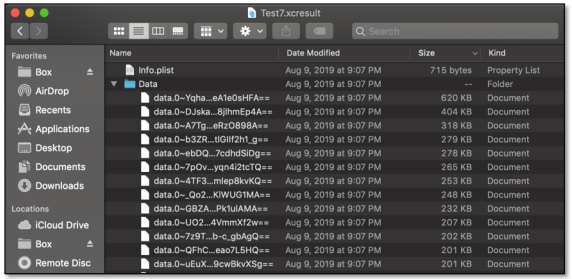


Extracting Screenshots

xcresult & xcresulttool

- With Xcode 10 and below, xcresult had an easily inspected file structure
- With Xcode 11+, Apple changed the xcresult format
- *xcresulttool* is included in Xcode 11+ command line tools to do inspection & extraction of xcresult
  - Powerful tool
  - Not a one-line extractor





xcresult & xcresulttool

- All these commands, for just 1 screenshot

```
$ xcrun xcresulttool get --path $xcresultpath --format json
```

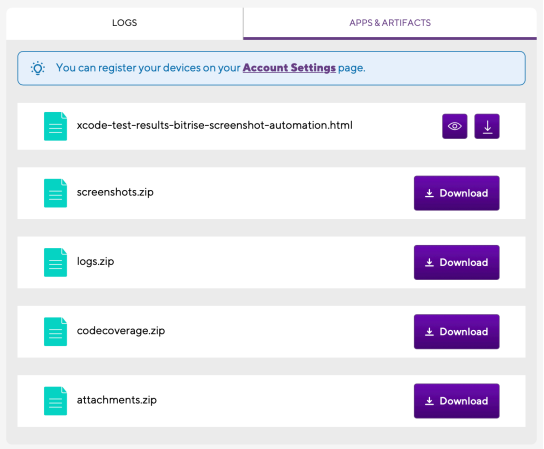
```
$ xcrun xcresulttool get --path $xcresultpath --format json --id 0~_6FLLXVxqNZFWTecoWAKeruyb1Jz8RMTQhv1WOY6rrDKD9fcctzLS71METqymkBgT27dA5y8aPQtO7QGqf22Aw==
```

```
$ xcrun xcresulttool get --path $xcresultpath --format json --id 0~kQ1eE6SCOHOqHvf_H1RgkahpjprRs3YG7BKlO_Twqzv4Cu3r7mbK3WSOSudizsrE7_4I6nPMnjdiXXdri8bH_w==
```

```
$ xcrun xcresulttool export --path $xcresultpath --output-path $destinationPath/$filename --id 0~JVQMKe6IgJ5d1MjzEtX2RGgfrtdmFxsP76yVkE-zoykhLqzSlva1VbnxkOVCNIjxXPfh2ldYqmCoyldz_YifmQ== --type file
```

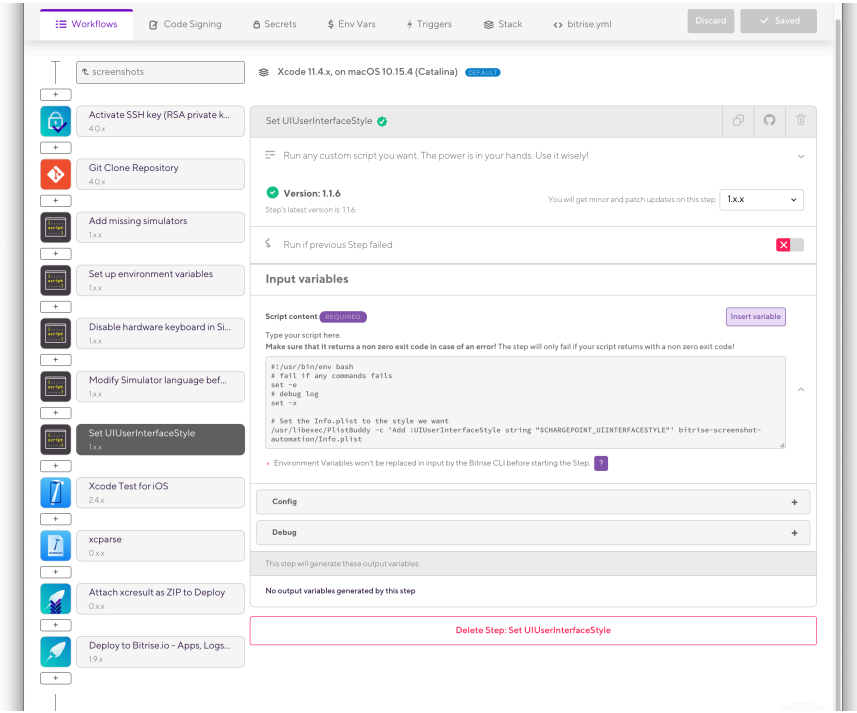
xcparse

- Swift command line tool to parse Xcode 11 xcresult
  - Extracts screenshots, attachments, logs (device, testmanagerd, etc.), & code coverage
  - *xcparse screenshots* makes extraction one command
- Options to separate screenshots by model, OS, language, test plan, test, & more
- Bitrise plugin support
  - Adds ZIPs with files directly to Bitrise Deploy
  - Provides the paths to the ZIPs as output for your own scripts to use



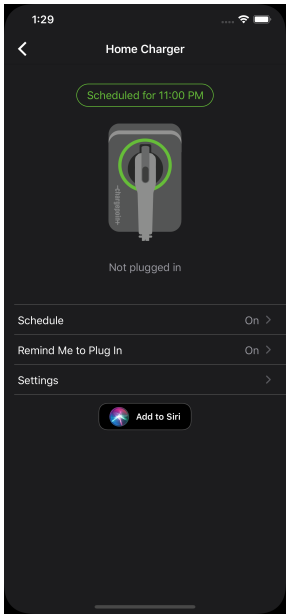
Integrating with Bitrise

Sample App: bitrise-screenshot-automation



https://github.com/ChargePoint/bitrise-screenshot-automation

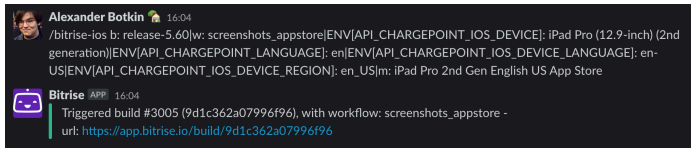
Dark Mode Screenshots



- Xcode 11.4 added simctl command to change device's UI style
  - *xcrun simctl ui <DEVICE> appearance dark*
- For Xcode 11.0 to 11.3, this option doesn't exist
  - Can modify the Info.plist of your app in a script step
    - Real device or simulator
    - Affects app UI style only
  - Can modify default in *com.apple.uikitservices.userInterfaceStyleMode*
    - Simulator only
    - Affects device & app UI style

Environment Variables

- Use environment variables to configure options like test plan, configuration, device, interface style
- Allows you to have one workflow that can do all the work
- You can set default values for the environment variables when workflow is invoked or customize it if you make a workflow clone
- Allows for replacement of values via curl or Bitrise Slack slash commands



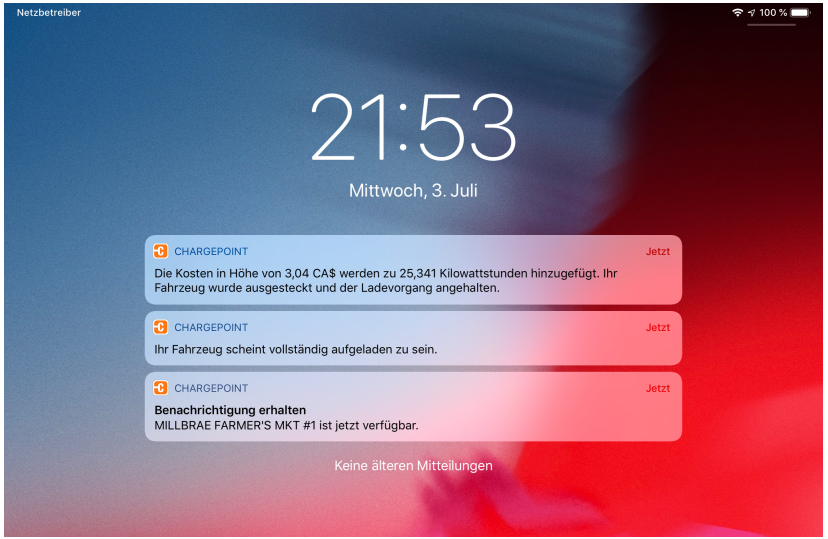
- Need a script step to set environment variable from those passed via APIs

```
#!/bin/bash

# Pull any arguments that may have been set from Slack slash commands and such
if [ ! -z "$API_CHARGEPOINT_UIINTERFACESTYLE" ] ; then
  envman add --key CHARGEPOINT_UIINTERFACESTYLE --value "$API_CHARGEPOINT_UIINTERFACESTYLE"
fi
```

Device Language in Simulator

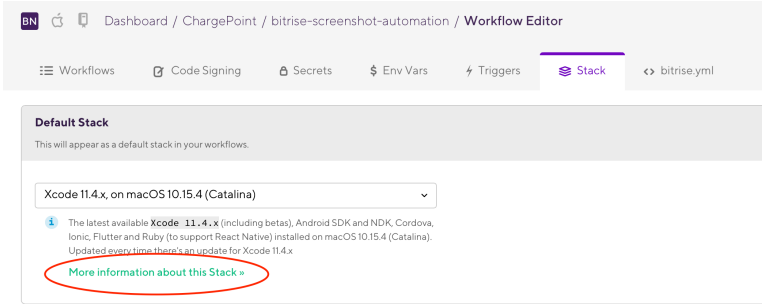
- As mentioned before, test plan language settings only affect the app language & region, not the device
  - Leaves OS dialogues that don't happen in your process unlocalized
- Modifying the Simulator's defaults can be a workaround for this

```
#!/bin/bash

# Iterate over every Simulator & modify the global
preferences for lang/region
find ~/Library/Developer/CoreSimulator/Devices/*
-type d -maxdepth 0 -exec /usr/libexec/PlistBuddy -c
"Delete :AppleLanguages" -c "Add :AppleLanguages
array" -c "Add :AppleLanguages:0 string
$CHARGEPOINT_IOS_DEVICE_LANGUAGE" -c
"Delete :AppleLocale" -c "Add :AppleLocale string
$CHARGEPOINT_IOS_DEVICE_REGION" {}/data/Library/
Preferences/.GlobalPreferences.plist \;
```



Stack Simulator Availability

+ Bitrise stacks may not have all the Simulators you want
+ Check the stack information to see what's in the VM by default




```
-- iOS 13.4 --
    iPhone 8 (ED09C5AA-7771-4EB6-ADCB-A9E5ED345AF0) (Shutdown)
    iPhone 8 Plus (D57BF0FB-ABEE-4143-BDE9-0BEC97E5AD6B) (Shutdown)
    iPhone 11 (5A4CF67B-B450-481D-B4FA-0E59AC3D90EC) (Shutdown)
    iPhone 11 Pro (A810F415-6675-4645-AA9D-D2632EA90842) (Shutdown)
    iPhone 11 Pro Max (AFA116C5-FFF2-4314-A4F5-7599DC80C60A) (Shutdown)
    iPhone SE (2nd generation) (2BA6EF22-9AFB-41AD-AE11-6707EB303F5C) (Shutdown)
    iPad Pro (9.7-inch) (A9D8B918-C006-4180-A0FA-A3C53E74E6ED) (Shutdown)
    iPad (7th generation) (1FF448AB-089A-4064-8224-B73E0618A89B) (Shutdown)
    iPad Pro (11-inch) (2nd generation) (B05DF69F-04A4-42AD-BF1F-786E59503422) (Shutdown)
    iPad Pro (12.9-inch) (4th generation) (C6446973-B376-48F5-9084-3117BC4F2459) (Shutdown)
    iPad Air (3rd generation) (EFD7BAB4-DFFC-4D7F-B96D-47999083FF27) (Shutdown)
```

+ If there's a Simulator you need, create it in a script step using simctl

```
#!/bin/bash

# Create the iPad Pro (12.9-inch) (2nd generation) simulator
xcrun simctl create "iPad Pro (12.9-inch) (2nd generation)" "com.apple.CoreSimulator.SimDeviceType.iPad-Pro--12-9-inch---2nd-generation-" "com.apple.CoreSimulator.SimRuntime.iOS-13-4"
```

Demo

Thank You

For further information on this topic,
check out the sample code on GitHub
https://github.com/ChargePoint/bitrise-screenshot-automation