Day 5

1. Provisioning
2. Distribution
3. Unit Testing
4. Instruments
5. UI Testing
6. Automation

Homework

Provisioning and Distribution

In order to provision and distribute an app to non-jailbroken devices you need to be a registered developer through the apple developer program, either through a development account or an enterprise account.

* ‘provisioning is the process of preparing and configuring an app to launch on devices and use app services’

Account Types

Development Account (Individual or Company)

The Apple Developer Program is for submitting apps on different Apple App stores.

Enterprise Account: In-house Distribution

The Apple Enterprise Program is for organizations developing and distributing in-house applications for their employees

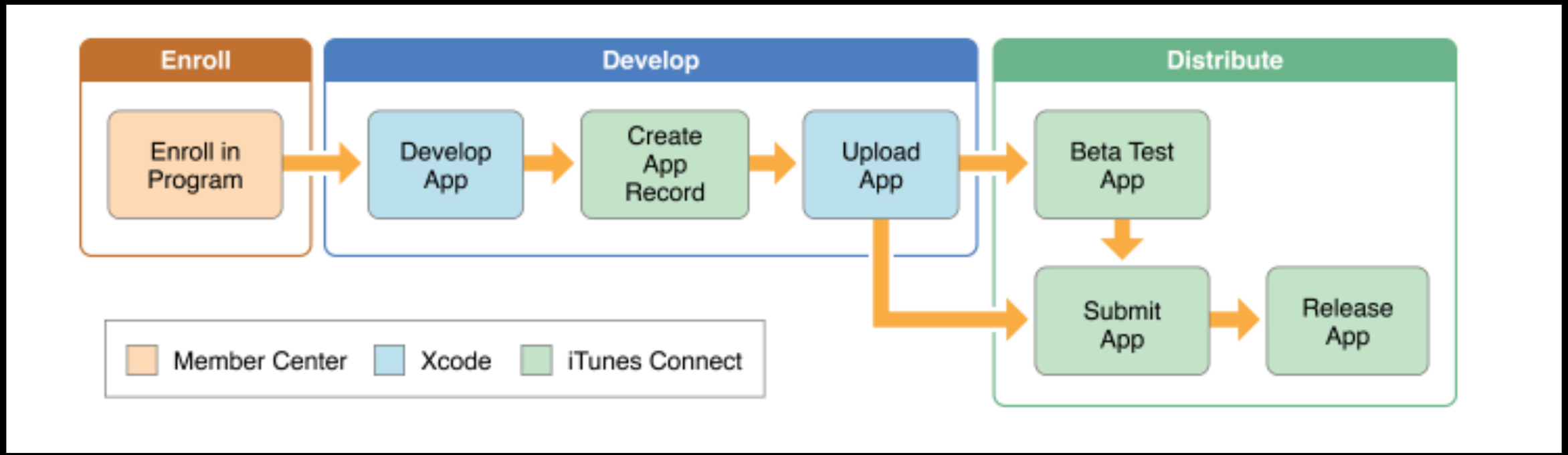
Roles

1. Team Agent - Legally responsible for the team. (Super user)
2. Admin - Can do just about everything
3. Member - Only sign apps during development

- Team agent defaults to the user that created the account.

- Manage roles through member center.

Provisioning



Provisioning Steps

1) Enroll as a develop
2) Create Developer Certificates
3) Download Intermediate Certificates
4) Create App ID
5) Create Development / Distribution Provisioning Profiles for your app
6) Distribute your app -
    - Development (Only to registered device)
        Use a cable
        Use iTunes
        Place on a server
    - Store Distribution
    - Enterprise Distribution

Apple Member Center

- Management portal for Apple accounts
- Provides technical learning resources for developers
- Provides development software
- Place for technical discussions and queries
- Beta software available

http://developer.apple.com

Certs and Provision Profiles

Bob

Admin

Dist. Cert

Dev. Cert

Jenny

Member

Dev. Cert

Jacob

Team Agent

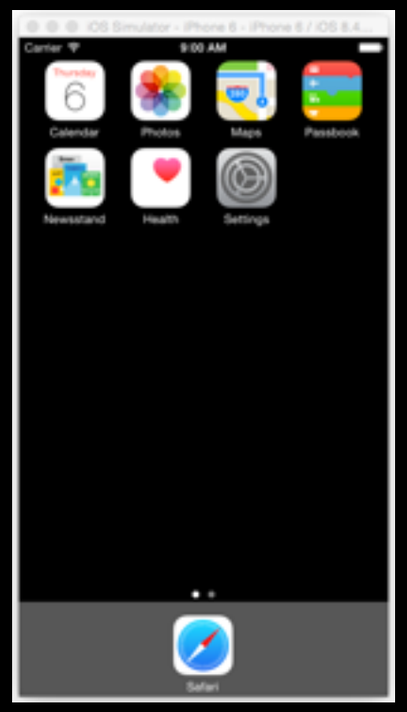
Provisioning Profile

Type: Development

App ID:com.elad.myFirstApp

Developers Allowed to sign this app: Bob

Services: In-App purchase, Apple Push Notification Service,
Game Center



Photos

Profile Types

Wildcard - This is for development use. Allows for provisioning to all registered devices from any of the team developers. Will run the app in a development sandbox with many missing features. Used with a wildcard app ID

Development - This is for development. It is linked to specific devices and to specific developers. The app will run in a debug state and will operate in the development sandbox. Certain features may behave differently

Store Distribution - For Apple store submission only. Will be linked to a specific certificate. Cannot be used for provisioning. Will run in the release environment. Must have the appropriate app ID

Adhoc Distribution - For the purposes of distributing an app to specific devices. Will be linked to the distribution certificate. Will run in the production sandbox.

Enterprise Distribution - For the purposes of distributing an app to any ‘company’ devices. Will be linked to the distribution certificate. Will run in the production sandbox. This is for an internal app store

iTunes Connect

“iTunes Connect is a web tool you use to enter information about your app for sale in the App Store. iTunes Connect stores all the metadata about your app including the versions and builds that you upload using Xcode”

“iTunes Connect is a marketing and business web tool that Apple Developer Program members use to sign contracts, set up tax and banking information, submit versions of their app, and obtain sales and finance reports”

XCTest Framework

- XCTest is the built in testing framework used in ObjC (it used to be OCUnit).
- The XCTestFramework is automatically linked by all new test targets.
- There is a test navigator to make testing easier
- XCode 6 and above offers performance testing
- Unit tests can be easily integrated into continuous integration.

Different Tests Available

1. Vanilla Unit Tests
2. Performance Tests
3. Asynchronous Tests

Example Code

```
- (void)setUp
    {
    [super setUp];
   //
   //
    }


- (void)tearDown
    {
     //
    [super tearDown];
    }

- (void)testAddition
    {
   XCTAssertTrue();
    XCTAssertEqual();
    XCTAssert();
   }

- (void)testPerformanceExample
    {
    [self measureBlock:^{
        // Do some image processing
        }];
    }
```

Asynchronous Tests

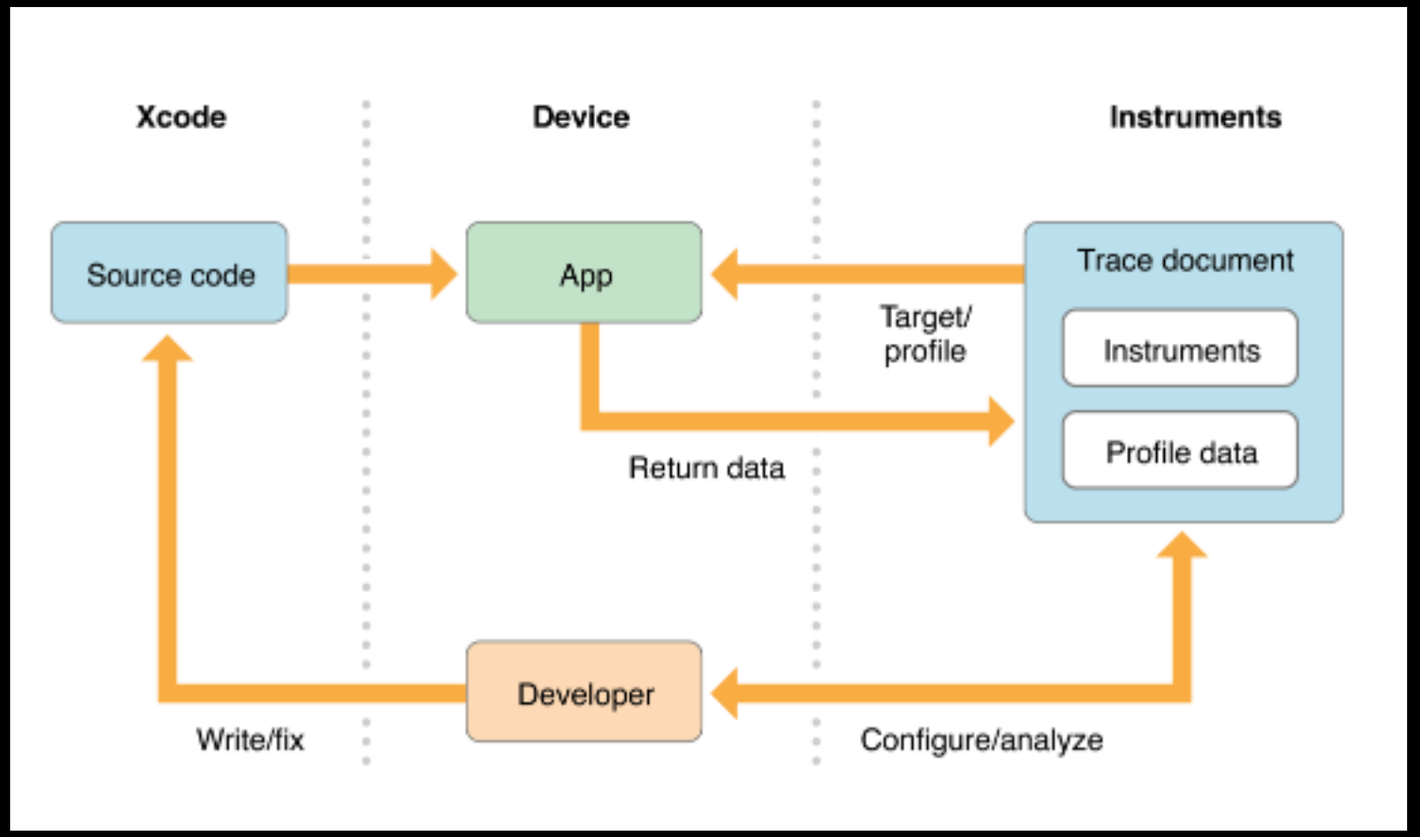
```
- (void) testGetLondonWeather
    {
    XCTestExpectation *getLondonWeatherExpectation;

    getLondonWeatherExpectation = [self expectationWithDescription:@"Get London
         Weather"];

    [[ELDBackendManager sharedInstance]
        getWeatherForLondonWithSuccess:^(NSDictionary *responseDictionary)
            {
            XCTAssertNotNil(responseDictionary,@"The response dictionary should not
                 be nil");
            XCTAssertNotNil(responseDictionary[@"London"],@"There should be a London
                 field in the response dictionary");
            [getLondonWeatherExpectation fulfill];
            }
        failure:^(NSError *error)
            {
            XCTAssertNil(error, @"Thee get London weather call should not fail");
            [getLondonWeatherExpectation fulfill];
            }];
    [self waitForExpectationsWithTimeout:10 handler:^(NSError *error) {}];
}
```

Instruments

(⌘ + i)

Instruments is a powerful and flexible performance-analysis and testing tool

You use specialized tools, known as instruments, to trace different aspects of your apps



Instruments Uses

- Examine the behavior of your app
- Examine device features
- Profile
- Performance analysis
- Examine the memory footprint of your app with potential leaks
- Improve the power efficiency of your app

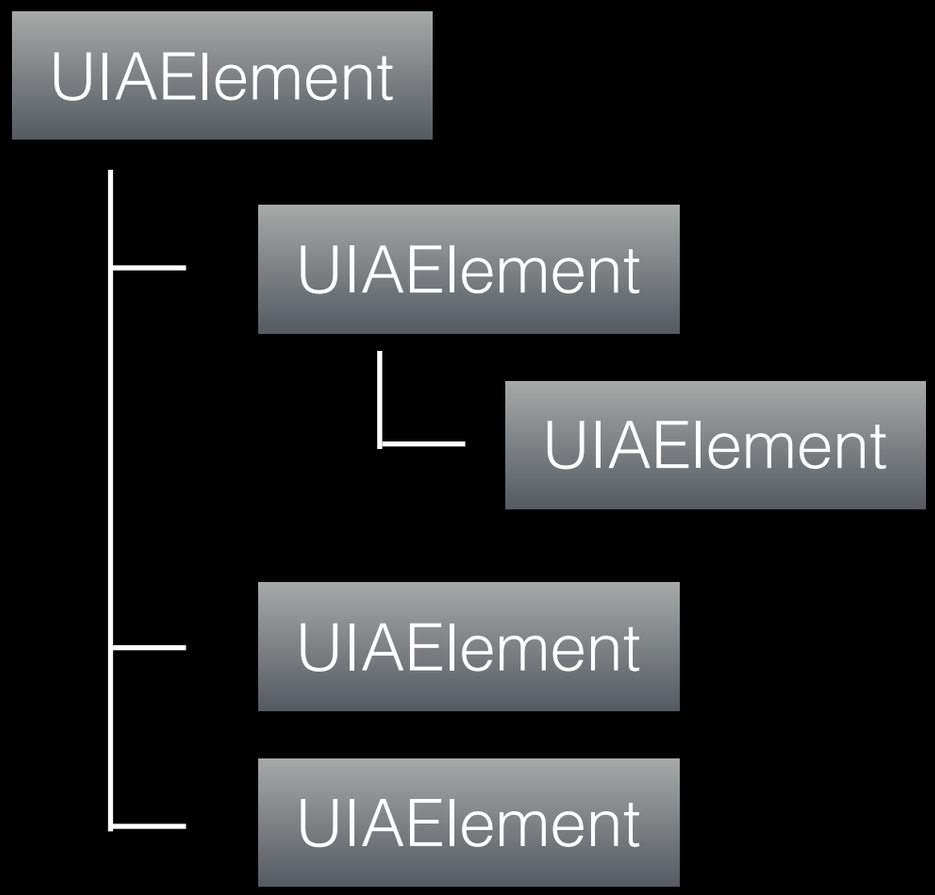
Templates Available

- Activity Monitor
- Allocations
- Automation
- Core Data
- Counters
- Energy Diagnostics
- Leaks
- Time Profiler
- Zombies

UI - Testing

Use the UI Automation javascript library to write test scripts that exercise your IOS app's user interface elements. The system returns log information.

```
if (Xcode < 7)
      {
      use instruments
      }
else
      {
      use test classes
      }
```



UI Testing in Xcode 7

```
UIATarget.localTarget().frontMostApp().mainWindow().buttons()[0]
UIATarget.localTarget().frontMostApp().mainWindow().buttons()["Edit"]
```

→ Getting a Button

```
UIATarget.localTarget().frontMostApp().mainWindow().buttons()[0].tap();
UIATarget.localTarget().frontMostApp().mainWindow().buttons()
    ["Edit"].tap();
var editButton
    =UIATarget.localTarget().frontMostApp().mainWindow().buttons()[0];
editButton.tap();
```

→ Tapping a Button

```
UIATarget.localTarget().frontMostApp().logElementTree()
UIALogger.logStart(“Test1");
UIALogger.logFail("Failed to foo.”);
UIALogger.logDebug("Done with level 3.")
```

→ Logging

```
UIATarget.onAlert = function onAlert(alert)
    {
    var title = alert.name();

    // add a warning to the log for each alert encountered
    UIALogger.logWarning("Alert with title '" + title + "' encountered!");
    UIATarget.localTarget().captureScreenWithName("alert_" + (new
    Date()).UTC());

    return true;
    }
```

→ Alert Handling

Automation

There are several command line tools that can be used to integrate building, testing and distributing iOS applications.

The command line tools come packaged with xcode. They can also be downloaded separately form member center.

Example tools:

- iprofiler
- xcode-select
- instruments
- xcrun

Examples

xcodebuild test -project MyAppProject.xcodeproj -scheme MyApp -destination 'platform=iOS,name=Development iPod touch'

xcodebuild -list -project <your_project_name>.xcodeproj

xcodebuild test -scheme MyMacApp -destination 'platform=OS X, arch=x86_64’
xcodebuild test -scheme MyiOSApp -destination 'platform=iOS Simulator,name=iPad'

```
instruments -w deviceID -t defaultTemplateFilePath targetAppName \
      -e UIASCRIPT scriptFilePath -e UIARESULTSPATH resultsFolderPath
```

References

- App Distribution Guide, https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/AppDistributionGuide/Introduction/Introduction.html#//apple_ref/doc/uid/TP40012582-CH1-SW1
- Testing with Xcode, https://developer.apple.com/library/prerelease/mac/documentation/DeveloperTools/Conceptual/testing_with_xcode/index.html#//apple_ref/doc/uid/TP40014132-CH1-SW1
- Test Navigator Help, https://developer.apple.com/library/prerelease/ios/recipes/xcode_help-test_navigator/Recipe.html#//apple_ref/doc/uid/TP40013329-CH1-SW1
- Instruments User Guide, https://developer.apple.com/library/prerelease/ios/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/TheInstrumentsWorkflow.html#//apple_ref/doc/uid/TP40004652-CH5-SW1
- Building from the Command Line, https://developer.apple.com/library/prerelease/watchos/technotes/tn2339/_index.html
- How to use instruments in Xcode, http://www.raywenderlich.com/23037/how-to-use-instruments-in-xcode
- UI Automation Javascript Reference for iOS, https://developer.apple.com/library/ios/documentation/DeveloperTools/Reference/UIAutomationRef/
- UIATarget, https://developer.apple.com/library/prerelease/ios/documentation/ToolsLanguages/Reference/UIATargetClassReference/index.html
- UIAElement, https://developer.apple.com/library/ios/documentation/ToolsLanguages/Reference/UIAElementClassReference/