XTC102

Xamarin.UITest

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Objectives

1. Create a Xamarin.UITest project
2. Create a cross-platform UI Test
3. Run UI Tests on physical devices

Create a Xamarin.UITest project

Xamarin University

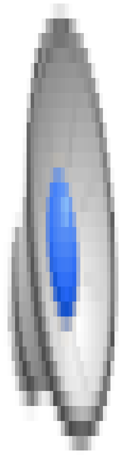
Tasks

1. Create a new UITest project
2. Use the REPL tool
3. Create a query
4. Build acceptance tests

What is Xamarin.UITest?

❖ Xamarin.UITest is a framework which lets you automate a mobile device and application

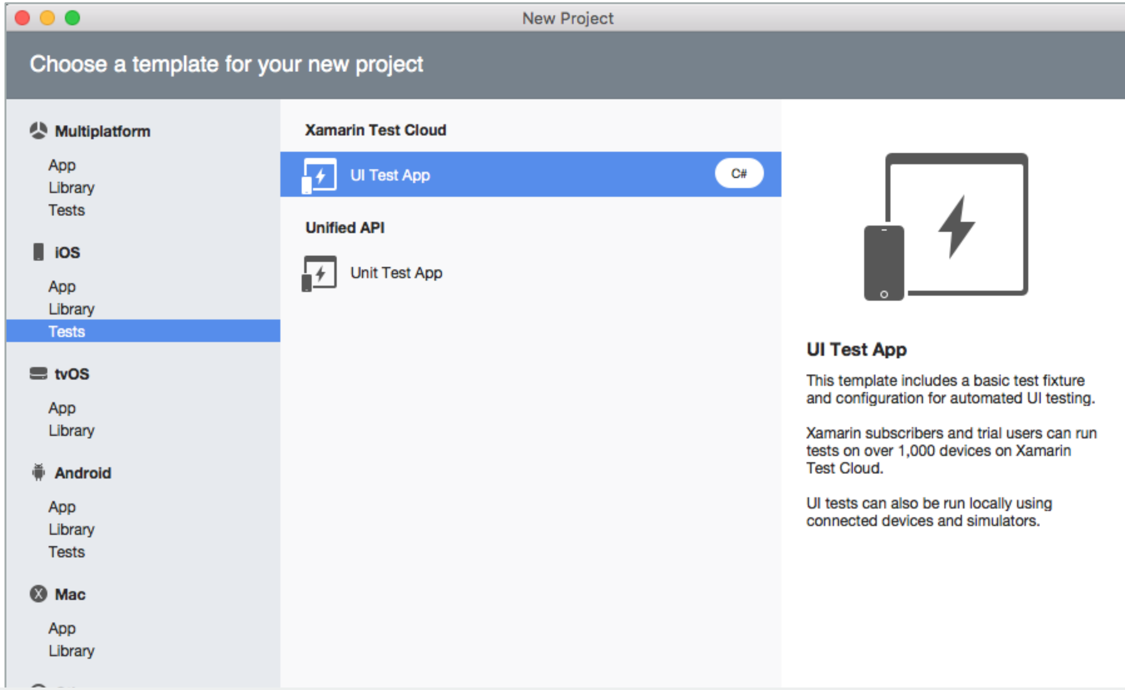
```
app.Tap (c => c.Marked ("Add"));
app.EnterText(c => c.Class("UITextField")
   .Index(0), "Get Milk");
app.Tap (c => c.Marked ("Save"));
```



Creating a Xamarin.UITest project

- Visual Studio for Mac has project templates for creating UITest projects for cross-platform, iOS and Android apps

- Creates a working project to start with, you just fill in some details and write the tests



UI Test App

This template includes a basic test fixture and configuration for automated UI testing.

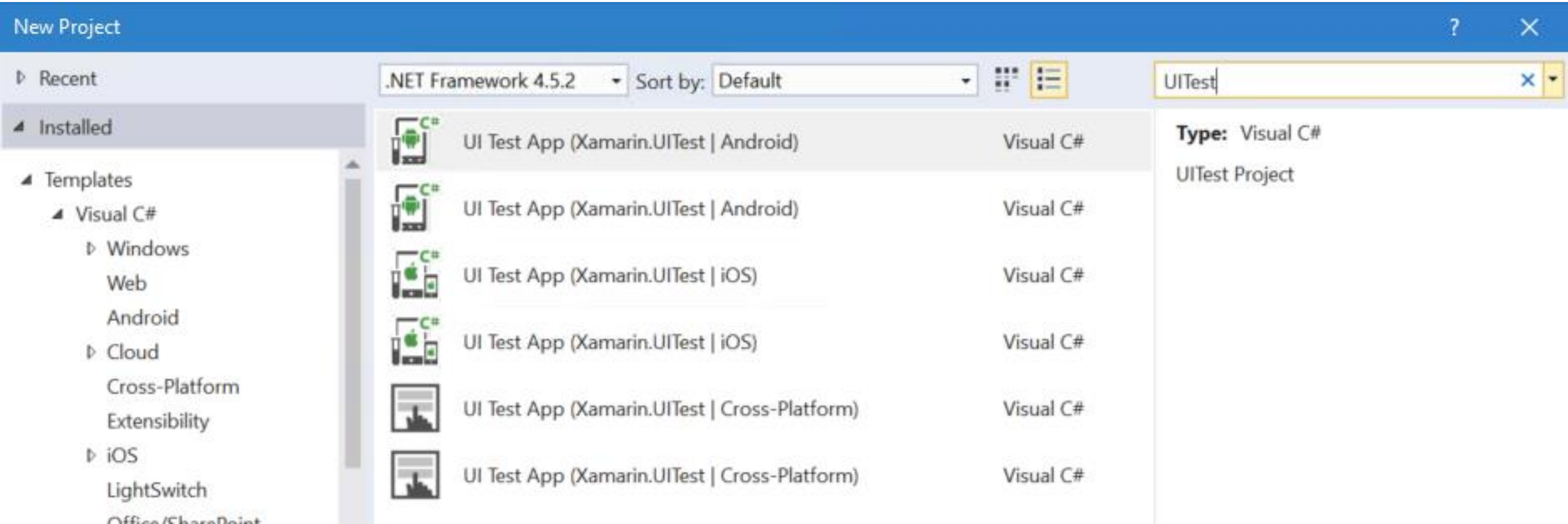
Xamarin subscribers and trial users can run tests on over 1,000 devices on Xamarin Test Cloud.

UI tests can also be run locally using connected devices and simulators.

Note: Xamarin.UITests for iOS applications can only be executed on a Mac currently; Android tests can be run on either Windows or Mac

Creating a Xamarin.UITest project

- Visual Studio has several UI test projects across multiple categories, use the Search box to show them all at once


New Project

.NET Framework 4.5.2 | Sort by: Default | UITest

- Recent
- Installed
  - Templates
    - Visual C#
      - Windows
      - Web
      - Android
      - Cloud
      - Cross-Platform
      - Extensibility
      - iOS
      - LightSwitch

UI Test App (Xamarin.UITest | Android)	Visual C#
UI Test App (Xamarin.UITest | Android)	Visual C#
UI Test App (Xamarin.UITest | iOS)	Visual C#
UI Test App (Xamarin.UITest | iOS)	Visual C#
UI Test App (Xamarin.UITest | Cross-Platform)	Visual C#
UI Test App (Xamarin.UITest | Cross-Platform)	Visual C#

Type: Visual C#

UITest Project

Running UI Tests

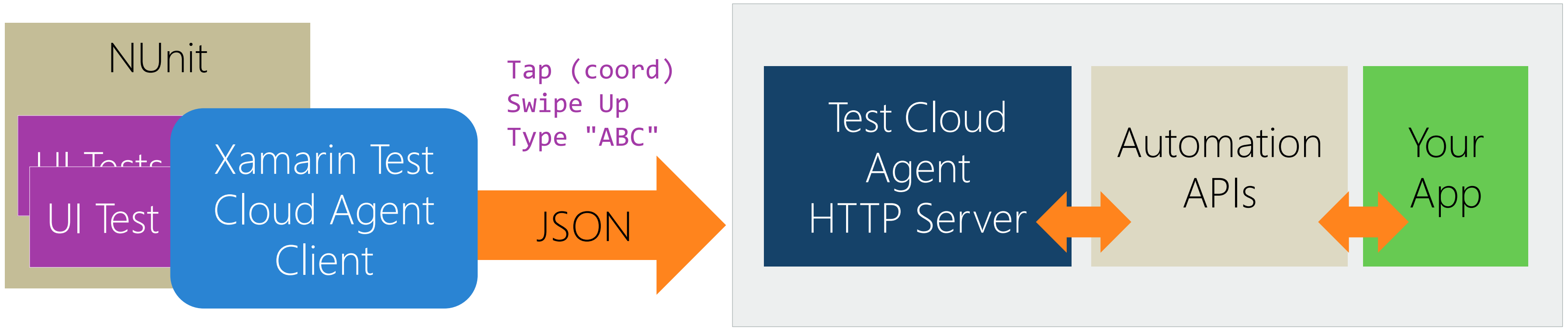
- Xamarin.UITest is a **framework of commands** you can use to *automate* an application in a cross platform fashion; the actual *testing* part is done through a unit testing framework

MSTest

Can use any test harness to *execute* the testing logic

Xamarin.UITest Architecture

- Xamarin.UITest utilizes a client/server architecture to automate your application and run the UI tests using HTTP and JSON

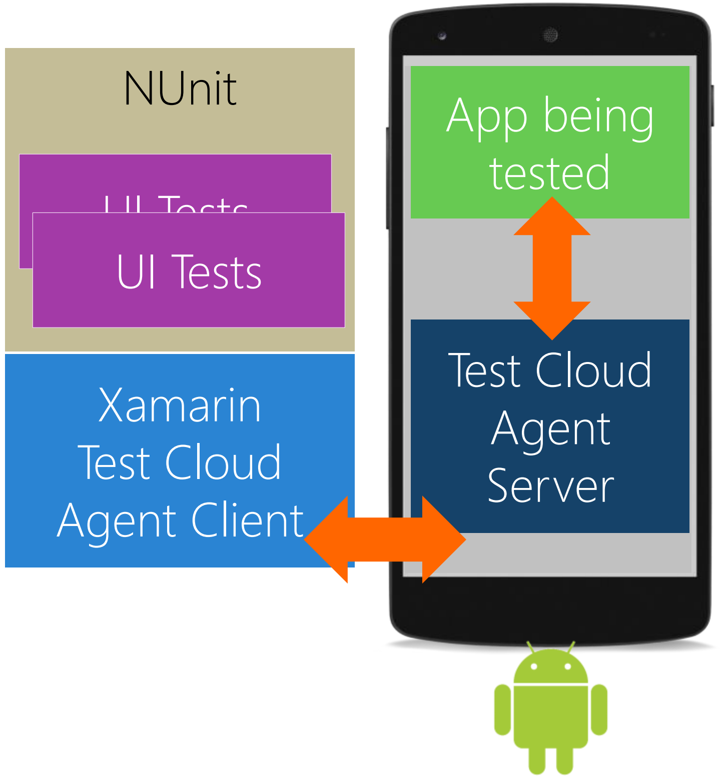


Runs on the same computer as the unit tests (e.g. desktop or cloud)

Runs on the device or simulator

Android architecture

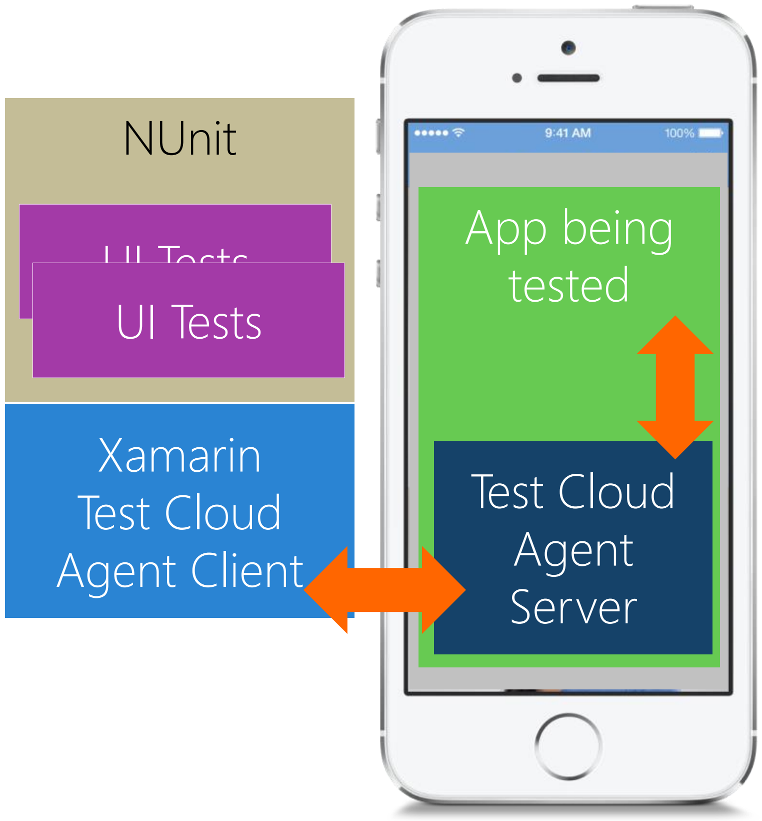
- On Android, Xamarin.UITest installs the **Xamarin Test Cloud Agent server** as a separate process
- Process is signed with the same keystore as your application so it can drive it with the Android Automation APIs



iOS architecture

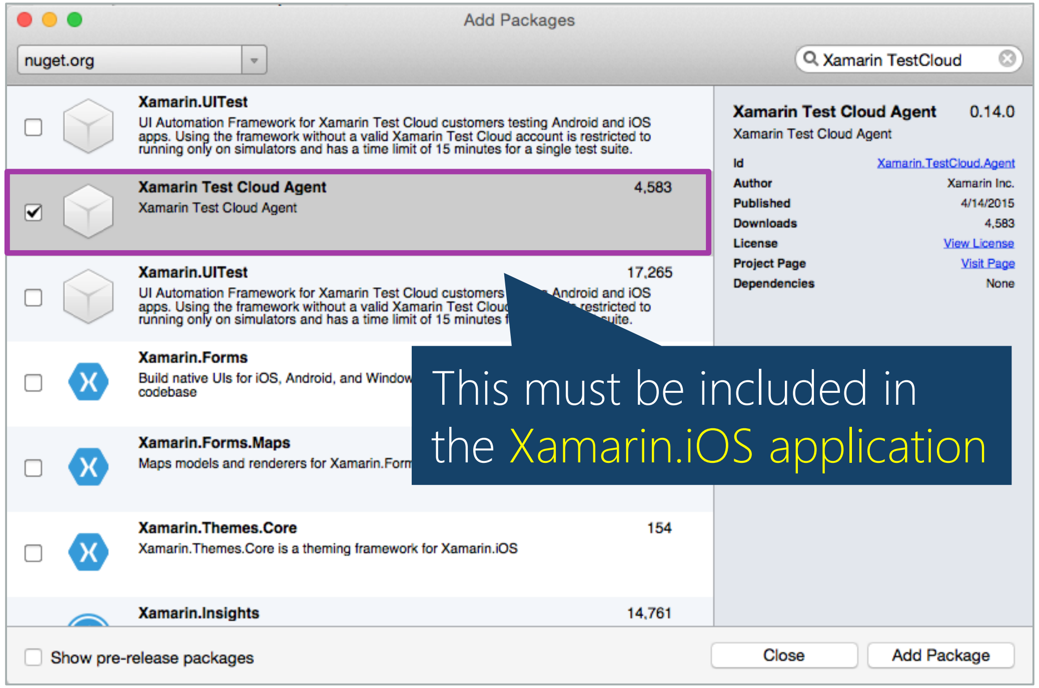
- On iOS, the **Xamarin Test Cloud Agent** component must be installed as part of the application bundle

- Since it's part of your app's process, it can utilize the iOS Automation APIs to automate the application



Automating an iOS application

- Must include the Test Cloud server as part of your iOS app that is being tested
- Can be installed through **Nuget** (preferred) or the Xamarin Component Store



Add Packages

nuget.org | Xamarin TestCloud

	Package	Downloads
☐	**Xamarin.UITest** UI Automation Framework for Xamarin Test Cloud customers testing Android and iOS apps. Using the framework without a valid Xamarin Test Cloud account is restricted to running only on simulators and has a time limit of 15 minutes for a single test suite.	
☑	**Xamarin Test Cloud Agent** Xamarin Test Cloud Agent	4,583
☐	**Xamarin.UITest** UI Automation Framework for Xamarin Test Cloud customers ... Android and iOS apps. Using the framework without a valid Xamarin Test Cloud ... restricted to running only on simulators and has a time limit of 15 minutes f... suite.	17,265
☐	**Xamarin.Forms** Build native UIs for iOS, Android, and Window... codebase	
☐	**Xamarin.Forms.Maps** Maps models and renderers for Xamarin.Form...	
☐	**Xamarin.Themes.Core** Xamarin.Themes.Core is a theming framework for Xamarin.iOS	154
	Xamarin.Insights	14,761

Xamarin Test Cloud Agent 0.14.0
Xamarin Test Cloud Agent

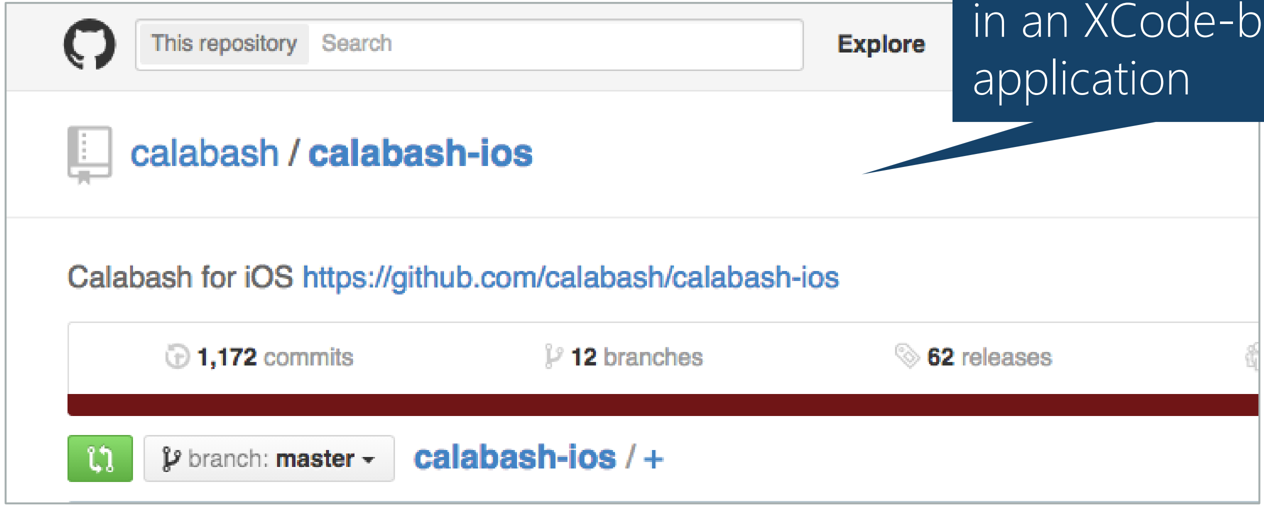
Id	Xamarin.TestCloud.Agent
Author	Xamarin Inc.
Published	4/14/2015
Downloads	4,583
License	View License
Project Page	Visit Page
Dependencies	None

☐ Show pre-release packages

This must be included in the Xamarin.iOS application

Automating an iOS application

❖ Native iOS applications written in Objective-C or Swift can download **Calabash** from Github and **install it through a script**



calabash / calabash-ios

Calabash for iOS https://github.com/calabash/calabash-ios

1,172 commits 12 branches 62 releases

calabash-ios / +

This must be included in an XCode-based application

Starting the Automation Server

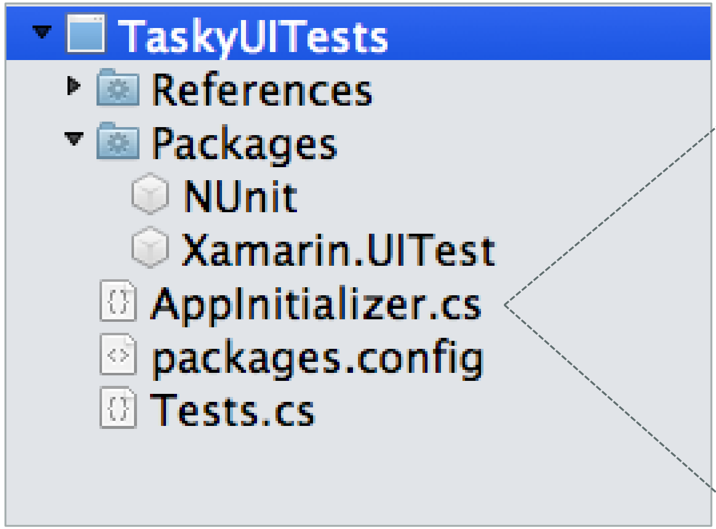
- Add code to start the Calabash server in your Xamarin.iOS application into the **FinishedLaunching** method

```
public override bool FinishedLaunching ( ... ) {
   ...
   #if ENABLE_TEST_CLOUD
   Xamarin.Calabash.Start();
   #endif
}
```

Setup is different for native Objective-C or Swift apps – check the calabash-ios Github readme for information on incorporating the server into your app

UITest project structure

- Template will create a test class with a `[SetUp]` step to initialize UITest; the contents vary based on the project style (Mobile vs. iOS vs. Android)



```
public class AppInitializer
{
   public static IApp StartApp(Platform platform)
   {
      if (platform == Platform.Android) {
         // ... Android init ...
      }
      // ... iOS init ...
}
```

Group Exercise

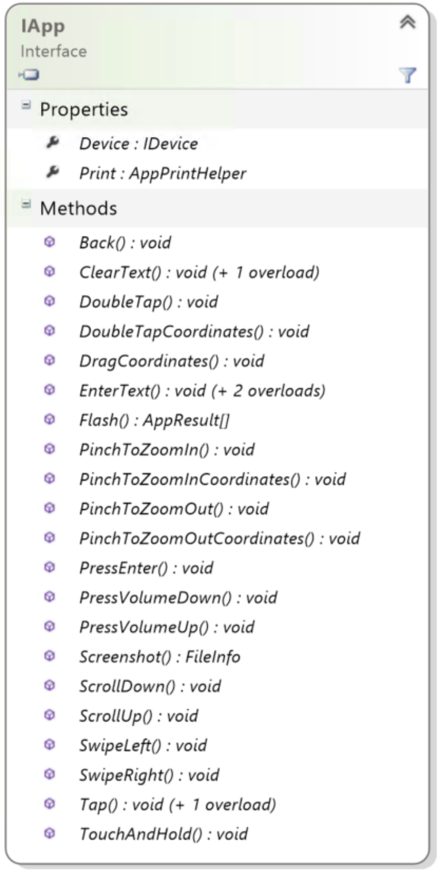
Creating a UITest Project

Interacting with UITest

- Testing API is provided through **`IApp`** interface which defines the methods used to interact with the app's UI

- Two implementations available today
  - `iOSApp`
  - `AndroidApp`

- Implementations obtained through static builder class **`ConfigureApp`**



IApp
Interface

Properties
- *Device : IDevice*
- *Print : AppPrintHelper*

Methods
- *Back() : void*
- *ClearText() : void (+ 1 overload)*
- *DoubleTap() : void*
- *DoubleTapCoordinates() : void*
- *DragCoordinates() : void*
- *EnterText() : void (+ 2 overloads)*
- *Flash() : AppResult[]*
- *PinchToZoomIn() : void*
- *PinchToZoomInCoordinates() : void*
- *PinchToZoomOut() : void*
- *PinchToZoomOutCoordinates() : void*
- *PressEnter() : void*
- *PressVolumeDown() : void*
- *PressVolumeUp() : void*
- *Screenshot() : FileInfo*
- *ScrollDown() : void*
- *ScrollUp() : void*
- *SwipeLeft() : void*
- *SwipeRight() : void*
- *Tap() : void (+ 1 overload)*
- *TouchAndHold() : void*

Configuring UITest

- **`ConfigureApp`** is used to initialize and configure UITest; this should be done prior to each test to keep the tests independent

```
IApp app;  // Field used by each [Test] method

[SetUp]
public void BeforeEachTest() {
   app = ConfigureApp.iOS       or
   app = ConfigureApp.Android
   ...
}
```

First step is to identify the platform, can be either iOS or Android

Selecting the application to test

- UITest runs tests against a specific, running application; can identify that application in several ways:

Bundle (.app), IPA or APK

Can specify the full path on the local disk to a built iOS or Android application

Selecting an app bundle or package

- Use the **`AppBundle`** or **`ApkFile`** method to identify a binary to test – this is installed on the simulator/emulator/device and then tests are executed

```
app = ConfigureApp.iOS
        .AppBundle("../../path/mybundle.app");
```

```
app = ConfigureApp.Android
        .ApkFile("../../path/myapp.apk");
```

Must supply the full path leading up to the binary; can use relative paths for projects in the same solution - starting at the UITest binary output folder

Working with UITest

- UITest runs tests against a specific, running application; can identify that application in several ways:

Bundle (.app), IPA or APK

Installed App name

Can identify a specific package or bundle identifier for an installed application

Select an installed application

❖ Use the `InstalledApp` method to identify an application that is **already installed** on the simulator/emulator/device

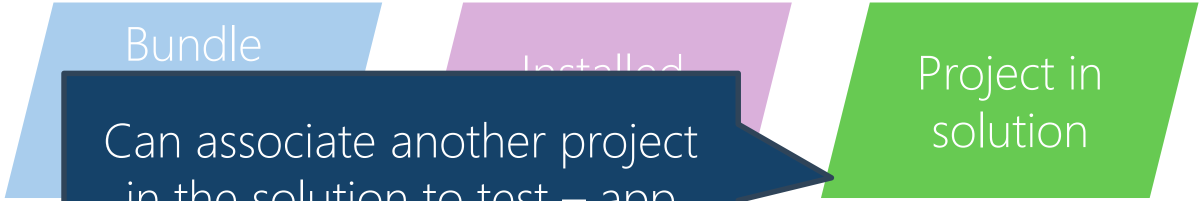
```
app = ConfigureApp.iOS  // or Android
        .InstalledApp("com.xamarin.taskypro");
```

↑

pass the package name or bundle identifier to specify the application

Working with UITest

- UITest runs tests against a specific, running application; can identify that application in several ways:

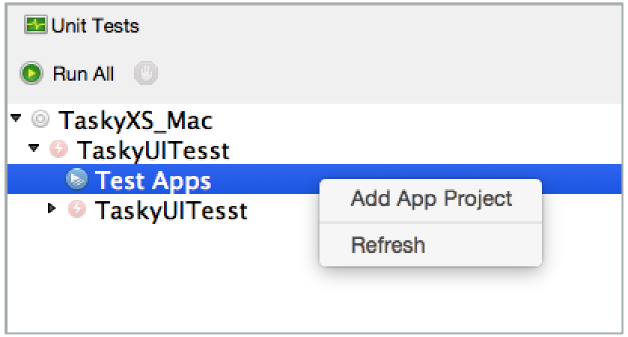


Can associate another project in the solution to test – app project must be in the same folder as the UITest project

Xamarin University

Selecting a project in your solution

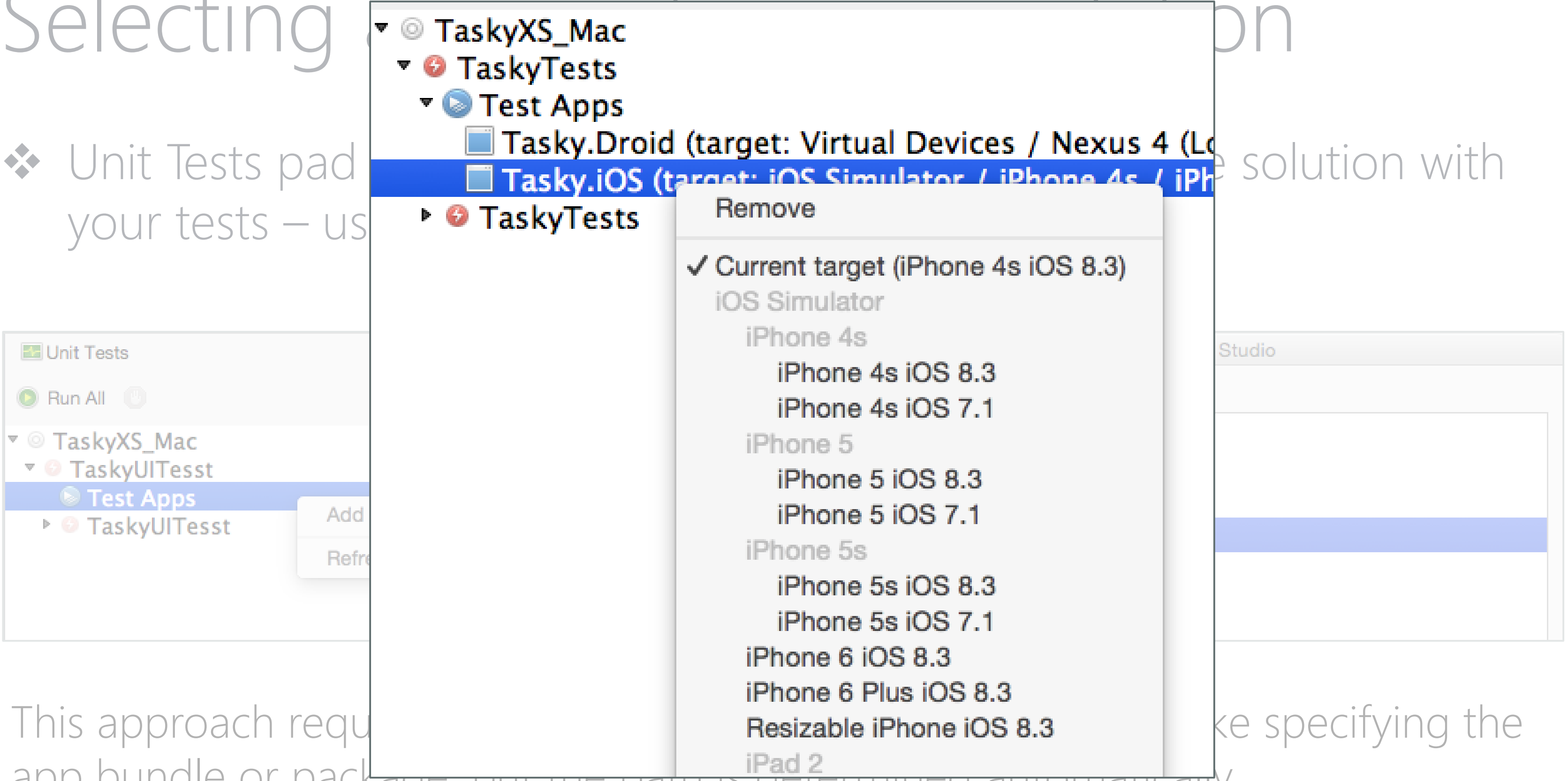
- Unit Tests pad has UI to associate another project in the solution with your tests – use the **Test Apps** section



This approach requires no code to launch the application – it's like specifying the app bundle or package, but the path is determined automatically

Selecting ... on

- Unit Tests pad ... e solution with your tests – us...



Unit Tests

Run All

- TaskyXS_Mac
  - TaskyUITesst
    - Test Apps
    - TaskyUITesst

- TaskyXS_Mac
  - TaskyTests
    - Test Apps
      - Tasky.Droid (target: Virtual Devices / Nexus 4 (L
      - Tasky.iOS (target: iOS Simulator / iPhone 4s / iPh
    - TaskyTests

This approach requ... ke specifying the app bundle or package, but the path is determined automatically

Starting your app

- Last step in the configuration is the *start* the application, this will launch the application on the simulator/emulator/device

```
IApp app;

[SetUp]
public void BeforeEachTest()
{
   app = ConfigureApp.Android
             .ApkFile("../../path/myapp.apk")
             .StartApp();
}
```

Writing tests

- UITest uses **NUnit** to execute the tests, but these are *not* unit tests

```
[TestFixture]
public class TaskyProBasicTests
{
    [Test]
    public void AddMilk_ShouldShowMilkInTasks()
    {
        ...
    }
}
```

Writing tests

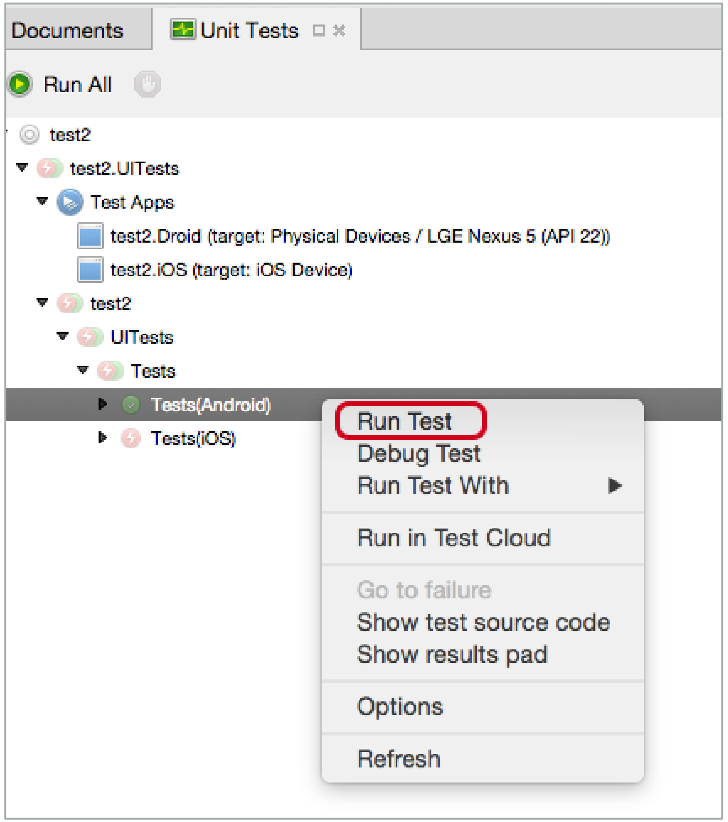
- Use the **`IApp`** interface to interact directly with your application UI: locating elements, tapping, typing, gestures and more

```
[Test]
public void AddMilk_ShouldShowMilkInTasks()
{
   app.Tap (c => c.Marked ("Add"));
   app.EnterText(c => c.Class("UITextField")
                        .Index(0),"Get Milk");
   app.Tap (c => c.Marked ("Save"));
}
```

Running the test

- Can run tests either in the IDE or from the command line (they are just unit tests)
  - `nunit-console.exe`

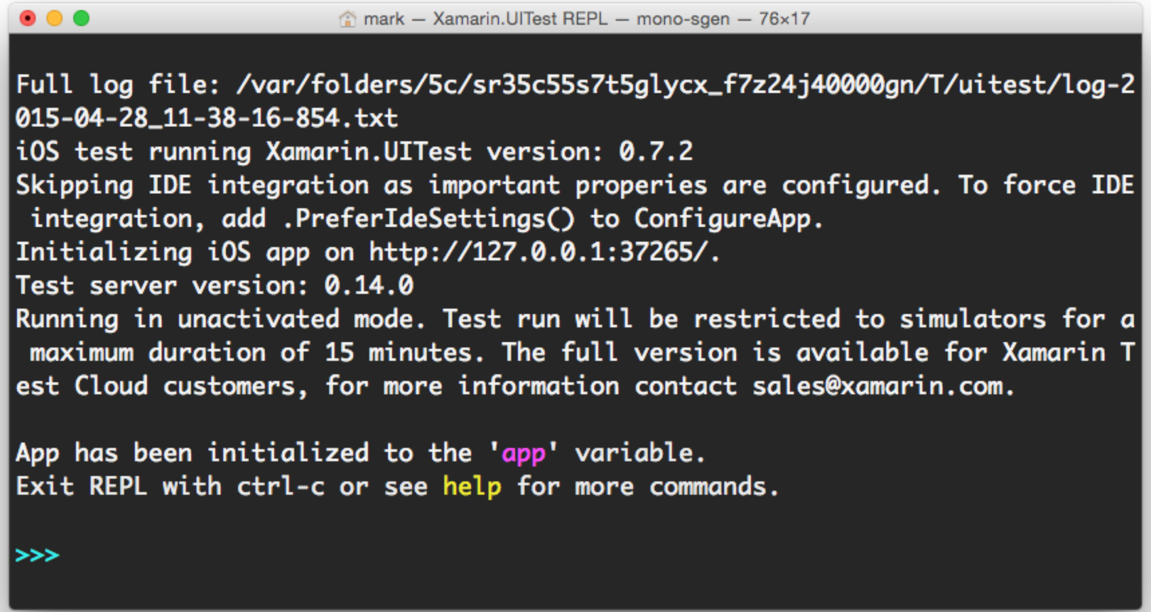
- iOS UI Tests can only be run from Visual Studio on the Mac, but Android is supported on both platforms and IDEs



Using the REPL

- Built in REPL (Read-Evaluate-Print-Loop) allows you to explore and manipulate the running application interactively through a runtime shell

```
app.Repl();
```



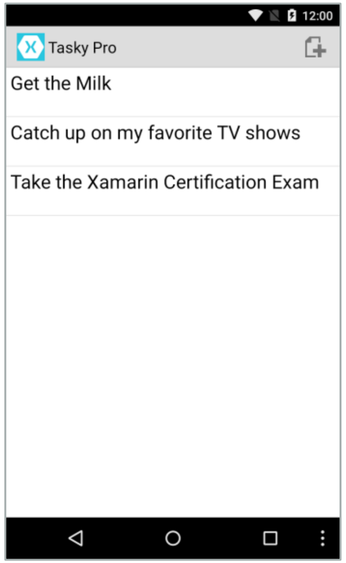
```
Full log file: /var/folders/5c/sr35c55s7t5glycx_f7z24j40000gn/T/uitest/log-2
015-04-28_11-38-16-854.txt
iOS test running Xamarin.UITest version: 0.7.2
Skipping IDE integration as important properies are configured. To force IDE
 integration, add .PreferIdeSettings() to ConfigureApp.
Initializing iOS app on http://127.0.0.1:37265/.
Test server version: 0.14.0
Running in unactivated mode. Test run will be restricted to simulators for a
 maximum duration of 15 minutes. The full version is available for Xamarin T
est Cloud customers, for more information contact sales@xamarin.com.

App has been initialized to the 'app' variable.
Exit REPL with ctrl-c or see help for more commands.

>>>
```

Examining the UI with the REPL

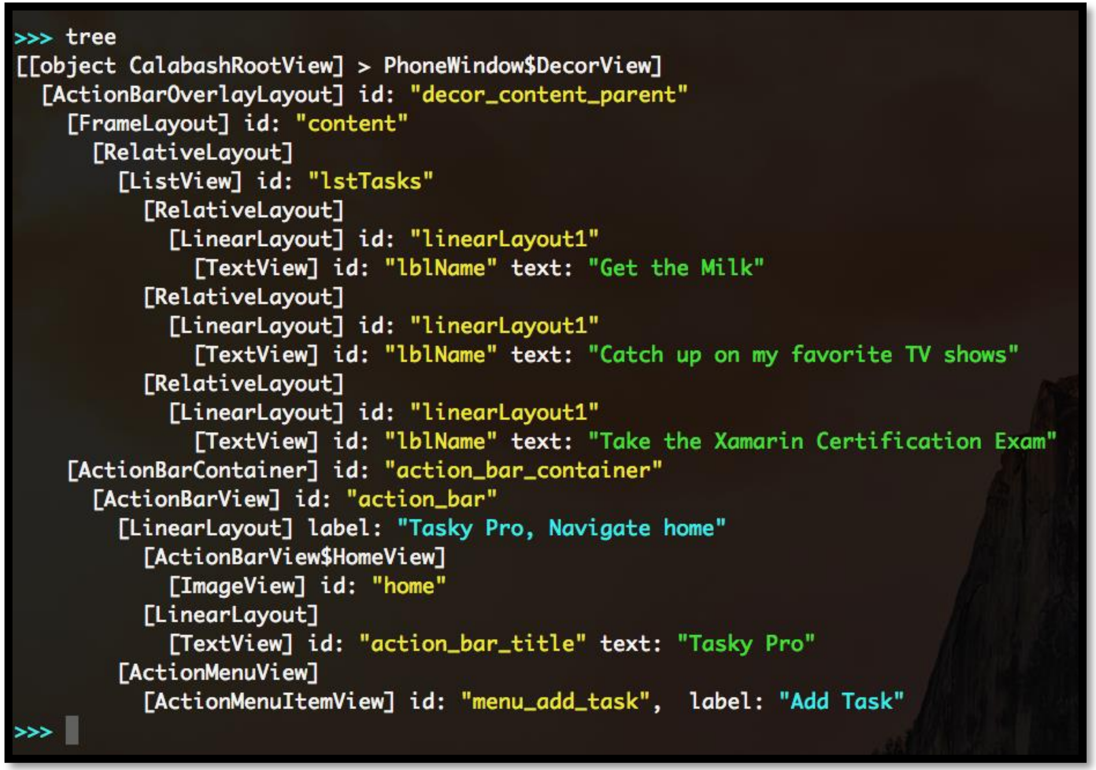
- Use **Tree** to dump the visual tree for your app



Get the Milk

Catch up on my favorite TV shows

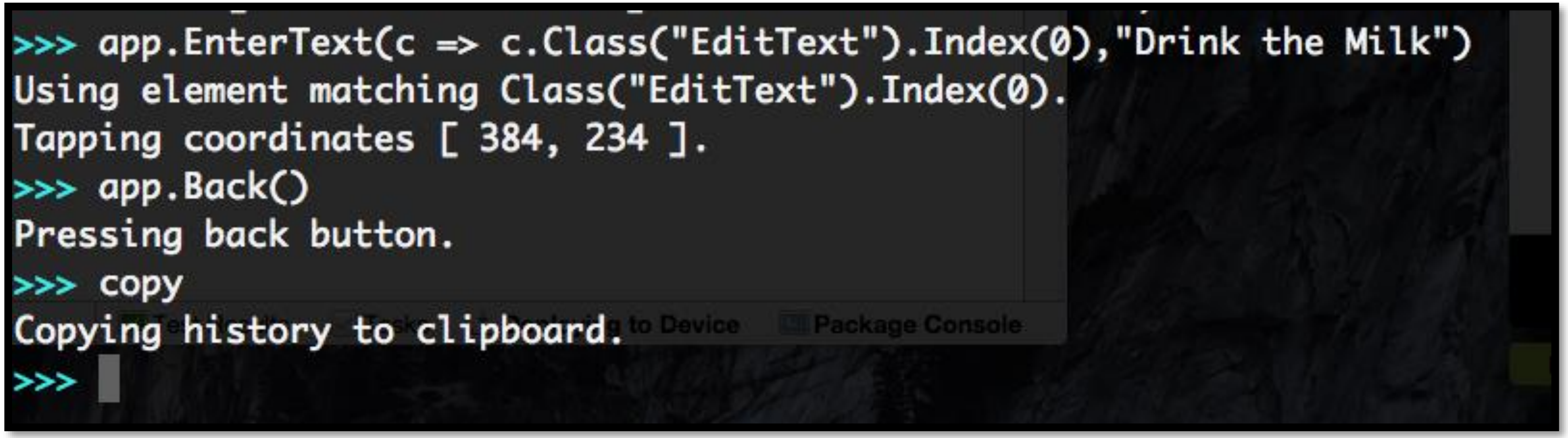
Take the Xamarin Certification Exam



```
>>> tree
[[object CalabashRootView] > PhoneWindow$DecorView]
  [ActionBarOverlayLayout] id: "decor_content_parent"
    [FrameLayout] id: "content"
      [RelativeLayout]
        [ListView] id: "lstTasks"
          [RelativeLayout]
            [LinearLayout] id: "linearLayout1"
              [TextView] id: "lblName" text: "Get the Milk"
          [RelativeLayout]
            [LinearLayout] id: "linearLayout1"
              [TextView] id: "lblName" text: "Catch up on my favorite TV shows"
          [RelativeLayout]
            [LinearLayout] id: "linearLayout1"
              [TextView] id: "lblName" text: "Take the Xamarin Certification Exam"
    [ActionBarContainer] id: "action_bar_container"
      [ActionBarView] id: "action_bar"
        [LinearLayout] label: "Tasky Pro, Navigate home"
          [ActionBarView$HomeView]
            [ImageView] id: "home"
          [LinearLayout]
            [TextView] id: "action_bar_title" text: "Tasky Pro"
        [ActionMenuView]
          [ActionMenuItemView] id: "menu_add_task",  label: "Add Task"
>>>
```

Copying the REPL data into a test

- Can use the **copy** command in the REPL to copy your command history to the clipboard, then use this text as the basis for a UI test



```
>>> app.EnterText(c => c.Class("EditText").Index(0),"Drink the Milk")
Using element matching Class("EditText").Index(0).
Tapping coordinates [ 384, 234 ].
>>> app.Back()
Pressing back button.
>>> copy
Copying history to clipboard.
>>>
```

Demonstration

Working with the REPL

Xamarin University

API Commands

❖ Xamarin.UITest has a rich API that allows for complete interrogation and interaction with the application

Commonly used methods		
`Query`	`Tap`	`WaitFor`
`WaitForElement`	`WaitForNoElement`	`Screenshot`
`SwipeLeft`	`SwipeRight`	`ScrollUp`
`ScrollDown`	`Flash`	

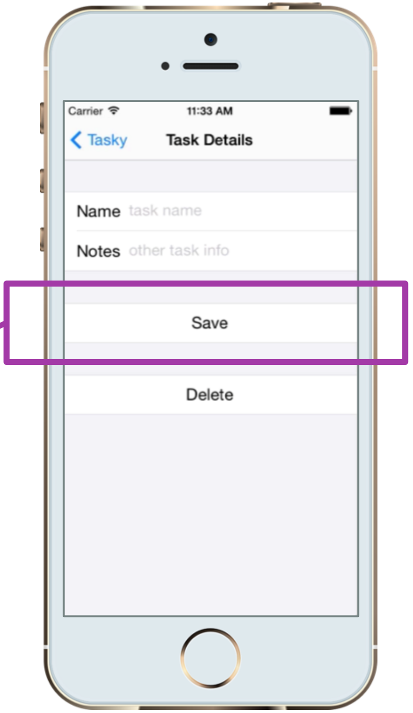
Identifying UI elements

- Most APIs are executed on a single UI element; can identify visible elements through a query

Returns an array of zero or more UI elements that match the provided filter

```
AppResult[] matchedItems =
     app.Query(
          c => c.Button().Marked("Save"));
```

Query identifies one or more visible elements on your current screen – typically through text or an id



Identifying UI elements

- Two ways to identify elements in your UI, can use them independently or together to be very specific with your query

Class Queries

Identify UI elements based on the specific control ("class") type

Marked Selectors

Identify UI elements using unique identifiers or associated text property

Class Queries

- Class queries are used to identify UI elements based on type

```
var matches = app.Query(c => c.Class("UILabel"));
```

```
var matches = app.Query(c => c.Class("TextView"));
```

helper methods provide abstraction over common platform types

```
var matches = app.Query(c => c.Button());
```

```
var matches = app.Query(c => c.TextField());
```

Marked Selectors

- **Marked selectors** identify elements based on text or id, often used together with class queries to uniquely identify an element

```
var matches =
     app.Query(c => c.Button().Marked("Save"));
```

① Get all buttons (`UIButton` or Android `Button`) in the UI

② Return any button with the text `"Save"`

Should prefer to identify visual elements using unique id instead of text values – text tends to change over time (or for localization), ids will remain constant

Marked Selectors

- **Marked selectors** identify elements based on text or id, often used together with class queries to uniquely identify an element

```
var matches =
    app.Query(c => c.Button().Marked("Save"));
```

or

```
var matches =
    app.Query(c => c.Button("Save"));
```

Shorthand syntax allows all queries to take a string – this turns into a `Marked` selector

Xamarin University

Commonly used queries

- Queries are used in Xamarin.UITest to locate and interact with the application's user interface

Query	What does the query do?
`app.Query();`	Selects all visible elements
`app.Tap (c => c.Id ("MyButton"));`	Selects all visible controls with the identifier `"MyButton"`
`app.Tap ("Click me");`	Selects all visible elements with the text or id `"Click me"`
`app.Flash (c => c.Button ());`	Flash all the visible buttons

Commonly used queries

Query	What does the query do?
`app.Query (c => c.Class("UILabel"));`	Selects all visible **`UILabel`**s
`app.Query (c => c.All());`	Selects all controls, visible and invisible
`app.Query (c => c.Id ("MyWeb").Css("input"));`	Selects the items that match the CSS selector **`"input"`** on the Web View called **`"MyWeb"`**

Writing Acceptance Tests

Arrange

Launch the Application

```
IApp app = ConfigureApp.iOS.AppBundle ("PathToIPA.app").StartApp ();
```

Act

Perform UI queries against the application through IApp

```
app.Tap (c => c.Marked ("Add"));
```

Assert

Verify the user interface reflects what you should see

```
Assert.IsTrue(app.Query (c => c.Marked ("entry_field")).Length > 0,
                "No text value was added to the entry field on Add");
```

Flash Quiz

Flash Quiz

① The REPL Tool can perform what operations (choose all that apply)

a) Query the User Interface
b) List the contents of the View Hierarchy
c) Copy previous operations to the clipboard
d) All of the above

Flash Quiz

① The REPL Tool can perform what operations (choose all that apply)

a) Query the User Interface

b) List the contents of the View Hierarchy

c) Copy previous operations to the clipboard

d) <u>All of the above</u>

Flash Quiz

② The command to type in a control is

a) TypeText

b) EnterText

c) UseKeyboard

Flash Quiz

② The command to type in a control is

a) TypeText

b) EnterText

c) UseKeyboard

Flash Quiz

③ The Query operation by itself will show all controls, both visible and invisible on the device

a) True

b) False

Flash Quiz

③ The Query operation by itself will show all controls, both visible and invisible on the device

a) True

b) <u>False</u>

Individual Exercise

Creating acceptance tests with Xamarin.UITest

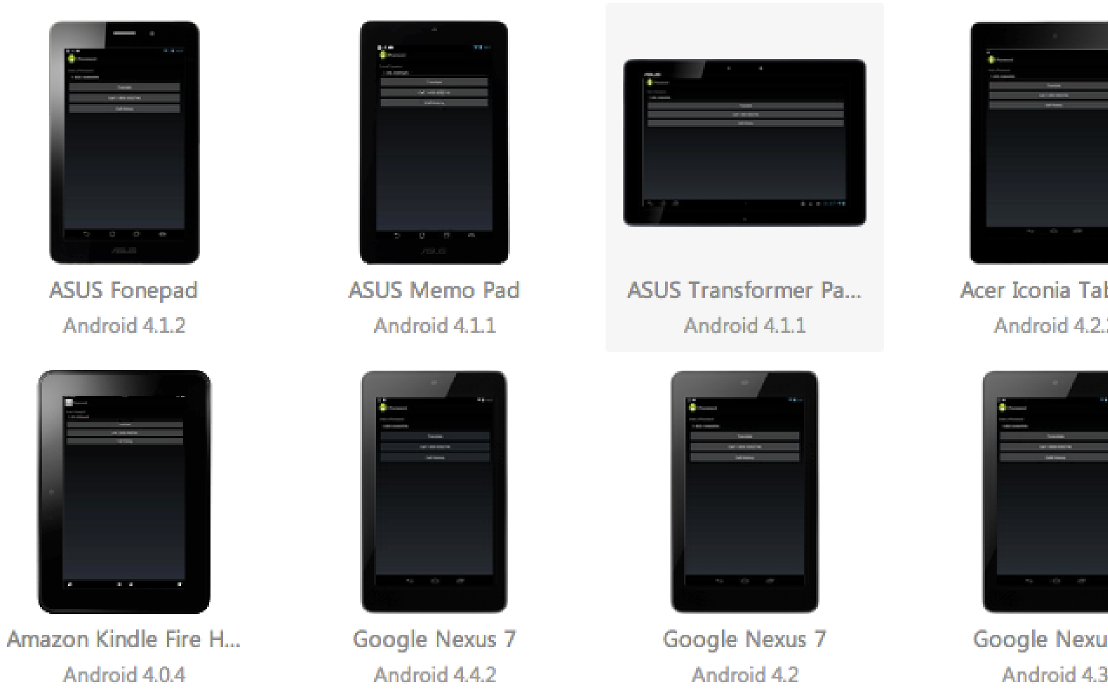
Xamarin University

Create a cross-platform UI Test

Xamarin University

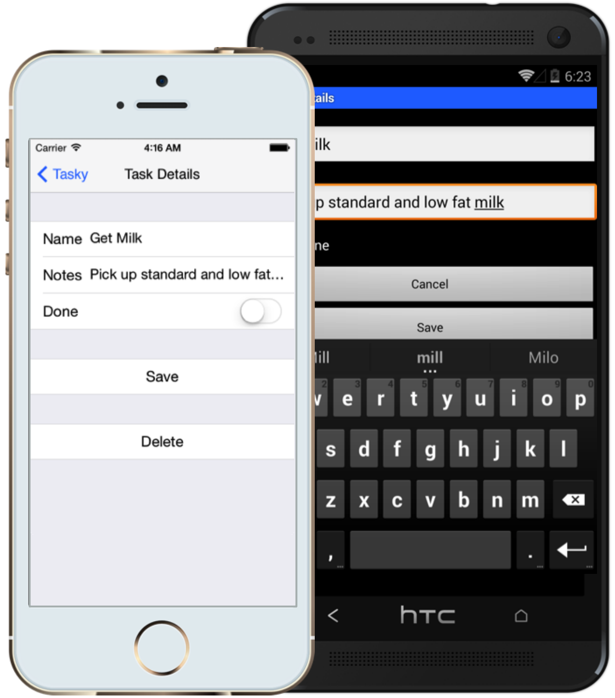
Tasks

1. Cross-platform tests
2. Platform differences
3. Advanced operations



Creating cross-platform tests

- UI is often constructed uniquely on each platform
- Unique tests are appropriate for many cases
- Ideally could run the same logical tests on all the platforms but have each test compensate for the unique UI presented



Cross-platform testing

- Process of identifying the UI to test changes as you move from iOS to Android, this means *two areas* will be affected in your test code

Class Queries

Marked Selectors

Class queries

- Available controls are different on each platform, so our class queries will often need to change to properly identify the UI to test

```
app.Query (c => c.Class("UILabel"));
```

vs.

```
app.Query (c => c.Class("android.widget.TextView"));
```

Marked selector

- Warning: `Marked` selector works *differently* between platforms

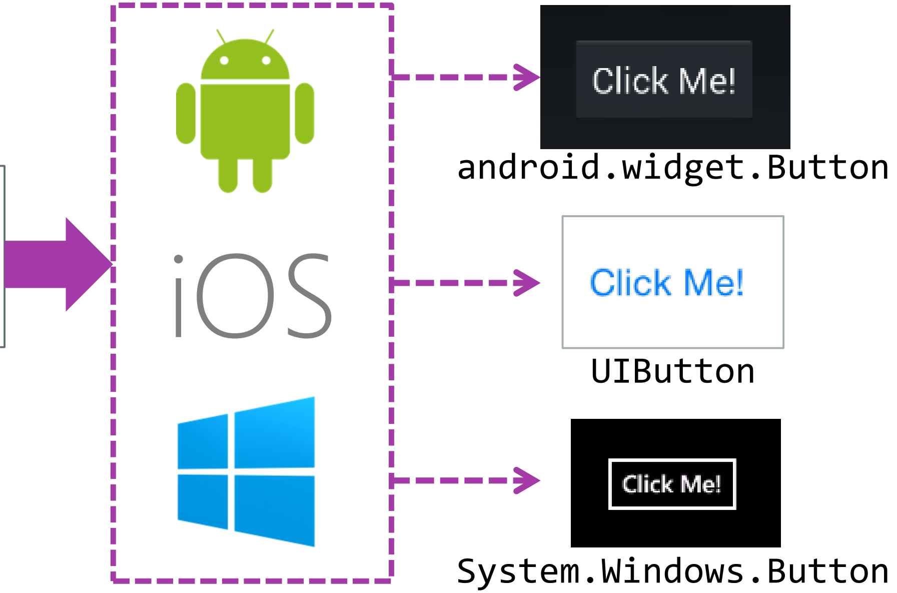
On iOS, it matches against the `AccessibilityIdentifier` and `AccessibilityLabel` of the `UIView`

On Android, the marked selector matches against the `Id`, `ContentDescription`, and `Text` of each view

Adding support for Xamarin.Forms

- Xamarin.Forms renders UI for you based on the logical tree of controls you create in code or XAML

```
Button button = new Button {
    Text = "Click Me!"
};
```



Adding support for Xamarin.Forms

- Can add **AutomationId** to each control to enable cross-platform lookup using **Marked** selector

```
var b = new Button {
    Text = "Click me",
    AutomationId = "MyButton"
};
var l = new Label {
    Text = "Hello, Xamarin.Forms!",
    AutomationId = "MyLabel"
};
```

```
<Button Text="Click Me"
        AutomationId="MyButton" />
<Label Text="Hello, Xamarin.Forms!"
        AutomationId = "MyLabel" />
```

Only iOS and Android are supported

Demonstration

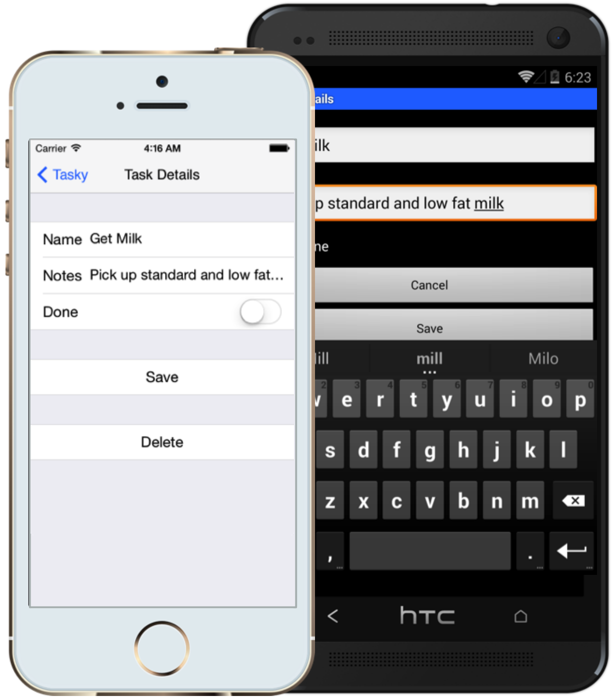
Show the platform differences

Xamarin University

Abstracting our tests

- Can use isolation and abstraction techniques such as interfaces, partial classes and conditional code to define the unique *non-sharable* elements

- Variety of ways to structure this – can be as complex or as simple as you need it to be (and are willing to maintain)



Detecting the platform

- Platform enumeration passed to constructor of test – can be used to create platform-specific setup code

```
[TestFixture (Platform.Android), [TestFixture (Platform.iOS)]
public class Tests
{
   IApp app;
   Platform platform;

   public Tests(Platform platform) {
      this.platform = platform;
      if (platform == Platform.iOS) { ... }
      else if (platform == Platform.Android) { ... }
   }
}
```

Creating cross-platform tests #1

```
readonly Func<AppQuery,AppQuery> AddButton;
readonly Func<AppQuery,AppQuery> NameField;
```

```
if (platform == Platform.iOS) {
   AddButton = c => c.Button("Add");
   ...
} else {
   AddButton = c => c.Marked("Add Task");
   ...
}
```

```
app.Tap(AddButton);
app.EnterText(NameField, name);
```

- Create unique platform-specific queries to identify the UI element the tests need to access
- … then use these captured queries in your tests

Individual Exercise

Creating a cross-platform UITest #1

Xamarin University

Creating cross-platform tests #2

- Define an interface to abstract the higher functions needed for testing
- Tests use abstraction to access and test screen features – testing logic is completely shared

```
public interface IEnterTaskScreen
{
    IEnterTaskScreen SetName(string name);
    IEnterTaskScreen SetNotes(string notes);
    IEnterTaskScreen MarkAsDone();
    IEnterTaskScreen Cancel();
    IEnterTaskScreen Save();
}
```

```
IEnterTaskScreen MainTaskScreen = ...;
MainTaskScreen
  .SetName("Get Milk")
  .SetNotes("Buy standard and low fat milk")
  .Save();
```

Waiting for UI activity

- Waiting for a fixed amount of time changes the way that you would realistically wait between device differences

- Can also differ from device to device, based upon processor speed, network connectivity, etc.

Proper way to wait for UI

- Tests should not block the test thread, instead, wait for some UI element to appear or disappear before continuing the test

```
app.WaitForElement("add_item",
    "The button to add an item did not appear",
    TimeSpan.FromSeconds(5));
```

This waits 5 seconds for an element with the text/id "Add Item" to appear in the visual tree of the application

Individual Exercise

Creating a cross-platform UITest #2

Taking screenshots

- Test methods can take a screenshot to facilitate manually test verification

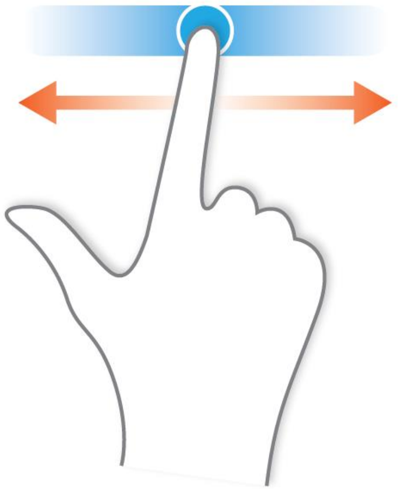
```
return ConfigureApp
       .Android  // or .iOS
       .EnableLocalScreenshots()
       .StartApp();
```

Must be enabled for the local testing scenarios as part of the configuration chain

```
[Test]
public void WelcomeTextIsDisplayed() {
   app.Screenshot(
         "Welcome Text is Displayed");
   ...
}
```

Gesture support

- Xamarin.UITest supports a limited set of gestures for touch interaction



Method	
`SwipeLeft`	`DoubleTap`
`SwipeRight`	`DragCoordinates`
`ScrollUp`	`TwoFingerTap`
`ScrollDown`	`FlickCoordinates`
`PinchToZoomIn`	`PinchToZoomOut`
`TouchAndHold`	`...`

Using coordinate-based gestures

- Basic gestures are applied within a visual element – if you want to cross multiple elements, then use the coordinate gestures

```
void SwipeLeftFromCenter(string containerId) {
   int width, height;
   GetHeightWidth(app, x => x.Marked(containerId), out width, out height);
   app.DragCoordinates (width / 2, height / 2 , 0, height / 2);
}

static bool GetHeightWidth(IApp app, Func<AppQuery, AppQuery> query, out int outX, out int outY) {
   outX = outY = 0;
   AppResult[] queryResult = app.Query(query);
   if (queryResult != null) {
      AppResult result = queryResult[0];
      outX = Convert.ToInt32(result.Rect.Width);
      outY = Convert.ToInt32(result.Rect.Height);
   }
}
```

Hybrid app support

- Xamarin.UITest supports testing **hybrid apps** which are HTML pages embedded in a native application shell, however this tends to be hard to portably test as the browser device capabilities differ significantly

```
app.Tap(x => x.Id("my-webview").Css("#my-button"));
```

Takes a regular CSS selector query – this would tap a button with the id "my-button"

Invoking methods directly

- **`IApp.Invoke`** lets you call methods on the **`AppDelegate`** (iOS) and running **`Activity`** (Android)
- Provides a "backdoor" to setup specific testing scenarios without driving the UI
- Xamarin methods must be marked with **`[Export]`** to expose them to the OS runtime

AppDelegate.cs

```
[Export("testMethod:")]
public NSString TestMethod(NSString arg)
 ...
```

AndroidApp.cs

```
[Export]
public string testMethod(string arg)
 ...
```

```
[Test]
public void SetupTest()
{
   app.Invoke("testMethod");
}
```

Flash Quiz

Flash Quiz

① Changing platforms or idioms do not require you to alter tests

a) True

b) False

Flash Quiz

① Changing platforms or idioms do not require you to alter tests

a) True

b) [✓ False]

Flash Quiz

② What is the purpose of the `Class` selector

a) It looks for exactly the same class, including the namespace

b) It looks for full implementations of the classes interface

c) It looks for implementations of the class fully visible on the screen

Flash Quiz

② What is the purpose of the `Class` selector

a) It looks for exactly the same class, including the namespace

b) It looks for full implementations of the classes interface

c) It looks for implementations of the class fully visible on the screen

Run UI Tests on physical devices

Xamarin University

Tasks

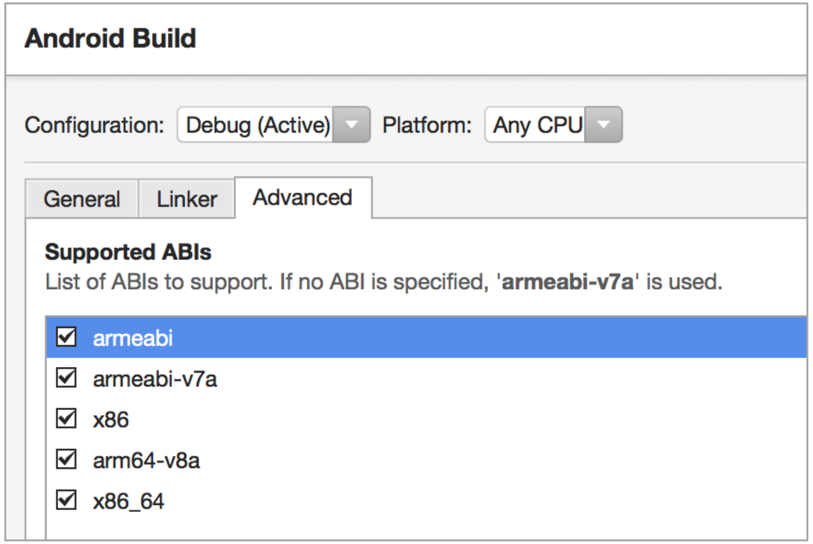
1. Android Requirements
2. iOS Requirements

Testing on physical devices

- To deploy your applications and tests onto real devices there are a few platform-specific requirements you will need to perform

Android build settings

- Select the Application Binary Interfaces required for your target Android hardware



Android Build

Configuration: Debug (Active) Platform: Any CPU

General | Linker | Advanced

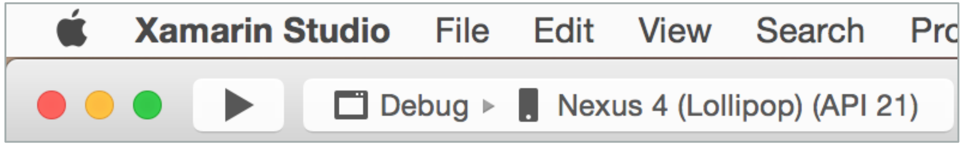
Supported ABIs
List of ABIs to support. If no ABI is specified, '**armeabi-v7a**' is used.

- [✓] armeabi
- [✓] armeabi-v7a
- [✓] x86
- [✓] arm64-v8a
- [✓] x86_64

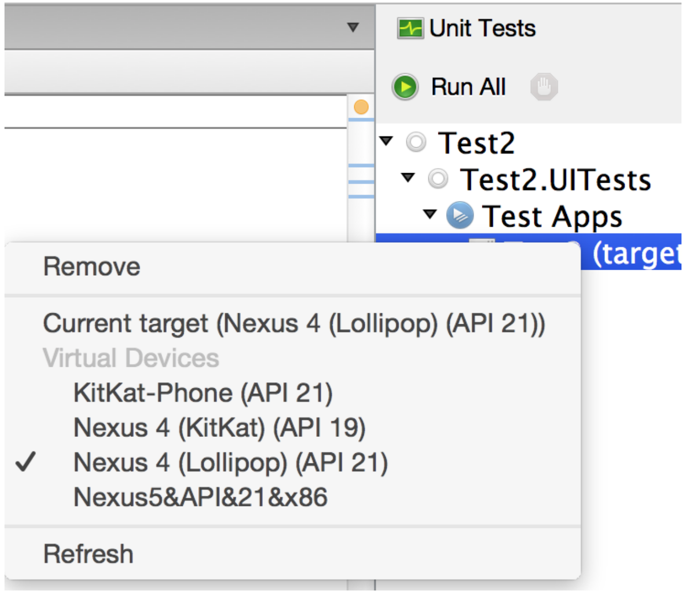
← must support all processor variations you want to run on

Identifying the device to run on

- Visual Studio for Mac **Unit Tests pad** will let you select the device/sim to run on



Defaults to the active device or simulator selected in the toolbar



Identifying the Android device

- Can also specify the device identifier as part of the test configuration, useful when more than one device or emulator is connected

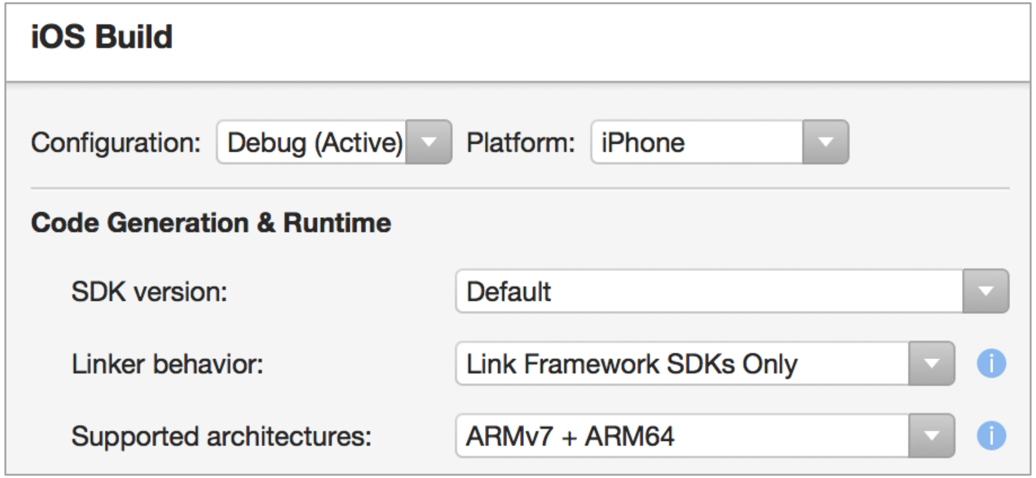
```
$ adb devices
List of devices attached
05845172        device
```

can use the ADB command line tool to identify all the connected devices

```
IApp app = ConfigureApp.Android.ApkFile("/path/to/app.apk")
                 .DeviceSerial("05845172").StartApp();
```

iOS Requirements

- Must use debug **build** and include all processor variations you plan to run against



iOS Build

Configuration: Debug (Active) Platform: iPhone

Code Generation & Runtime

SDK version: Default

Linker behavior: Link Framework SDKs Only

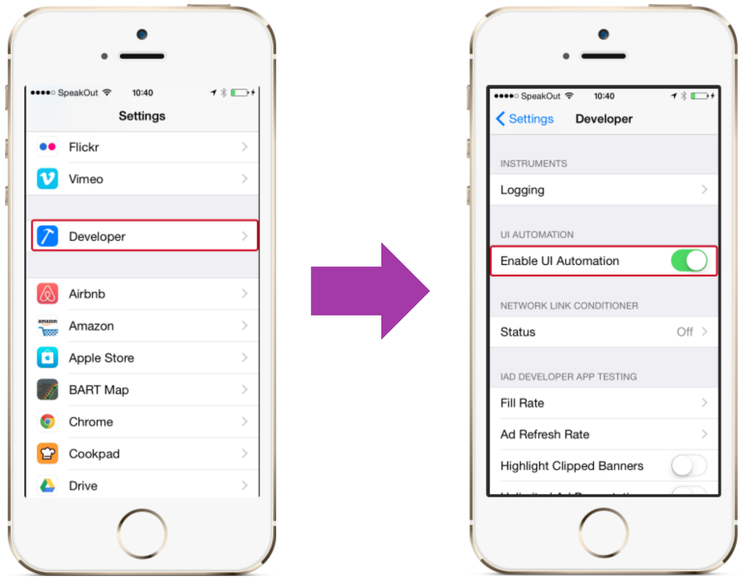
Supported architectures: ARMv7 + ARM64

Remember: You currently must use a Mac to build, run and submit iOS application + UI Tests to Xamarin Test Cloud

Enabling UI Automation on iOS

- To run UI tests on iOS physical devices, you must *enable UI Automation*

```
NUnit Test failed (click to run)
SetUp: System.Exception : Unable to run UIAutomation
script on device. For iOS 9 and above please make sure that
"Enable UI Automation" setting is enabled. The setting can be
found here: Settings -> Developer -> Enable UI automation.
```



••••○ SpeakOut 10:40

Settings

- Flickr
- Vimeo
- Developer
- Airbnb
- Amazon
- Apple Store
- BART Map
- Chrome
- Cookpad
- Drive

••••○ SpeakOut 10:40

< Settings **Developer**

INSTRUMENTS
- Logging

UI AUTOMATION
- Enable UI Automation: On

NETWORK LINK CONDITIONER
- Status: Off

IAD DEVELOPER APP TESTING
- Fill Rate
- Ad Refresh Rate
- Highlight Clipped Banners: Off

Identifying the iOS device

- Test code should identify the application by **bundle** and **device id** so it connects to the proper running app + device

```
IApp app = ConfigureApp.iOS
                .EnableLocalScreenshots()
                . DeviceIdentifier(
                   "5665472bcab727247ba037c18a4a405b46d8611e")
                .InstalledApp("com.xamarin.taskypro")
                .StartApp();
```

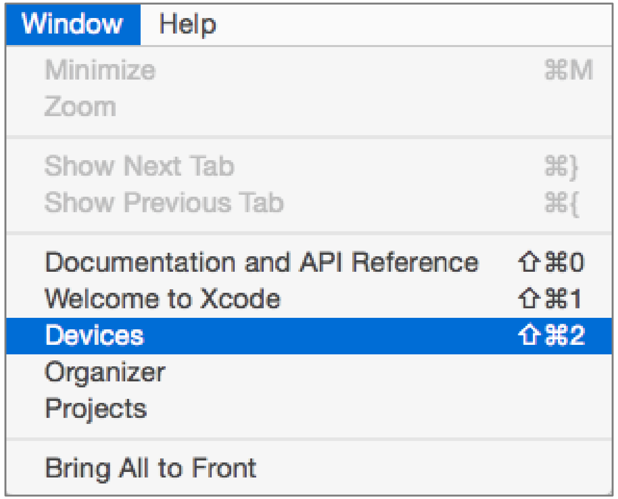
Getting device identifiers

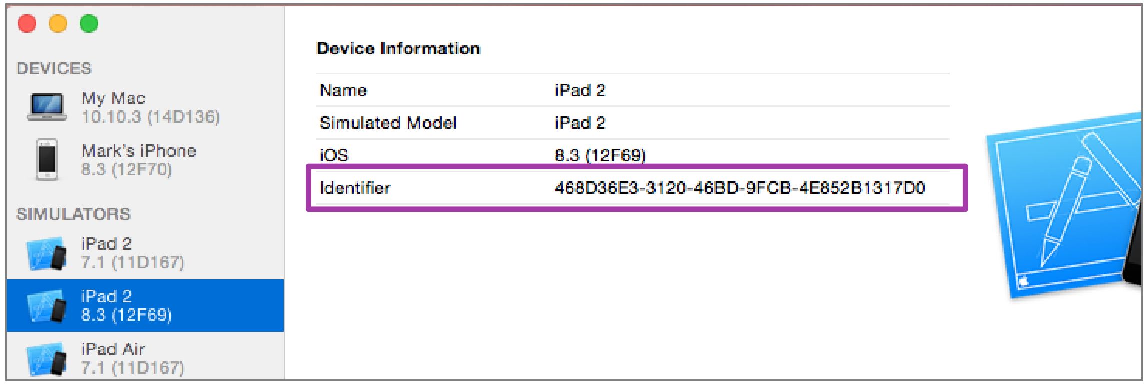
- Can identify devices on the command line using Instruments

```
$  xcrun instruments -s devices
Known Devices:
Mark’s MBPr [85E853D8-E91A-5DE8-A465-7CAAD4CC7ECC]
Mark’s iPhone (8.3) [5665472bcab727247ba037c18a4a405b46d8611e]
iPad 2 (7.1 Simulator) [EC6C3A52-C6E8-4A70-BB21-A4E7DE1CE8A5]
iPad 2 (8.3 Simulator) [468D36E3-3120-46BD-9FCB-4E852B1317D0]
iPad Air (7.1 Simulator) [CE1837EB-E7C5-4057-B374-C5C28398DC84]
iPad Air (8.3 Simulator) [733F7AA8-948C-4089-A74E-2D6558F6FE4B]
iPhone 4s (7.1 Simulator) [053B64CF-A564-4F82-B665-C967F1DFFBD7]
iPhone 4s (8.3 Simulator) [0BE9E503-2A5E-4F1C-AFFA-6D2BAECBE7B5]
...
```

Getting device identifiers

- ... or using the Xcode Devices window





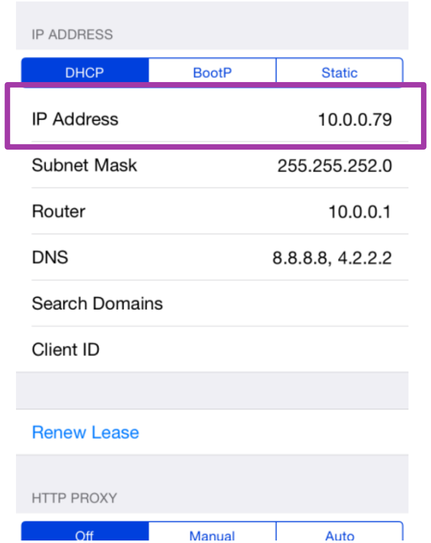
Device Information

Name	iPad 2
Simulated Model	iPad 2
iOS	8.3 (12F69)
Identifier	468D36E3-3120-46BD-9FCB-4E852B1317D0

Identifying the iOS device

- Can also identify by bundle and IP address for a WiFi connected device

```
IApp app = ConfigureApp.iOS
              .EnableLocalScreenshots()
              .DeviceIp("10.0.0.79")
              .InstalledApp("com.xamarin.taskypro")
              .StartApp();
```


IP ADDRESS

[✓ DHCP] BootP Static

IP Address	10.0.0.79
Subnet Mask	255.255.252.0
Router	10.0.0.1
DNS	8.8.8.8, 4.2.2.2
Search Domains	
Client ID	

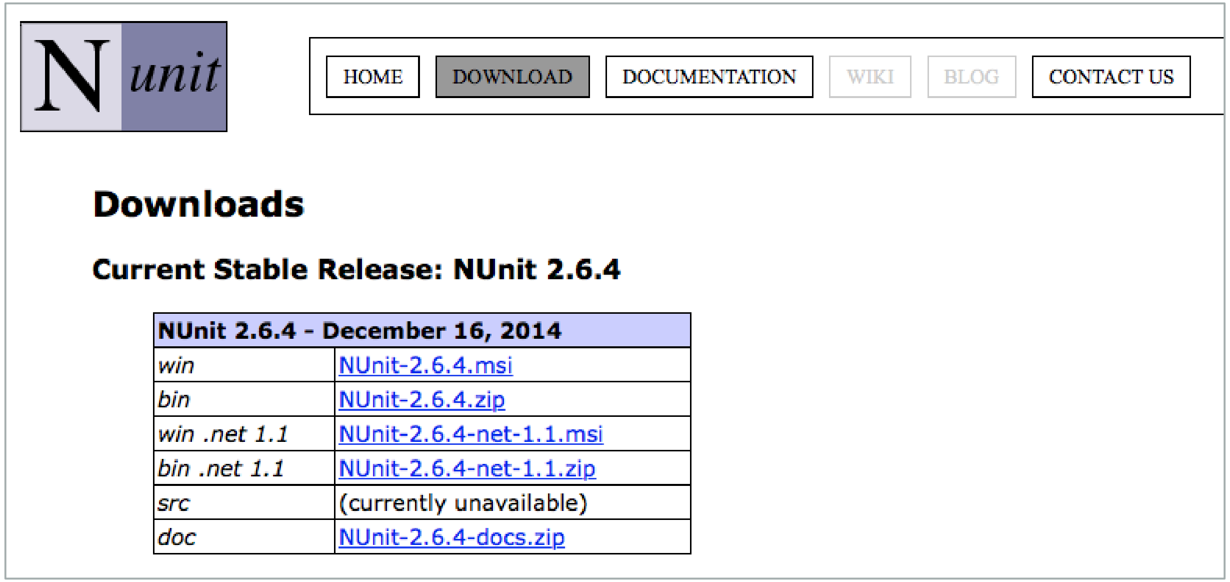
Renew Lease

HTTP PROXY

[✓ Off] Manual Auto

Running your UI Tests

- Can run tests on devices from Visual Studio for Mac – just like running on the simulators, or from the command line using **nunit-console**



Downloads

Current Stable Release: NUnit 2.6.4

NUnit 2.6.4 - December 16, 2014	
win	NUnit-2.6.4.msi
bin	NUnit-2.6.4.zip
win .net 1.1	NUnit-2.6.4-net-1.1.msi
bin .net 1.1	NUnit-2.6.4-net-1.1.zip
src	(currently unavailable)
doc	NUnit-2.6.4-docs.zip

Mixing command-line and IDE settings

- Can add the **`PreferIdeSettings`** flag to the configuration chain to ensure that IDE settings *override* the direct settings applied

```
return ConfigureApp
    .iOS
    .PreferIdeSettings()
    .DeviceIdentifier("96d5b77bc5b727247b8037018ada405b46d8611e")
    .InstalledApp("com.xamarin.samples.taskyprotouch")
    .StartApp();
```

Individual Exercise

Deploy Xamarin.UITests to a local device

Thank You!

Please complete the class survey in your profile:
university.xamarin.com/profile

Microsoft