XTC102

Xamarin.UITest

Download class materials from
university.xamarin.com

Microsoft

Xamarin University

Information in this document is subject to change without notice. The example companies, organizations, products, people, and events depicted herein are fictitious. No association with any real company, organization, product, person or event is intended or should be inferred. Complying with all applicable copyright laws is the responsibility of the user.

Microsoft or Xamarin may have patents, patent applications, trademarked, copyrights, or other intellectual property rights covering subject matter in this document. Except as expressly provided in any license agreement from Microsoft or Xamarin, the furnishing of this document does not give you any license to these patents, trademarks, or other intellectual property.

Xamarin, MonoTouch, MonoDroid, Xamarin.iOS, Xamarin.Android, Xamarin Studio, and Visual Studio are either registered trademarks or trademarks of Microsoft in the U.S.A. and/or other countries.

Other product and company names herein may be the trademarks of their respective owners.

Objectives

1. Create a Xamarin.UITest project
2. Create a cross-platform UI Test

Create a Xamarin.UITest project

Xamarin University

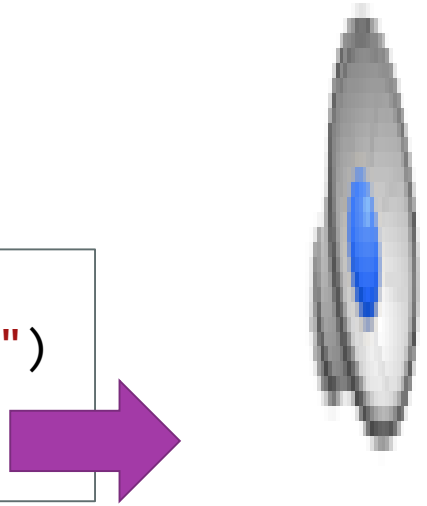
Tasks

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

What is Xamarin.UITest?

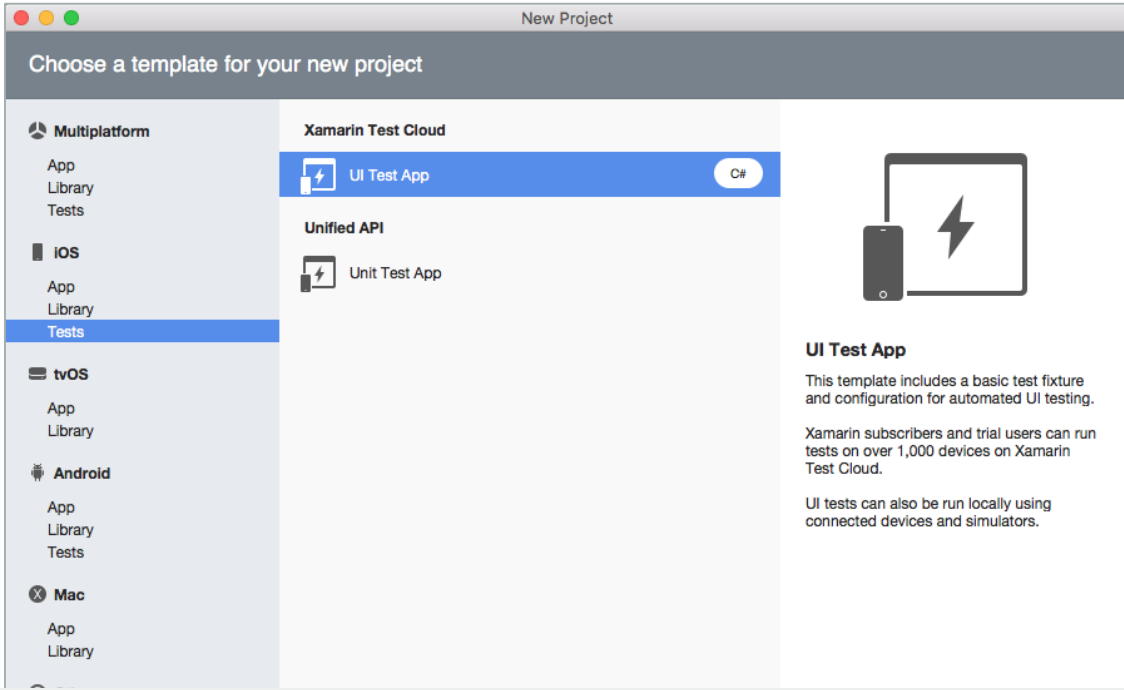
- Xamarin.UITest is a framework which lets you automate a mobile device and application

```
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

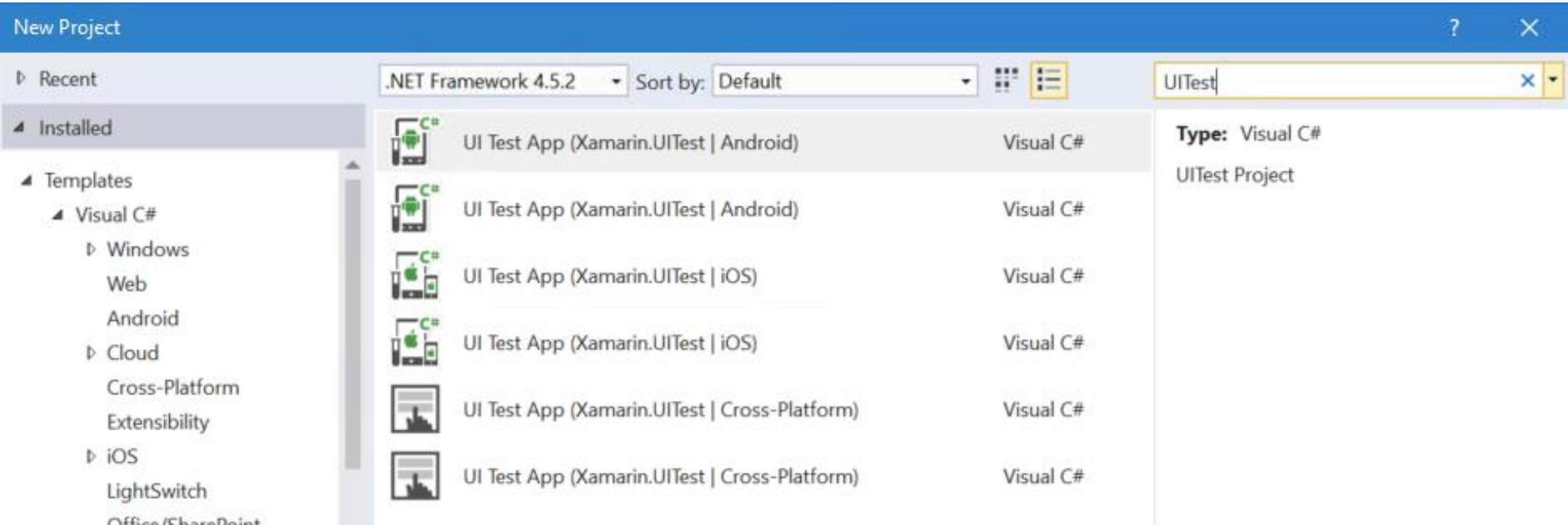
- Visual Studio for Mac has project templates for creating UITest projects for cross-platform, iOS and Android apps
- Creates a working project to start with, you just fill in some details and write the tests



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

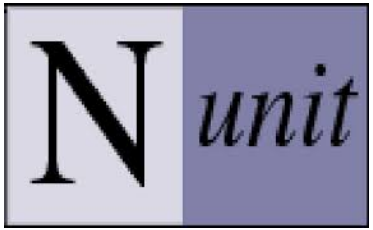
Creating a Xamarin.UITest project

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



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

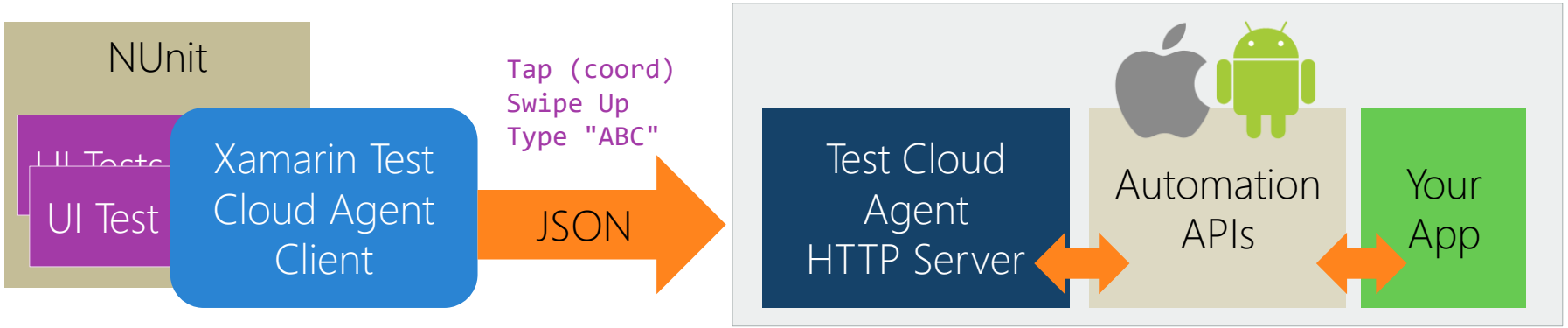


MSTest

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

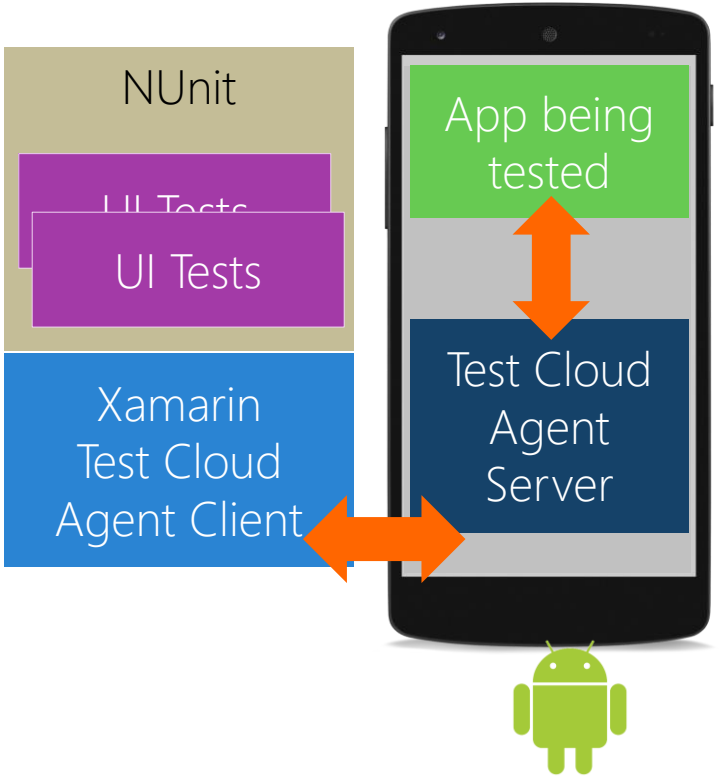


Runs on the same computer as the unit tests (e.g. desktop or cloud)

Runs on the device or simulator

Android architecture

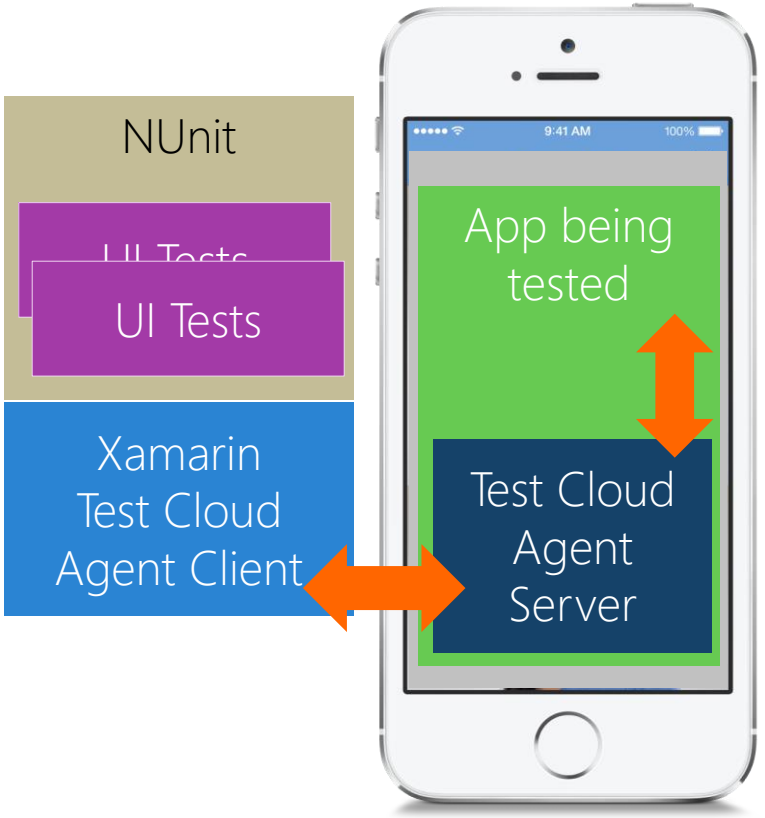
- On Android, Xamarin.UITest installs the **Xamarin Test Cloud Agent server** as a separate process
- Process is signed with the same keystore as your application so it can drive it with the Android Automation APIs



iOS architecture

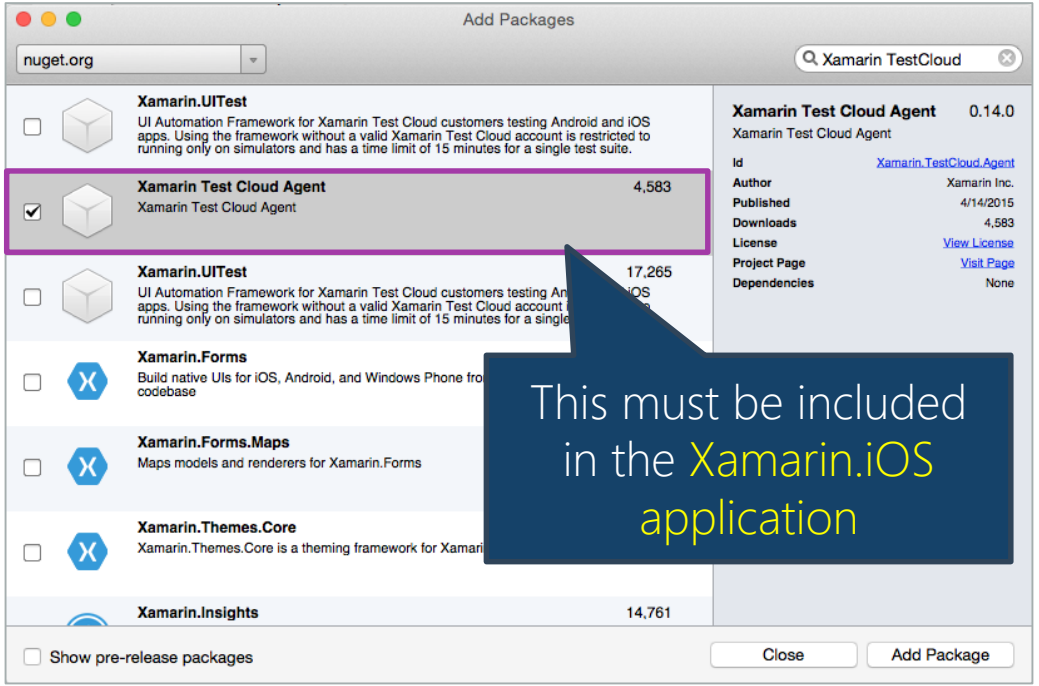
- On iOS, the **Xamarin Test Cloud Agent** component must be installed as part of the application bundle

- 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



Automating an iOS application

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

Starting the Automation Server

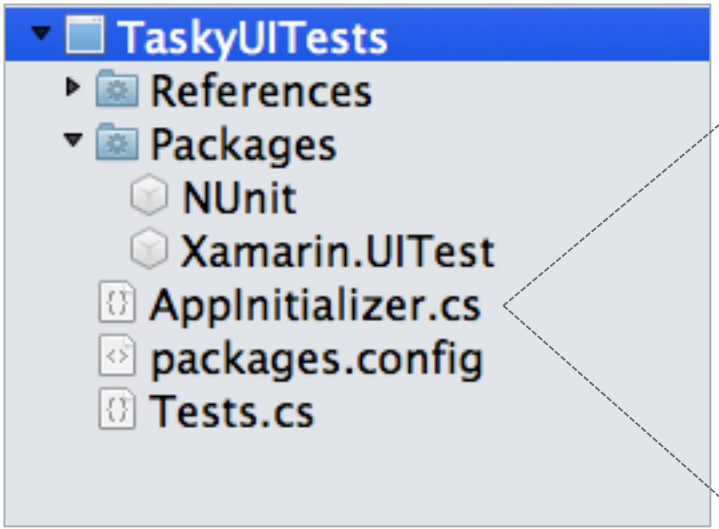
- Add code to start the Calabash server in your Xamarin.iOS application into the **`FinishedLaunching`** method

```
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)



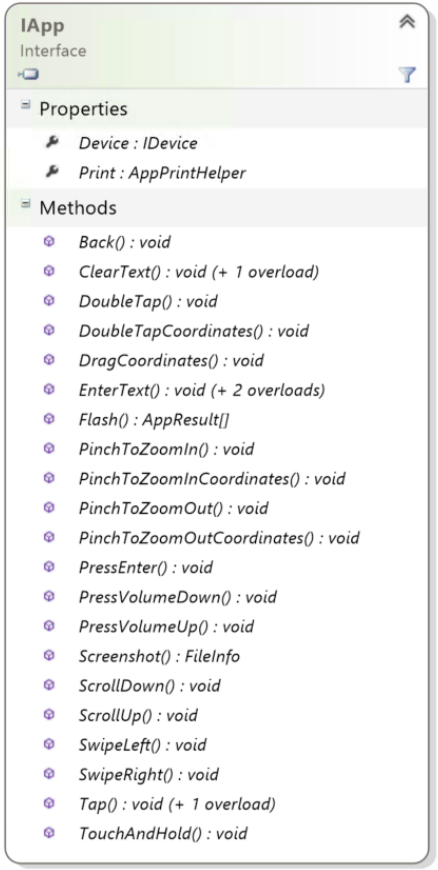

```
public class AppInitializer
{
    public static IApp StartApp(Platform platform)
    {
        if (platform == Platform.Android) {
            // ... Android init ...
        }
        // ... iOS init ...
}
```

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`**



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
    app = ConfigureApp.Android

    ...
}
```

or

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:

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:

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");
```

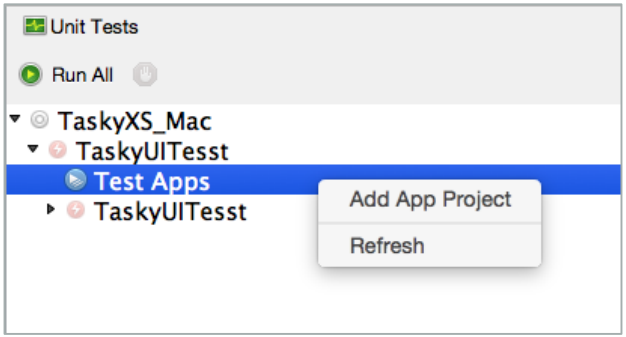
pass the package name or bundle identifier to specify the application

Working with UITest

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

Selecting a project in your solution

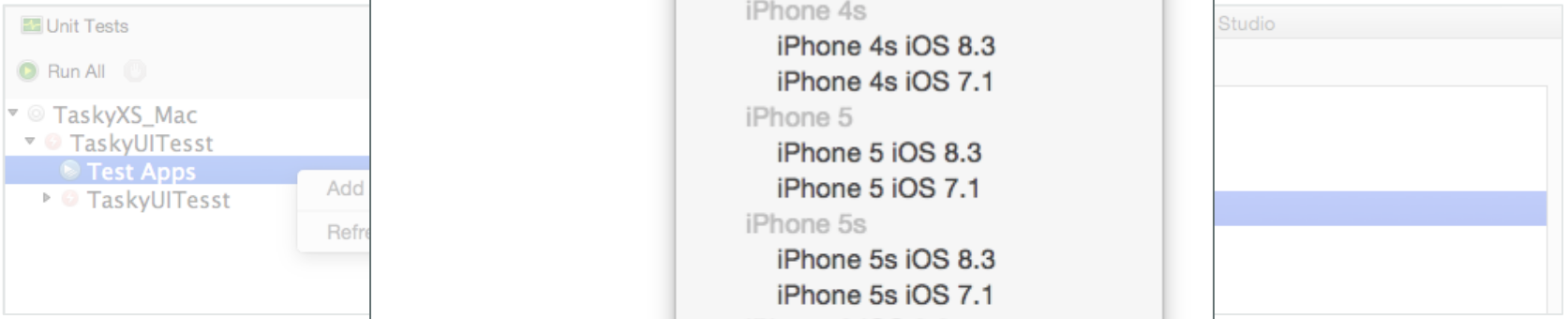
- Unit Tests pad has UI to associate another project in the solution with your tests – use the **Test Apps** section



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

- Unit Tests pad
  your tests – us



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

ke specifying the

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()
    {
        ...
    }
}
```

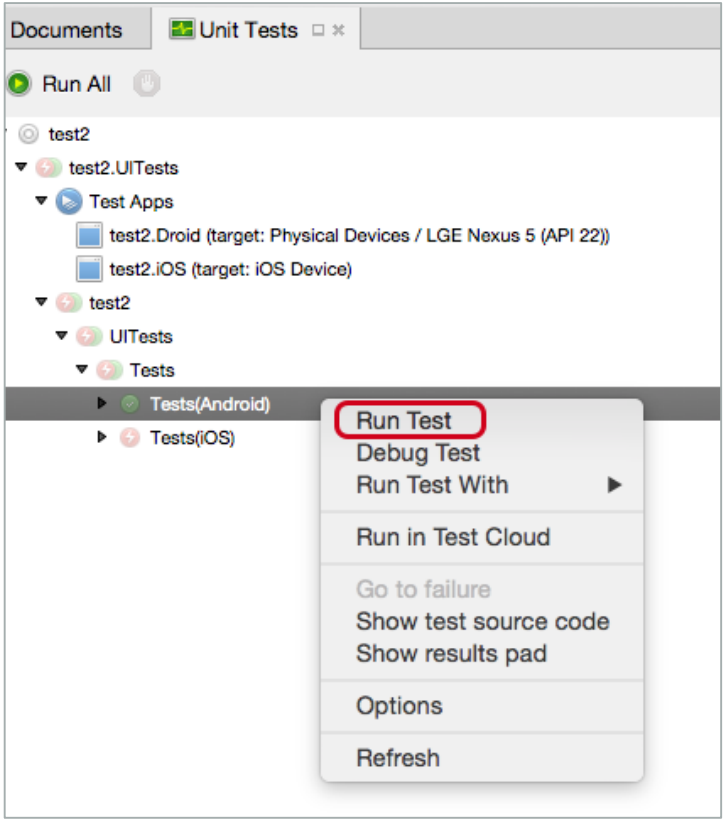
Writing tests

- 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

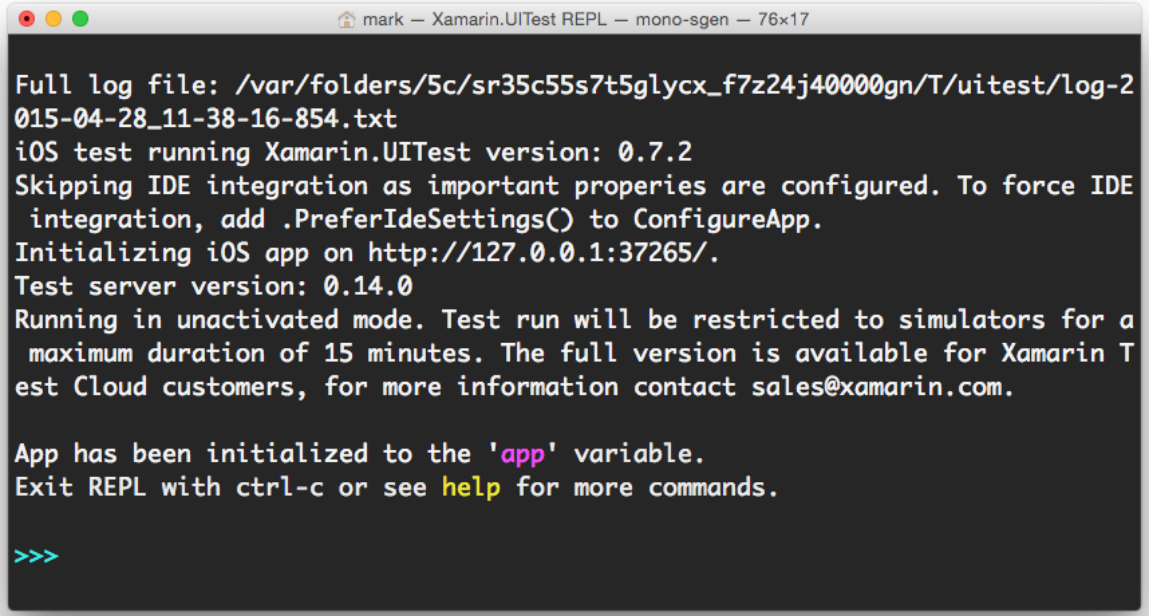
- Can run tests either in the IDE or from the command line (they are just unit tests)
  - `nunit-console.exe`
- iOS UI Tests can only be run from Visual Studio on the Mac, but Android is supported on both platforms and IDEs



Using the REPL

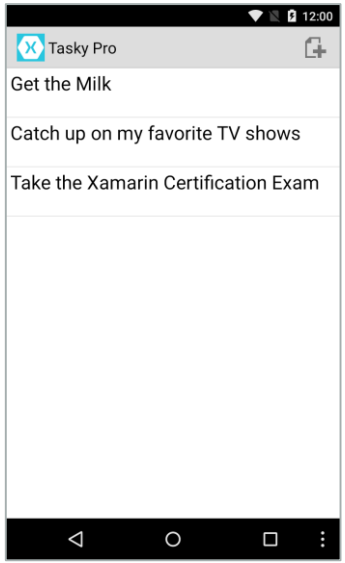
- Built in REPL (Read-Evaluate-Print-Loop) allows you to explore and manipulate the running application interactively through a runtime shell

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



Examining the UI with the REPL

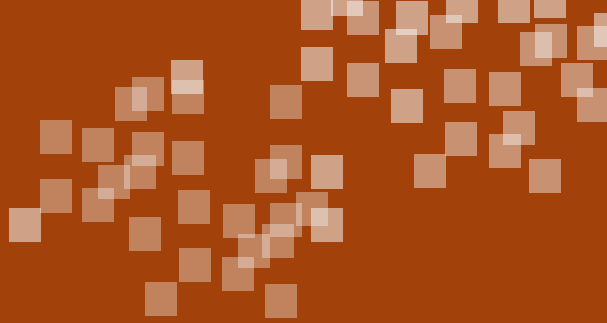
- Use **Tree** to dump the visual tree for your app




```
>>> 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



Demonstration

Working with the REPL

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`	

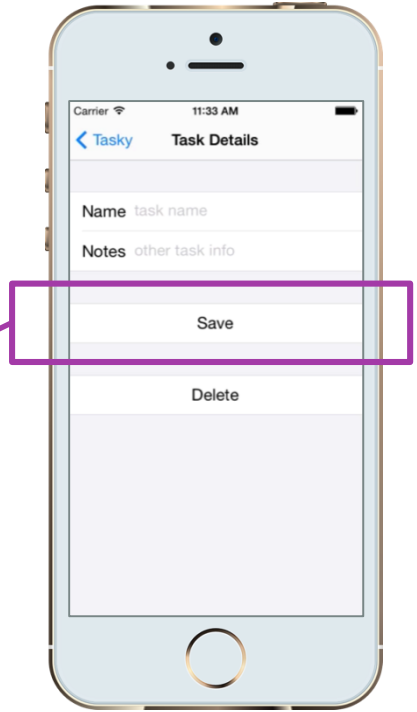
Identifying UI elements

- 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

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

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

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"`**

Check out **https://developer.xamarin.com/guides/testcloud/uitest/cheatsheet/** for some examples and common queries you might use

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

Xamarin University

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) All of the above

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

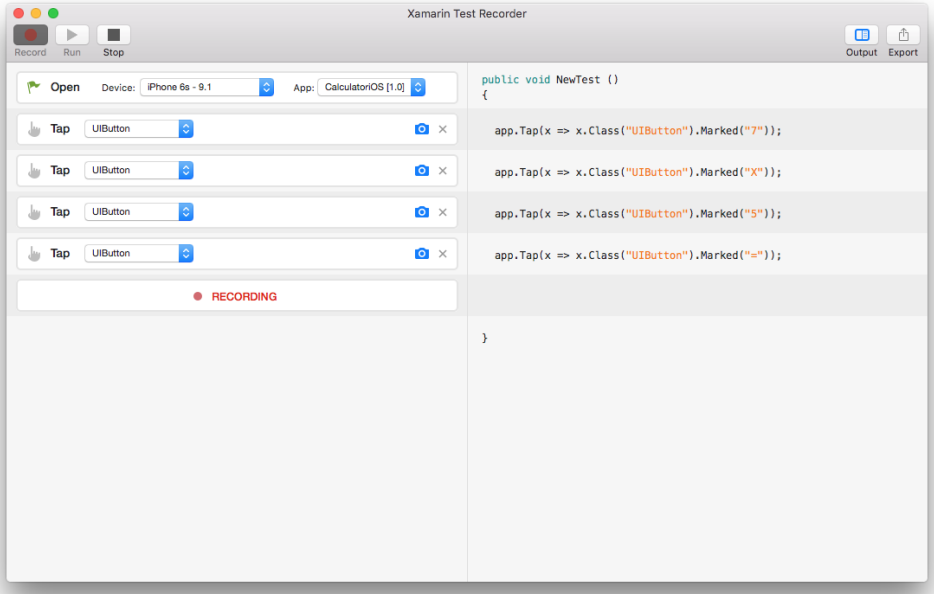
③ The Query operation by itself will show all controls, both visible and invisible on the device

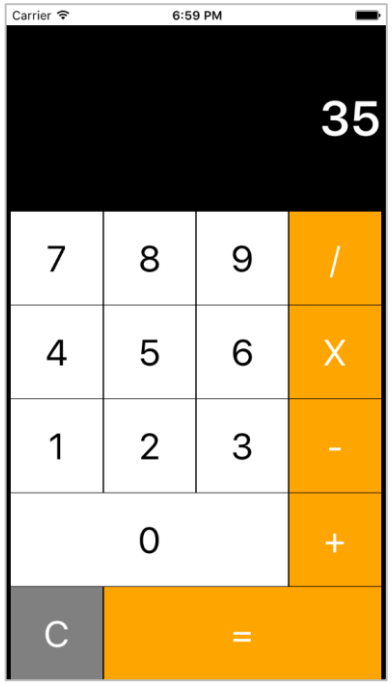
a) True

b) False

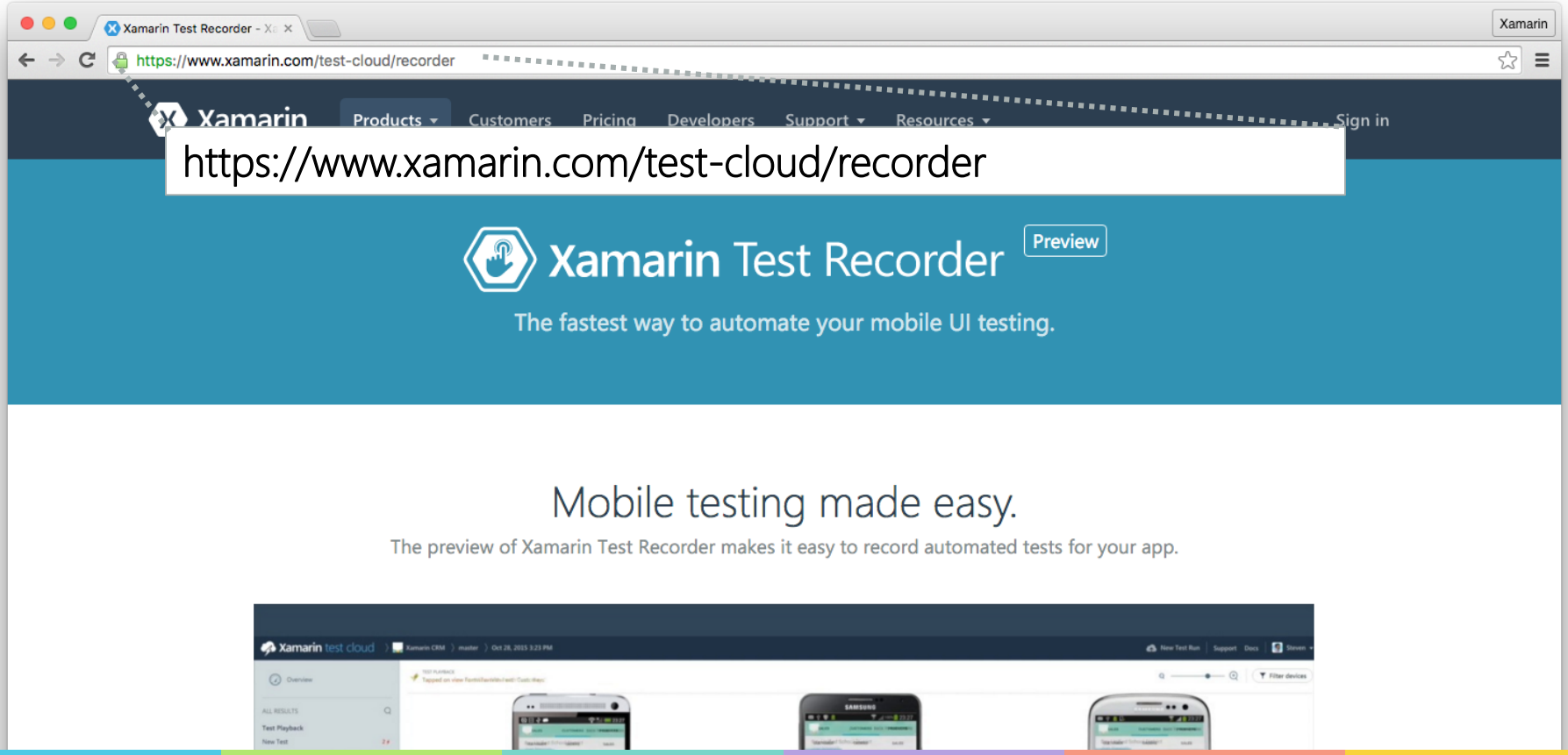
Using the Test Recorder

- Test Recorder is an easy way to interactively build Xamarin.UITest scripts by working directly with the application





Downloading the Test Recorder



Visual Studio Test Recorder

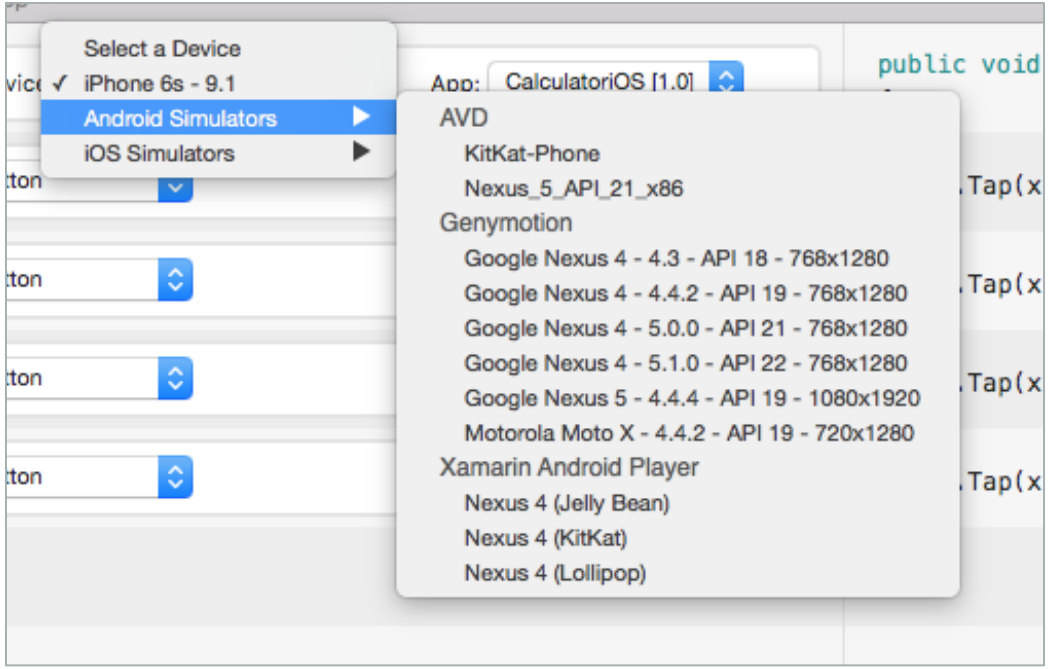
- Test Recorder is supplied as an extension for Visual Studio 2015 which is available from the **Tools > Extensions and Updates** menu

Requirements

- The Test Recorder in Visual Studio on Windows requires an Enterprise License
- Test Recorder for Visual Studio for Mac can be used in every version

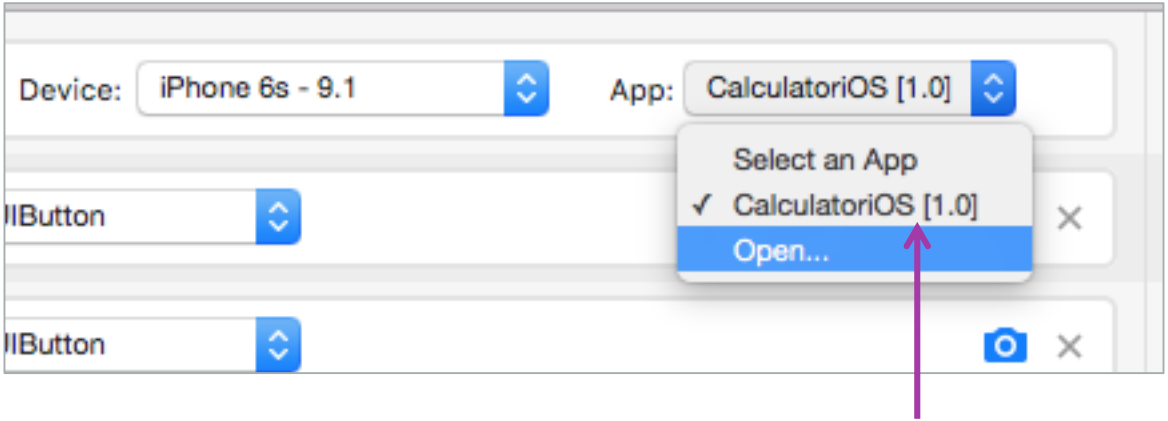
Test Recorder: select the device

- Can use a physically connected device, or simulator/emulator



Test Recorder: select the app

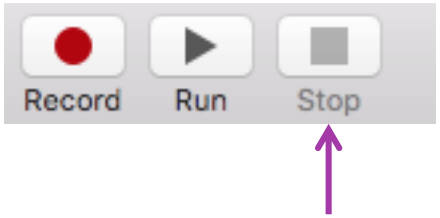
- Can then select an application from disk, Test Recorder will install the selected app onto the device/simulator and launch it



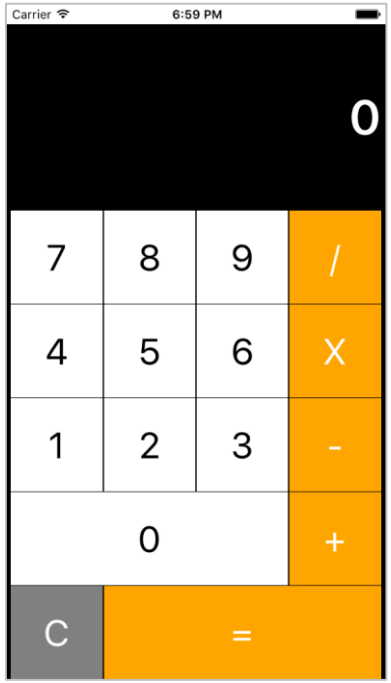
It also remembers prior applications you have installed

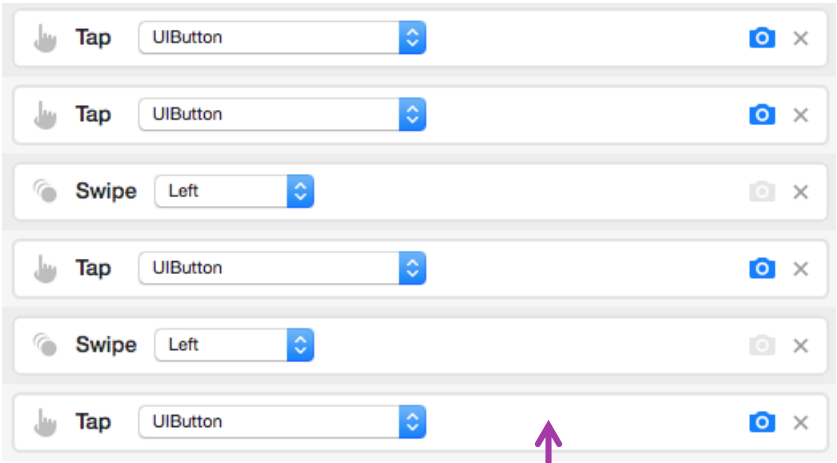
Using the Test Recorder

- Can then press Record to start recording all your actions



.. and click **Stop** when finished

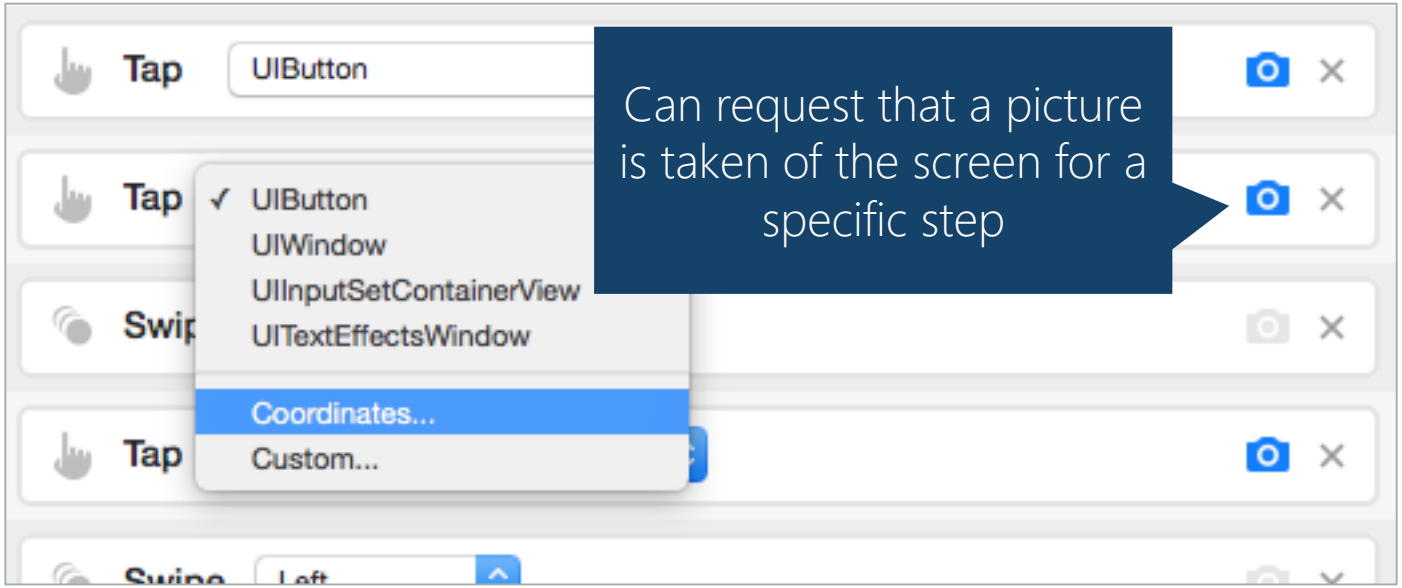




Steps are listed in the order you execute them

Using the Test Recorder

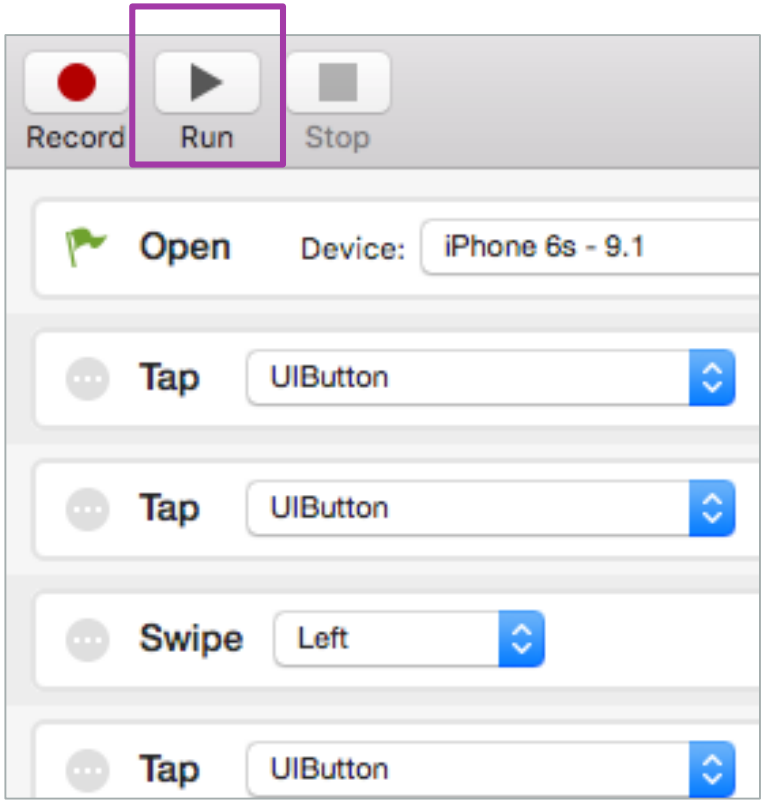
- Can edit steps to fine-tune recording, either while you are recording or after the recording is complete



Using the Test Recorder

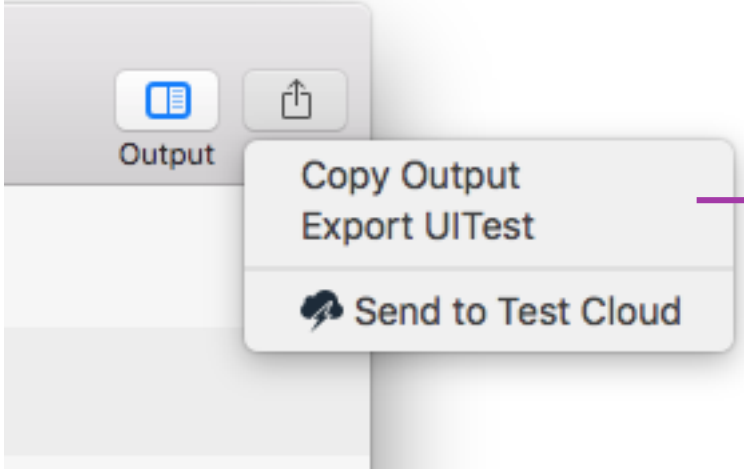
- Once the recording is finished, you can play it back to verify it does what you want it to

- If not, you can edit the test and try again until you get it right



Using the Test Recorder

- Last step is to export the test – can copy steps to the clipboard, export as a Xamarin.UI test (C#), or even send the test (and app) to Xamarin Test Cloud

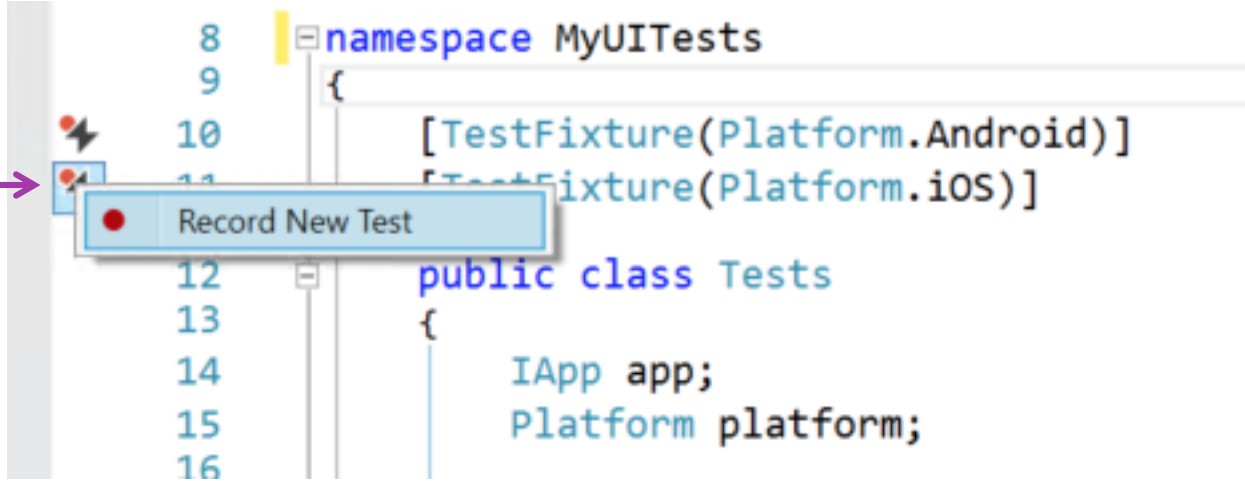



```
public void NewTest ()
{
  app.Tap(x => x.Class("UIButton").Marked("C"));
  app.Tap(x => x.Class("UIButton").Marked("7"));
  app.SwipeLeft();
  app.Tap(x => x.Class("UIButton").Marked("5"));
  app.SwipeLeft();
  app.Tap(x => x.Class("UIButton").Marked("="));
}
```

Test Recorder for Visual Studio

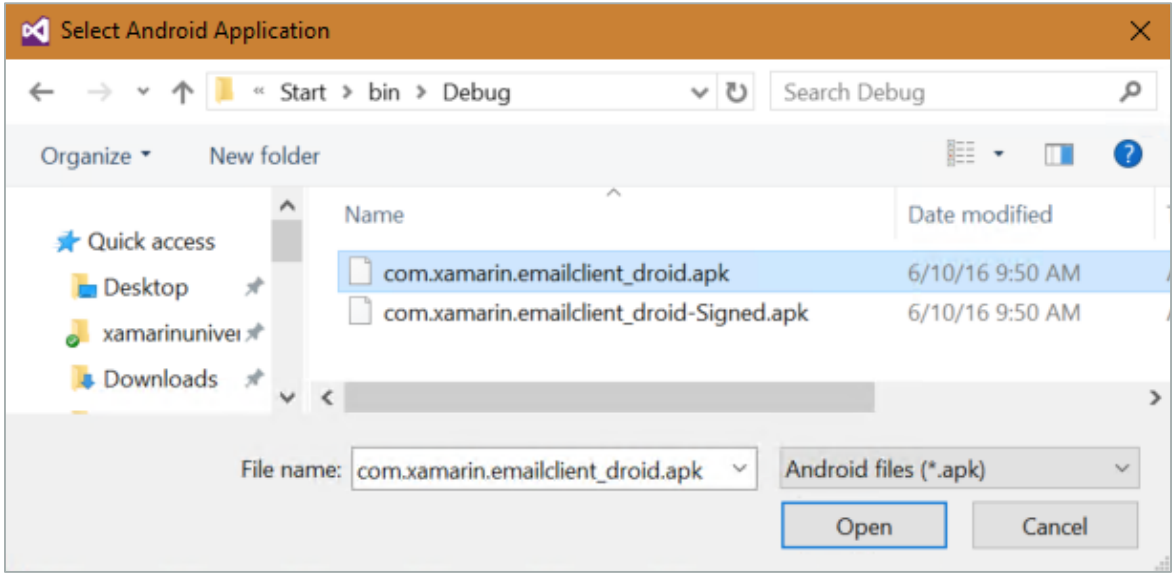
- Can install a Visual Studio add-in to support recording – allows you to record a test directly in the IDE

Right click on test marker on **`TestFixture`** select "Record New Test" – this creates a new UI test in your source file



Test Recorder for Visual Studio

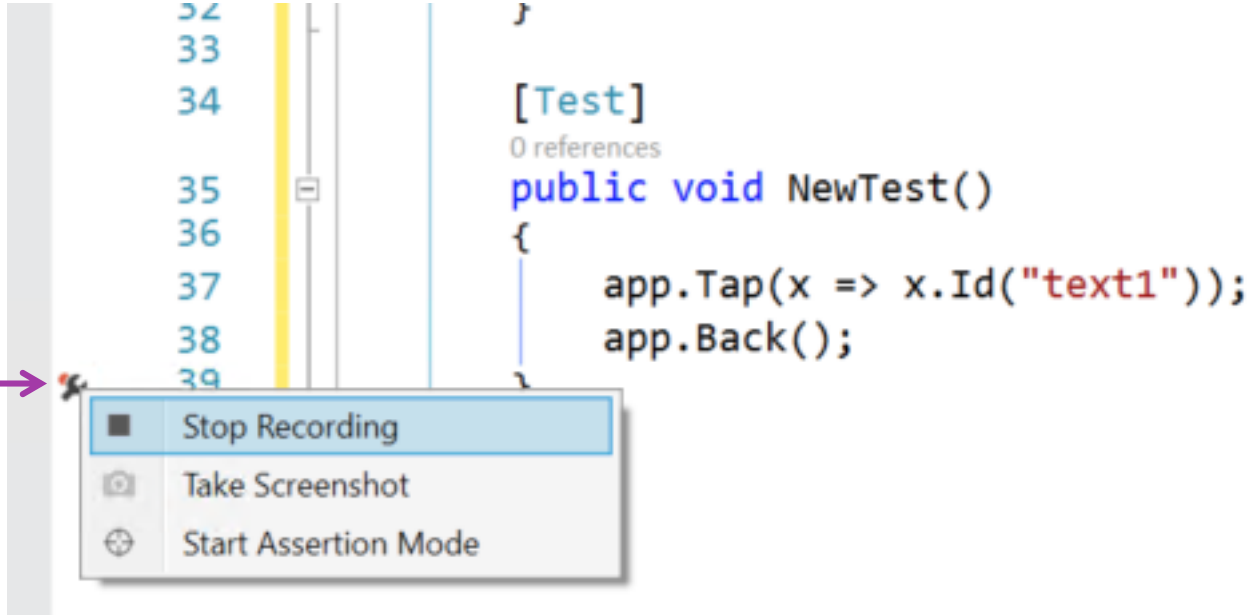
- Add-in will prompt for APK to load onto selected device or simulator



Test Recorder for Visual Studio

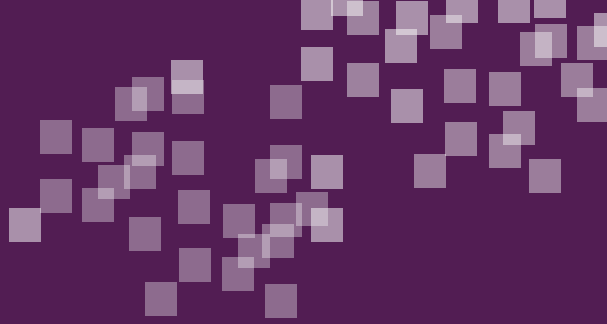
- Visual Studio will write UI test code as you interact with the app and place it into a newly created test

Can then stop the test, take a screen shot, or add assertions & wait steps by clicking the recorder settings icon in the new test



Demonstration

Using the Test Recorder



Individual Exercise

Creating acceptance tests with Xamarin.UITest

Xamarin University

Summary

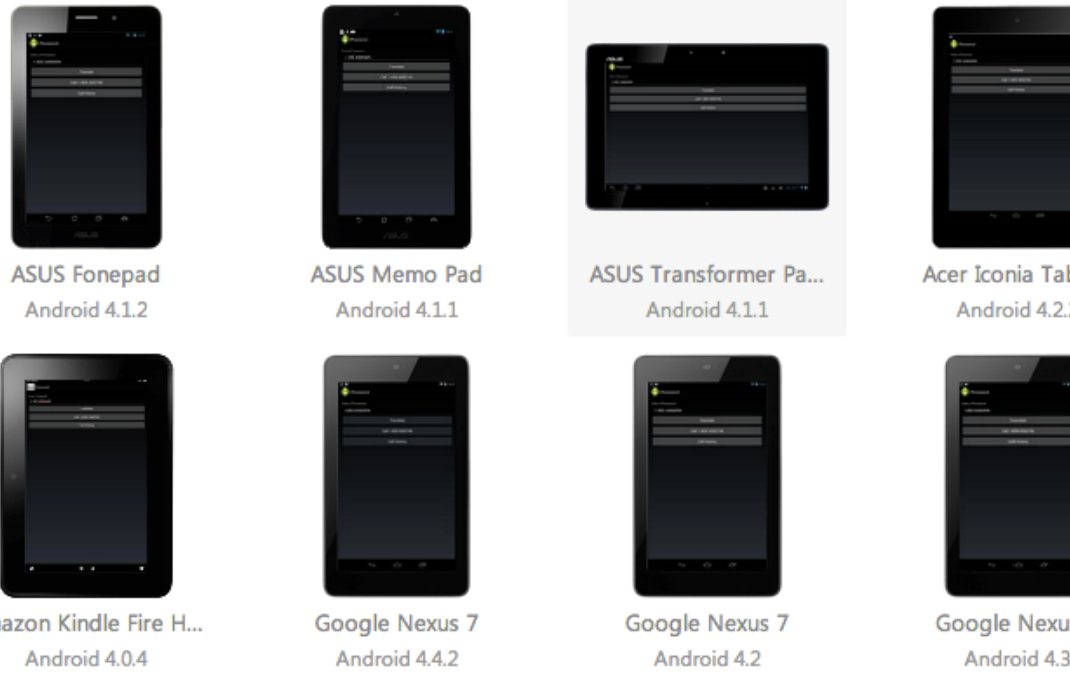
1. Create a new UITest project
2. Use the REPL tool
3. Create a query
4. Use the Test Recorder
5. Build acceptance tests

Create a cross-platform UI Test

Xamarin University

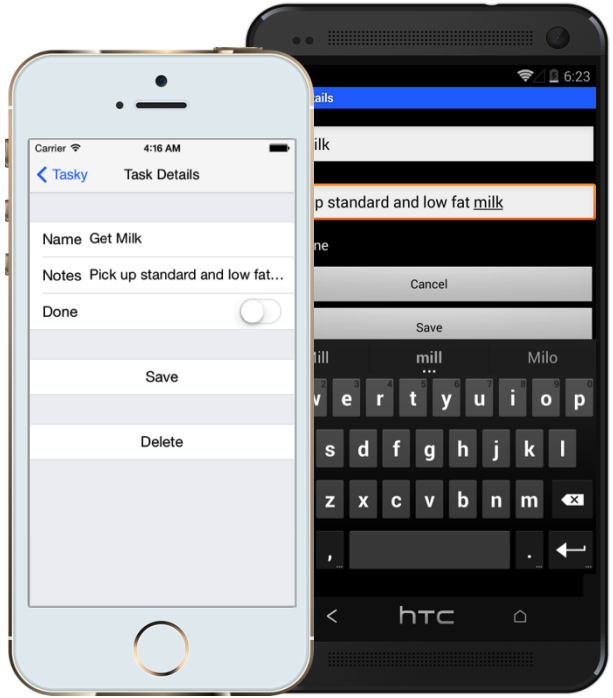
Tasks

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

- 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

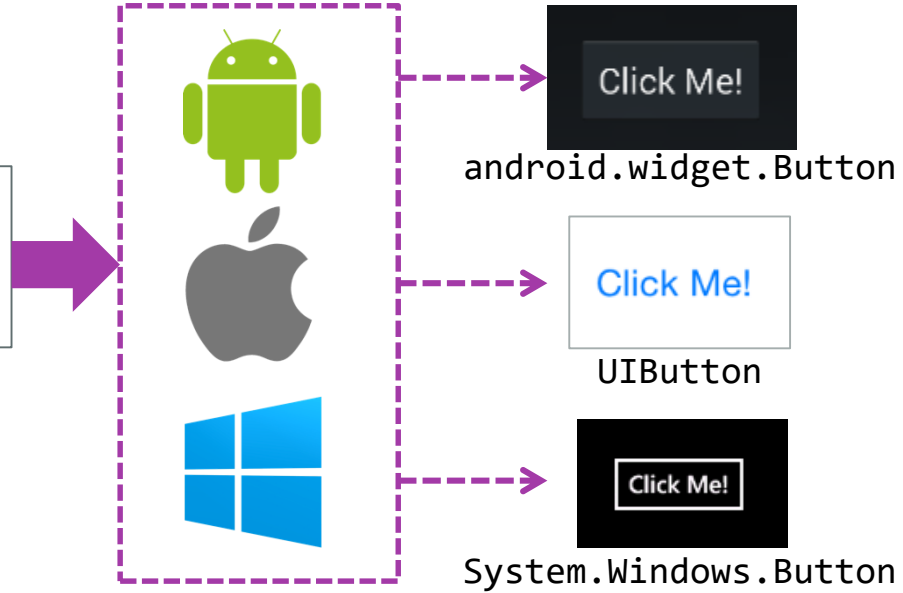
On iOS, it matches against the **`AccessibilityIdentifier`** and **`AccessibilityLabel`** of the **`UIView`**

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 at this time.

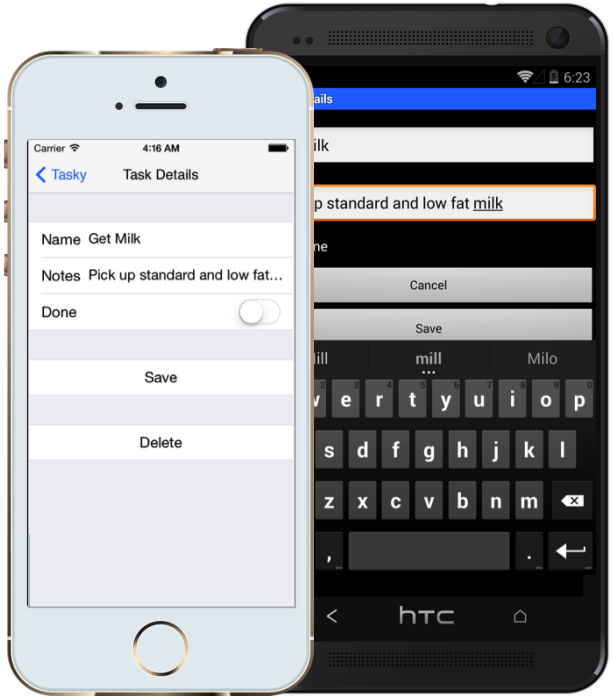
Demonstration

Show the platform differences

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) { ... }
    }
}
```

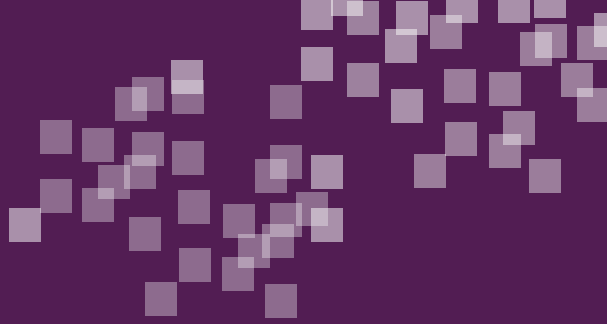
Creating cross-platform tests #1

```
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

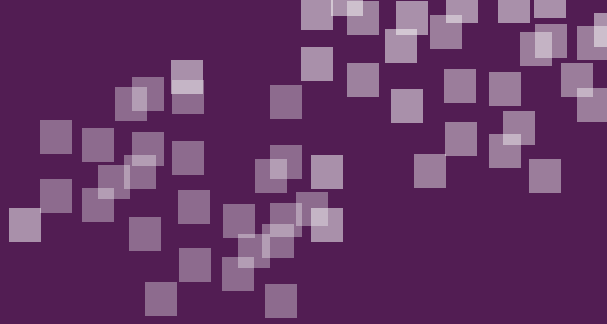
- Can also differ from device to device, based upon processor speed, network connectivity, etc.

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

Xamarin University

Taking screenshots

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

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

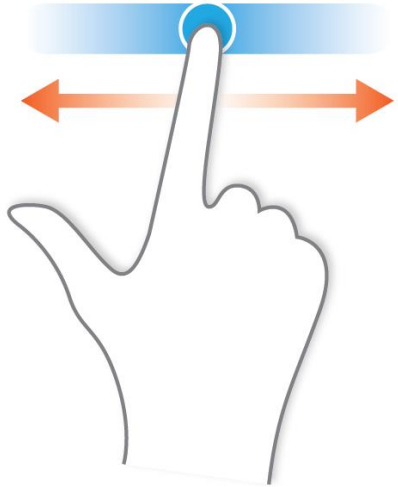
Must be enabled for the local testing scenarios as part of the configuration chain

Should name the screenshots – this is used in Test Cloud to identify the picture

```
[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"

See: https://developer.xamarin.com/guides/testcloud/uitest/working-with/webviews/ for more information on working with web views in your apps

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

Xamarin University

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

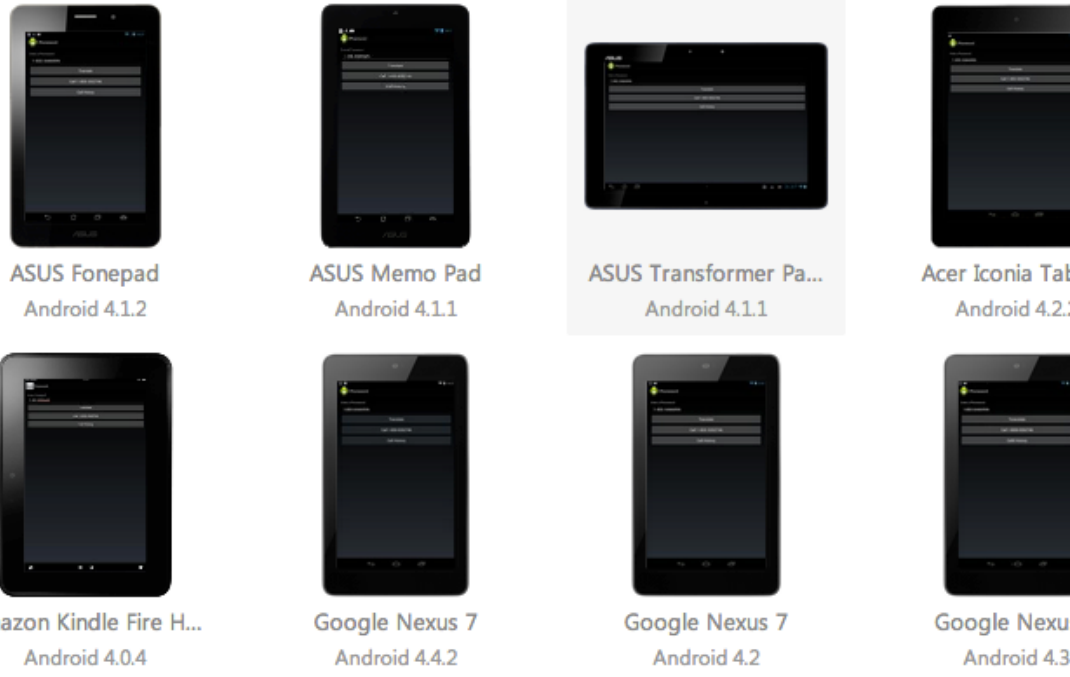
a) It looks for exactly the same class, including the namespace

b) It looks for full implementations of the classes interface

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

Summary

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



Learning more about UI Testing

- **XTC103** walks you through deploying your tests to local devices connected to your computer and then uploading those same tests to Xamarin Test Cloud to get full coverage

Thank You!

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

Microsoft