

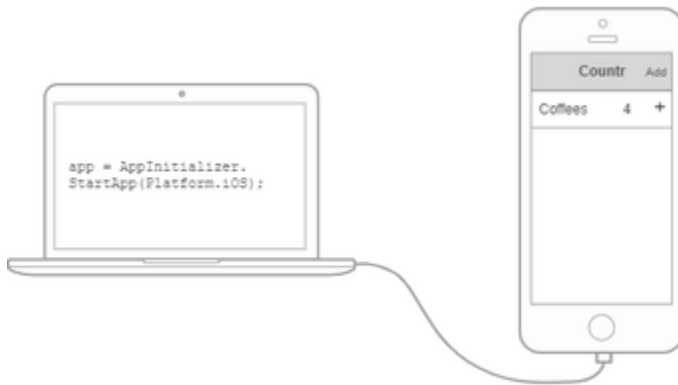


# Xamarin.UITest

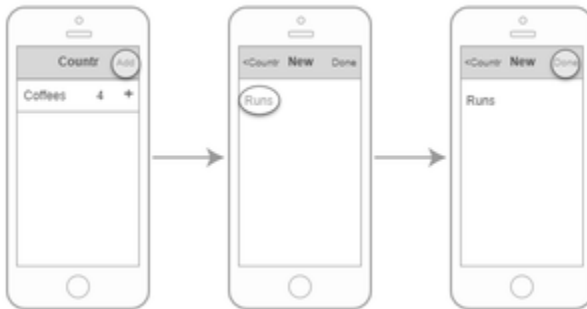
*This document is a summary of the official documentation of Xamarin and other sources. All links consulted are at the end of the document.*

## About Xamarin.UITest

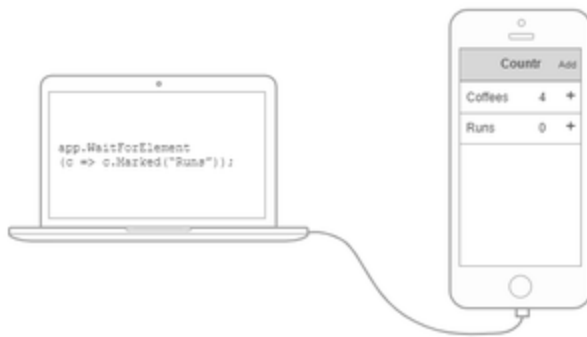
- Xamarin.UITest is a **testing framework that enables Automated UI Acceptance Tests written in C# using the NUnit** unit testing framework to be run against iOS and Android applications.
- It can simulate user interactions on Xamarin application (Forms, iOS, Android), and also native swift/Obj-C based iOS & Java-based Android application.
- It's API that powers Xamarin Test Cloud, enabling an uploaded test to run on hundreds of physical device simultaneously.
- Each test should follow the **Arrange-Act-Assert pattern**:
  - **Arrange**: the app is launched on the target device and the test framework connects to it.



- **Act**: the test connects to the device and interacts with it as if it was a user, for example tapping buttons and entering text.



- **Assert**: the test examines the results of the actions performed in the Act step to determine correctness. For example, the application may verify that a particular error message is displayed.



- You write UI tests in the same way that you write a unit test.
- Automated UI testing relies heavily on being able to **locate and interact with views on the screen**. Xamarin.UITest addresses this requirement with two important sets of APIs that work with each other:
  1. **Actions** that can be performed on views such as tapping on the view, entering text, or swiping **on** the view.
  2. **Queries** to locate views on the screen – Part of the Xamarin.UITest framework are APIs that will locate the views on a screen. Queries locate views at run time by inspecting attributes associated with the view and returning an object that the actions may work upon.
- To help with writing tests, Xamarin.UITest provides a *read-eval-print-loop* ([REPL](#)). The **REPL** allows developers and testers to interact with a screen while the application is running and simplifies creating the queries.
- **Xamarin.UITest API:**
  - All test interactions with the mobile application occur through an instance of **Xamarin.UITest.IApp**.
  - **Xamarin.UITest.IApp**: defines the methods that are crucial for the test to collaborate with the application and interact with the user interface.
    - There are two implementations:
      - **Xamarin.UITest.iOS.iOSApp** This class will automate tests against iOS.
      - **Xamarin.UITest.Android.AndroidApp** This class is for automating tests on Android.
      - Both are created using the helper `ConfigureApp` class.
      - It's recommended that a new `IApp` instance be used for each test, this will prevent state from one test spilling over into another affecting the results and reliability of the test.
      - We can initialize an instance of `IApp` in two places:
        - **SetUp method:**
        - **TestFixtureSetup method:**

- **Initialize IApp for iOS Applications:**

```
IApp app = ConfigureApp
    .iOS
    .AppBundle( "/path/to/iosapp.app" )
    .StartApp();
```

The **AppBundle** method can be used to specify where on the file system the app bundle may be found (example: `..\NareiaMobileBase.iOS\bin\iPhoneSimulator\Debug`).

- **Initialize IApp for Android Applications**

```

IApp app = ConfigureApp
    .Android

    .ApkFile("../../../iOSAppProject/bin/iPhoneSimulator/Debug
    /iosapp.app" )
    .StartApp() ;

```

The `ApkFile` method of `IApp` is used to specify where on the file system the APK may be found.

## Interacting with User Interface

`IApp` has many methods for interacting with an application.

Method	Description
<code>Button</code>	Will locate one or more buttons on the screen.
<code>Class</code>	Will try to locate views that are of a specified class.
<code>Id</code>	Will try to locate a view with the specified Id.
<code>Index</code>	Will return one view from a collection of matching views. Usually used in conjunction with other methods. Takes a zero-based index.
<code>Marked</code>	Will return a view according to the heuristics discussed below.
<code>Text</code>	Will match views that contain the provided text.
<code>TextField</code>	Will match an Android <code>EditText</code> or iOS <code>UITextField</code> .

For example, the following method shows how to simulate a tap on button called "SaveUserdataButton":

```

app.Tap( c=>c.Marked( "SaveUserDataButton" ) ) ;

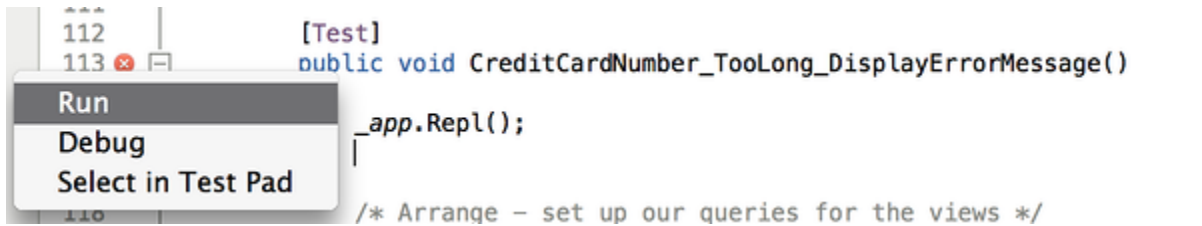
```

## Using the REPL

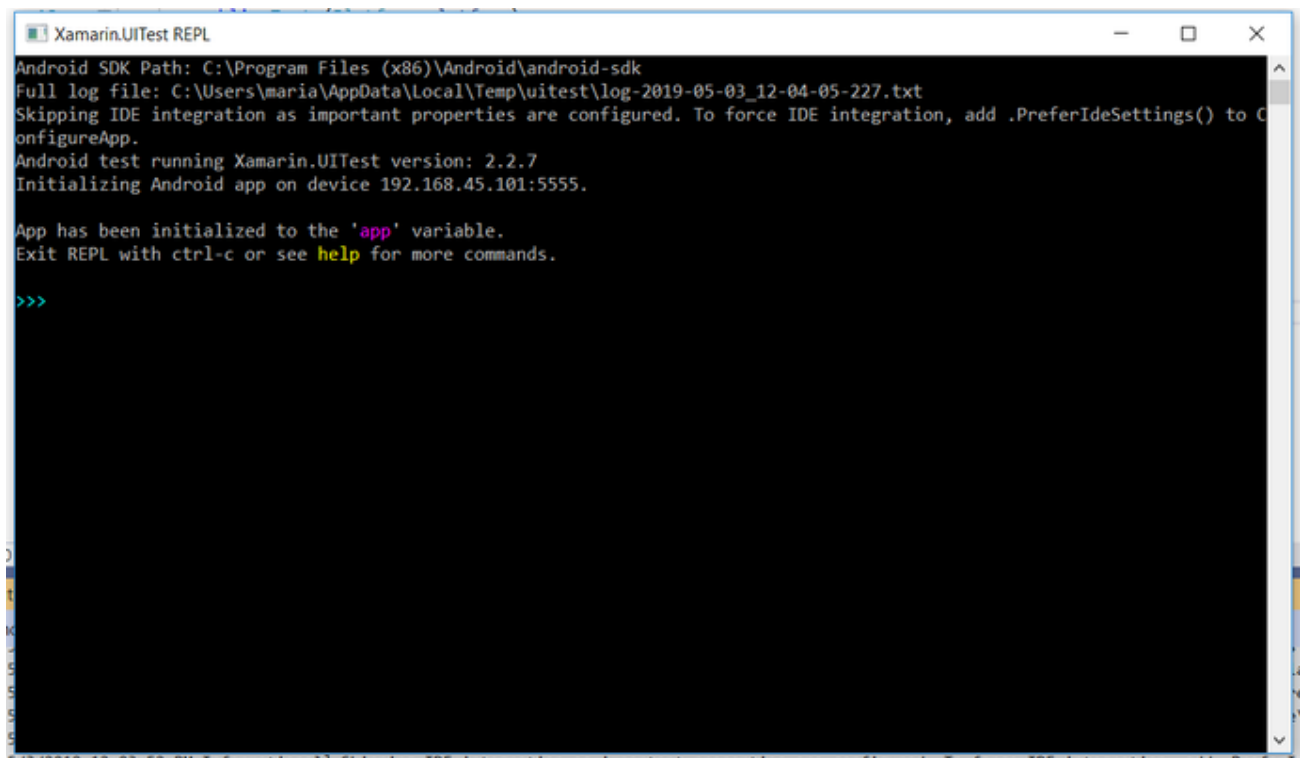
- The only way to start the REPL is to invoke the `IApp.Repl` method within an existing test.

```
[Test]
public void CreditCardNumber_TooLong_DisplayErrorMessage()
{
    app.Repl();
}
```

- To run the test by right clicking in the gutter of Visual Studio and selecting **Run**:



The test will run, and when the `Repl` method is invoked, Xamarin.UITest will start the REPL in a terminal session, as shown in the following screenshot:



- You can use the **tree** command to explore the user interface.

```
Xamarin.UITest REPL
Full log file: C:\Users\maria\AppData\Local\Temp\uitest\log-2019-05-03_12-04-05-227.txt
Skipping IDE integration as important properties are configured. To force IDE integration, add .PreferIdeSettings() to C
onfigureApp.
Android test running Xamarin.UITest version: 2.2.7
Initializing Android app on device 192.168.45.101:5555.

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

>>> tree
[[object CalabashRootView] > DecorView]
  [LinearLayout > FrameLayout]
    [FitWindowsFrameLayout] id: "action_bar_root"
    [ContentFrameLayout > ... > PlatformRenderer] id: "content"
      [NavigationPageRenderer] id: "NoResourceEntry-1"
      [PageContainer] id: "NoResourceEntry-6"
        [PageRenderer > Platform_DefaultRenderer] id: "NoResourceEntry-2"
          [ScrollViewRenderer > ... > Platform_DefaultRenderer] label: "mainSv"
            [EntryRenderer] label: "CreditCardEntry_Container"
              [FormsEditText] id: "NoResourceEntry-3", label: "CreditCardEntry"
                [ButtonRenderer] label: "ValidateCC_Container"
                  [AppCompatButton] id: "NoResourceEntry-4", label: "ValidateCC", text: "Validate CC"
                [BoxRenderer]
                  [LabelRenderer] label: "mainSvScrollStatus_Container"
                    [FormsTextView] id: "NoResourceEntry-5", label: "mainSvScrollStatus", text: "Scroll down to bottom"
              [Toolbar] id: "toolbar"
            [View] id: "navigationBarBackground"
            [View] id: "statusBarBackground"
  >>>
```

- You can use the `app.Flash()` method to get some details about the views in the results.

```
Xamarin.UITest REPL

Label => "CreditCardEntry_Container",
Text => null,
Class => "md51558244f76c53b6aeda52c8a337f2c37.EntryRenderer",
Enabled => true
}

>>> app.Flash(c=>c.Class("FormsEditText"))
Flashing query for Class("FormsEditText") gave 1 results.
[
  [0] {
    Id => "NoResourceEntry-3",
    Description => "md51558244f76c53b6aeda52c8a337f2c37.FormsEditText{c963de VFED..CL. .... 0,0-1440,181 #3}",
    Rect => {
      Width => 1440,
      Height => 181,
      X => 0,
      Y => 400,
      CenterX => 720,
      CenterY => 490
    },
    Label => "CreditCardEntry",
    Text => "",
    Class => "md51558244f76c53b6aeda52c8a337f2c37.FormsEditText",
    Enabled => true
  }
]

>>>
```

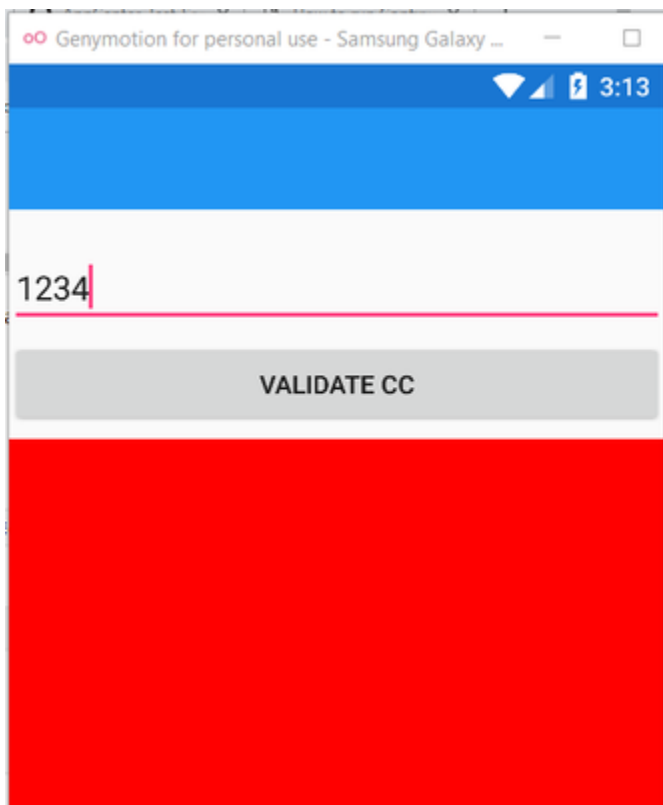
- You can use REPL to create an try individual steps in a test.

For example we can enter text in the Entry with the AutomatedId "CreditCardEntry" using `IApp.EnterText`.

```
Xamarin.UITest REPL
Android SDK Path: C:\Program Files (x86)\Android\android-sdk
Full log file: C:\Users\maria\AppData\Local\Temp\uitest\log-2019-05-03_12-13-10-493.txt
Skipping IDE integration as important properties are configured. To force IDE integration, add .PreferIdeSettings() to C
onfigureApp.
Android test running Xamarin.UITest version: 2.2.7
Initializing Android app on device 192.168.45.101:5555.

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

>>> app.EnterText("CreditCardEntry", "1234");
Using element matching Marked("CreditCardEntry").
Tapping coordinates [ 720, 490 ].
>>>
```



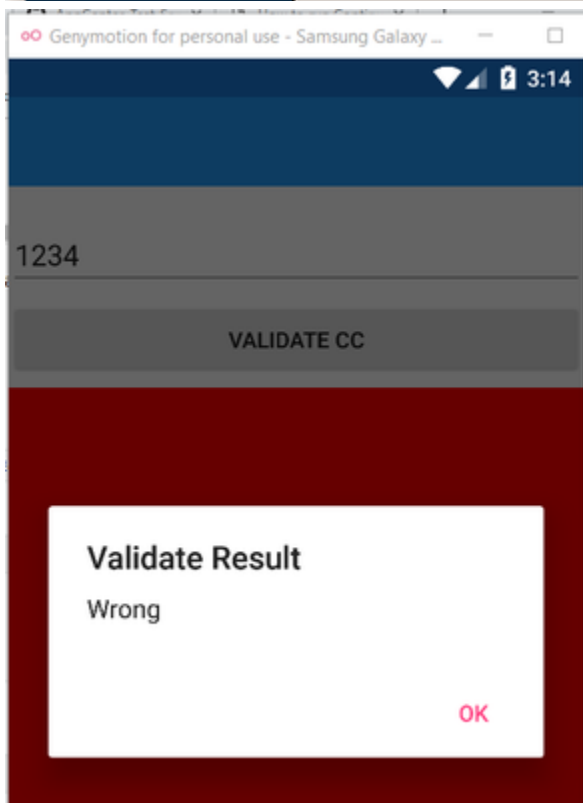
To simulate the user tapping on a button you can use `IApp.Tap`:

```
Xamarin.UITest REPL

Android SDK Path: C:\Program Files (x86)\Android\android-sdk
Full log file: C:\Users\maria\AppData\Local\Temp\uitest\log-2019-05-03_12-13-10-493.txt
Skipping IDE integration as important properties are configured. To force IDE integration, add .PreferIdeSettings() to C
onfigureApp.
Android test running Xamarin.UITest version: 2.2.7
Initializing Android app on device 192.168.45.101:5555.

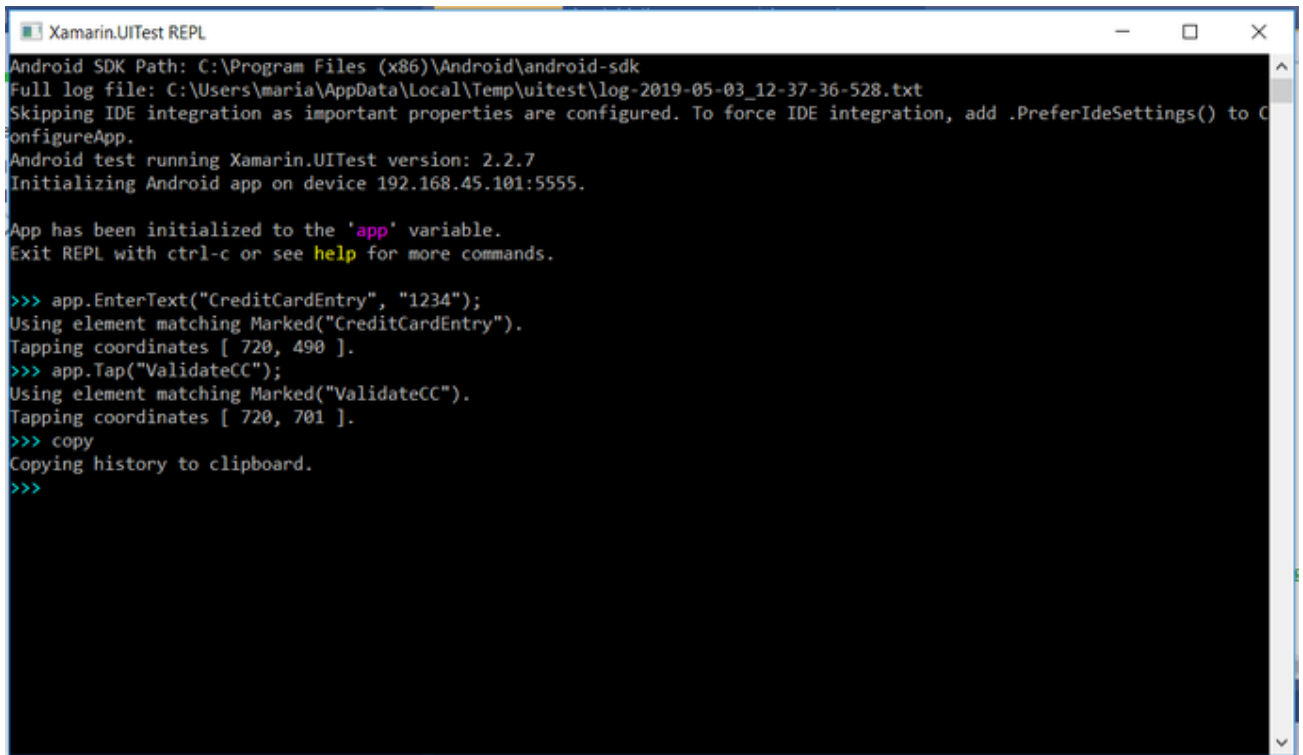
App has been initialized to the 'app' variable.
Exit REPL with ctrl-c or see help for more commands.

>>> app.EnterText("CreditCardEntry", "1234");
Using element matching Marked("CreditCardEntry").
Tapping coordinates [ 720, 490 ].
>>> app.Tap("ValidateCC");
Using element matching Marked("ValidateCC").
Tapping coordinates [ 720, 701 ].
>>>
```



- The REPL will keep a history of the commands that you have entered into it during the session. The **copy command** paste those commands into the clipboard.





```
Xamarin.UITest REPL
Android SDK Path: C:\Program Files (x86)\Android\android-sdk
Full log file: C:\Users\maria\AppData\Local\Temp\uitest\log-2019-05-03_12-37-36-528.txt
Skipping IDE integration as important properties are configured. To force IDE integration, add .PreferIdeSettings() to C
onfigureApp.
Android test running Xamarin.UITest version: 2.2.7
Initializing Android app on device 192.168.45.101:5555.

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

>>> app.EnterText("CreditCardEntry", "1234");
Using element matching Marked("CreditCardEntry").
Tapping coordinates [ 720, 490 ].
>>> app.Tap("ValidateCC");
Using element matching Marked("ValidateCC").
Tapping coordinates [ 720, 701 ].
>>> copy
Copying history to clipboard.
>>>
```

## Appcenter

Read [Get started with UITest and Xamarin.Forms](#).

## UITest setup

- Add new project > Visual C# > Test > **Xamarin.UI Cross-Platform Test Project**
- Once the test project has been added it'll install two NuGet packages that UITest needs - NUnit and Xamarin.UITest. It's worth **updating the Xamarin.UITest** NuGet package to the latest version, as they often push out bug fixes to ensure it works on the latest mobile OS versions. **Don't update NUnit. UITest only works with NUnit 2**, not NUnit 3 and if you update this package your tests won't work and you'll need to remove the package and re-install NUnit 2.
- UI Test project has two files auto-generated for you:
  - **AppInitializer.cs**: is a static helper class with a single static method (`StartApp`) which is used to start your app. The `StartApp` method returns an instance of `IApp` that your tests can use. This method uses a helper class called `ConfigureApp` to start the app, and this helper class has a fluent API that allows you to configure and run your app.
  - **Test.cs**: contains an UI test to run. This test fixture has a parameterized constructor that takes the platform to run the tests on as one of the values from the `Platformenum`, either `Platform.iOS` or `Platform.Android`, and has two `TestFixture` attributes, one for each platform. This means that we have two test fixtures - one Android and one iOS. This fixture has a `setup` method that uses the `AppInitializer` to start the app before each test, and a single test that calls the `Screenshot` method on the `IApp` returned from the app initializer to take a screenshot.

```

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

    public Tests(Platform platform)
    {
        this.platform = platform;
    }

    [SetUp]
    public void BeforeEachTest()
    {
        app = AppInitializer.StartApp(platform);
    }
}

```

- **Initialize application for test:** StartApp function in AppInitializer.cs is where the test environment is configured. This function will be called during the test setup, in order to provide a clean test environment for tests to run.
  - Here we're asking the initializer to do the following:
    - Specified an installed application on the device (or you can supply an APK file)
    - Enable on-demand screenshot capture as specified in the tests. The screenshots are saved in <MyTestProject>\bin\Debug.
    - Decide automatically if the test should run on the emulator or connected device
    - Start the application

```

public static IApp StartApp(Platform platform)
{
    if (platform == Platform.Android)
    {
        return ConfigureApp
            .Android
            .InstalledApp("com.companyname.NareiaMobileBase")
            .EnableLocalScreenshots()
            .StartApp();
    }

    return ConfigureApp
        .iOS
        .AppBundle("../..\\..\\NareiaMobileBase.iOS\\bin\\iPhoneSimulator\\Debug\\NareiaMobileBase.iOS.app")
        .EnableLocalScreenshots()
        .StartApp();
}

```

## Using marked to locate views

- We can use Marked attribute or AppQuery to locate for the UI element to act on. In Xamarin.Forms, Marked attribute is represented by the AutomationId in Xamarin.Forms. In this case we are going to use **Marked** so in each of the item that we wish to automate, we should assign a unique name to its AutomationId attribute.

```

<ScrollView x:Name="mainSv" AutomationId="mainSv" Grid.Row="0" Scrolled="ScrollView_Scrolled">
    <StackLayout x:Name="mainSvContent">
        <!-- CC Controls -->
        <Entry x:Name="entryCC" AutomationId="CreditCardEntry" Placeholder="At least 5 character"/>
        <Button AutomationId="ValidateCC" Text="Validate CC" Clicked="Button_Clicked"/>
    </StackLayout>
</ScrollView>

```

- Sometimes UI rendering operation is much slower than our test sequence and that rendering time may depend on many external factors, such as network performance, code execution speed, hardware limitation etc. So we should introduce some delay before checking for result, and we can do this by using **WaitForElement API**.

```

[Test]
public void CCEntry_MoreOrEqual5Digit_Correct()
{
    app.EnterText("CreditCardEntry", "12345");
    app.Tap("ValidateCC");
    app.WaitForElement("message");
    Assert.IsTrue(app.Query(x => x.Id("message").Text("Correct")).Any());
    SaveScreenshot();
}

```

## Possible errors

Run test finished: 0 run

```

[5/3/2019 11:23:48 AM Informational] ----- Run test started -----
[5/3/2019 11:23:49 AM Informational] ===== Run test finished: 0 run (0:00:00.4254568) =====

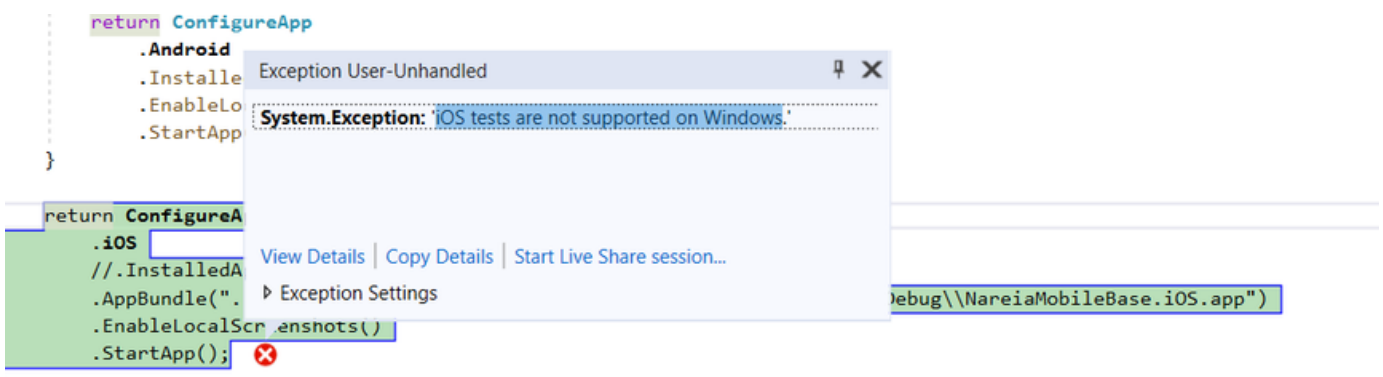
```

### Solution:

Go to Tools Options Test General and uncheck the "For improved performance, only use test adapters in test assembly folder or as specified in runsettings file".

You can also change the log level to Diagnostic to get more information in the output window.

## iOS tests are not supported on Windows



**Solution:** running iOS Xamarin.UITests from Windows using the Mac Agent is not supported you have to use a Mac for running them.

### Links consultados

- [Xamarin.UITest.](#)
- [Introduction to UI Testing with Xamarin.](#)
- [Xamarin.Forms How to: Perform automated UI test.](#)
- [Working with iOS Simulators.](#)
- [Xamarin not starting remotely on mac agent from windows: "iOS tests are not supported on Windows".](#)
- [How to run test using VS on iOS simulator\(on mac\).](#)

- Github repository with **UITest example**