Xamarin

Xamarin is a very interesting and intriguing product allowing development of native and cross-platform (iOS, Android, and Windows Phone) apps in Visual Studio using C#

As a side note, Xamarin is now part of Microsoft Microsoft to acquire Xamarin and empower more developers to build apps on any device

To using and developing with Xamarin there are 2 options:

- Visual Studio (https://www.visualstudio.com/vs/)
- Xamarin Studio (https://www.xamarin.com/platform)
    - Mac OS
    - Windows

Xamarin Studio

Language support C# and F#
Native Platform Integration Xamarin Studio features integration with native platform tools for both iOS and Android. This allows using various code-signing, deployment, and diagnostics tools.

Visual Designers

iOS Designer

The iOS Designer is fully integrated with Xamarin Studio and enables visual editing of .xib and Storyboard files to create iOS, tvOS, and WatchOS UIs and transitions. The entire user interface can be built using drag-and-drop functionality between the Toolbox and Design Surface, while using an intuitive approach to handling events. The iOS Designer also supports custom controls with the added benefit of design-time rendering.

Android Designer

Android Designer works with Android .axml files to visually construct user interfaces.

Xamarin Studio has built in integration for Xamarin Test Cloud.

Also, Xamarin Studio allows work with Nuget packages and integrates with git for version control

Cross-platform development with Xamarin

Compilation

The C# source makes its way into a native app in very different ways on each platform:

- **iOS** – C# is ahead-of-time (AOT) compiled to ARM assembly language. The .NET framework is included, with unused classes being stripped out during linking to reduce the application size. Apple does not allow runtime code generation on iOS, so some language features are not available (see Xamarin.iOS Limitations).
- **Android** – C# is compiled to IL and packaged with MonoVM + JIT'ing. Unused classes in the framework are stripped out during linking. The application runs side-by-side with Java/ART (Android runtime) and interacts with the native types via JNI (see Xamarin.Android Limitations).
- **Windows** – C# is compiled to IL and executed by the built-in runtime, and does not require Xamarin tools. Designing Windows applications following Xamarin's guidance makes it simpler to re-use the code on iOS and Android. Note that the Universal Windows Platform also has a **.NET Native** option which behaves similarly to Xamarin.iOS' AOT compilation.

Platform SDK Access

Xamarin makes the features provided by the platform-specific SDK easily accessible with familiar C# syntax:

- **iOS** – Xamarin.iOS exposes Apple's CocoaTouch SDK frameworks as namespaces that you can reference from C#. For example the UIKit framework that contains all the user interface controls can be included with a simple using MonoTouch.UIKit; statement.
- **Android** – Xamarin.Android exposes Google's Android SDK as namespaces, so you can reference any part of the supported SDK with a using statement, such as using Android.Views; to access the user interface controls.
- **Windows** – Windows apps are built using Visual Studio on Windows. Project types include Windows Forms, WPF, WinRT, and the Universal Windows Platform (UWP).

User Interface

With Xamarin, an application user interface uses native controls on each platform, creating apps that are indistinguishable from an application written in Objective-C or Java (for iOS and Android respectively).

Visual Designer

Each platform has a different method for visually laying out screens:

- **iOS** – Xamarin's iOS Designer for Xamarin Studio and Visual Studio facilitates building Views using drag-and-drop functionality and property fields. Collectively these Views make up a Storyboard, and can be accessed in the .STORYBOARD file that is included in your project.
- **Android** – Xamarin provides an Android drag-and-drop UI designer for both Xamarin Studio and Visual Studio. Android screen layouts are saved as .AXML files when using Xamarin tools.
- **Windows** – Microsoft provides a drag-and-drop UI designer in Visual Studio and Blend. The screen layouts are stored as .XAML files.

Library and Code Re-Use

C# Source and Libraries

Given development done in C#, C# Source and Libraries can be reused in Xamarin.iOS or Xamarin.Android projects.

Objective-C Bindings + Binding Projects

Xamarin provides a tool called *btouch* that helps create bindings that allow Objective-C libraries to be used in Xamarin.iOS projects. Refer to the Binding Objective-C Types documentation for details on how this is done.

.jar Bindings + Binding Projects

Xamarin supports using existing Java libraries in Xamarin.Android. Refer to the Binding a Java Library documentation for details on how to use a .JAR file from Xamarin.Android.

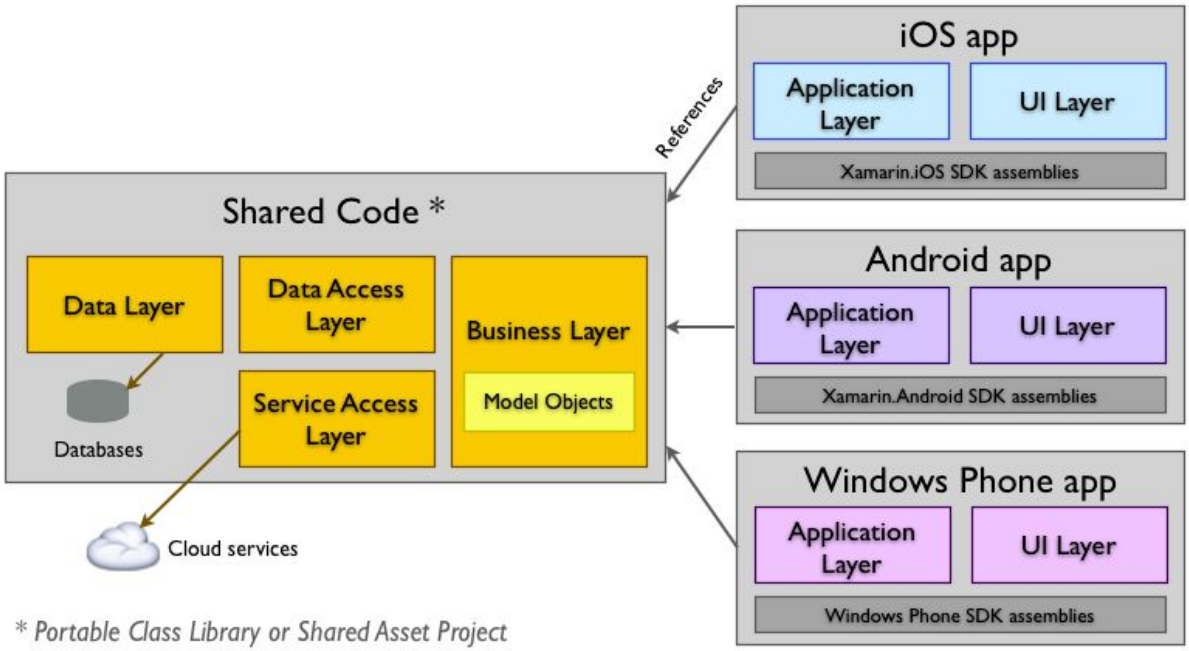
Open-source Xamarin.Android bindings are available on github.

C via PInvoke

"Platform Invoke" technology (P/Invoke) allows managed code (C#) to call methods in native libraries as well as support for native libraries to call back into managed code.

Developing for cross-platform implies sharing code, and these are the options.

Shared Asset Projects



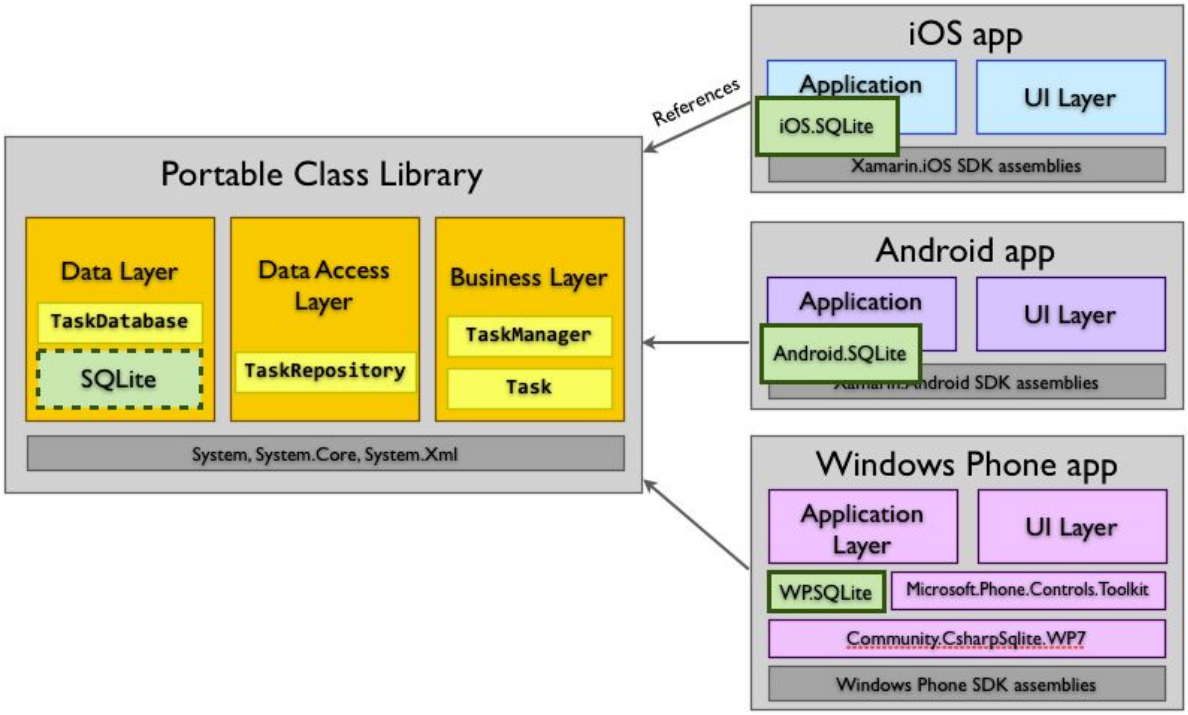
Benefits

- Allows you to share code across multiple projects.
- Shared code can be branched based on the platform using compiler directives (eg. using #if __ANDROID__ , as discussed in the Building Cross Platform Applications document).
- Application projects can include platform-specific references that the shared code can utilize (such as using Community.CsharpSqlite.WP7 in the Tasky sample for Windows Phone).

Disadvantages

- Unlike most other project types, a Shared Project has no 'output' assembly. During compilation, the files are treated as part of the referencing project and compiled into that DLL. If you wish to share your code as a DLL then Portable Class Libraries are a better solution.
- Refactorings that affect code inside 'inactive' compiler directives will not update the code.

Portable Class Library



Benefits

- Allows you to share code across multiple projects.
- Refactoring operations always update all affected references.

Disadvantages

- Cannot use compiler directives.
- Only a subset of the .NET framework is available to use, determined by the profile selected (see the Introduction to PCLfor more info).

Xamarin UI

Xamarin provides its own solution that allows developing a cross-platform UI in the form of Xamarin Forms.

User Interface

There are four main control groups used to create the user interface of a Xamarin.Forms application.

1. **Pages** – Xamarin.Forms pages represent cross-platform mobile application screens. For more information about Pages, see Xamarin.Forms Pages.

2. **Layouts** – Xamarin.Forms layouts are containers used to compose views into logical structures. For more information about Layouts, see Xamarin.Forms Layouts.
3. **Views** – Xamarin.Forms views are the controls displayed on the user interface, such as labels, buttons, and text entry boxes. For more information about Views, see Xamarin.Forms Views.
4. **Cells** – Xamarin.Forms cells are specialized elements used for items in a list, and describe how each item in a list should be drawn. For more information about Cells, see Xamarin.Forms Cells.

Pages: Page represents an *Activity* in Android, a *View Controller* in iOS, or a *Page* in the Windows Universal Platform (UWP). The sample in the screenshots above instantiates a ContentPage object and uses that to display a Label

Also Xamarin offers a few options in terms of pages that will allow presenting the information in various formats, remaining to choose the most suitable.

ContentPage

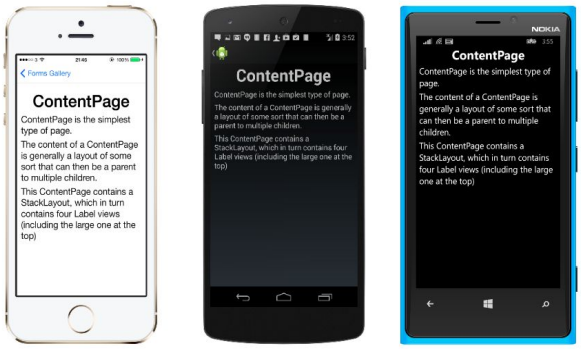
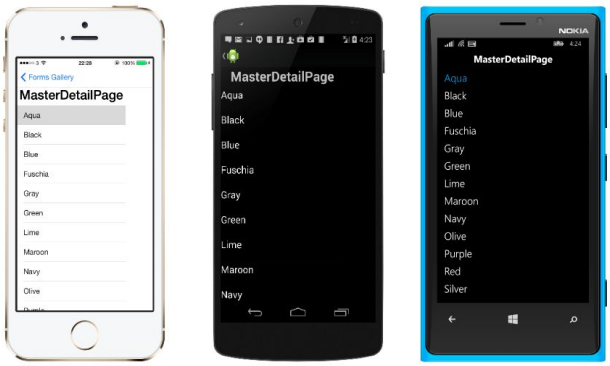
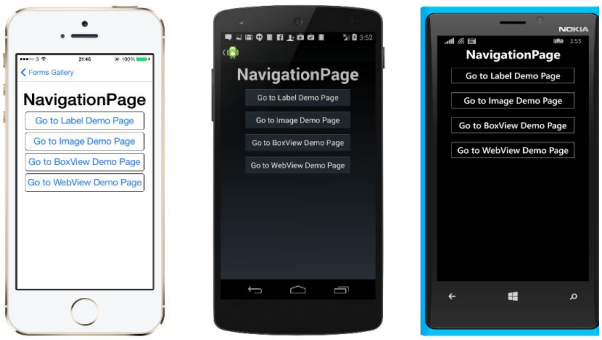
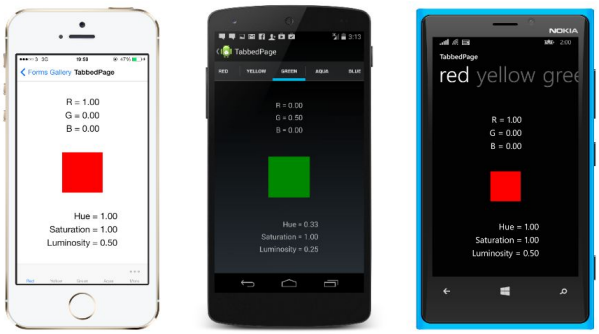
MasterDetailPage

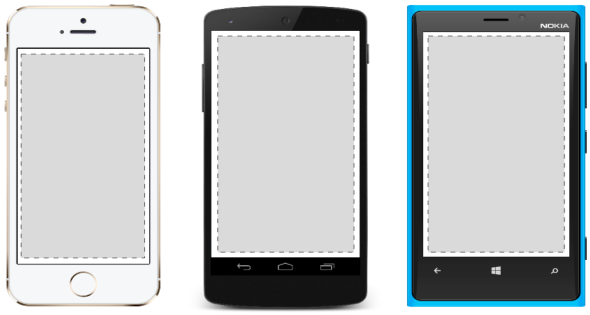
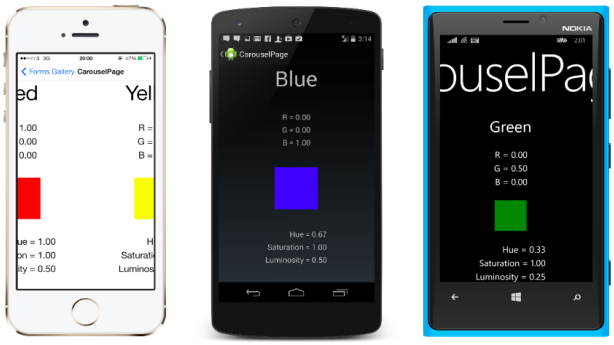
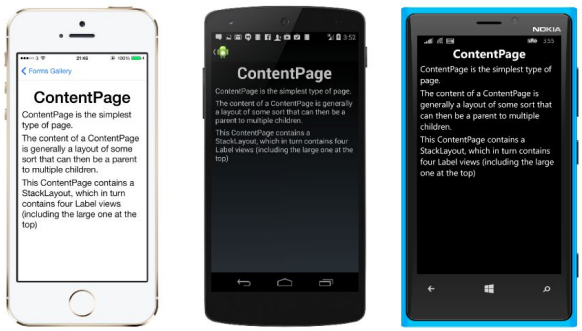
NavigationPage

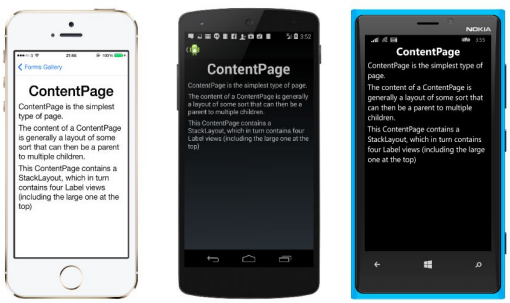
TabbedPage

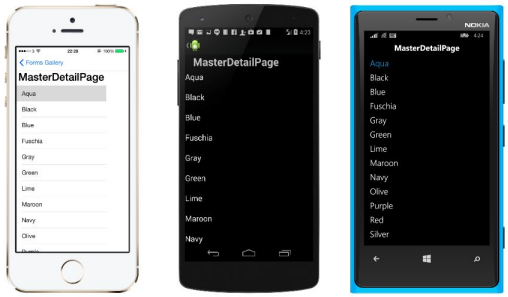
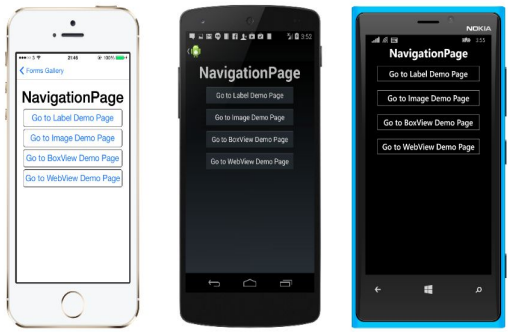
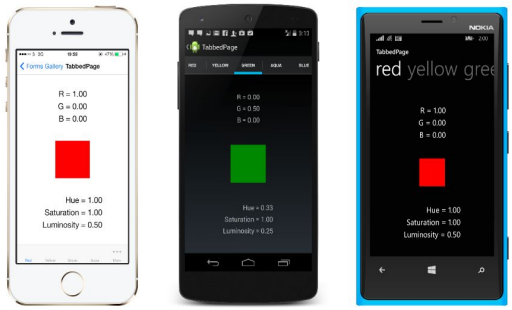
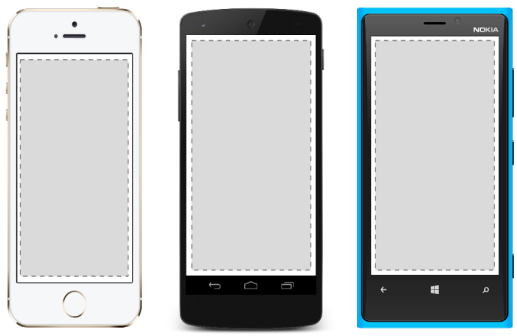
TemplatedPage

CarouselPage

ContentPage	A ContentPage displays a single View, often a container such as a StackLayout or a ScrollView.	
MasterDetailPage	A Page that manages two panes of information.	
NavigationPage	A Page that manages the navigation and user-experience of a stack of other pages.	
TabbedPage	A Page that allows navigation between children pages, using tabs.	

TemplatedPage	A Page that displays full-screen content with a control template, and the base class forContentPage.	
CarouselPage	A Page allowing swipe gestures between subpages, like a gallery.	
ContentPage	A ContentPage displays a single View, often a container such as a StackLayout or a ScrollView.	

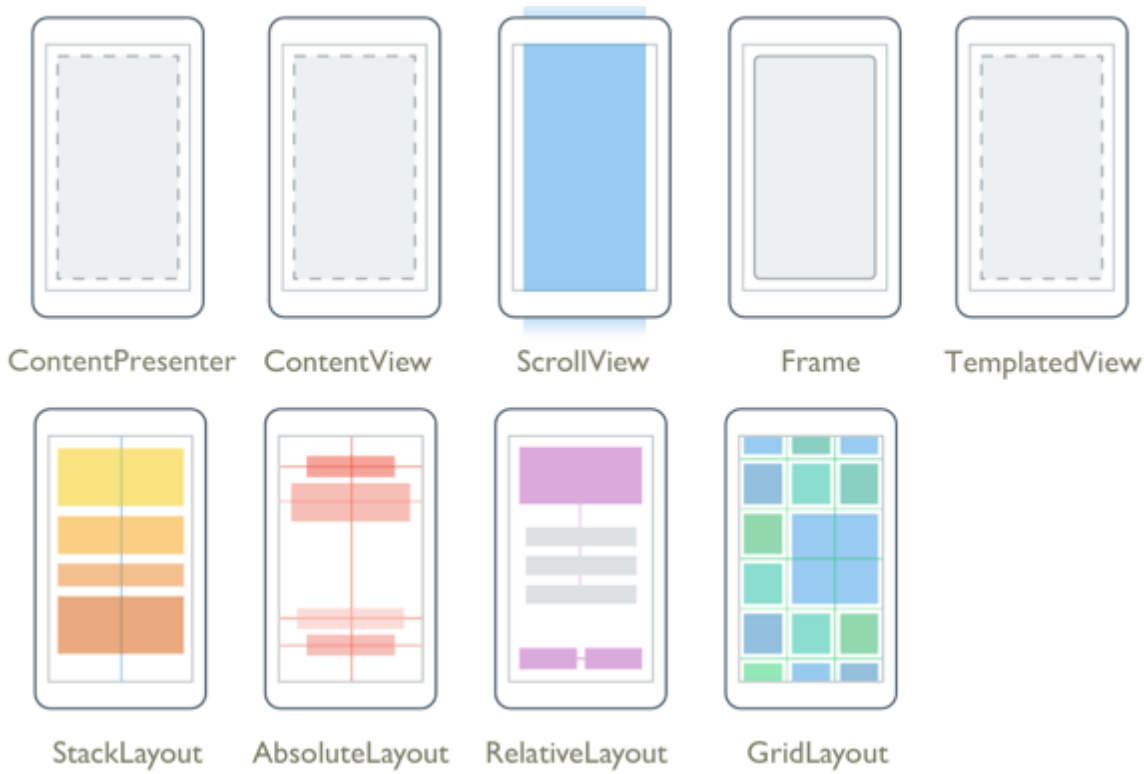
TYPE	DESCRIPTION	SCREENSHOT
ContentPage	A ContentPage displays a single View, often a container such as a StackLayout or a ScrollView.	

MasterDetailPage	A Page that manages two panes of information.	
NavigationPage	A Page that manages the navigation and user-experience of a stack of other pages.	
TabbedPage	A Page that allows navigation between children pages, using tabs.	
TemplatedPage	A Page that displays full-screen content with a control template, and the base class forContentPage.	

CarouselPage	A Page allowing swipe gestures between subpages, like a gallery.	

.

The layouts also



ContentPresenter

ContentView

Frame

ScrollView

TemplatedView

AbsoluteLayout

Grid

RelativeLayout

StackLayout

Views -- Are what in other frameworks

ActivityIndicator

BoxView

Button

DatePicker

Editor

Entry

Image

Label

ListView

OpenGLView

Picker

ProgressBar

SearchBar

Slider

Stepper

Switch

TableView

TimePicker

Cells -- Xamarin

EntryCell

SwitchCell

Text Cell

ImageCell

Xamarin Renderers

The way Xamarin manages to generate UI available cross-platform is through Renders, which render the controls for a particular platform.

As such, Xamarin provides renders for each platform that it supports and can be located in the following namespaces.

Xamarin treats the *MapRenderer* class, a class used for displaying maps using platform native map APIs on each platform, differently. That is for providing a fast and familiar map experience for users and as a result it's been. As a result the *MapRenderer* is provided in different namespaces than other renderers.

- **iOS** – Xamarin.Forms.Maps.iOS
- **Android** – Xamarin.Forms.Maps.Android
- **Windows Phone 8** – Xamarin.Forms.Maps.WP8
- **WinRT** – Xamarin.Forms.Maps.WinRT
- **Universal Windows Platform (UWP)** – Xamarin.Forms.Maps.UWP

While the Xamarin renderers can be found in

- **iOS** – Xamarin.Forms.Platform.iOS
- **Android** – Xamarin.Forms.Platform.Android
- **Android (AppCompat)** – Xamarin.Forms.Platform.Android.AppCompat
- **Windows Phone 8** – Xamarin.Forms.Platform.WinPhone
- **WinRT** – Xamarin.Forms.Platform.WinRT
- **Universal Windows Platform (UWP)** – Xamarin.Forms.Platform.UWP

For more information please visit Renderer Base Classes and Native Controls.

Xamarin Behaviors

Behaviors lets you add functionality to user interface controls without having to subclass them. Behaviors are written in code and added to controls in XAML or code.

Xamarin provides a few types of behaviors:

- Attached Behaviors
- Xamarin.Forms Behaviors
- Reusable Behaviors

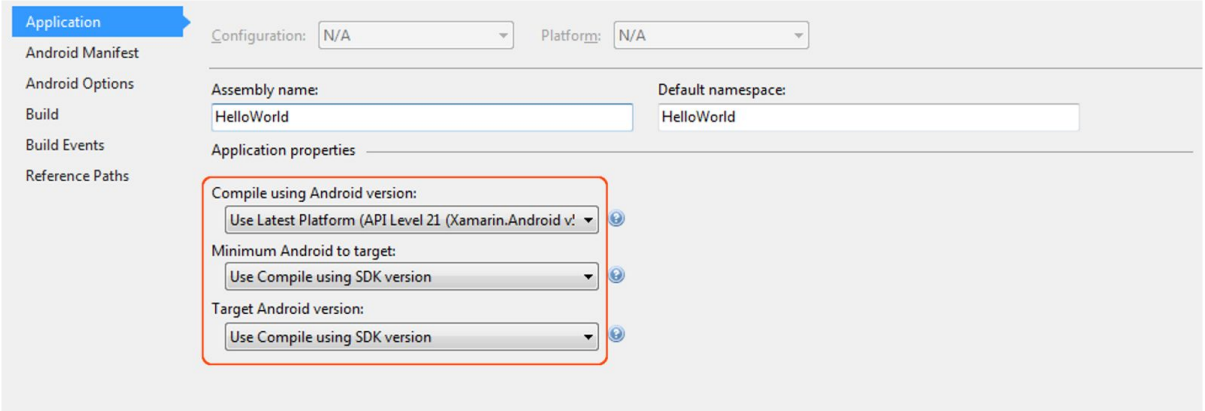
Xamarin Android

Xamarin.Android exposes three Android API level project settings:

Target Framework – Specifies which framework to use in building your application.This API level is used at compile time by Xamarin.Android.

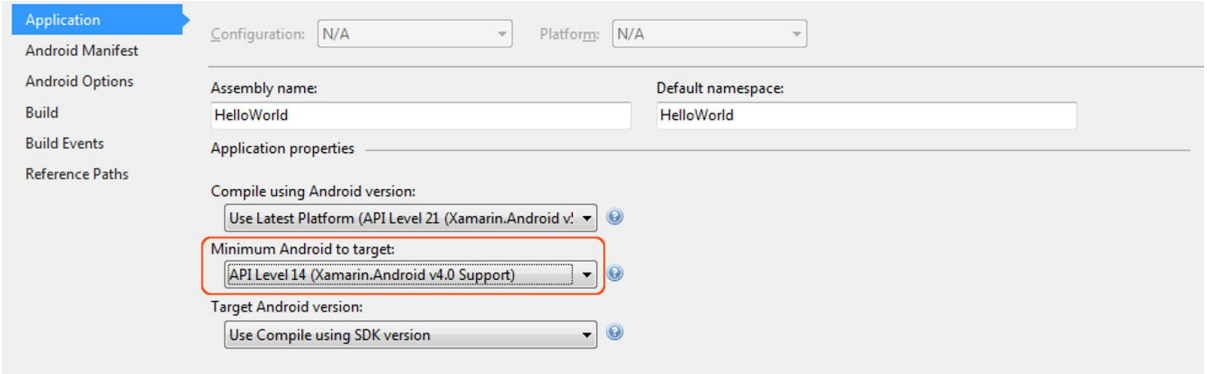
Minimum Android Version – Specifies the oldest Android version that you want your app to support. This API level is used at run time by Android.

Target Android Version – Specifies the version of Android that your app is intended to run on.This API level is used at run time by Android.



If you want to maintain backward compatibility with an earlier version of Android, change Minimum Android version to target to the oldest version of Android that you want

your app to support. The following example configuration supports Android versions from API Level 14 thru API level 21:

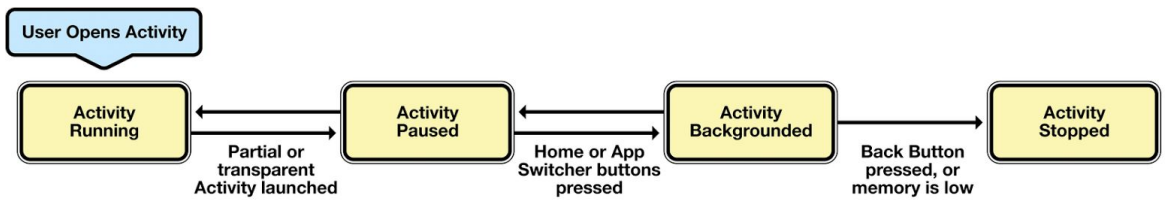


Activity Lifecycle

Activities are a fundamental building block of Android applications and they can exist in a number of different states.The activity lifecycle begins with instantiation and ends with destruction, and includes many states in between.When an activity changes state, the appropriate lifecycle event method is called, notifying the activity of the impending state change and allowing it to execute code in order to adapt to that change. This article examines the lifecycle of activities and explains the responsibility that an activity has during each of these state changes in order to be part of a well-behaved, reliable application.

Activity States

The Android OS arbitrates Activities based on their state. This helps Android identify activities that are no longer in use, allowing the OS to reclaim memory and resources. The following diagram illustrates the states an Activity can go through during its lifetime:

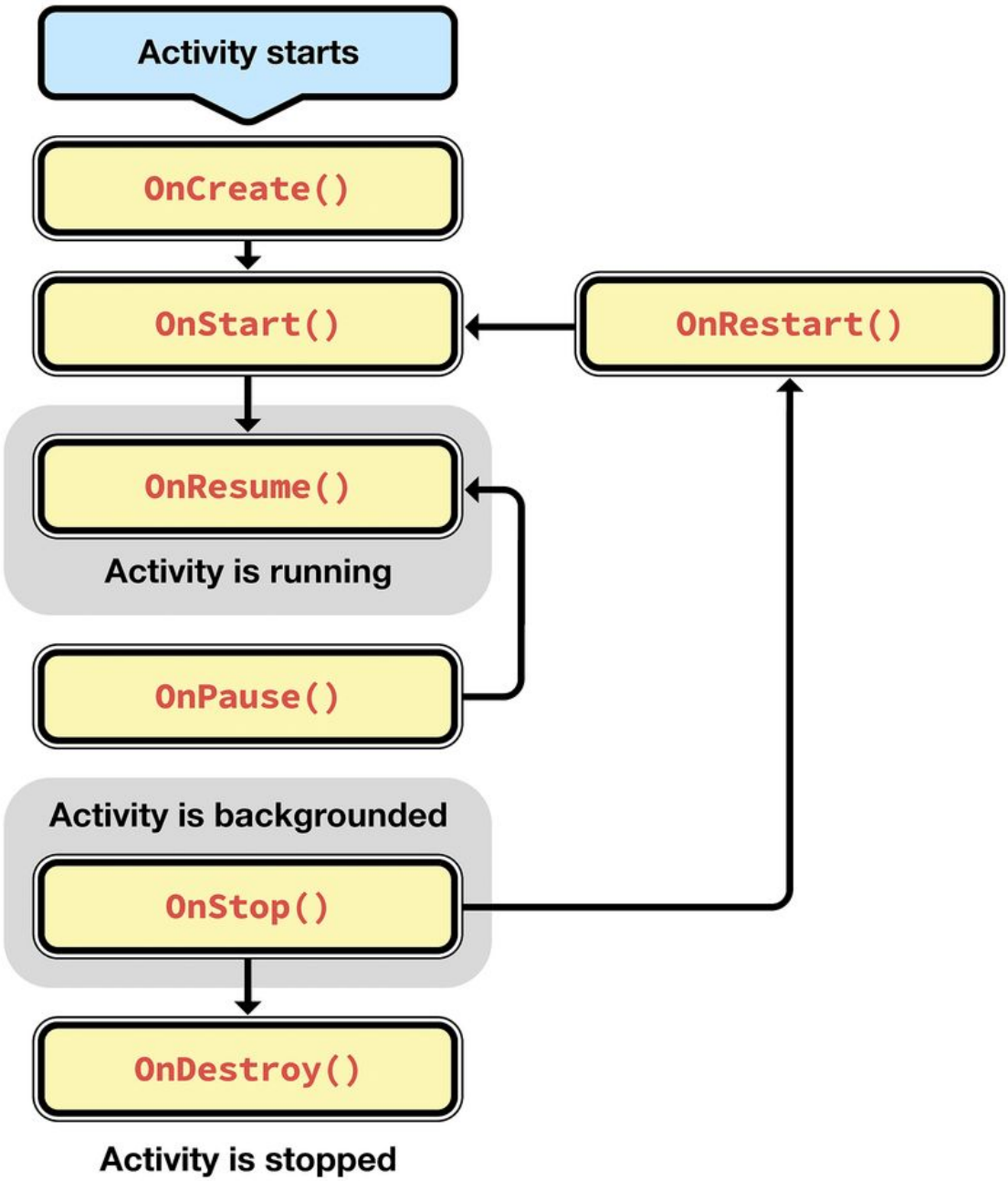


Activity Re-Creation in Response to Configuration Changes

To make matters more complicated, Android throws one more wrench in the mix called Configuration Changes.Configuration changes are rapid activity destruction/ re - creation cycles that occur when the configuration of an activity changes, such as when the device is rotated (and the activity needs to get re-built in landscape or portrait mode), when the keyboard is displayed (and the activity is presented with an opportunity to resize itself), or when the device is placed in a dock, among others.

Activity Lifecycle Methods

The Android SDK and, by extension, the Xamarin.Android framework provide a powerful model for managing the state of activities within an application. When an activity's state is changing, the activity is notified by the OS, which calls specific methods on that activity. The following diagram illustrates these methods in relationship to the Activity Lifecycle:



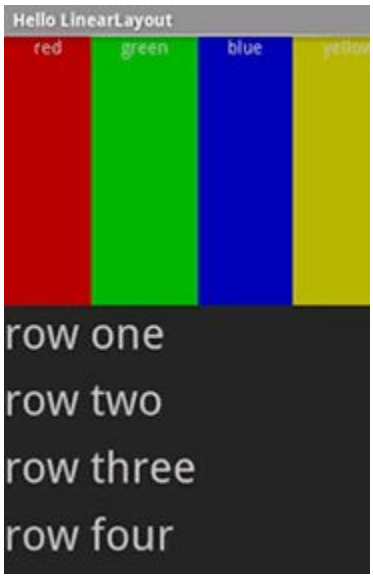
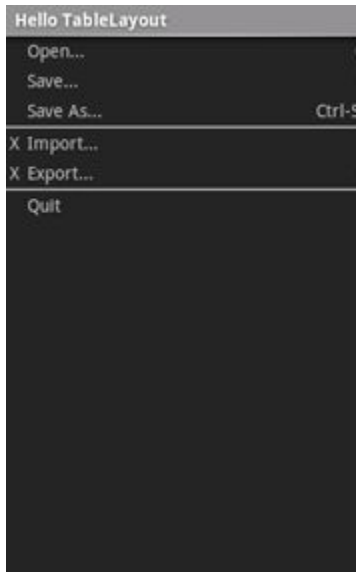
User Interface

Below is a presentation of common Xamarin.Android layouts and widgets that can be used to build UI.

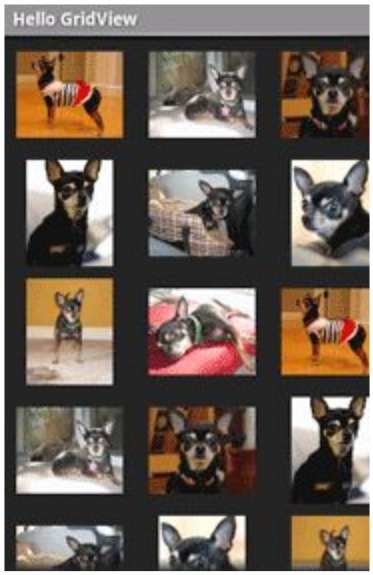
ListView and Adapters

List views and adapters are one of the most fundamental building blocks of Android Applications, check out the guide on them here.

Layouts

Linear Layout	Relative Layout	Table Layout
		

Grid View	Tab Layout	List View



Widgets & Other Views

Date Picker

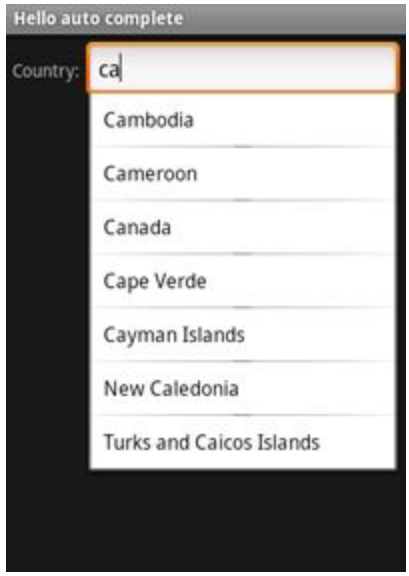
Time Picker

Form Elements

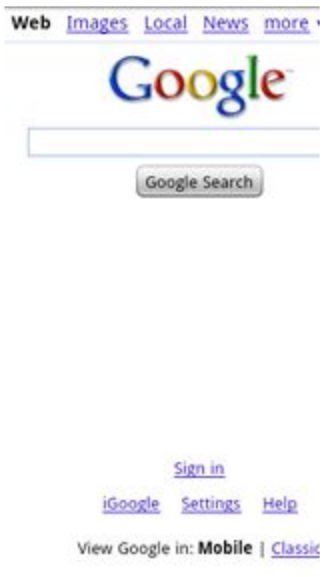
Spinner

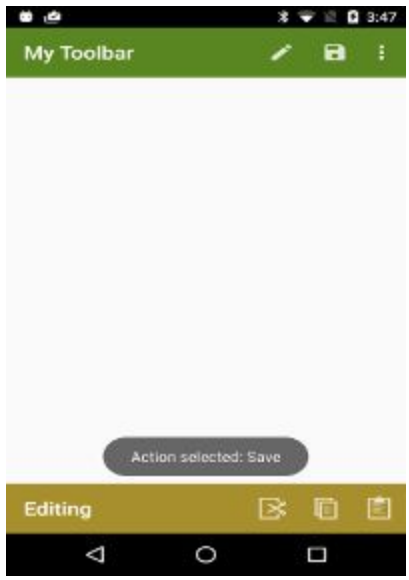
Auto Complete

Gallery



Web View Toolbar





Android user interfaces can be created declaratively by using XML files or programmatically by writing code. Xamarin.Android offers an Xamarin.Android Designer that allows developers to create and modify declarative layouts visually, without having to deal with the tedium of hand-editing XML files. The Designer also provides real-time feedback,

which lets the developer evaluate UI changes without having to redeploy the application to a device or to an emulator.

In order to help design user interfaces that target many devices, the Designer comes with a variety of device configurations built in. It also supports adding additional device configurations; these configurations are based on *qualifiers* that you specify to distinguish one device configuration from another. There are many different types of qualifiers. For more information about these resource types, see Android Resources.

As a note, the Designer allows creation of Alternative Layouts, that allows display of UI in other languages than English and support for Material Design.

Xamarin iOS

Xamarin.iOS uses controls to expose events for most user interactions. Xamarin.iOS applications consume these events in much the same way as do traditional .NET applications. For example, the Xamarin.iOS UIButton class has an event called TouchUpInside and consumes this event just as if this class and event were in a .NET app.

Besides this .NET approach, Xamarin.iOS exposes another model that can be used for more complex interaction and data binding. This methodology uses what Apple calls delegates and protocols. Delegates are similar in concept to delegates in C#, but instead of defining and calling a single method, a delegate in Objective-C is an entire class that conforms to a protocol. A protocol is similar to an interface in C#, except that its methods can be optional. So for example, in order to populate a UITableView with data, you would create a delegate class that implements the methods defined in the UITableViewDataSource protocol that the UITableView would call to populate itself.

This is what Xamarin uses it for building the UI.

- **Events** – Using .NET events with UIKit controls.
- **Protocols** – Learning what protocols are and how they are used, and creating an example that provides data for a map annotation.
- **Delegates** – Learning about Objective-C delegates by extending the map example to handle user interaction that includes an annotation, then learning the difference between strong and weak delegates and when to use each of these.

Xamarin Test Cloud

The Xamarin Test Cloud ecosystem consists of the following parts:

Calabash – This is a framework that allows tests to be written with Cucumber and Ruby.Calabash tests are well suited to Behavior Driven Development – a methodology that focuses on creating executable specifications. The specifications are written in the everyday language that business would use, and then developers write tests to automate a mobile app by parsing the specifications.

Xamarin.UITest – This is a framework that allows tests to be written in C# using the popular NUnit testing library. This framework is well suited to teams that are already skilled with writing NUnit tests and/or already developing their mobile applications using Xamarin. Test Cloud – Test Cloud is a cloud based service consisting of thousands of physical mobile devices. Users upload their apps and tests to Test Cloud, which will install the apps on the devices and run the tests. When the tests are complete, Test Cloud, the results made available to users through an easy to use and informative web - based front end.

Xamarin Test Recorder – This tool, still under development, can simplify creating tests and is great for someone who is new to Xamarin.UITest and unfamiliar with the APIs.Testers can start Test Recorder, connect it to device, simulator, or emulator, and then start using the mobile app.Test Recorder will capture the interactions between the user and the mobile app, and output a Xamarin.UITest in C# for that scenario.

The Anatomy of the Test Cloud Framework

Automated interactions with a mobile application require some sort of automation library that will simulate a user's action and allow the test to verify the state of the user interface to prove the application is working correctly. Both Android and iOS have their own proprietary UI automation frameworks.

These two very different set of testing APIs would prove challenging to a tester who wants to write cross-platform tests. To help address is concern, one or two helper components will be installed on the mobile device along with the app:

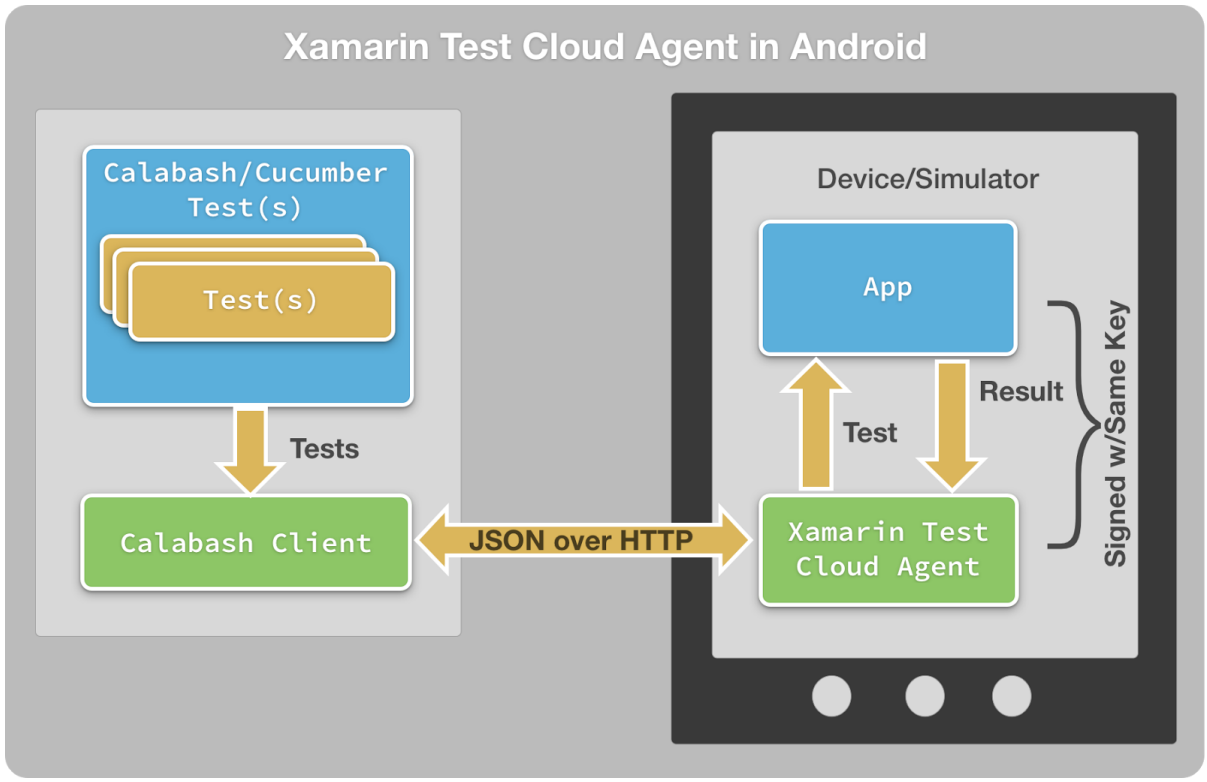
Xamarin Test Cloud Agent – This is a lightweight HTTP server that interacts with tests via JSON over HTTP.The Xamarin Test Cloud Agent is a middle - man that will take queries (and in some cases, actions) from the tests and perform them on the application being tested.The Xamarin Test Cloud Agent is required for both Android and iOS applications, and has a slightly different role on each platform.

DeviceAgent – This is installed only iOS applications that are built with Xcode 8.Logically it is very similar to t Xamarin Test Cloud Agent – it is responsible for perform gestures and advanced queries on iOS views but does so using a different set of testing APIs. We will learn more about the DeviceAgent below.

Let's see what role each component plays on the respective platforms.

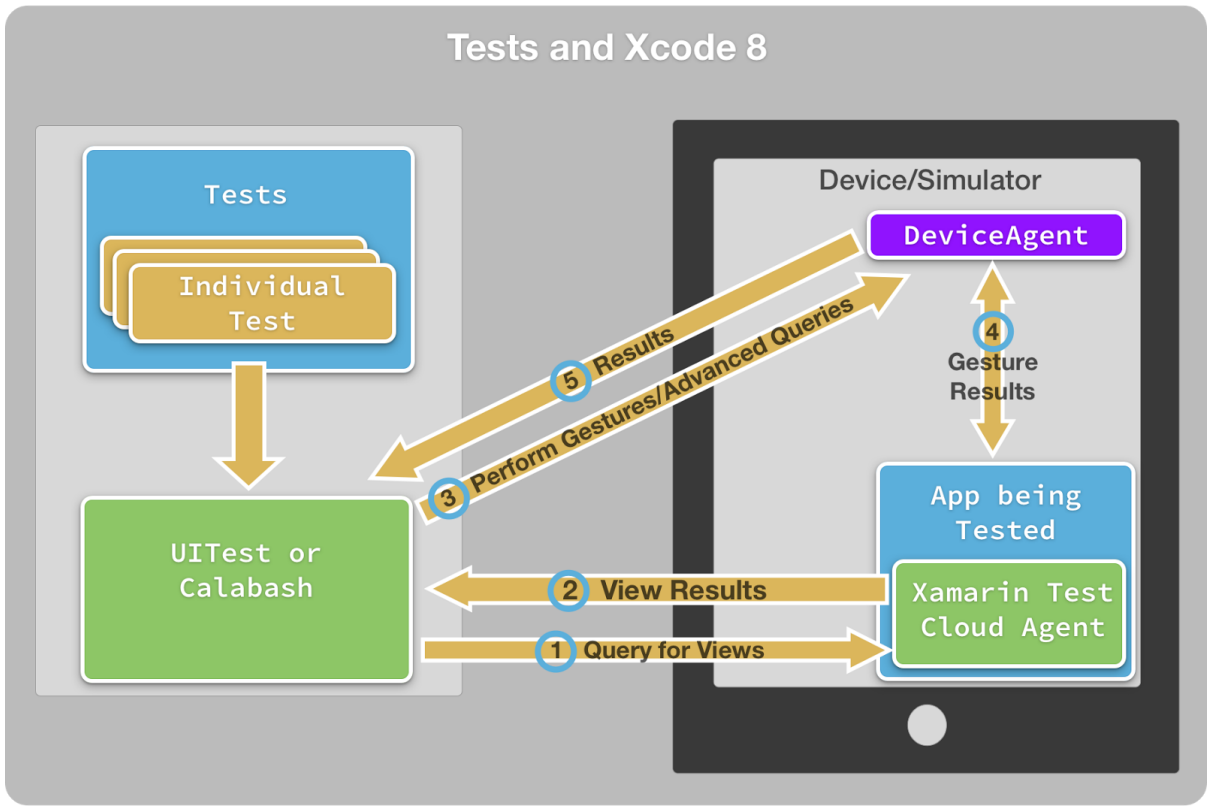
Xamarin Test Cloud Agent on Android

In Android, The Xamarin Test Cloud Agent is responsible for using the Android automation APIs to control the user interface and to locate views so that a test may interact with them.It is bundled into a separate APK and runs as a separate application, which has permission to automate the application under test.This is possible, because when the test is deployed to the mobile device, Calabash or UITest will sign both application packages with the same key.



Xamarin Test Cloud Agent on iOS

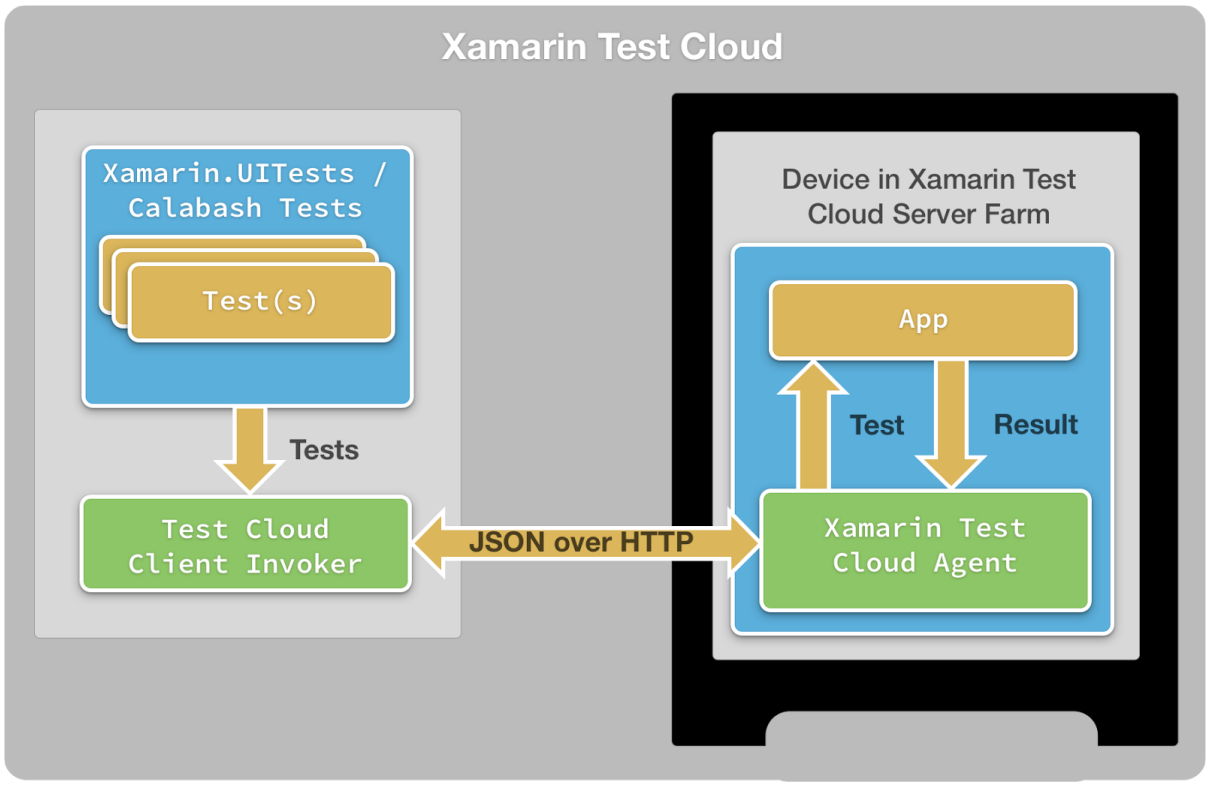
The Test Cloud Agent has a slightly different role in iOS applications – by itself it does not automate the app under test. Instead, the Test Cloud Agent will interrogate the active window and retrieve information about the views for to the user. The view information is returned to the test script. The test script will then automate the iOS application with the help of another component called the *DeviceAgent*. The DeviceAgent will simulate the gestures and actions for the test (using the automation API's provide with Xcode 8) and if necessary return the result of those interactions to the test:



The Xamarin Test Cloud Agent is available via a NuGet Package, and must be included in the App Bundle before a test can be executed. The Xamarin Test Cloud Agent should only be included in Debug builds of the application. Apple will reject apps that are submitted with the Xamarin Test Cloud Agent linked into the App Bundle.

Running Tests in Xamarin Test Cloud

Running Tests in Xamarin Test Cloud is conceptually similar to running tests locally, except that Xamarin Test Cloud hosts the tests and will execute them on selected devices:



To facilitate this, tests are uploaded along with the app when it's pushed to Xamarin Test Cloud for testing. Xamarin Test Cloud will then reset the device to a clean state, install the app, and run the tests.

Xamarin Test Recorder

As you can expect is a tool for recording automated user interface tests.
The pattern for recording UI Tests is the same for both Android and iOS projects:

- Start Test Recorder, specify the application to be tested, and select the device to run application on.
- Interact with the application and the Test Recorder will create a C# test method.
- Have the Test Recorder submit the test script to Xamarin Test Cloud, or incorporate the test into a Xamarin UITest project.

Requirements

You must be familiar with Xamarin Test Cloud and Xamarin.UITest.

The computer running Test Recorder must have the Android SDK installed with the following:

- Android 4.3 (API level 19) or higher.
- Android SDK Build-tools
- Android SDK Platform-tools

In closing, a book about Xamarin has been recently published and can be downloaded from here:
https://blog.xamarin.com/xamarin-forms-book-now-available-in-easy-to-digest-chapter-summaries/

Xamarin Intro
https://www.dropbox.com/s/gk450dvx2w31tz0/XamarinIntro.pdf

Xamarin Intro