Introduction to mobile development with Xamarin

Enterprise & Mobile .Net

DE HOGESCHOOL
MET HET NETWERK

Hogeschool PXL – Elfde-Liniestraat 24 – B-3500 Hasselt
www.pxl.be - www.pxl.be/facebook

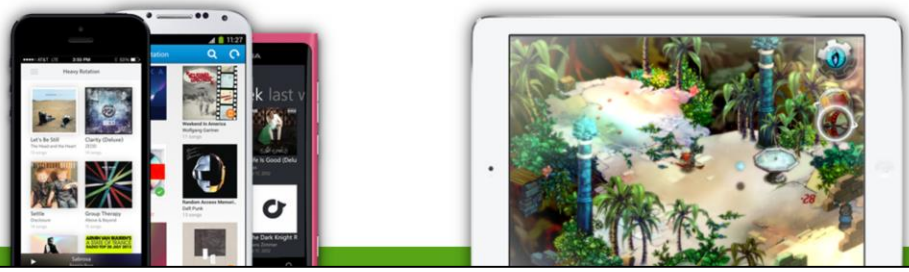
Agenda

- Hello Xamarin!
- C# everywhere
- How Xamarin works
- Xamarin components
  - Visual Studio integration (Windows)
  - Xamarin Studio (Mac)
  - Xamarin Forms and XAML
- Preparing your environment for Xamarin
  - Android
  - iOS

PXL IT

What is Xamarin?

- Xamarin enables developers to reach all major mobile platforms!
  - Native User Interface
  - Native Performance
  - Shared Code Across Platforms
  - C# & .NET Framework



Native User Interfaces

Xamarin apps are built with standard, native user interface controls. Apps not only look the way the end user expects, they behave that way too.

Native API Access

Xamarin apps have access to the full spectrum of functionality exposed by the underlying platform and device, including platform-specific capabilities like iBeacons and Android Fragments.

Native Performance

Xamarin apps leverage platform-specific hardware acceleration, and are compiled for native performance. This can't be achieved with solutions that interpret code at runtime.

What is Xamarin?

- Toolset on top of Visual Studio
  - Output and build to Android, iOS and Windows
  - Native apps
- Commercial product
  - $2000/year for business subscriptions
- Owned by Microsoft (2016)
- The “No-Compromise” approach





PXL IT

Advantages of using Xamarin

- Full control
- Familiar development environment
- Native controls
- Native performance
- Code reuse
- Active component store

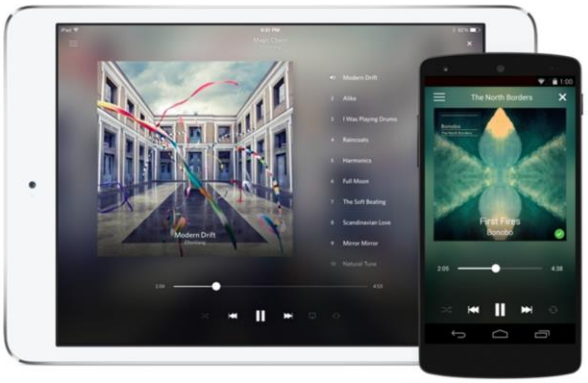


Image courtesy of Xamarin.com

Disadvantages of Xamarin

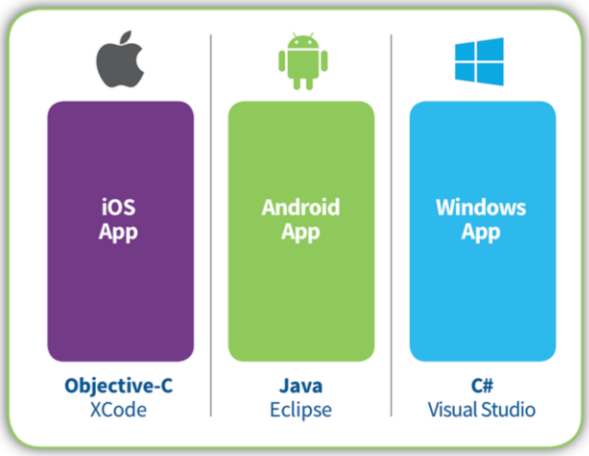
- You need a licence
- It's not a shared UI Platform (yet: Xamarin.Forms)
- You need to understand each platforms UI controls and UX paradigms
- You need a Mac for iOS development

HOW WE GOT TO C# EVERYWHERE…

First, we had the silo'd Approach

Build Apps Multiple Times

- Multiple Teams
- Multiple Code Bases
- Different toolsets



- Multiple Teams
- Multiple Code Bases
- Expensive & Slow
- Positive = Great apps delivered to user's platform
- Negative = Development hampered by multiple code bases & fragmentation

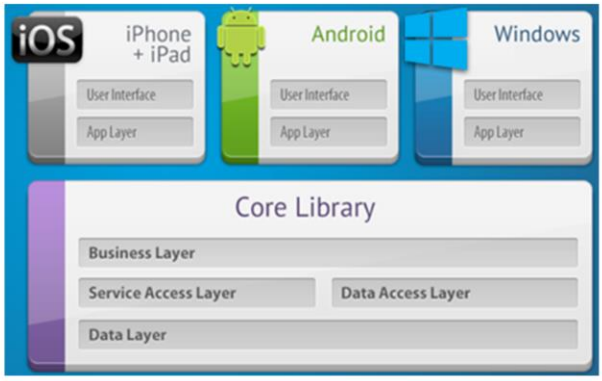
Write Once, Run Anywhere Approach

- Lowest common denominator
- Browser fragmentation
- Developing & designing for 1 platform, happen to get other platforms

- Unhappy Users
- Unhappy Developers
- Increase in Abandoned Apps
- Limited to what is implemented

Xamarin's Unique Approach

- Native User Interface
- Native Performance
- Shared code across platforms
- C# & .NET Framework
- Full API Coverage



UI build natively per platform, leveraging C#

C# + XAML

C# + XML

C# + XIB

One shared app logic code base, iOS, Android, Mac, Windows Phone, Windows Store, Windows

100% API Coverage

Anything you can do in Objective-C or Java can be done in C# and Visual Studio with Xamarin!

- Xamarin offers 100% API coverage on iOS and Android
- If you want to do something specific on a platform it is all available
    - NFC, iBeacon, WiFi, Bluetooth, you name it the API is available
- Xamarin beautiful bindings for every API in Android and iOS

HOW XAMARIN WORKS

Before Xamarin

- Before Xamarin, we had to resort to Java for Android and Objective-C for iOS development
  - New ecosystem when coming from .NET
  - New languages
- Expensive to do cross-platform
  - HTML was/is an option or perhaps Cordova
  - Not always full coverage of the platform

PXL IT

When considering how to build iOS and Android applications, many people think that the indigenous languages, Objective-C and Java, respectively, are the only choice. However, over the past few years, an entire new ecosystem of platforms for building mobile applications has emerged.

Xamarin is unique in this space by offering a single language – C#, class library, and runtime that works across all three mobile platforms of iOS, Android, and Windows Phone (Windows Phone's indigenous language is already C#), while still compiling native (non-interpreted) applications that are performant enough even for demanding games.

What Xamarin brings

- Single language: C#
  - For iOS, Android, UWP (and Windows Phone)
- Full performance
- Full platform feature coverage
- Compilation to native code

PXL IT

Each of these platforms has a different feature set and each varies in its ability to write native applications – that is, applications that compile down to native code and that interop fluently with the underlying Java subsystem. For example, some platforms only allow you to build apps in HTML and JavaScript, whereas some are very low-level and only allow C/C++ code. Some platforms don't even utilize the native control toolkit

How does Xamarin work then?

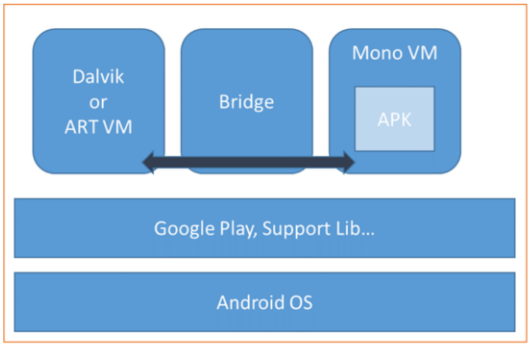
- 2 products: Xamarin.iOS and Xamarin.Android
  - Built on top of Mono: open source version of .NET ECMA standards
    - -Runs on Linux, Mac OS X…
- Build process utilizes Apple and Google tool chain to create 100% native applications
  - Xamarin.Android uses JIT compilation to execute code at runtime
  - Xamarin.iOS uses “Ahead of time” (AOT) compilation to create ARMx binary

PXL IT

On iOS, Xamarin’s Ahead-of-Time (AOT) Compiler compiles Xamarin.iOS applications directly to native ARM assembly code. On Android, Xamarin’s compiler compiles down to Intermediate Language (IL), which is then Just-in-Time(JIT) compiled to native assembly when the application launches.

Android runtime model

- Mono VM + Java VM execute side-by-side (supports both Dalvik and ART)
- Mono VM JITs IL into native code and executes most of your code
- Can utilize native libraries directly as well as .NET BCL



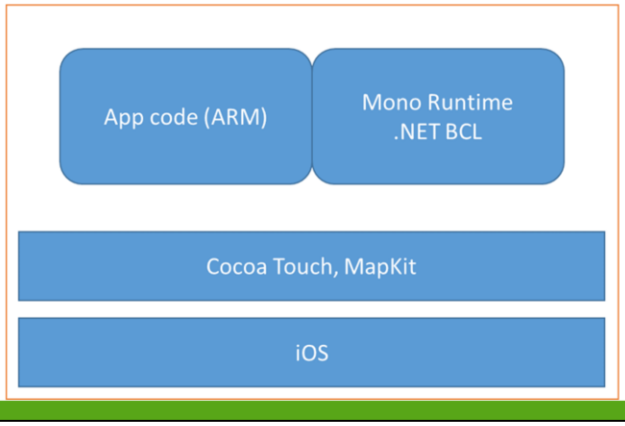
PXL IT

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/Dalvik and interacts with the native types via JNI.

Reference:
https://developer.xamarin.com/guides/android/advanced_topics/limitations/

iOS Runtime model

- Native ARMx code – no JIT is being used here
- Mono runtime provides system services such as Garbage Collection
- Full access to iOS frameworks such as MapKit as well as .NET BCL



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.

Reference:
https://developer.xamarin.com/guides/ios/advanced_topics/limitations/

Windows Phone / UWP

- C# is compiled into IL
- Is actually not covered by Xamarin
- However, taking Xamarin into account when setting up your architecture will help a lot!

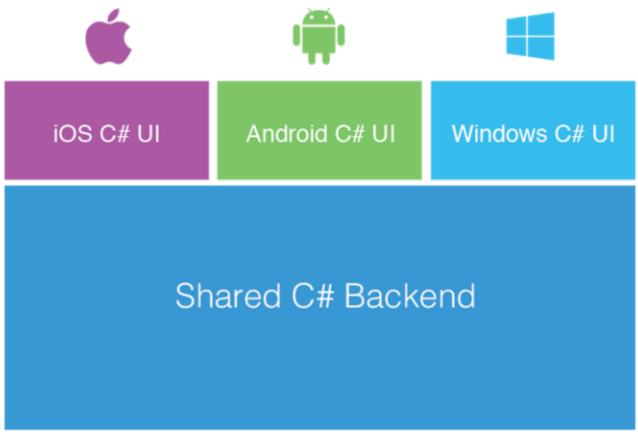
Code sharing options

- Basically, 2 options: Code-sharing with or without UI
  - Regular Xamarin: without (Platform UI)
  - Xamarin Forms: with
- Platform UI:
  - UI is specific to the platform, optimized
  - App logic is C# (Shared projects or PCLs)
  - Average: 60-90% code reuse
- Xamarin Forms:
  - UI abstraction layer
  - Native UI being rendered on iOS, Android and Windows Phone
  - Up to 99% code sharing

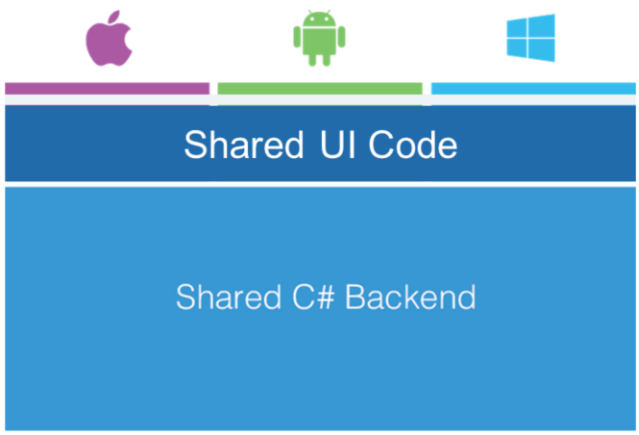
PXL IT

Code sharing options

Traditional Xamarin approach



With Xamarin.Forms:
more code-sharing, native controls



PXL IT

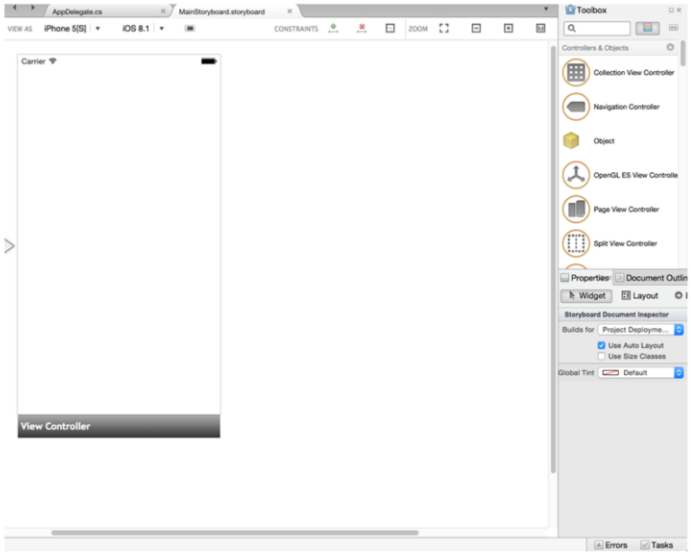
UI creation

- UIs in Xamarin use native controls
  - No distinction possible between UIs built in Objective-C or Java
- Options to create the UI
  - Programmatically: often difficult, no visual indication, in iOS this is sometimes the only way
  - Visual designer:
    - -iOS: designer included for XS and VS, drag and drop support, create .STORYBOARD files
    - -Android: designer included for XS and VS, drag and drop support, create .AXML files
    - -Windows Phone: using Visual Studio, XAML and Blend

PXL IT

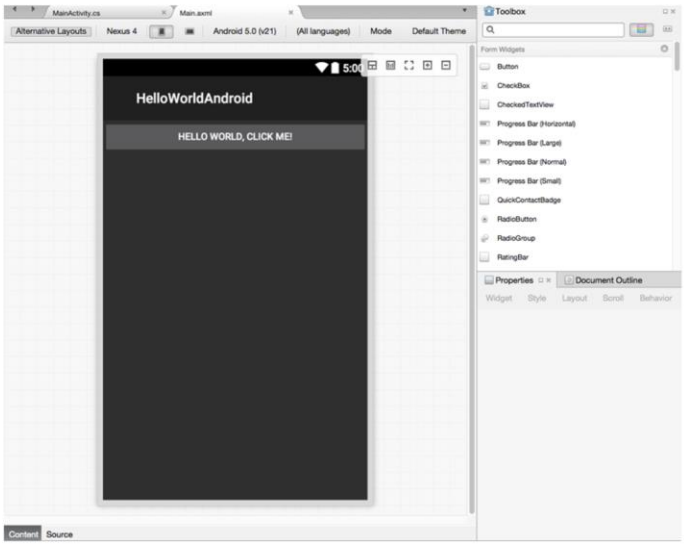
https://developer.xamarin.com/guides/cross-platform/application_fundamentals/building_cross_platform_applications/part_1_-_understanding_the_xamarin_mobile_platform/

Visual designers for iOS, Android and UWP



PXL IT

Visual designers for iOS, Android and UWP



Compilation output

- After compilation, we get:
  - On Android: an .apk file
  - On iOS: an .app file
- No way to identify that an app was built with Xamarin afterwards
  - No way for Apple to remove Xamarin-based apps from the store
  - Packages are identical

Application deployment

- Xamarin apps can be deployed to the regular stores
  - iOS: App Store, Enterprise store
  - Android: Google Play, Amazon Marketplace, .apk
  - UWP: Windows Store

PREPARING FOR XAMARIN DEVELOPMENT

What you need for Xamarin development

- Xamarin license (duh) or trial
- PC or Mac
- For Android development
  - Visual Studio or Xamarin Studio
  - Android SDK and Emulators (installed via Xamarin setup)
  - Visual Studio Android Emulator
  - Optionally, a device
- For iOS development
  - Visual Studio or Xamarin Studio
  - XCode SDK (free)
  - Optionally, a VM (VMWare or Parallels) with Windows 7 or 8
  - Optionally, a device
  - Optionally, a developer account

PXL IT

References:

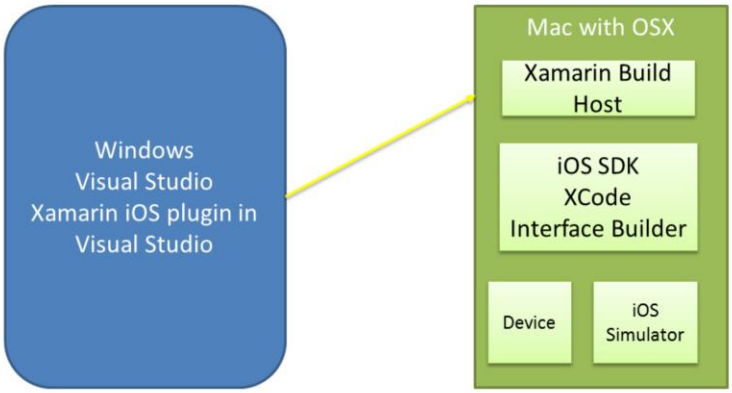
Xamarin download (if you didn't install it with Visual Studio)
https://www.xamarin.com/download

Visual Studio Android Emulators
https://www.visualstudio.com/vs/msft-android-emulator/

Xamarin has a community license which is active when you don't log in.

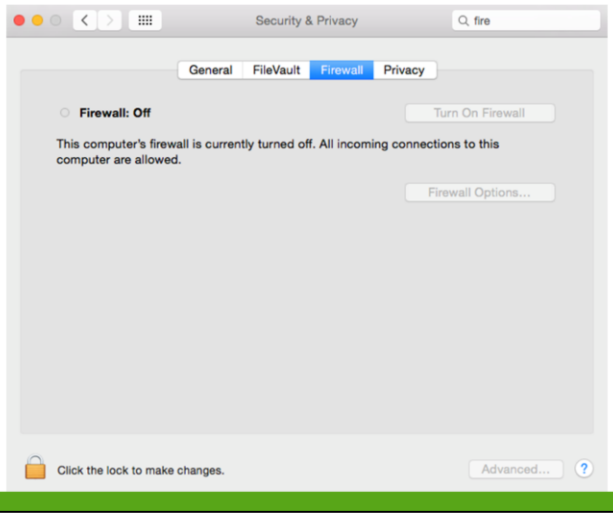
iOS development on Windows

- Required: a Mac with latest OSX!
  - There are no iOS simulators on Windows!



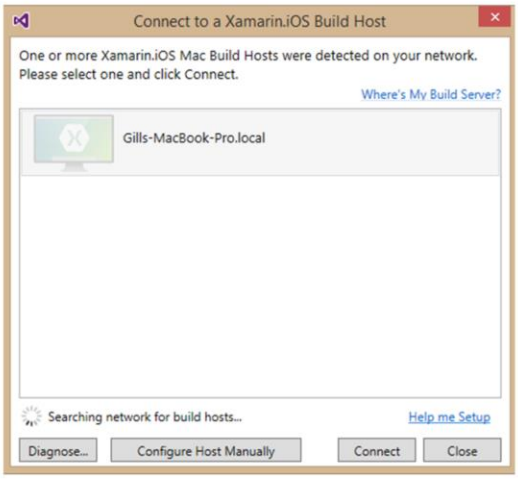
iOS development on Windows

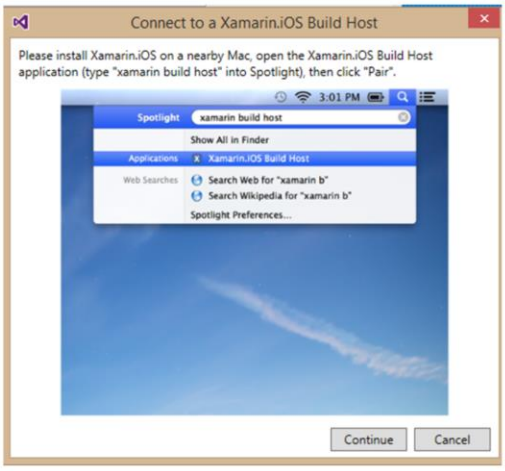
- Firewall on Mac must be disabled



Xamarin Build Host

Connecting Windows with your Mac



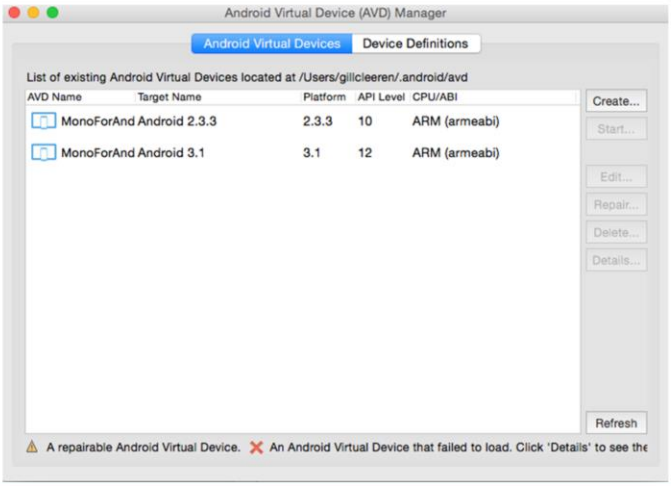


Android development on Windows

- Xamarin Installer installs all the needed tools
- Can be done entirely on Windows

Emulators

- A number of emulators are installed automatically by the setup



PXL IT

Android Emulator configuration.pdf

The default emulators are SLOOOOW!!

- Possible options
  - Genymotion
    -Requires VirtualBox under the hood
  - HAXM drivers
  - Android Player from Xamarin
  - Microsoft Android emulator

PXL IT

Android emulator performance.pdf

GenyMotion - Genymotion provides a virtual Android environment built on top of VirtualBox, a generic x86 hardware virtualization system. A version is available free for personal use. Advanced features are available in the Indie and Business licenses.

Intel HAXM Drivers - The Intel HAXM Drivers provide hardware acceleration for x86 based emulators on Intel VT-enabled sytems. The HAXM drivers are free to use and published by Intel.

Device setup

- 3 steps
  - Enable Debugging on the Device
  - Install USB Drivers (Windows only)
    - Connect the Device to the Computer

Device setup Android.pdf

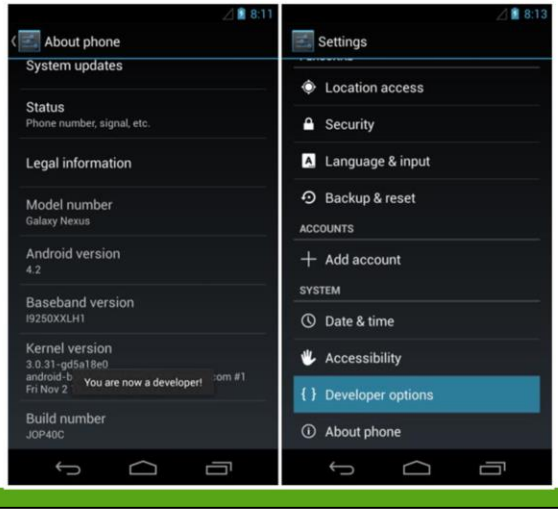
Enable Debugging on the Device - By default, it will not be possible to debug applications on a Android device.

Install USB Drivers - This step is not necessary for OS X computers. Windows computers may require the installation of USB drivers.

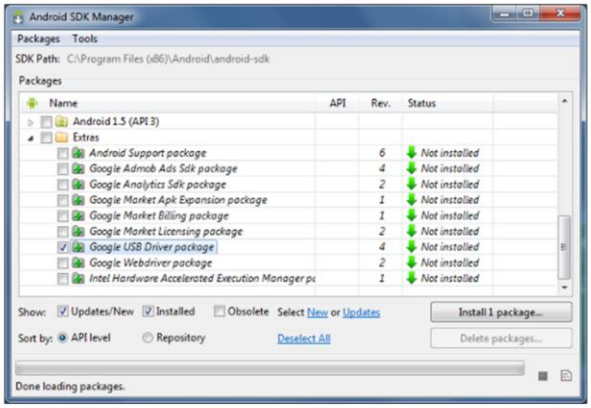
Connect the Device to the Computer - The final step involves connecting the device to the computer by either USB or WiFi.

Enable Debugging

- Tap the Build number 7 times to reveal developer options



Get the USB drivers



Developing for Android

DEMO

Live Demo: Show the finished Phoneword-app → try this out yourself!

More info:

http://developer.xamarin.com/guides/cross-platform/getting_started/introduction_to_mobile_development/

http://developer.xamarin.com/guides/android/getting_started/hello,android/hello,android_quickstart/