Developing PhoneGap Applications

**Workshop 1 – SetUp**

Antoine Hauck   
antoine.hauck@bbv.ch

# Installation der Komponenten

Für eine einfache und effiziente Entwicklung von Cross-Plattform Mobile Apps mittels Xamarin zu ermöglichen sind grundsätzlich folgende Komponenten erforderlich:

* Xamarin Subscription (Xamarin Trial Account oder höher lösen)
  + <https://store.xamarin.com/account/register>
* Xamarin Komponenten mit dem Xamarin Installer
  + [Download the Xamarin Platform](http://xamarin.com/download)
* Xamarin Applikationen können grundsätzlich mit Visual Studio oder Xamarin Studio entwickelt werden. Für die Visual Studio Integration wird mind. Visual Studio 2010 vorausgesetzt und mind. eine Business Lizenz.
* Depending of your desired targets you will need additional software packages or hardware requirements. Here you will find more detailed installation manuals for your target and development platform.
  + Android: [Installing Xamarin.Android](http://developer.xamarin.com/guides/android/getting_started/installation/mac/)
    - Note: Android Emulators are usually very slow when no Hardware Acceleration is installed and enabled ([How-To](https://software.intel.com/en-us/android/articles/installation-instructions-for-intel-hardware-accelerated-execution-manager-windows))
  + iOS: [Installing Xamarin.iOS](http://developer.xamarin.com/guides/ios/getting_started/installation/)
  + Windows Phone: You will need Visual Studio and the [Windows Phone SDK](http://dev.windows.com/en-us/develop/download-phone-sdk)

# [optional] Pre-Exercise Phoneword Translator

Optionally you can make yourself more comfortable with building apps with Xamarin and its associated tools and techniques with a pre-exercise. The guides for this pre-exercise will guide you step-by-step through the different processes how to build an app for Android and iOS. The app translates alphanumeric phone numbers into numeric numbers (1-855-XAMARIN 🡪 1-855-9262746).

The both applications will not share any code between the platforms and therefore you will have no gain from code-reusability between the platforms. It just gives you a basic overview of the concepts and steps involved.

The following guides are available:

* Android: [Introduction to Android Development with Xamarin](http://developer.xamarin.com/guides/android/getting_started/hello,android/)
* iOS: [Introduction to iOS Development with Xamarin](http://developer.xamarin.com/guides/ios/getting_started/hello,_iOS/)

# [optional] Add an unit test to PhoneWord

# Phoneword goes Cross-Platform

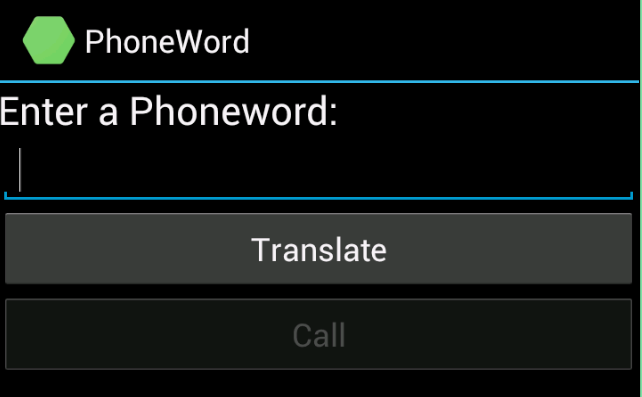
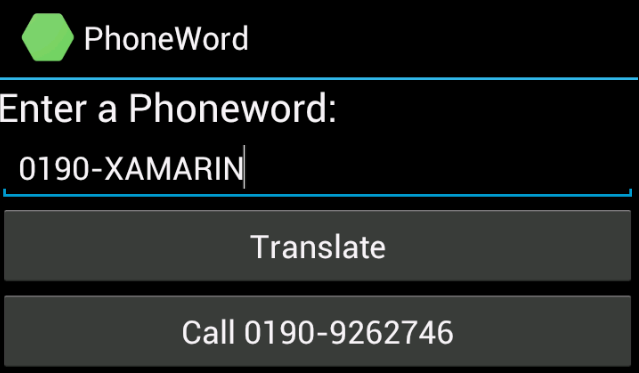
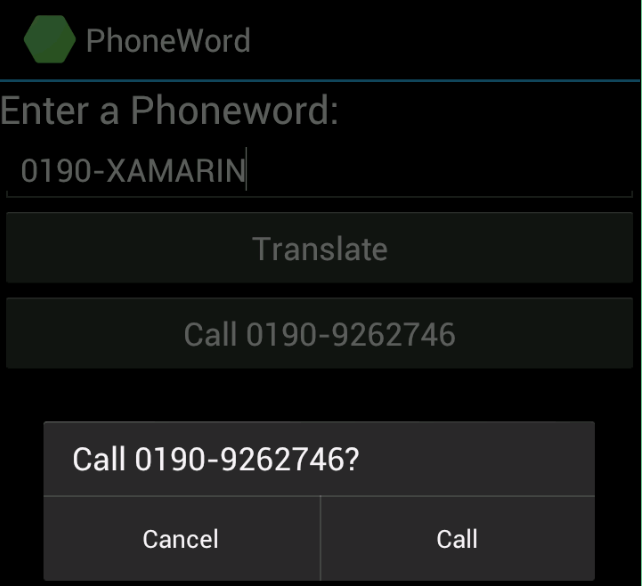
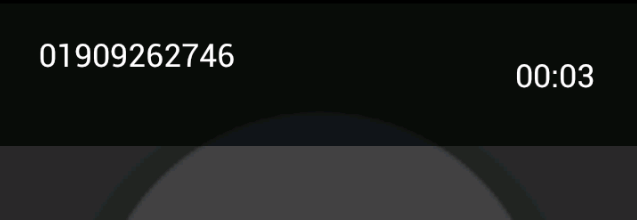
We will extend the Phoneword application from the previous exercise with the following features:

* Make the business logic reusable (as a Portable Class Library) for the different target devices
* Adding Model-View-ViewModel (MVVM) architecture
* Adding a test project for unit tests and acceptance tests
* Add a history page, where we can see all the dialed numbers

As a reminder: The main requirement of the app to translate alphanumeric phone numbers into numeric numbers (1-855-XAMARIN 🡪 1-855-9262746).

The version that we will create will use a [Portable Class Library](http://developer.xamarin.com/guides/cross-platform/application_fundamentals/pcl/introduction_to_portable_class_libraries/) (PCL) to encapsulate and share code across iOS, Android and Windows Phone platforms.

The app should look similar like this:

## Create the project structure

Create a new solution with a portable class library project, which will hold all the core business logic, which we will use across the platforms.

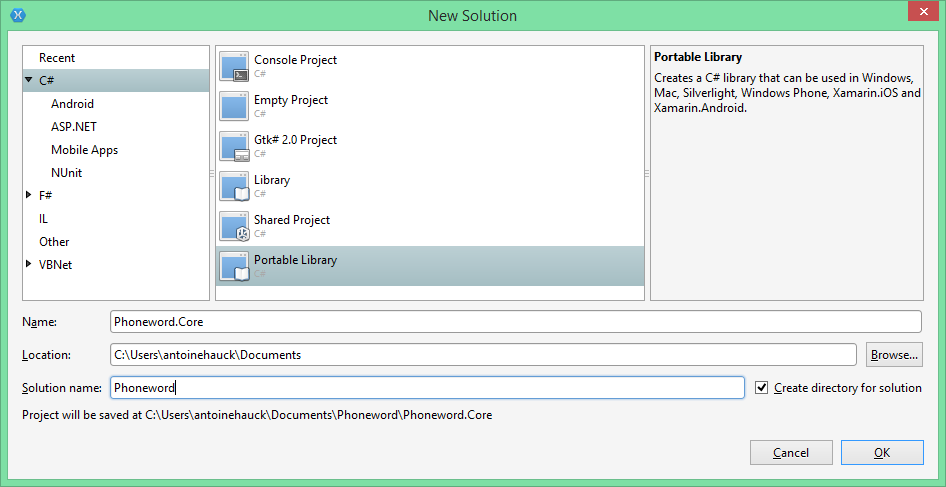
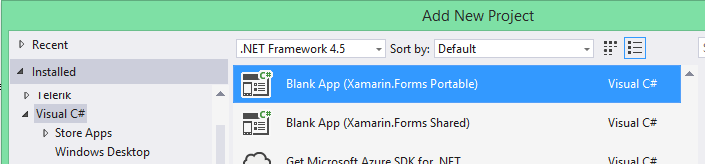


Figure 1: Creating a Solution with a PCL Project in Xamarin Studio



Now create a project for each desired target that you want to support in your application:

* Xamarin.Android for Android
* Xamarin.iOS for iOS (only possible on a Mac or in Visual Studio)
* Windows Phone (only possible in Visual Studio)

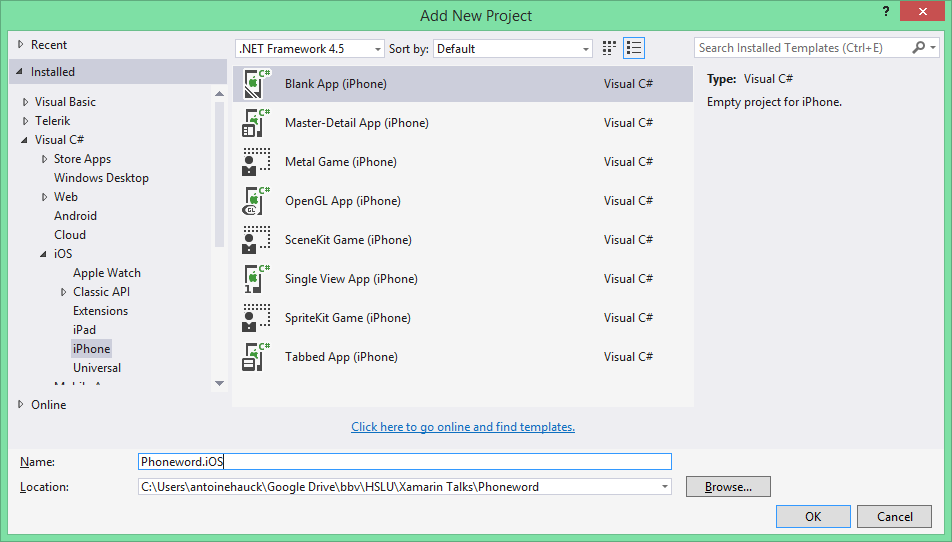


Figure 2: Create a blank iOS Project in Visual Studio

Of course, we don’t want to test manually if the conversion of alphanumeric to numeric numbers work as desired. For this, we will want to make an automated unit test to test the functionality.

For that we will also create a unit test project, where we will put our tests in it.

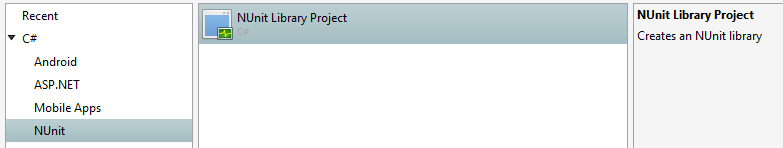
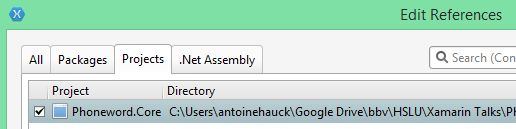


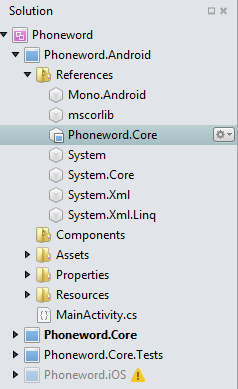
Figure 3: Adding a NUnit test project

Now every target specific project and also the test project should reference to your reusable business logic in the PCL library.

Right click on every target specific project the References tab and select Add Reference -> Projects and choose your PCL project.



Finally the structure should look similar like this:



Now we are ready to begin with the implementation of our app.

## Test Driven Development

Because we are professional developers, we will first write a test before implementing our features.

Our architect provided us with the highly sophisticated interface IPhonewordTranslator, which defines the behavior hot the translation should occur.

/// <summary>

/// Phoneword translator, which provides functions to translate phone numbers.

/// </summary>

public interface IPhonewordTranslator

{

/// <summary>

/// Translate a alphanumeric number to a numeric number.

/// </summary>

/// <returns>The numeric number.</returns>

/// <param name="alphanumericNumber">Alphanumeric number to be translated.</param>

string ToNumericNumber(string alphanumericNumber);

}

Copy the interface definition into your Core project. Also create a first dummy implementation of the interface which always returns an empty string.

### Write your unit test

Now write your first unit test, which will do nothing more than just check if your dummy method returns an empty string.

If you are not familiar in writing unit tests with NUnit, check the following [Quick Start guide](http://www.nunit.org/index.php?p=quickStart&r=2.2.10).

One common pattern in writing unit tests is the Arrange Act Asser (AAA) pattern. A quick explanation can be found [here](https://store.xamarin.com/account/my/subscription?product=Xamarin.Androidhttp://c2.com/cgi/wiki?ArrangeActAssert).

When you have written your test run all the tests in Xamarin Studio with “Ctrl + T” or in Visual Studio.

Our implementation is very unproductive at the moment. Rewrite or add additional tests so that you can test the proper behavior.

When you have a closer look to the specification / user story for this feature you will find out that the conversion should happen as follows:

|  |  |
| --- | --- |
| Letter | Converted Number |
| a, b, c | 2 |
| d, e, f | 3 |
| g, h, i | 4 |
| j, k, l | 5 |
| m, n, o | 6 |
| p, q, r, s | 7 |
| t, u, v | 8 |
| w, x, y, z | 9 |

You should also consider the following cases:

* Lower and uppercase letters should be handed similarly.
* Spaces and dashes should be allowed (e.g. 0190 555-222-1 is valid).
* Other or Special Characters should be skipped.

As a last hint, have a look at the [Row Test feature](http://www.nunit.org/index.php?p=testCase&r=2.5) in NUnit. It might help you running the same test with different input.

Of course your tests will fail. Now you need to implement the desired behavior in your interface implementation. When you implemented the functionality, run your tests again to see that everything works as expected.

## Write the target specific code

As you might expect, when we click the Call button, a call should be made to the translated numeric number. The initiation of a call is platform-specific. This means you can’t really perform a call in your shared business logic, you need to do this at the platform level. Anyway, the shared business logic defines the behavior of the calling functionality:

/// <summary>

/// Abstraction for the dialing functionality.

/// </summary>

public interface IDialer

{

/// <summary>

/// Dials the provided number.

/// </summary>

/// <param name="number">The number you want to perform a call.</param>

bool Dial(string number);

}

Create the interface definition in your shared business logic (Core project) and supply a platform specific implementation for each of your target devices.

The implementation for Android looks like this:

public class PhoneDialer : IDialer

{

public static Activity Activity { get; set; }

public bool Dial(string number)

{

if (Activity == null)

return false;

var intent = new Intent(Intent.ActionCall);

intent.SetData(Android.Net.Uri.Parse("tel:" + number));

if (IsIntentAvailable(Activity, intent))

{

Activity.StartActivity(intent);

return true;

}

return false;

}

public static bool IsIntentAvailable(Context context, Intent intent)

{

var packageManager = context.PackageManager;

var list = packageManager.QueryIntentServices(intent, 0)

.Union(packageManager.QueryIntentActivities(intent, 0));

if (list.Any())

return true;

var mgr = TelephonyManager.FromContext(context);

return mgr.PhoneType != PhoneType.None;

}

}

The implementation for iOS looks like this:

public class PhoneDialer : IDialer

{

public bool Dial(string number)

{

return UIApplication.SharedApplication.OpenUrl(

new NSUrl("tel:" + number));

}

}

|  |  |
| --- | --- |
| warning icon | **Caution**  Für eine hardwarebeschleunigte Ausführung der Emulatoren müssen die erweiterten Prozessorbefehle für die Virtualisierung im BIOS aktiviert sein (Intel VT oder AMD-V) |

|  |  |
| --- | --- |
| warning icon | **Caution**  Es wird dringend empfohlen bei Genymotion unter „Settings -> ADB -> Use custom Android SDK Tools“ auszuwählen und auf das Android SDK Verzeichnis zu verweisen. Es kann ansonsten vorkommen, dass der Emulator nicht erkannt wird und man im Taskmanager den „adb.exe“ Prozess abschiessen muss. |

Nach der Installation muss ein virtuelles Gerät, zum Beispiel ein Samsung Galaxy S4 – 4.4.4. API 19, heruntergeladen werden. Der Genymotion Emulator ist nun bereit.

#### Google Android Emulator

Die Google Android Emulatoren sind grundsätzlich sehr langsam. Dies liegt daran, dass die Emulatoren als ARM Images auf dem Entwicklungsrechner virtualisiert werden. Die Performance kann um ein Vielfaches verbessert werden wenn man stattdessen x86 Images verwendet. Um dies zu ermöglichen muss die Hardwarebeschleunigung für den Hypervisor installiert und aktiviert sein. Die Beschleunigung funktioniert nur wenn alle drei Anforderungen erfüllt sind:

* Intel Prozessor mit erweiterten befehlen für Virtualisierung (Intel VT)
* Intel VT ist im BIOS aktiviert
* [Intel Hardware Accelerated Execution Manager](https://software.intel.com/en-us/android/articles/intel-hardware-accelerated-execution-manager) (HAXM) ist installiert

|  |  |
| --- | --- |
| warning icon | **Caution**  Für eine hardwarebeschleunigte Ausführung der Emulatoren wird momentan nur für Intel VT fähige Prozessoren unterstützt. Genymotion unterstützt auch AMD-V fähige Prozessoren. |

Ob Intel HAXM erfolgreich installiert und aktiviert wurde kann man mit folgendem Befehl verifizieren:

sc query intelhaxm

Als Output sollte folgendes erscheinen:

STATE: 4 RUNNING

Die Android Emulatoren können im AVD Manager erstellt und gestartet werden. Den AVD Manager kann man unter dem Installationsverzeichnis der Android SDK als Datei „AVD Manager.exe“ vorfinden und starten. Eine genaue Anleitung dazu findet man in den [Google Developer Docs](http://developer.android.com/tools/devices/managing-avds.html).

#### ANT Installation

Apache Ant wird benötigt um unsere App auf die Android Emulatoren und Geräte direkt von PhoneGap aus builden zu können.

Die neueste [Apache Ant](http://ant.apache.org/bindownload.cgi) Version als Zip Datei herunterladen und entpacken (z.B. C:\apache-ant). Anschliessend das bin Verzeichnis in die PATH Systemvariable hinzufügen.

PATH: C:\apache-ant\bin;

In der Kommandozeile überprüfen ob Apache Ant aufgelöst wird:

ant –version

Dieser Befehl sollte die Versionsnummer in der Konsole anzeigen.

### Windows Phone

Der Windows Phone Emulator ist womöglich der schnellste Emulator auf einem Windows Rechner. Es werden jedoch viele, grosse Komponenten sowie der Hyper-V Hypervisor benötigt. Man muss genau abklären, ob man den Windows Phone Emulator verwenden möchte.

|  |  |
| --- | --- |
| warning icon | **Caution**  Um Windows Phone Emulatoren zu betreiben muss der Hyper-V Hypervisor installiert werden. Dieser kann **nicht** mit dem Intel HAXM Hypervisor koexistieren. Dier hat zur Folge dass man die Android Emulatoren (Genymotion & Google) nicht mehr hardwarebeschleunigt betreiben kann.  Eine Switchen zwischen den beiden Hypervisors wäre mit [BCDEdit Einträgen](http://www.hanselman.com/blog/SwitchEasilyBetweenVirtualBoxAndHyperVWithABCDEditBootEntryInWindows81.aspx) möglich. |

Detailliertere Informationen zu den Anforderungen und den Installationsschritten findet man unter:

* [Windows Phone 8 SDK tools](http://msdn.microsoft.com/en-us/library/windows/apps/ff402523(v=vs.105).aspx)
* [System requirements for the emulator for Windows Phone 8](http://msdn.microsoft.com/en-us/library/windows/apps/ff626524(v=vs.105).aspx)

### iPhone & iPad

Möchte man direkt auf einem [iPhone & iPad Gerät/Simulator](https://developer.apple.com/library/ios/documentation/IDEs/Conceptual/iOS_Simulator_Guide/Introduction/Introduction.html) deployen und debuggen benötigt man einen Mac. Diese Restriktion von Apple erzwingt das Maschinencode für iPhone/iPad nur von einem Mac Host erstellt werden darf. Die wohl günstigste Variante wäre entweder sich einen Mac Mini anzuschaffen oder eine [Mac Instanz in der Cloud](http://www.macincloud.com/) zu mieten.

Natürlich gibt es viele Alternativen. Mit PhoneGap Build oder Telerik AppBuilder können Apps in der Cloud für iOS Geräte erstellt und deployed (mittels Companion App) werden. Telerik AppBuilder betet auch einen Simulator an, mit dem man direkt auf dem Gerät debuggen kann.

# Verifikation

Um sicherzustellen dass alles einwandfrei funktioniert werden wir folgende Schritte durchführen:

* Initialprojekt erstellen
* Projekt in der IDE öffnen und Hello World implementieren.
* Hello World implementieren
* Auf lokalen Browser deployen & debuggen
* Auf Emulator deployen (Android)
* [optional] auf reales Gerät deployen
  + Android Gerät
  + iPhone (ohne Mac möglich)

## Initialprojekt erstellen

Der Befehl um ein neues Ionic Projekt ist wie folgt strukturiert:

ionic start <ProjectName> <TemplateName>

ionic start erstellt ein neues Verzeichnis ProjectName. Darin wird ein neues PhoneGap Projekt mit integriertem AngularJS und Ionic UI Framework erstellt. TemplateName defniniert mit welchem Template das Projekt erstellt wird. Es stehen drei Templates zur Verfügung: blank, tabs und sidemenu

Wir möchten ein leeres Projekt erstellen. Hierzu öffnet man die Kommandozeile und navigiert zu dem Pfad an dem man das neue Projekt erstellen will (z.B. C:\dev\Projects) und gibt folgendes ein:

ionic start MyApp blank

Am Schluss zeigt uns das CLI einige nützliche Befehle für das weitere Vorgehen. Zudem findet man ein sehr gutes [Video Tutorial](https://www.youtube.com/watch?v=C-UwOWB9Io4&feature=youtu.be), [Getting Started](http://ionicframework.com/getting-started/) sowie [Docs](http://ionicframework.com/docs/) Page wie man die Ionic Toolchain installieren und nutzen kann.

## Projekt in IDE öffnen & Hello World Implementieren

IntelliJ/WebStorm öffnen und unter File -> Open das Projektverzeichnis auswählen (z.B. C:\dev\Projects\MyApp)

Unter dem www Verzeichnis die index.html Datei öffnen und im <ion-content> Element ein „Hello World“ einfügen:

<ion-content>

<h1>Hello World</h1>

</ion-content>

## Auf lokalen Browser ausführen

Nun navigieren wir mit der Kommandozeile in das neu erstellte Projektverzeichnis (z.B. C:\dev\Projects\MyApp) und starten die soeben erstelle Ionic App im lokalen Webbrowser:

ionic serve

Eventuell wird man in der CLI nach der zu verwendenden IP Adresse gefragt. In diesem Falle localhost auswählen. Bei allfälligen Firewall Meldungen den Zugriff erlauben.

Solange man nicht Chrome Canary (oder ein anderer, bevorzugter Browser) als Default-Browser definiert, so wird der Default-Browser von Ionic automatisch geöffnet, welcher im Betriebssystem eingestellt ist. Man kann Ionic instruieren gar keinen Browser zu öffnen.

ionic serve -b

Somit muss man den Link (üblicherweise <http://localhost:8100>) direkt im gewünschten Browser eingeben um das Ergebnis zu sehen.

Ein ganz tolles Feature ist die Livereload Funktion. PhoneGap/Ionic beobachtet alle Dateien im www Verzeichnis des Projektes. Wird eine Datei geändert und anschliessend gespeichert so wird PhoneGap/Ionic die Änderungen sofort mit dem Browser synchronisieren. Somit sieht man die Änderungen direkt und ohne Verzögerung im Browser ohne das Projekt neu kompilieren zu müssen (wie man es üblicherweise in anderen Projekten gewohnt ist).

Wir können direkt die Hello World Nachricht ändern und die Datei speichern. Der Browser sollte nun sofort die neue Nachricht anzeigen.

## Auf Android Emulator ausführen

Um die hybride App auf einem Emulator, realen Gerät oder AppStore/Play Store deployen zu können muss der HTML/JS/CSS Inhalt in eine native Applikation (mittels Apache Cordova/PhoneGap) innerhalb einer WebView gepackt werden. Um dies zu bewerkstelligen müssen wir zuerst dem Projekt die gewünschten Plattformen hinzufügen. Alle unterstützen Plattformen von Apache Cordova können mit nachfolgendem Befehl abgefragt werden:

cordova platform list

Da wir auf einem Android Emulator deployen möchten, so müssen wir Android als Plattform dem Projekt hinzufügen:

ionic platform add android

Wenn wir den Android Emulator von Google verwenden, so geben wird folgendes ein:

ionic emulate android

Verwenden wir hingegen Genymotion, so verwenden wir den run Befehl als würde die App auf ein reales Gerät deployed werden. Dies liegt daran, dass ein Genymotion Emulator als eigenständige virtuelle Maschine mit VirtualBox läuft und somit als ein „echtes“ Gerät betrachtet wird:

ionic run android

## Auf realem Gerät ausführen

Der ultimative und aussagekräftigste Test ist wenn die App auf einem echten Gerät läuft. Nur so kann man sicherstellen, dass keine Komplikationen mit der darunterliegenden Hardware auftreten.

### Android Phone

Bei einem Android Gerät muss das Gerät vom Betriebssystem erkannt werden können, d.h. der Gerätetreiber muss installiert sein. I.d.R. reicht der Google USB Driver aus. Ansonsten Treiber vom Hersteller herunterladen und installieren.

Zudem muss beim Gerät der USB Debugging Modus aktiviert sein. In der Regel zu finden unter System Settings -> Developer options -> USB debugging

Ob das Gerät erkannt wird kann mit diesem Befehl verifiziert werden:

adb devices

Es sollte ein Eintrag erscheinen, welcher als device markiert ist. Achtung: Nicht zu verwechseln mit einem Genymotion Emulator, dieser wird auch als device angezeigt.

Nun kann die App auf dem realen Gerät deployed und ausgeführt werden:

ionic run android

Es kann sein, dass man vom Phone gefragt wird ob man der angeschlossenen Entwicklungsmaschine den USB debugging Zugriff erlauben möchte. Dies mit „Ja“ bestätigen.

|  |  |
| --- | --- |
| warning icon | **Caution**  Der aktive USB debugging Modus stellt ein Sicherheitsrisiko dar. Er sollte nur auf dedizierte Entwicklungsgeräte aktiviert werden. Ansonsten nach der Debugging Session **unbedingt** den Modus wieder deaktivieren. |

### iPhone (ohne Mac)

Es kann auch ohne Mac auf ein echtes iPhone/iPad deployed werden.  
Dazu wird die [PhoneGap Developer](https://itunes.apple.com/app/id843536693) App für iOS benötigt. Wenn man diese installiert so kann man daraus direkt auf den Webserver, indem die eigene App läuft, zugreifen. Hierbei ist es wichtig dass einerseits der entsprechende Port in der Firewall auf dem Entwicklungsrechner freigegeben ist und das iPhone und Entwicklungsrechner sich beide auf der Netzwerkebene (gleiches Subnet oder via Routing) erreichen können.

Im ersten Schritt auf dem Entwicklungsrechner die hybride App starten

ionic serve

Anschliessend die PhoneGap Developer App auf dem iPhone starten und die IP Adresse bzw. Hostnamen und Port des Entwicklungsrechners eingeben und Connect drücken. Ein Beispiel könnte so aussehen:

192.168.20.174:8001

Es gäbe noch viel einfachere Varianten die App auf ein iPhone zu deployen wie zum Beispiel mit Telerik Appbuilder oder PhoneGap Build uvm.

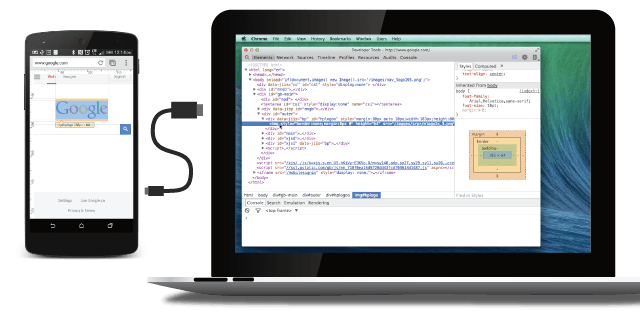
## Auf realem Gerät debuggen

Die App auf einem echten Gerät auszuführen mag schön und gut sein, aber richtig   
interessant wird es erst wenn die App in Echtzeit darauf debugged werden kann.

Es wurden schon einige kostenpflichtige Tools erwähnt, welche dies ermöglichen. Aber es geht auch kostenlos.

Eines davon ist zum Beispiel [GapDebug](https://www.genuitec.com/products/gapdebug/?utm_source=github&utm_medium=forum&utm_campaign=gapdebug). Es ermöglicht das Debuggen auf iOS und Android Geräten. Es hat zudem noch den Vorteil, dass der App Code nicht mit einem Debug Flag modifiziert werden muss.

Um nicht noch weitere Tools installieren zu müssen werden wir die Remote Debugging Funktion in Google Chrome verwenden.



**AWESOMENESS !!!**

Wie man die Remote Debugging Funktion für Google Chrome und das Android Gerät einrichtet und verwendet ist sehr gut in [Remote Debugging on Android with Chrome](https://developer.chrome.com/devtools/docs/remote-debugging) beschrieben.

Es sollte beachtet werden, dass dieses Feature nur einwandfrei funktioniert wenn alle diese Kriterien erfüllt sind:

* Apache Cordova mit Mindestversion 3.3.0
  + Für Version 3.2.0 wird das [WebViewDebug Plugin](https://github.com/jrstarke/webview-debug) benötigt
* Android API Target Level ist mindestens 19 (Version 4.4 KitKat)
  + Nicht nur das Projekt sondern auch Emulator bzw. Gerät müssen diesen Level haben.
* Google Chrome mit Mindestversion 30
* USB Debugging auf dem Gerät ist aktiviert und Berechtigung für die angeschlossene Entwicklungsmaschine erteilt.
* Für Android Screencast (Echtzeitanzeige der WebView des Gerätes direkt im Browser) wird mindestens die Android Version 4.4.3 benötigt.

Die Applikation auf dem angeschlossenen Gerät (via USB) deployen und starten:

ionic run android

Im Google Chrome Browser

about:inspect

eingeben. Sicherstellen das “Discover USB devices“ aktiv ist und bei der entspechenden WebView auf „inspect“ klicken. Schon kann man in Echtzeit debuggen, Breakpoints setzten uvm.