Don't Panic

MOBILE DEVELOPER'S GUIDE TO THE GALAXY

published by:

ENOUGH SOFTWARE

Services and Tools for All Mobile Platforms

Enough Software GmbH + Co. KG
Stavendamm 22
28195 Bremen
Germany
www.enough.de

Please send your feedback,
questions or sponsorship requests to:
mdgg@enough.de
Follow us on Twitter: *@enoughsoftware*

14th Edition February 2014

Art Direction and Design by

Andrej Balaz
(Enough Software)

Editors:

Richard Bloor

Marco Tabor
(Enough Software)

Mobile Developer’s Guide
Contents

Prologue

When we started Enough Software in 2005, almost no one amongst our friends and families understood what we were actually doing. Although mobile phones were everywhere and SMS widely used, apps were still a niche phenomena – heck, even the name ‘apps’ was lacking – we called them MIDlets or “mobile applications” at the time. We kept on architecting, designing and developing apps for our customers – and it has been quite a few interesting years since then: old platforms faded, new platforms were born and a selected few took over the world by storm. Overall: the mobile ecosystem really kicked ass.

With the Mobile Developers Guide to the Galaxy, we began to follow this ecosystem closely. Thanks to our contributing authors, this guide now covers a whopping eight mobile platforms, even after dropping some platforms such as Symbian or webOS along the way. This is the first edition in which we cover Tizen, by the way.

This edition is the biggest thus far, with over 10,000 copies printed on first press – and without our sponsors, this guide would not come to be. Thanks to Paypal – do visit *developers.paypal.com* to join one of their many great offerings! And thanks to SAP, please find out more regarding their (great!) mobile platform offerings on *developers.sap.com*. Of course we are especially happy to welcome Twilio as a first-time sponsor for this edition. Check out *www.twilio.com* to find out how their tools can help you with everything from app distribution, improving security through 2 factor authentication, to implementing VoIP and messaging features into your apps.

The future is exciting, we look forward to your sharing your excitement with us via twitter @enoughsoftware or via email: *mdgg@enough.de*.

We are looking forward to hearing from you!

Robert + Marco / Enough Software

Bremen, February 2014

BY Robert Virkus & Marco Tabor

The Galaxy of Mobile: An Introduction

Welcome to the world of mobile development, a world where former giants stumble and new stars are seemingly born on a regular basis.

The focus of this book is on developing mobile apps, which encompasses a number of phases including: planning and specification, prototyping and design, implementation, internal testing and deployment, deployment to an app store, discovery by users, installation, use and feedback. Ultimately, we want our users to enjoy using our apps and to give us positive ratings to encourage other users to do likewise.

Keep reading to learn how to develop apps for the major platforms. Should this be the first time that you have considered getting involved, we advise against delay. The world is moving rapidly towards mobile becoming the predominant form of computing and others will surely overtake you if you wait too long.

While developing mobile apps shares many common feature with developing other software, it has specific characteristics. We will cover some of these next.

Topology: Form Factors and Use Patterns

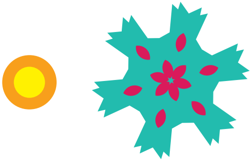
You have to differentiate between smartphones, tablets and feature phones. Each form factor poses its own usability challenges; for instance, a tablet demands different navigation to a phone. TV systems are gaining traction as another form factor for mobile developers.

Use patterns in an Android app, of course, differ from those on iOS, which also differ from those for Windows Phone apps, et cetera.

You should, therefore, refrain from providing an identical experience on all form factors or even all your target smartphone systems. Otherwise, you risk delivering a mediocre service to some sections of your target user base.

Star Formation: Creating a Mobile Service

There are several ways to realize a mobile service:



- App
- Website
- SMS, USSD[1] and STK[2]

App

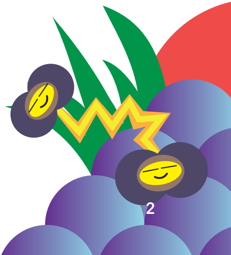
Apps run directly on the device. You can realize them as native, web-based or hybrid apps.

Native Apps

A native app is programmed in a platform specific language with platform specific APIs. It is typically purchased, downloaded and upgraded through the platform specific central app store. Native apps usually offer the best performance, the deepest integration and the best overall user experience compared to other options. However, native development is often also the most complex development option.

[1] en.wikipedia.org/wiki/USSD

[2] en.wikipedia.org/wiki/SIM_Application_Toolkit



Web Apps

A web app is based on HTML5, JavaScript and CSS and does not rely on any app store. It is a locally stored mobile site that tries to emulate the look-and-feel of an app.

A famous example of a web app is the Financial Times app, which left the app store in order to keep all subscriber revenue to themselves for the web world; conversely, the web-based Facebook iOS app was revamped into native app in order to dramatically improve its performance and usability. There are several web app frameworks available to build a native wrapper around such apps so that you can publish them in app stores, such as Phonegap[3].

Hybrid Apps

A hyped controversy circles around whether native or web apps are the future.

For many mobile app developers, this controversy is no longer relevant as a hybrid approach to app development has become quite common: an app can use native code for enhanced performance and integration of the app with the platform, while using a webview together with HTML5-based content for other parts of the app. Parts of the resulting app behave like a native app, while other parts are powered by web technologies. The web-based part can use Internet connectivity to offer up-to-date content. While this could be viewed as a drawback, the use of web technologies enables developers to revise content and features without the need to submit updates to app stores. The key challenge is to combine the unique capabilities of native and web technologies to create a truly user-friendly and attractive app.

[3] www.phonegap.com

Website

A website runs for the most part on your server but you can access various phone features on the device with JavaScript, for example to store data locally or to request the current location of the device. In contrast to apps, mobile websites are inherently cross-platform. However you should not assume that a mobile browser is always based on WebKit, see Microsoft's plea to mobile web developers not to make their websites run on WebKit only[4].

SMS, USSD and STK

Simple services can be realized with SMS, USSD or STK. Everyone knows how SMS (Short Message Service) text messaging works and every phone supports SMS, but you need to convince your users to remember textual commands for more complex services. Some operators offer APIs for messaging services that work for WiFi-only devices, such as the network APIs of Deutsche Telekom[5]. USSD (Unstructured Supplementary Service Data) is a GSM protocol used for pushing simple text based menus, the capabilities depend on the carrier and the device. STK (SIM Application Toolkit) enables the implementation of low-level but interactive apps directly on the SIM card of a phone.

STK may appear irrelevant when so much focus is on smartphone apps, however, for example, m-pesa is an STK app which is transforming life and financial transactions in Kenya and other countries.[6]

[4] blogs.windows.com/windows_phone/b/wpdev/archive/2012/11/15/adapting-your-webkit-optimized-site-for-internet-explorer-10.aspx

[5] www.developergarden.com/apis

[6] memeburn.com/2012/03/how-m-pesa-disrupts-entire-economies/

The Universe of Mobile Operating Systems

The mobile space is much more diverse than other areas in IT. When you are developing software for personal computers, you basically have 3 operating systems to chose from. When it comes to mobile, there are many more. This book provides an introduction to the mobile operating systems that are currently the most relevant, but be aware that the mobile space changes continuously and at a speed that you will seldom observe in other businesses. We have seen many promising technologies appear and quickly disappear, regardless of how big the companies behind them are or the historic market relevance of those companies.

So read on, learn how the market is today and then be prepared to keep it under observation (or make sure you have the latest edition of our guide at hand).

Quasars: Android and iOS

When people talk about mobile apps, they are mainly referring to Android and iOS. Why? When it comes to market share, these two platforms combined dominate the smartphone market with easily 90% in key markets[7] (see the table below for global numbers). The Developer Economics 2014 research[8] also shows that iOS and Android are at the top in terms of developer mindshare – that is, the percentage of developers using each platform, irrespective of which platform they consider to be their 'primary'. Android was at the top, with 71% of developers currently working on the platform, followed by iOS with 55%.

Of course this also means: if you are going to use Android or iOS, you will have lots of competition.

[7] www.idc.com/getdoc.jsp?containerId=prUS24442013

[8] DeveloperEconomics.com

Dark Matter: Feature Phone Platforms

While smartphones generally get the most news coverage, many parts of the world still belong to the feature phone universe. Globally almost 50% of all phones sold in Q2 2013 were feature phones[9], with an install base much higher than that. Biggest vendors are Samsung and Nokia. Nokia claims to have quite a lot of success with their Nokia Store as there are more than 500 developers who have had more than 1 million downloads of their app[10]. Research from 2011 showed that the unhyped platforms actually provided a better chance for developers: Feature phone apps on Nokia's OVI store had 2.5 times higher download numbers compared to apps on Apple App Store[11].

While you can develop native apps for feature phones when you have close relationship with the vendor, you typically develop apps using Java ME or BREW for these phones.

Magnetars: Windows Phone and Windows RT

Windows Phone has now become the 'third ecosystem'[12] in the smartphone universe, it even sells out iPhone in some regions, such as Italy or Latin America. Windows 8.1 and Windows 8 market share has now surpassed the share of all Mac OS X versions combined according to Net Applications[13].

Super Novas: Sailfish OS, Firefox OS, BlackBerry 10 and Aliyun

Will these platforms become spectacular success stories or doomed chapters of the mobile industry? Nobody knows for sure, but there are mixed messages open for interpretation.

[9] gartner.com/newsroom/id/2573415

[10] developer.nokia.com/Distribute/Statistics.xhtml

[11] www.research2guidance.com/apps-on-nokias-ovi-store-had-2-5-times-higher-download-numbers-in-q2-2011-compared-to-apps-on-apple-app-store/

[12] kantarworldpanel.com/global/News/news-articles/Apple-iPhone-5S-outsells-5C-three-to-one-in-Great-Britain

[13] netmarketshare.com/operating-system-market-share.aspx

The Finish company Jolla[14] entered the market in Q4 2013 with its Sailfish OS[15]. The OS received varying reviews, so it will be interesting to see if and how Sailfish OS development is improved throughout 2014.

Firefox OS[16] received a lot of love by previewers and developers alike, which is why we decided to include a dedicated chapter about the platform in this guide. After the launch in several locations globally it has so far failed to gain serious marketshare. Firefox OS will also make it to tablets.

Initial BlackBerry 10 reception varied between skepticism and enthusiasm – but even though all relevant operators carry BlackBerry 10 devices, the adoption has been too slow. As a sale of BlackBerry also seemed to have failed, it will be interesting to see what happens with this new OS.

Aliyun has been released on a single device in China with an unknown market share. It drew publicity mostly from the fact that Google pressured Acer into not releasing an Aliyun device based on Acer's membership of the Open Handset Alliance and the fact that Aliyun's app store featured some pirated Google Android apps[17]. While Aliyun is claimed to be based on Linux, the source code has not yet been released.

White Dwarfs: Symbian and bada

Only shadows of their former selves are Symbian and Samsung bada. While bada was very shortlived, Samsung kept parts of it alive when creating the Tizen platform. Symbian has been pushed into maintenance mode – and left the world with a (photographic) bang in the form of the PureView 808; the importance and market share continue to fall sharply worldwide.

[14] jolla.com

[15] sailfishos.org

[16] mozilla.org/firefox/os

[17] news.cnet.com/8301-1035_3-57513651-94/alibaba-google-just-plain-wrong-about-our-os

Newborn Stars: Tizen and Ubuntu

We will see some interesting new entries in 2014.

Tizen[18] devices have been announced for Q1 2014. Seemingly gently yet continuously pushed forward by Samsung and Intel, Tizen aims to power not only smartphones but also TVs, tablets, netbooks and in-vehicle infotainment systems. The fact that we included a dedicated chapter about Tizen in this edition of this guide reflects the fact that we are taking the platform seriously and see a chance that it will live longer than bada.

Last but not least Canonical presented Ubuntu[19] for mobile devices. The idea is to bring the full power of a PC to the phone. The crowdsourcing[20] effort to fund the Ubuntu Edge phone did not reach its goal, but Ubuntu plans to enter the market anyway.

Solar System: Smartphone OS Market Shares

When you look at the global smartphone market shares, the picture might look simple[21]:

Platform	Market Share Q3 2013	Absolute Year-over-Year Change
Android (Google)	81.3%	+6.3%
iOS (Apple)	13.4%	-2.2%
Windows Phone (Microsoft)	4.1%	+2.0%
BlackBerry	1.0%	-3.3%
Other	0.2%	-2.8%

[18] tizen.org

[19] ubuntu.com/devices/phone

[20] indiegogo.com/projects/ubuntu-edge

[21] blogs.strategyanalytics.com/WSS/post/2013/10/31/Android-Captures-Record-81-Percent-Share-of-Global-Smartphone-Shipments-in-Q3-2013.aspx

You may agree with the majority of developers that decide spending time on platforms other than iOS and Android is a waste of time. Be assured: It is not that simple. While worldwide smartphone shipments exceeded feature phones[22] for the first time in Q1 2013, feature phones still outsell smartphones in many regions.

Note also that Sailfish OS or even Firefox OS market share – while likely small – is not yet known. One of these platforms might still be the best choice for your business case – it may be better to be a small planet on the edge of the galaxy than dicing with the black hole of stellar app numbers at the galactic core.

You also have to remember that these are global figures – the regional market share of each platform is another matter altogether. In a world where localized content is increasing in importance, it is essential to know the details and characteristics of your target market. For example, China is the largest smartphone market today responsible for more than 40% of worldwide Android shipments in Q3 2013[23], but Chinese handsets typically come without the Google Play store or other Google services.

To find out about market share in your target region, check out online resources such as comscore[24], StatCounter[25], VisionMobile[26] or Gartner[27].

[22] idc.com/getdoc.jsp?containerId=prUS24085413

[23] engadget.com/2013/11/14/android-ios-market-share-gartner-q3-2013/

[24] comscoredatamine.com/category/mobile

[25] gs.statcounter.com

[26] visionmobile.com

[27] gartner.com

About Time and Space

As developers, we tend to have a passion for our chosen darlings. However, let us not forget that these technologies are just that – technologies that are relevant at a given time and in a given space, but not more. Yes, flamewars are fun but in retrospect, they are always silly. Hands up those who fought about Atari versus Amiga back in the good ol' 80s! Probably not many of you but, surely, you get the point. Initiatives such as FairPhone[28] or IndiePhone[29] may prove more important than the OS or vendor of your choice in the future.

Lost in Space

If you are lost in the vast space of mobile development, do not worry, stay calm and keep on reading. Go through the options and take the problem that you want to solve, your target audience and your know-how into account. Put a lot of effort into designing the experience of your service, concentrate on the problem at hand and keep it simple. It is better to do one thing well rather than doing 'everything' only so-so. Invest in the design and usability of your solution. Last but not least, finding the right niche is often better than trying to copy something that is already successful. This guide will help you make an informed decision!

[28] fairphone.com

[29] indiephone.eu

BY Anna Alfut

Conceptional Design for Mobile

Stumbling upon an idea is a wonderful Aha! moment. You suddenly know what to do and have the confidence that your idea will solve the problem faced by your potential users. Going from this early stage to the final app implementation is challenging. Not only do you aim to build a stable application, you also want it to be helpful and easy to use.

User Experience can be described as how users perceive your application during and after they have interacted with it. Was it well designed, easy and enjoyable to use? Did it help them achieve a task in an efficient or fun way? Was everything working smoothly? Before you get into design and coding, it is worth spending some time in refining your concept. Below are some guidelines to help you define your idea enough to move into design and build stages.

Capturing the Idea

Write a **concept summary** that describes your app in few sentences (the shorter the better). Try to explain it to several people, outside your team members, to see how well they can understand and relate to it.

Get to know your **audience**. Who are the people that you are designing for and what would be their motivations for using your app? If you can, go and talk to them to get some first-hand information. A useful technique to document your findings in this area is to create personas – generic profiles of your user groups.

Define your **content**. Ask yourself what is the core content of your application? Depending on your application type, it may be photos, user generated feeds, original data (books, metadata, music) etc. Once you recognize the main information to display or the key interaction it is easier to get the right focus in the design stage. For example, if you are creating an e-book reader app you probably want to make sure that the typography is of good quality, and your screen designs provide enough space for text without UI elements getting in the way of the reading experience.

Describe the main **functionality**. What will users do via your interface? You can think about it in terms of verbs and try to list them out: browse, share, buy etc. You will notice that some activities are related. For instance, if your application has a strong community aspect there will be a number of features that you can group (like sharing, commenting, messaging, following). This can be another UI hint for you. It helps users if related functionalities are presented in a similar way.

When designing for mobile experience you need to think about the **context** in which your app will be used. And how it will affect both your interface usability and the users. Do you think you will get users full attention, or will they be jogging at the same time? Is your app a stand alone product? Does it relate or depend on other services? What will happen if there is no internet connection? How will your app's UI handle this situation?

It is worthwhile to spend some time on market **research**. Play with other apps that might be similar to yours. Find out how they are doing: what users think about them. This is a good way of knowing the space you are entering.

After answering so many questions you should have a clearer understanding about the app you want to build. As you go further with your idea development keep asking those questions. It is a good way to keep focused and check if you are getting closer to what you wanted to achieve. Sometimes what you wanted to achieve changes with time as well. The undefined concept is just a hint that once explored can lead you to unexpected findings and new ideas.

Designing User Experience

To capture and refine your app's overall UX you have to think about the user flows, information architecture, interactions, layout structure and visual design. How will it all work together in your product's environment? What are the details of each use case scenario, what issues users might have while using your app, and how feedback will be communicated in a helpful way. And once your product is ready, how will you let people know about it and convince them that it is worth their time?

User Flows

Some apps have very linear flow to achieve a certain task (e.g. a camera app). Some might have more iterative journeys. Describe your "ideal scenario(s)" where user starts at a point A and after a number of steps, ends up in point B. Think of other possible journeys that can deviate from the ideal path. Draw flowcharts or use wireframes to map out various scenarios in detail.

Wireframes

Wireframes are flat, sketchy versions of your interface. Their purpose is to capture functionalities and overall interface concept. A wireframe for a given screen will have different versions/states depending on a scenario. For a network error you will have different instances of the same screen.

Before you delve into detailed layouts, get familiar with UI guidelines for the OS that your app will be developed for.

Each platform is a different environment and you should read guidelines to use the correct conventions. Unless you have a strong case to do otherwise, follow the established practices. Make your research and get familiar with pattern galleries that are available online. Keeping close to the "native" feel gives you instant usability benefits. Users are likely to recognize standard behaviours or visual treatments from using other applications on their devices. You will find platform-specific links in the respective chapters of this book.

Wireframes can be done with pen and paper or you can use one of the many wireframing tools that are out there. Sketching on paper is probably the best way to start as you do not need to spend time learning new software. Drawings are easier to change and scrap. It is also a lot of fun to make them. The advantage of using dedicated applications is the ability to collaborate on your designs and transform your mockups into clickable prototypes.

Prototyping

A prototype is the best way to visualize and evaluate your app's interactions. It does not matter whether you have a big budget or are working on a personal project over the weekends, having a fairly complete prototype version of your app is the best way to communicate your concept and discuss it with others. Prototype is done before you spend time on developing the final code and pixel perfect designs. An agreed clickable walkthrough is a useful reference that teams can work towards without risking going too much off track.

There is no best way of putting a prototype together. You can use whatever technique works for you. From paper prototyping, using one of the specialised tools or other applications that have the functionality to put clickable journeys together (like standard presentation tools). If you have coding skills, building a HTML prototype is also a good way to go. You may be able to use available frameworks and libraries to design a prototype that looks and acts similar to the final product.

You do not need to complete the whole prototype before you start coding. Depending on how you choose to organise your work you can focus on certain parts of your app as you go along and even move towards fast iterations in the actual code. Still, the initial execution ideas are simply faster to validate in the sketchy mockups.

Some available tools are free and most of the commercial ones offer trial version or have free account options for limited number of projects. New applications are becoming available often. Here is a list of few applications to try and choose from:

Application	About	Availability
App in seconds appinseconds.com	Web based, prototyping for iPhone.	commercial
Axure axure.com	Desktop application, wireframing and prototyping.	commercial
Balsamiq Mockups balsamiq.com	Desktop application or plugin to wikis and bug tracking tools, wireframing.	commercial
Fluid UI fluidui.com	Web based, prototyping for iOS, Android, Windows and web projects.	commercial
Mockingbird gomockingbird.com	Web based, wireframing and prototyping.	commercial
OmniGraffle omnigroup.com/products/omnigraffle	Desktop application, diagraming and wireframing.	commercial
Pencil pencil.evolus.vn	Desktop application or Firefox extension; open source GUI prototyping tool.	free

Application	About	Availability
POP popapp.in	iPhone and Android app, making hand drawn designs photos into clickable prototypes.	free
Proto io proto.io	Web based, prototyping.	commercial
Proty protytype.com	Web based, responsive wireframing.	free
UX Pin uxpin.com	Web based, design and wireframing.	commercial

Visual design

Unless you are building an app that uses non-visual input, your app UI will rely on graphics. Taking care of visual design details will improve your app's experience and make it stand-out from amongst the masses.

Spacing and visual hierarchy improves your interface usability. Layout defines details of positioning the elements on the screen and its relation to each other. After users learn your UI it should stay consistent throughout the flow. For example, if your main action button changes color from screen to screen, consider the impact on the users, will they be confused? will they understand the significance of the change?. If the color changes are intentional, make sure you are doing them for good reasons.

Similar to designing layouts and interactions on the wireframes level, certain styling decisions might be informed by a specific platform guidelines. Your app can look very different depending on which platform it was designed for. Make sure that your designs follow the recommended practices for font use, standard icons, layout conventions. Again: see the platform-related chapters of this guide to find more information and links to specific online resources.

It is best if the company branding is interpreted in the UI in a non-obstructive way so users can concentrate on interacting with you app. Use the background, control's colors, maybe certain images or layout choices to add the desired look and feel. Splash screen (if present) is the place where you can display some additional graphics.

Finally, the launching icon is the first-impression visual element that your app will be identified by and judged on. Make it look good. If you are planning on doing releases on multiple platforms check the design requirements early so you can come up with an easily portable artwork.

Designing for multiple screen sizes

With the ever-changing mobile devices market you also have to consider how your UI will look on different screen sizes and displays densities. While it can be too early to get into much details about it before you have your concept refined thinking about the layout scalability-to-usability ratio during the wireframing and visual design stage (so once you have some sort of graphic representation of your layouts) can save a lot of development and testing time later on. If this topic is completely new to you it is worth reading more about best practices in Responsive Web Design (RWD). Web designers have been solving this problems for a while now. And again, it is good to check if the platform specific guidelines provides more information around this topic as well.

User testing

The best way of validating your interface concept is to confront it with real users as soon as possible. You do not need to wait until you have a finished and polished product. In fact testing early can save you a lot of time in the long term as it can expose the ideas that don't work quickly. The more time you invest into developing your designs the harder it gets to let go and start over. It is more difficult to accept feedback on something that you considered almost done than on a clickable prototype that you can update fairly quickly.

Ask few people to do certain tasks using your prototype. If the app you are designing is a music player you can ask them to play a song. If you are unsure of certain functionality you can try to divert the user's attention by asking them to perform reversed tasks, like changing the selected track and picking another one instead. To get the most honest feedback try not to guide users when they are using your prototype.

You can also run testing sessions on other apps that are currently out there. It can surprise you how much others notice about the application that you might have never thought of. Iterate on your designs and apply learnings from user testing as often as you can.

Learn more

There is plenty of resources available online. Here are some to whet your appetite:

Online magazines

- **Konigi:** *konigi.com*
- **Smashing Magazine (UX design section):** *uxdesign.smashingmagazine.com*
- **UX Magazine:** *uxmag.com*
- **UX Matters:** *uxmatters.com*
- **UX Mastery:** *uxmastery.com*

Books

- Susan Weinschenk. **100 Things Every Designer Needs to Know About People**. Research findings on why people react in certain ways when interacting with technology.
- Steve Krug. **Rocket Surgery Made Easy**. A guide how to run usability testing sessions.
- **This is Service Design Thinking.** Characteristics and techniques of service design *thisisservicedesignthinking.com*
- **Mobile Developer's Guide To The 5th Dimension.** The little brother of this guide focusing on UX/ UI design: *wip.org*

Other resources

- **Nielsen Norman Group:** *nngroup.com*
- **Interaction Design Foundation:** *interaction-design.org*

BY Tim Messerschmidt

Android

The Ecosystem

The Android platform is developed by the Open Handset Alliance led by Google and has been publicly available since November 2007. Its use by the majority of hardware manufacturers has made it the fastest growing smartphone operating system. More than 81% of all smartphones sold in Q4 2013 worldwide were based on Android[1]. At their Google I/O event in May 2013, Google announced that over 900 million Android devices have been activated so far[2] which also includes tablets, media players, set-top boxes, desktop phones and car entertainment systems. Google's own smart eyeglasses, Google Glass, runs a minimal version of Android supporting both web and native apps. Some non-Android devices are also able to run Android applications with reduced functionality, such as RIM’s Playbook with its BlackBerry Android runtime, the new Open Source OS Sailfish[3] and the crowdfunded gaming console Ouya.

In January 2014, there were over 1,000,000 apps available in the Android Market[4].

Android is an operating system, a collection of pre-installed applications and an application framework (Dalvik) supported by a comprehensive set of tools. The platform continues to evolve rapidly, with the regular addition of new features every 6 months or so with the newest release being Android 4.4

[1] www.gsmarena.com/android_worldwide_marketshare_crosses_80_for_the_first_time-news-7171.php

[2] gigaom.com/2013/05/15/google-io-statshot-900-million-android-devices-activated/

[3] sailfishos.org

[4] www.appbrain.com/stats/number-of-android-apps

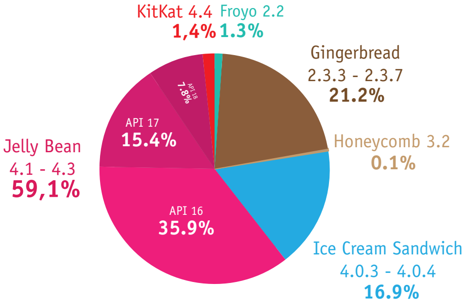
(codename 'KitKat'). As with its last few predecessors, KitKat can be considered as a minor update due to its enhancements, tweaks, and fixes being less visual and more under the hood. Adding unified ways to access the phone's storage system, a new printing framework and hardware sensor batching, Google makes sure that developers can write more efficient and consistent applications. On top of these additions NFC Host Card Emulation has been added, a technology that can be used in context of payments and loyalty programs. Also native support for infrared blasters means you can create apps for the remote control of TVs, set-top boxed and many other electronic devices.

One of the most discussed issues when developing for Android is the system's fragmentation: The multitude of different devices by various manufacturers and the fast progress of the platform itself leads to uncertainty over whether or not your Android application will run everywhere. In addition, only a very small number of phone and tablet models (1.4%) support the latest OS version. However, today, you will reach nearly 100% of the installation base if you decide to target Android 2.2 or above[5].

To encourage a solid user experience and consistent appearance of Android apps, Google publishes a design guide[6]. Going into the importance of color schemes, design patterns and common use patterns, the guide provides a great orientation when building apps for the Android ecosystem.

[5] developer.android.com/resources/dashboard/platform-versions.html

[6] developer.android.com/design



Prerequisites

The main programming language for Android is based on Java. But beware, only a subset of the Java libraries and packages are supported and there are many platform specific APIs that will not work with Android. You can find answers to your "What and Why" questions online in Android's Dev Guide[7] and your "How" questions in the reference documentation[8]. Furthermore, Google introduced a section in their documentation called "Android Training"[9] that helps new developers learn about various best practices. This is where you can learn about basics such as navigation and inter-app communication, as well as more advanced features such as intelligent Bitmap downloads and optimizing your app for better battery life.

To get started, you need the Android SDK[10], which is available for Windows, Mac OS X, and Linux. It contains the tools needed to build, test, debug and analyze apps. The Android

[7] developer.android.com/guide

[8] developer.android.com/reference

[9] developer.android.com/training/index.html

[10] developer.android.com/sdk

Development Tools (ADT)[11] are responsible for the integration with IDEs and making sure that your development flow is as comfortable as possible.

IDE support

Today, Google offers prepacked IDEs based on IntelliJ called "Android Studio", and Eclipse (referred to as "Eclipse + ADT Plugin"), effectively bundling the Android Developer Tools with the IDE. Using these tools saves some time in the setup of the SDKs and offer a more Android-tailored experience: Android Studio comes directly with Gradle-support and enables the display of resources such as strings and colors next to their references. As Android Studio is still an **early beta** version, you might encounter a bug or two – better stick to Eclipse should this be a showstopper for you.

IDE	plugin support	bundled version
Eclipse	seperate ADT package	Eclipse + ADT Plugin
Intellij	seperate Android plugin	Android Studio

More information and the required downloads can be found in the Android documentation's "Tools"[12] section.

Native development

The Android NDK[13] enables native components to be written for your apps by leveraging both JNI for invocations of native methods and using native subclasses that offer callbacks to it's

[11] developer.android.com/tools/sdk/eclipse-adt.html
[12] developer.android.com/tools
[13] developer.android.com/tools/sdk/ndk

non-native pendants. This is important for game developers and anyone who needs to rely on efficient processing.

Implementation

App Architecture

Android apps usually include a mix of Activities, Services, BroadcastReceivers and data providers; these all need to be declared in the application's manifest.

An Activity is a piece of functionality with an attached user interface. A Service is used for tasks that run in the background and, therefore, are not tied directly to a visual representation. A Message Receiver handles messages broadcast by the system, your own or other apps. A Data Provider is an interface to the content of an application that abstracts from the underlying storage mechanisms (e.g. SQLite).

An application may consist of several of these components, for instance an Activity for the UI and a Service for long running tasks. Communication between the components is achieved by Intents or remote procedure calls handled by Android Interface Definition Language (AIDL).

Intents bundle data, such as the user’s location or a URL, with an action. These intents trigger behaviors in the platform and can be used as a messaging system in your app. For instance, the Intent of showing a web page will open the browser. A powerful aspect of this building block philosophy is that any functionality can be replaced by another application, as the Android system always uses the preferred application for a specific Intent. For example, the Intent of sharing a web page triggered by a news reader app can open an email client or a text messaging app depending on the apps installed and the user’s preference: Any app that declares the sharing Intent as their interface may be used.

The user interface of an app is separated from the code in Android-specific XML layout files. Different layouts can be created for different screen sizes, country locales and device features without touching the Java code. To this end, localized strings and images are organized in separate resource folders. Of course, you are also able to define and design layouts in code or make use of both strategies to enable dynamic UI updates.

The SDK and Plug-Ins

To aid development, you have many tools at your disposal in the SDK, the most important ones are:

- **android:** To create a project or manage virtual devices and versions of the SDK.
- **adb:** To query devices, connect and interact with them (and virtual devices) by moving files, installing apps and alike.
- **emulator:** To emulate the defined features of a virtual device. It takes a while to start, so do it once and not for every build.
- **ddms:** To look inside your device or emulator, watch log messages, and control emulator features such as network latency and GPS position. It can also be used to view memory consumption and kill processes. If this tool is running, you can also connect the Eclipse debugger to a process running in the emulator. Beyond that, ddms is the only way (without root-access) to create screenshots in Android versions below 4.0.

These four tools along with many others, including tools to analyze method trace logs, inspect layouts and test apps with random events, can be found in the tools directory of the SDK.

IDE plug-ins are available to help manage all these files.

Version 11.x of IntelliJ includes a visual layout-editor, so you are free to choose between Eclipse and IntelliJ should you want to perform rapid prototyping by dragging UI-elements in the editor.

If you are facing issues, such as exceptions being thrown, be sure to check the ddms log or use the logcat mechanism. It enables you to check whether you neglected to add all necessary permissions, for example, `android.permission.INTERNET` in the uses-permission element[14].

If you are using features introduced after Android 2.3 such as Fragments[15] for large screens, be sure to add the Android Compatibility package from Google. It is available through the SDK and AVD Manager and helps development for Android 3.0+ without causing problems with deployment to Android 1.6[16] through to Android 2.3. Be sure to use the v4 packages in your apps to provide maximum backwards support. There is also a version for Android 2.1 and above called v7 appcompat library that introduces a way to implement the ActionBar pattern as documented online[17].

Developing your application against Android 3.1+, will enable you to make homescreen widgets resizable, and connect via USB to other devices, such as digital cameras, gamepads and many others. Android 4.X releases introduced further interesting features such as expandable notifications, lock-screen widgets, and a camera with face detection. The native computing framework, Renderscript (introduced in 3.1), was heavily changed and no longer provides direct graphic rendering capabilities but may now be used for heavy processing instead.

To provide some backwards compatibility for devices with older Android versions, Google began to use the Google Play

[14] developer.android.com/reference/android/Manifest.permission.html

[15] developer.android.com/guide/topics/fundamentals/fragments.html

[16] android-developers.blogspot.com/2011/03/fragments-for-all.html

[17] developer.android.com/tools/support-library/features.html

Services framework[18] which gets updated via the Play Store and adds libraries such as the latest Google Maps. If you are interested in authenticating users, you might want to have a look at the Google+ Sign capabilities that bring the benefit of real user data to your app. The functionality is managed via OAuth 2.0 tokens that allow use of the Google Account on the user's behalf.

Testing

The first step in testing an app is to run it on the emulator or a device. You can then debug it, if necessary, through the ddms tool.

All versions of the Android OS are built to run on devices without modification, however some hardware manufacturers may have changed pieces of the platform. Therefore, testing on a mix of devices is essential. To get an idea of which devices are most popular, refer to AppBrain's list[19].

To automate testing, the Android SDK comes with some capable and useful testing instrumentation[20] tools. Tests can be written using the standard JUnit format, using the Android mock objects that are contained in the SDK.

The Instrumentation classes can monitor the UI and send system events such as key presses. Your tests can then check the status of your app after these events have occurred. MonkeyRunner[21] is a powerful and extensible test automation tool for testing the entire app. These tests can be run on both virtual and physical devices.

[18] developer.android.com/google/play-services/

[19] www.appbrain.com/stats/top-android-phones

[20] developer.android.com/guide/topics/testing/testing_android.html

[21] developer.android.com/guide/developing/tools/monkeyrunner_concepts.html

In revision 21 of the SDK, Google finally introduced a more efficient UI automation testing framework[22] which allows functional UI testing on Android Jelly Bean and above. The tool itself can be executed from your shell with the command `uiautomatorviewer` and will present you the captured interface including some information about the views presented. Executing the tests is relatively easy: After you have written your test, it is then built via ANT as a JAR-file. This file has to be pushed onto your device and then executed via the command `adb shell uiautomator runtest`.

In October 2013 a new tool called Espresso[23] was released by Google. It provides a very lean API that helps to quickly write procedural tests for your UI.

Open source testing frameworks, such as Robotium[24], can complement your other automated tests. Robotium can even be used to test binary apk files if the app's source is not available. Roboelectric[25] is another great tool which runs the tests directly in your IDE in your standard/desktop JVM.

Your automated tests can be run on continuous integration servers such as Jenkins or Hudson. Roboelectric runs in a standard JVM and does not need an Android run-time environment. Most other automated testing frameworks, including Robotium, are based on Android's Instrumentation framework, and will need to run in the Dalvik JVM. Plugins such as the Android Emulator Plugin[26] enable these tests to be configured and run in Hudson and Jenkins.

[22] android-developers.blogspot.de/2012/11/android-sdk-tools-revision-21.html

[23] googletesting.blogspot.de/2013/10/espresso-for-android-is-here.html

[24] code.google.com/p/robotium

[25] robolectric.org/

[26] wiki.hudson-ci.org/display/HUDSON/Android+Emulator+Plugin

Building

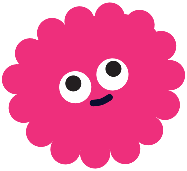
Aside from building your app directly in the IDE of your choice, there are also more comfortable ways to build Android apps. Gradle[27] is now the officially supported build automation tool for Android. There is also a maven plugin[28] which is well supported by the community. Both tools can use dependencies from different Maven repositories, fro example the Maven Central Repository[29].

Google ships libraries for Gradle as Android Archive (.aar) files that can be obtained using the Android SDK Manager. You are also able to package your own libraries or SDKs utilizing the android-library plugin for Gradle. A great source for finding Gradle-friendly Android libraries is "Gradle, please"[30].

Signing

Your apps are always signed by the build process, either with a debug or release signature. You can use a self-signing mechanism, which avoids signing fees (and security).

The same signature must be used for updates to your app – so make sure to not lose the keystore file or the password. Remember: you can use the same key for all your apps or create a new one for every app.



[27] tools.android.com/tech-docs/new-build-system

[28] code.google.com/p/maven-android-plugin/

[29] www.maven.org

[30] gradleplease.appspot.com

Distribution

After you have created the next killer application and tested it, you should place it in Android's appstore called "Play". This is a good place to reach customers and sell them your apps. Android 1.6 upwards also supports in-app purchase through Google Wallet. This enables you to sell extra content, feature sets and alike from within your app by using the Android Play[31] infrastructure. It is also used by other app portals as a source for app metadata. To upload your application to Android Play, go to play.google.com/apps/publish/.

You are required to register with the service using your Google Checkout Account and pay a $25 registration fee. Once your registration is approved, you can upload your app, add screenshots and descriptions, then publish it.

Make sure that you have defined a `versionName`, `versionCode`, an icon and a label in your `AndroidManifest.xml`. Furthermore, the declared features in the manifest (uses-feature nodes) are used to filter apps for different devices.

One of the recent additions to the Google Play Store is alpha and beta testing plus staged rollouts. This allows you to do some friendly user testing before publishing the app to all users. Furthermore, you can target specific countries and devices by setting the right flags in the Developer Console and export detailed statistics that help in understanding your userbase. Using the inbuilt localization service, you can easily add new languages to your app by paying a fee – make sure to check the Localization Checklist[32] for detailed information about the importance of this topic.

As there are lots of competing applications in Android Play,

[31] developer.android.com/guide/google/play/billing/

[32] developer.android.com/distribute/googleplay/publish/localizing.html

you might want to use alternative application stores[33]. They provide different payment methods and may target specific consumer groups. One of those markets is the Amazon Appstore, which comes pre-installed on the Kindle Fire tablet family.

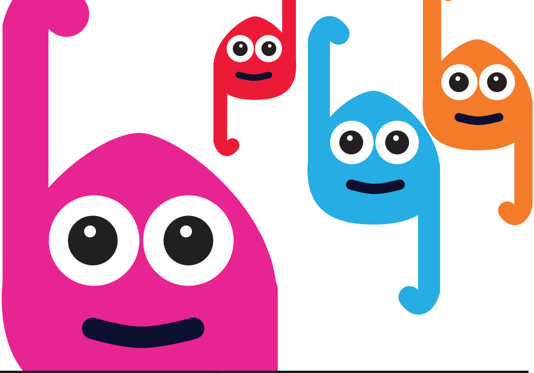
Adaptation

As adaptation of Android increases, vendor specific ecosystem have also been growing that involves their own SDKs, fully-customized Android versions and tools around topics such as alpha and beta testing. This has both upsides, such as a very tight integration that allows an amazing experience for users, and downsides, such as increased fragmentation of ecosystem. Vendor specific marketplaces often prohibit the upload of generic apps that utilize utilities other than their own.

One example is Amazon's Kindle Fire ecosystem which is basically a customized fork of Android and represents the Android tablet with the biggest market share: Instead of using Google's Play Services for enabling in-app purchases or maps, you have to use Amazon's own libraries that offer similar functionality. The reasoning behind it is pretty simple: Kindle devices are not delivered with the required libraries to run Google's services. Amazon also offers their own advertisement and gaming services (comparable to Google Play Games) that help to target your audience. Offering Emulators for their four different devices (1st Gen, 2nd Gen, HD 7" and HD 8.9"), Amazon helps perfect your app by providing a realistic environment. On top of the testing that Amazon offers their developer community, they also review any apps that get uploaded to their Appstore.

Here is a little overview that can help you find the right resources:

[33] onepf.org/appstores/



Vendor	Documentation
Amazon	developer.amazon.com/sdk/fire.html
HTC	htcdev.com
LG	developer.lge.com
Motorola	developer.motorolasolutions.com/ community/android
Samsung	developer.samsung.com/android
Sony	developer.sonymobile.com

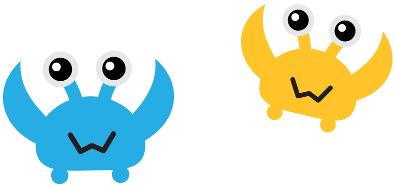
Interestingly enough more and more vendors (e.g. Samsung and HTC) have also started to offer vanilla Android versions of their devices called "Google Play Edition". These devices use the same hardware as the regular models but do not come with any software customization. These devices are directly distributed through Google's Play Store and offer bleeding edge devices to users that want to stick to Google's experience.

Monetization

In addition to selling an app in one of the many app stores available, there are several different ways of monetizing an Android app. One suitable way is by using advertising, which may either be click- or view-based and can provide a steady income. Other than that, there are different In-App Billing possibilities such as Google's own service[34] that utilizes the Google Play Store or PayPal's Mobile SDK[35] and Mobile Payments Library[36]. Most services differ in transaction-based fees and the possibilities they offer for example subscriptions, parallel payments or pre-approved payments. If you're looking to bring extra cool functionality to your app, you should consider implementing card.io's SDK[37] for camera-enabled credit card scanning.

For the vendor specific ecosystems, such as Samsung Apps or Amazon's Appstore, you should consider using their SDKs to enjoy the benefits of optimized monetization.

Be sure to check that the payment method of your choice is in harmony with the terms and conditions of the different markets you want to publish your app to. Those particularly for digital downloads, for which different rules exist, are worth checking out.



[34] developer.android.com/google/play/billing/

[35] github.com/paypal/PayPal-Android-SDK

[36] developer.paypal.com/webapps/developer/docs/classic/mobile/gs_MPL/

[37] card.io

BY Ovidiu Iliescu & Michael Koch

BlackBerry Java Apps

The Ecosystem

The BlackBerry platform is developed by Canadian company Research In Motion (today named BlackBerry Ltd.) and was launched in 1999. BlackBerry devices became extremely popular because they were equipped with a full keyboard for comfortable text input, had a long battery life, included BlackBerry Messenger (their mobile social network offering) and offered a robust push service for email and other data.

However, with the exception of a few markets (Nigeria, Indonesia, South Africa), the overall market share of BlackBerry phones has declined continuously in recent years. In Q3 2013, BlackBerry devices accounted for only 2% of global smartphone shipments[1]. To gain back lost ground, in 2012 RIM introduced a completely new operating system: BlackBerry 10 (BB10). Since 2013, all their new devices are based on BB10; see the dedicated BB10 chapter to learn more about it.

This chapter concentrates on developing apps for the older BlackBerry devices released before 2013. Although it will be phased out in the future, currently the BlackBerry Java API is the most commonly used method to develop apps for older BlackBerry devices. As such, this chapter is dedicated to it.

[1] bgr.com/2013/10/30/blackberry-market-share-q3-2013/

Prerequisites

First, download the Java SDK[2]. Next, you need Eclipse and the BlackBerry plugin[3]. These can be downloaded separately, or as a convenient bundle which also includes the SDK and simulators for the latest BlackBerry OS. Instructions on how to download SDKs for older devices are available on the download page. Extra device simulators are available for download from RIM's website[4].

To deploy your app package on to a device for testing you should download and install the BlackBerry Desktop Manager[5]. For faster deployment, you might also use a tool called javaloader that comes with the JDE.

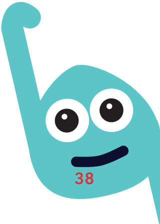
Implementation

The BlackBerry JDE is partly based on Java ME and some of its popular JSR extensions. This means that BlackBerry apps can be created using Java ME technologies alone. Another option is to use BlackBerry's proprietary extensions and UI framework that enable you to make full use of the platform. A complete JavaDoc of both APIs is available online[6].

Native UI components can be styled to an extent, but they inherit their look from the current theme. This can be prevented in code, by overriding the `Field.applyTheme()` method for each component/field.

From OpenGL-ES to homescreen interaction and cryptography, the BlackBerry APIs provide you with everything you need

[2] oracle.com/technetwork/java
[3] us.blackberry.com/developers/javaappdev/javaplugin.jsp
[4] us.blackberry.com/sites/developers/resources/simulators.html
[5] us.blackberry.com/apps-software/desktop/
[6] blackberry.com/developers/docs/7.0.0api/index.html



to create compelling apps. In addition to the official BlackBerry tools, there are third party extensions that enable you to enhance your apps, for example J2ME Polish[7] or Glaze[8] which enable you to design and animate your UI using CSS.

Services

BlackBerry offers many services that can be useful in developing your applications including advertising, mapping, payment and push services[9].

The push service[10] is useful mainly in mail, messaging or news applications. Its main benefit is that the device waits for the server to push updates to it (rather than continuously polling the server), which reduces network traffic, battery usage and costs. BlackBerry offers the push mechanism as a limited free service, with a premium paid extension that enables you to send more push messages.

Porting

Porting apps between BlackBerry devices is easy because the OS is made by a single company that has been careful to minimize fragmentation issues. However, this does not entirely eliminate challenges:

- Some classes (for example, FilePicker) and functionality are only available on specific OS versions.
- You need to handle different screen resolutions and orientation modes (landscape and portrait).
- You need to handle touch and non-touch devices and, depending on the target device, some device-specific

[7] j2mepolish.org

[8] glaze-ui.org

[9] developer.blackberry.com/services/#platform

[10] us.blackberry.com/developers/platform/pushapi.jsp

features as well (such as the Storm's physically clickable touchscreen).

Porting to other Java platforms such as Java ME and Android is complicated as it is not possible to port the BlackBerry UI. In general, cross-platform portability strongly depends on how frequently your app uses native BlackBerry components and APIs. For example it is not possible to reuse BlackBerry push services classes on other platforms.

Testing

BlackBerry provides simulators for various handsets. The Blackberry testing and debugging capabilities are on par with those of other modern platforms such as Android and iOS: the simulators allow developers to simulate a large variety of events (incoming calls, changes to GPS coordinates, changes to network conditions, etc) while on-device debugging makes it easy to test your code on real hardware.

In addition, automated testing is also possible, though somewhat limited and complicated. You can use the bundled FledgeController tool[11] to inject events programatically from your computer, or you can use the EventInjector class[12] to inject events from a BlackBerry application running on the device (or simulator). However, there is very little documentation available on the topic, so expect some hacking and head-scratching to be a part of your BlackBerry automated testing experience.

[11] docs.blackberry.com/en/developers/deliverables/15476/Using_the_BBSmrtphnSmltr_programmatically_607582_11.jsp

[12] blackberry.com/developers/docs/4.1api/net/rim/device/api/system/EventInjector.html

Signing

Many security-critical classes and features of the platform (such as networking or file APIs) require an application to be signed such that the publisher can be identified. To achieve this, you need to obtain a signing key directly from BlackBerry[13]. The signing itself is undertaken using the rapc tool, which also packages the application.

Distribution

BlackBerry's own distribution channel is called AppWorld[14] where you can publish your apps. For paid applications, you get a 70% revenue share. In addition GetJar[15] is a well-known independent website that also publishes BlackBerry apps.

Learn More

If you want to learn more about BlackBerry Java development, the following are a few resources that might help you.

Bundled sample apps

The SDKs come with a great selection of sample apps, showcasing everything from simple "Hello, World!" applications to a complex geo-location and multimedia apps.

[13] https://www.blackberry.com/SignedKeys/codesigning.html
[14] appworld.blackberry.com
[15] getjar.com

Online

- The official BlackBerry documentation microsite[16]
- The BlackBerry developer forums[17]

Books

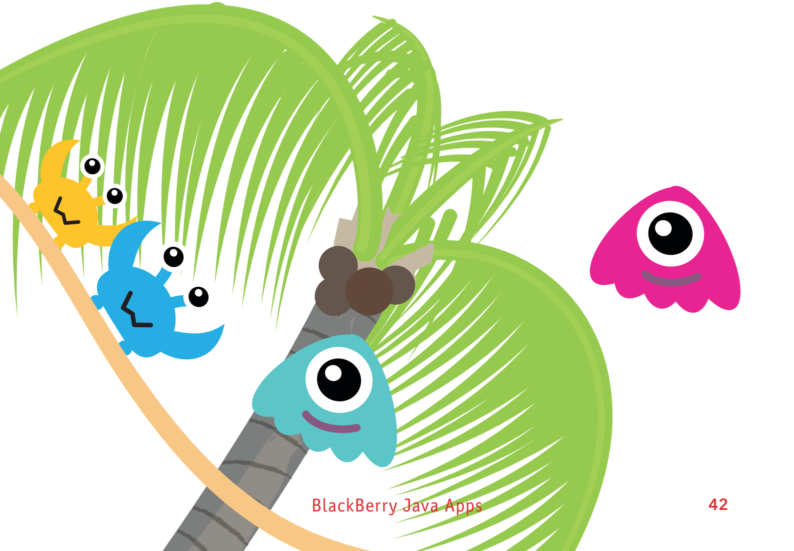
Printed works dealing with BlackBerry Java development include:

- **BlackBerry Development Fundamentals**[18] by John Wargo
- **Beginning BlackBerry 7 Development 2nd Edition** by Anthony Rizk
- **Advanced BlackBerry 6 Development 2nd Edition** by Chris King

[16] developer.blackberry.com/java/documentation/

[17] supportforums.blackberry.com/t5/Java-Development/bd-p/java_dev

[18] bbdevfundamentals.com/



BY Marcus Ross

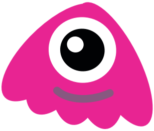
BlackBerry 10

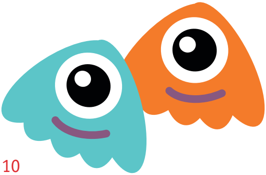
The Ecosystem

The BlackBerry 10 platform (BB10) is a general relaunch from BlackBerry (company former named RIM). BlackBerry has taken this approach in order to catch-up with competing mobile operating systems: iOS, Android and Windows Phone 8. BB10 devices came to market in Q1 2013 – there are no upgrade plans for older generation devices. Currently four BB10 handsets are available: The new flagship is now the Z30. The Z10 and two phones with a physical keyboard, the Q5 and Q10.

As outlined in the previous chapter, BlackBerry is under massive market pressure: In Q1 2013, only 2.9% of all smartphones sold globally were BlackBerry devices- compared to a market share of 6.4% a year earlier. They have to make BB10 a success if they do not want to lose even more ground, so they are investing a lot in this relaunch. This means new and interesting opportunities for app developers who are willing to develop for the new platform. Although the OS is entirely new, in its core it is based on QNX, a realtime OS for embedded devices. The other parts of the BlackBerry ecosystem, like the Marketplace called BlackBerry World or the push-service, have not changed. A big asset for BlackBerry in enterprises are the Mobile-Management-Software called BlackBerry Enterprise Server.

The latest BlackBerry SDK version 10.2 is available since October 2013.





Development

With BB10, apps can be developed using a wide variety of software technologies:

- C Native SDK
- C++ Cascades SDK
- HTML5 (WebWorks SDK)
- Adobe Air
- Android Runtime
- BlackBerry App Generator

To attract developers to their new OS, RIM provides a rich set of resources including a simulator, many sample projects on GitHub[1] and frequently updated documentation[2].

A major point of discontent, for which RIM has received a lot of backlash, is that the current Java API is no longer supported. This means that Java developers writing code for older, non-BB10 BlackBerry devices need to re-orient themselves to one of the technologies previously mentioned. As not all developers are willing to do this, there is concern in the community that too many developers will "jump ship" and re-orient themselves to competing platforms. Furthermore, since there is no migration path for current generation apps, developers will need to rewrite them from scratch for the new platform. This is necessary because the core of the new OS is based on QNX[3], a realtime embedded OS. On the other hand, the new platform offers new opportunities, e.g. for web developers and Android developers who can easily migrate their apps.

[1] github.com/blackberry

[2] developer.blackberry.com/platforms/bb10

[3] www.qnx.com

C Native SDK

The BlackBerry native SDK supports many open standards that allow developers to bring their existing apps to the platform. To get started, there is a Native Dev Site[4]. Writing your code with the native SDK enables your app is as close to the hardware as possible. The BlackBerry 10 native SDK includes everything you need to develop programs that run under the BlackBerry 10 OS: a compiler, linker, libraries, and an extensive Integrated Development Environment (IDE). It is available for Windows, Mac and Linux.

The core development steps are the following:

- Request a signing account and keys
- Set up the native SDK[5]
- Install and configure the simulator[6]
- Configure your environment for development and deployment
- Create your first project
- Run sample applications

As a new addition, BlackBerry added Scoreloop[7] support to the NDK. Scoreloop is a technology that enables mobile social gaming. It lets developers integrate social features into their games, while preserving each game's specific look and feel. Some of the features currently available include:

- User profile
- Leaderboards
- Challenges
- Awards and achievements

[4] developer.blackberry.com/native/beta

[5] developer.blackberry.com/native/download

[6] developer.blackberry.com/native/download

[7] developer.blackberry.com/native/documentation/bb10/com.qnx.doc.scoreloop.lib_ref/topic/overview.htmll

C++ Cascades SDK

Developing with C++ and Cascades is another option. Cascades has been designed to allow developers to build a BlackBerry native application with strong support for easy UI implemenation. The Cascades framework separates application logic from the UI rendering engine. In an application, the declared UI controls, their properties and their behavior are defined in an Markup-Language called QML[8]. When your application runs, the UI rendering engine displays your UI controls and applies any transitions and effects that are specified. The Cascades SDK provides the following features:

- Cascades UI and platform APIs
- Tools to develop your UI in C++, Qt Modeling Language (QML), or both
- Ability to take advantage of core UI controls and to create new controls
- Communication over mobile and Wi-Fi networks
- Recording and playback of media files
- Storage and retrieval of data
- Certificate managing and cryptographic tools

The Cascades framework is built using the Qt application framework. This architecture allows Cascades to leverage the Qt object model, event model, and threading model. The slots and signals mechanism in Qt allows for powerful and flexible inter-object communication. The Cascades framework incorporates features of fundamental Qt classes (such as QtCore, QtNetwork, QtXml, and QtSql, and others) and builds on them. This lets developers define things instead of programming them e.g. they only need to define the duration and type of an animation, instead of programming it. This approach is similar to iOS

[8] en.wikipedia.org/wiki/QML

with Core Animation. QML can even be written by experienced Javascript developers because of its JSON-like markup.

To help developers with this new approach of UI building, there is a tool called Cascades Builder. It is built into the QNX Momentics IDE and lets developers design a UI using a visual interface. When a change to the code is made, you can see the effects immediately in the design view. The developer has no need to program a control, he can simply use a drag and drop approach.

If you are a designer, the Cascades Exporter[9] is for you. This Adobe Photoshop Plugin slices and rescales your images and packages them up to a tmz-File (compressed, sliced and metadata enhanced image assets). These asset files can be easily used by a developer with the QNX Momentics IDE.

To get further information, there is a Cascades Dev Site[10] available.

HTML5 WebWorks

If you are a Web/JavaScript developer, you can use your existing skills to write apps for BlackBerry. There are two important tools that you can use:

The first tool is the WebWorks SDK[11]. Among other features, it allows you to write regular webpages and then package them as native BlackBerry apps with ease. The new version 2.0 of webworks fits tightly in Apache Cordova framework a.k.a Phonegap. BlackBerry published all webworks-apis as plugins for the cross-plattform Cordova tooling. If you want to mimic the BlackBerry-UI style in HTML, there is a project on GitHub to help you. It is called BBUi.js[12] and provides extensive CSS to

[9] developer.blackberry.com/cascades/documentation/design/cascades_exporter

[10] developer.blackberry.com/cascades

[11] developer.blackberry.com/html5/download/sdk

[12] github.com/blackberry/bbUI.js

make your regular webpage look like a native BlackBerry-UI application. You use data-attributes to enhance the HTML for that approach. As an alternative for bbui.js there is also support for jQueryMobile with a BB10 Theme. The SenchaTouch framework[13] also supports BB10.

The second tool is the Ripple Emulator[14]. It is a Chrome Browser extension that acts as a BB10 device simulator for WebWorks apps. It also emulates hardware-specific features, such as the accelerometer and the GPS sensor. You can even use it to package and deploy your app without going through the command-line.

It is good to know that RIM offers hardware accelerated WebGL support and you could do debugging and profiling on the mobile device via WebInspector as a built in feature.

To get more information about developing with WebWorks there is a HTML5 Dev micro-site[15] with more information.

Adobe Air

If you are an existing AIR develeoper you can add BB10 as a new distribution channel. You will use the BlackBerry 10 SDK for Adobe AIR to create applications for BlackBerry devices.

You can use the SDK with Adobe ActionScript and Adobe Flex APIs to create/port BlackBerry Apps. These APIs provide some unique UI components and predefined skins, as well as listeners for events that are specific to BlackBerry devices. Using the Adobe Flash Builder APIs, your application can also access the features that are unique to mobile devices, such as the accelerometer and geolocation information. Additionally,

[13] www.sencha.com/products/touch

[14] developer.blackberry.com/html5/download/ripple

[15] developer.blackberry.com/html5

you can harness the features of the BlackBerry 10 Native SDK by developing AIR Native Extensions (ANE).

To begin developing your Adobe AIR application:

- Download and install VMware Player for Windows or VMware Fusion for Mac
- Download the BlackBerry 10 Simulator
- Download the BlackBerry 10 SDK for Adobe AIR
- Begin development with Adobe Flash Builder, Powerflasher FDT or Command Line Tools

For further information, visit the dedicated website[16].

[16] developer.blackberry.com/air/

Android Runtime

You can use the BlackBerry Runtime for Android apps to run Android Jelly Bean 4.2.2 platform applications on BlackBerry 10.2. To use the runtime, you must first repackage your Android applications in the BAR file format, which is the file format required for an application to run on BlackBerry 10.

As a developer, you will need to use one of the following tools to repackage your application. These tools also check how compatible your application is for running on BlackBerry 10, as some of the APIs from the Android SDK may not be supported, or may be only partially supported on the BlackBerry platform.

- **Plug-in repackaging tool for Eclipse:** The main advantage of using this tool is the ability to check for compatibility, repackage, debug, and run apps on the BlackBerry PlayBook, BlackBerry Tablet Simulator, BlackBerry 10 Dev Alpha Simulator and BlackBerry 10 device, all without leaving Eclipse. You can also use this plug-in to sign your application before it is distributed. If you want to test your application without signing it, you can use the plug-in to create and install a debug token on the target device or simulator.
- **Online packager:** The main advantage of the BlackBerry Packager for Android apps is that you can use it to quickly repackage your Android application using only your browser. You can test the application for compatibility, repackage it as a BlackBerry Tablet OS or BlackBerry 10 compatible BAR file, and then sign it so that it can be distributed through the BlackBerry AppWorld storefront.
- **Command-line repackaging tools:** One of the main advantages of using the BlackBerry SDK for Android apps is that you can use it to repackage multiple Android applications from the APK file format to the BAR file format. In

addition, you can also use this set of command-line tools to check the compatibility of your Android applications, sign applications, create debug tokens, and create a developer certificate.

If you want to find out more about running Android apps on BB10, please visit the dedicated website[17].

BlackBerry App Generator

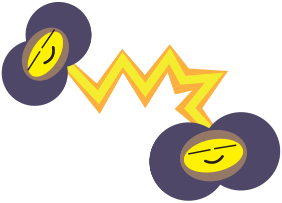
If you are not a developer, BlackBerry provides an easy way of generating a simple app for BB10 with the BlackBerry App Generator[18]. This webpage generates an app based on imput-sources like

- RSS feeds
- Tumbler
- Facebook
- YouTube
- flickr

and more. It generates a master-detail styled app that can be customized with a logo and color selection. For a simple news-app that approach is totally fine, but do not expect any "CNN"-like masterpieces.

[17] developer.blackberry.com/android

[18] blackberryappgenerator.com/blackberry/



Testing

BlackBerry continues to provide a simulator for BB10 handsets as a separate download[19]. This simulator enables you to run an app on a PC/Mac/Linux in the same way it would be run on a real BlackBerry device. To assist with testing, the simulator comes with a little application called controller. This utility enables you to simulate things like setting the battery level, GPS-position, NFC or tilting the device and thereby check how your application reacts in real-world scenarios.

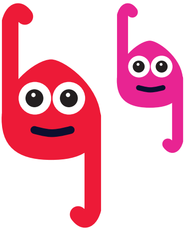
Signing

Many security-critical classes and features of the platform (such as networking or file APIs) require an application to be signed so that the publisher can be identified. This final step in developing an app for BlackBerry is often painful.

If you like to test your unsigned app on a physical device, you need to request a file called debug token. This token enables a specific BB10 device to run unsigned apps. For this setup procedure you need to request a signing file (client-PBDT-xxxxx.csj) via the BlackBerry Key Order Form[20]. After receiving the file by email you can install a debug token with

[19] developer.blackberry.com/devzone/develop/simulator/

[20] www.blackberry.com/SignedKeys/codesigning.html



the command-line tools. After this setup you can run unsigned apps on your device. Please be advised that this needs to be done on each device separately.

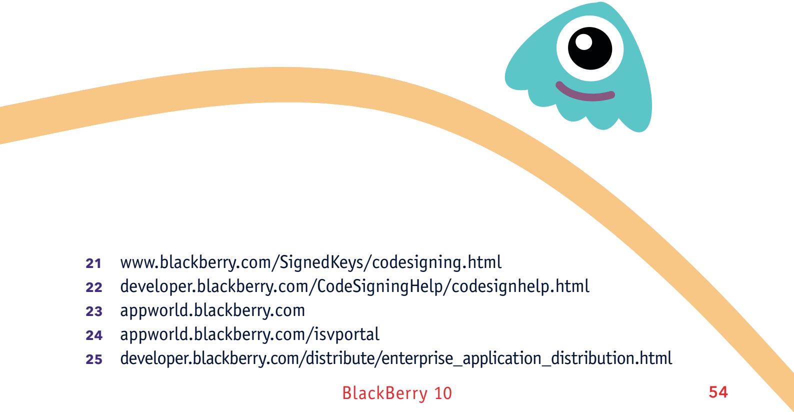
If you want to publish your app in BlackBerry's AppWorld, you need a signing key also ordered through the BlackBerry Key Order Form[21]. To help you with this process of setup BlackBerry provides a step by step webpage[22] that guides you through the process.

Distribution

As with all previous OS versions, BB10 apps are distributed via BlackBerry AppWorld[23]. The necessary vendor account can be created at the Vendor Portal for BlackBerry World[24].

For paid applications, developers get a 70% revenue share.

The second option is an enterprise distribution. This let you roll out an internal app in your organization instead of making it publicly available to any user. This is suitable for B2B Apps. If you want to find out more about enterprise distribution, please visit the dedicated website[25].



[21] www.blackberry.com/SignedKeys/codesigning.html

[22] developer.blackberry.com/CodeSigningHelp/codesignhelp.html

[23] appworld.blackberry.com

[24] appworld.blackberry.com/isvportal

[25] developer.blackberry.com/distribute/enterprise_application_distribution.html

BY Marcus Ross

Firefox OS

The Ecosystem

Do we need another mobile operating system? Mozilla Foundation thinks so and developed Firefox OS[1], a Linux based open source mobile operating system aimed at lower end smartphones. From the beginning, the release cycles for updates has been pretty ambitious: After releasing version 1.0 in February 2013, version followed 1.1 in October and 1.2 in December the same year.

The first Firefox OS device made available for the mass market was the ZTE Open, which sell for 80USD and is promoted for emerging markets. With the release of Alacatel's One Touch Fire device in Germany, Firefox OS officially entered the European market in October 2013. The phone's introductory price was set to 90 Euro.

Firefox apps are HTML-based, but instead of packaging HTML5 web apps with tools such as Phonegap, FirefoxOS uses HTML/JavaScript/CSS as the native development languages. This means it is pretty easy for a web developer to start writing native apps for the system. You need to extend your knowledge to the JavaScript API provided by Firefox OS, and to how apps are packaged.

[1] mozilla.org/firefox/os

Firefox OS basically consists of three main components:

- **Gonk**: The low-level Linux Kernel and hardware abstraction layer (HAL). In theory a hardware vendor just needs to port the Gonk to their hardware to make it Firefox OS compatible.
- **Gecko**: The application runtime. Gecko is parses, executes and render the HTML, JavaScript and CSS. All access to the hardware needed to deliver app functionality is handled by this runtime. It includes a networking stack, graphics stack, layout engine, virtual machine (for JavaScript), and porting layers.
- **Gaia**[2]: The user interface (UI), written in HTML, CSS and JavaScript. Gaia provides all UI elements needed for standard dialogues. It interfaces with the operating system through Open Web APIs.

Development

There are two ways to create an app for Firefox OS: hosted apps and packaged apps. In both cases you write code in HTML, CSS and JavaScript. Hosted apps are basically a website. They are easily updated but offer limited access to the web API and need an established internet connection. Packaged apps run locally and are essentially a zip file containing all the app's assets.

Unlike normal webapps, Firefox OS apps need a manifest[3]. This is metadata for your app which defines the name, description, icons and other information.

This is a how a minimal manifest would look:

[2] github.com/mozilla-b2g/gaia github.com/mozilla-b2g/gaia github.com/mozilla-b2g/gaia

[3] developer.mozilla.org/en-US/docs/Web/Apps/Manifest

```
{
   ″name″:″Hello World″,
   ″description″:″Yet another...″,
   ″launch_path″:″/index.html″,
   ″icons″:{
      ″128″:″icon.png″
   },
   ″developer″:{
      ″name″:″Your name″,
      ″url″: ″http://...″
   },
   ″default_locale″:″en″
}
```

The Firefox WebAPI[4] offers you access to: vibration, geolocation, battery status, alarm, IndexedDB, proximity sensor, ambient light sensor and an archive. Using the APIs you can, for example, access the Battery Status simply by calling `navigator.battery.level` in JavaScript.

If you need more features than the WebAPI provide, you can use Activities. Mozilla uses the Object MozActivity, similar to Android Intents: The user will be asked which app he wants to use for a certain task.

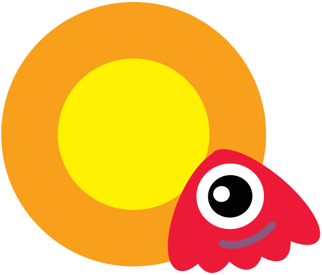
Here is an example of how to create a short message:

```
var sms = new MozActivity({
   name: ″new″,
   data: {
      type: ″websms/sms″,
      number: ″+46777888999″
   }
});
```

[4] wiki.mozilla.org/WebAPI

While is example shows how to access the picture-gallery (picker):

```
var pick = new MozActivity({
   name: ”pick”,
   data: {
      type:
   }

});
```



Simulator and Testing

Mozilla provides a downloadable simulator for Firefox OS as a browser plug-in[5]. Firefox OS 1.2 introduced the App Manager[6]. This new developer tool enables remote debugging of code and provides more GUI helpers, such as the manifest editor. However it is strongly recommended that you don't blindly trust the simulator: for example, it has far more RAM available compared to the actual Firefox devices.

[5] addons.mozilla.org/en-us/firefox/addon/firefox-os-simulator/

[6] developer.mozilla.org/en-US/Firefox_OS/Using_the_App_Manager

Distribution

Mozilla has created an global AppStore called Marketplace[7]. Your app will be reviewed according to Mozilla's guidelines[8]. Once it is published, you will get 70% of the generated revenue.

Learn More

Your first resource to learn more about how to develop your Firefox OS app is the Mozilla Developer network[9]. A quick introduction, including video tutorials on how to get started, can also be found at marketplace.firefox.com/developers/docs/quick_start. André Fiedler also offers some useful insights for beginners in his Sideshare.net presentation Doing mobile web Apps for Firefox OS – the right way[10]. Finally, Mozilla's developer evangelist Chris Heilmann offers a lot of information on his blog[11].

[7] marketplace.firefox.com

[8] developer.mozilla.org/en-US/docs/Web/Apps/Publishing/Marketplace_review_criteria

[9] developer.mozilla.org

[10] slideshare.net/andrefiedler1/doing-mobile-web-apps-for-firefox-os-the-right-way

[11] hacks.mozilla.org/author/cheilmann

BY Alexander Repty

iOS

The Ecosystem

A Short History of iOS

Apple announced iOS (which was then referred to as iPhone OS) at MacWorld 2007 together with the first iPhone, which was released on June 29, 2007. Since then Apple has released a new generation of the iPhone accompanied by a new major release of iOS every year, at some point between June and October. In October 2013, the latest major release (7.0) was delivered, after it was first presented at the Apple Worldwide Developer Conference (WWDC) 2013. A first since the debut of the original iOS, the latest version presents a major overhaul of the user interface and its aesthetics. While for users the changes in iOS 7 mainly mean getting used to different a look, developers face a lot of work getting their apps ready, as apps created for the design patterns of the earlier operating system releases will look very much out of place.

Devices Running iOS

Currently, Apple sells several distinct devices (in various configurations) that run iOS:

- iPhone
- iPod touch
- iPad
- Apple TV

With the exception of Apple TV, all of those devices include the App Store and can run third-party applications.

Most devices will run the most current version of iOS for at least two years from their initial release, so developers should consider this when planning to develop an application. Using older hardware generally means fewer resources, such as CPU cycles and RAM; and in some cases different display sizes and/or screen resolutions.

A detailed list of iOS devices, their capabilities and supported iOS versions can be found on Wikipedia[1].

Device and App Sales

According to information from Apple, which are usually milestones announced at special media events, over 700 million iOS devices should have been sold[2]. According to the graph in the article, over 200 million of those devices were sold in 2013 alone. Since sales of iOS devices are still gaining momentum, even after five years, a large number of those devices can be assumed to be in active use and still running either iOS 6 or iOS 7.

The App Store currently contains over 1,000,000 applications by third-party developers, which have collectively been downloaded over 60 billion times[3], paying out over ten billion dollars to developers according to Apple[4].

Fragmentation

With the introduction of 4.0" devices (the iPhone 5 and the fifth generation iPod touch) in 2012, Apple for the first time changed the form factor of its iPhone and iPod touch models, which can lead to quite a bit of extra effort in developing for both device sizes, especially if a universal app (which is optimized for both iPhone/iPod touch and iPad) is required.

[1] en.wikipedia.org/wiki/List_of_iOS_devices

[2] theverge.com/2013/9/10/4715256/apple-700-million-ios-devices-sold-by-end-of-september

[3] theverge.com/2013/10/22/4866302/apple-announces-1-million-apps-in-the-app-store

[4] digitaltrends.com/mobile/app-store-specs-wwdc-2013/

While updated CPU architectures haven't added a lot of effort to app projects, Apple has introduce a 64-bit ARM chip, the Apple A7[5], which can mean extra effort in supporting both 32-bit and 64-bit architectures, depending on use case. However, for most apps, this should not mean more than a recompile using a newer version of Apple's developer tools.

On the software side, the recent introduction of iOS 7 comes with a completely overhauled user interface and associated aesthetics. Instead of the previously favored rich look with depth and ornamentation, the new operating system used a flat look with vastly reduced details in the iconography. With this change comes a significant amount of work for apps that are targeted at both the new iOS 7 and older versions of the system, such as iOS 6.

Since Apple, as of December 2013, is still selling older models of their hardware that do not use Retina Displays (such as the iPad 2 or the original iPad mini), developers are advised to include graphic assets in two separate, optimized versions.

With updates to the operating system available free to users and supporting devices up to four years old in some cases, users tend to adopt new versions very quickly compared to other mobile platforms. According to Apple's App Store Support Center[6], almost 80% of all devices used iOS 7 in January 2014, just three months after its release. This accelerated adoption means that developers can focus on new versions very quickly and leave support for legacy systems behind.

[5] en.wikipedia.org/wiki/Apple_A7

[6] developer.apple.com/support/appstore/

Technology Overview

Frameworks and Language(s)

Since iOS builds on the foundation of Mac OS X, it uses a lot of the same frameworks and technologies, except for the Cocoa Touch layer (which manages and draws the user interface) and a number of other, small frameworks that are unique to either of the systems. This makes it easy for a number of applications to use a similar code base and just vary on the user interface, which would have to be completely redesigned for touch devices anyway.

Apple continues to bring the platforms closer together by making frameworks available on iOS which were previously only available on OS X and vice versa. One such example is MapKit, which Apple included in iPhone OS 3.0 in 2009 and brought to the Mac in OS X 10.9 (Mavericks) in 2013.

Most Apple supplied frameworks for iOS are written in Objective-C (or supply Objective-C APIs over a different back-end), which is a Smalltalk-inspired lightweight runtime on top of C and retains full C compatibility. Few frameworks supply C APIs, mostly those used for audio and video programming. The system also supports development in C++ and Objective-C++ and includes standard frameworks for all of those languages.

Before the release of iOS, Objective-C lead a somewhat shadowy existence with ratings as low as 0.03% in the TIOBE index[7], thanks to its use almost exclusively for Apple's desktop platform, Mac OS X. In December 2007, it was only the 57th most popular programming language and since has made its way to number three in 2013, just behind Java and C and after winning "Programming Language of the Year" in 2011 and 2012.

[7] tiobe.com/index.php/content/paperinfo/tpci/index.html

Over the past years, Apple has made numerous improvements both to the Objective-C runtime and the LLVM compiler to add new features to the language, such as automatic memory management, blocks (a form of closures) and automatically-synthesized properties, most of which directly benefits developers as they have to write less code.

On their developer website[8], Apple provides a plethora of resources for iOS developers, including software downloads, training videos, getting-started guides, documentation, sample code and forums.

Most of these resources contain very valuable information, such as the Human Interface Guidelines, which every developer should read.

Alternative Programming Languages

Coming from other platforms and languages, learning Objective-C and the iOS SDK APIs might seem very daunting, especially given the language's unusual syntax when compared with other popular programming languages. Therefore, it might be worth considering an alternative to using Apple's tools and languages.

Over the years, a number of products have popped up that add bridges or wrappers in order to support development against the iOS SDK APIs for different languages. One of the most popular of these products is Xamarin.iOS[9], previously known as MonoTouch. The product aims to facilitate porting existing C# code bases to iOS (and Android, using Xamarin.Android), while using native iOS APIs, retaining the highest possible runtime speed. Being a commercial product, Xamarin.iOS is usually updated very quickly to support the latest changes made by Apple, which makes it a viable option for

[8] developer.apple.com/devcenter/ios/

[9] xamarin.com/ios

developers interested in porting their .NET business logic to the platform.

Ruby developers interested in iOS development might take a look at RubyMotion[10], an implementation of the programming language that runs on both OS X and iOS. The product offers compatibility with existing Objective-C libraries (via the CocoaPods[11] package manager or manually) and compilation of source code (for runtime speed and obfuscation purposes). Like Xamarin.iOS, RubyMotion is a commercial product with a steady upgrade pace.

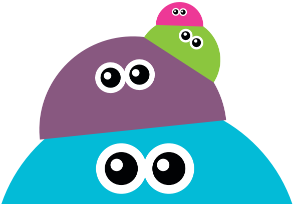
Xcode and Alternatives

For iOS (and Mac OS X) development, Apple supplies its own suite of developer tools, completely free of charge, including the following applications:

- **Xcode**: integrated development environment
- **Instruments**: performance analyzer running on top of DTrace
- **Dashcode**: development environment for Dashboard widgets (Mac OS X) and other HTML-related content
- **iOS Simulator**: simulates an iOS environment for quick testing

A commercial alternative IDE to Xcode is JetBrains' ApCode[12], a Java application with various more in-depth features than Xcode has to offer.

[10] rubymotion.com
[11] cocoapods.org
[12] jetbrains.com/objc



Getting Started With iOS Development

The requirements to get started with iOS development are:

- Intel-based Mac computer running Mac OS X 10.8/10.9
- Free Apple Developer Connection membership
- Xcode 4.x/5.x and iOS SDK (available for free from developer.apple.com)

This setup enables you to write apps and test them in the iOS Simulator, which is included with Xcode and the iOS SDK. This is sufficient to get started and get a feel for iOS software development, but in order to actually deploy apps to iOS devices and to the App Store, a paid membership in the Apple Developer Connection is needed (available from $99 US/€79 per year). A paid membership in the iOS Developer Program also provides access to pre-release software, such as beta versions of upcoming versions of iOS and Xcode.

Generally, it is advisable to do as much testing as possible on as many different devices as possible. See the section Testing and Debugging below for more details.

For developers eager to dive into iOS app development, Apple provides many starting points and guides on its website[13]. The best document to read for newcomers to the platform is Start Developing iOS Apps Today[14], which gives a broad overview of steps and skills required to develop iOS applications and links to various in-depth articles about the entire process, from basic setup tutorials to user interaction design guidelines.

[13] developer.apple.com/library/ios/navigation/#section=Resource%20 Types&topic=Getting%20Started

[14] developer.apple.com/library/ios/referencelibrary/GettingStarted/ RoadMapiOS/chapters/Introduction.html

Diving Deeper Into iOS Development

Once you have an understanding of the basic concepts of the iOS platform, Objective-C and the Cocoa touch framework, it is time to expand your knowledge by learning about other Apple-supplied frameworks and how they can help you build outstanding iOS apps.

A great starting point to expand your knowledge is an all-encompassing book on everything about iOS development. Most experienced iOS developers will suggest one of Aaron Hillegass's books, such as "iOS Programming: The Big Nerd Ranch Guide"[15]. Even though books about iOS development tend to be outdated rather quickly, thanks to an incredibly fast development cycle, this books explains concepts that will likely be true for a while and challenges the reader more than other books usually do.

For those who learn better in a classroom environment, Big Nerd Ranch also offers a number of training classes[16] in Europe and North America.

For even more in-depth literature, it might be worth checking out the current catalogs of publishers such as Pragmatic Programmers, Apress, Sams and O'Reilly. All of these regularly publish good quality books targeted at novice, intermediate and advanced levels and in-depth literature about specific frameworks.

Testing and Debugging

The iOS developer tools include support for unit testing as well as automated user interface testing with the UIAutomation framework. Using Xcode's command line tools, those tools can even be integrated into continuous integration systems for automated acceptance testing.

There are numerous external test automation tools and

[15] bignerdranch.com/book/ios_programming_the_big_nerd_ranch_guide_rd_edition_

[16] bignerdranch.com/training

frameworks as well. Some are proprietary commercial offerings; however the majority are now open source, including some from commercial companies who hope to sell services to make your automated testing easier and more powerful. Many of the external test automation tools require the developer to incorporate a library into a special build of their app. The library allows the tests to interact with your app. Be careful to keep the special builds separate from builds intended for release to the app store, otherwise you may have an unwelcome rejection when submitting your app to the app store.

Xcode includes both gdb and lldb and will automatically use the appropriate one based on which compiler is being used for the application. Although you retain complete control over the debugger using the prompt, Xcode offers a user interface for often-used actions, such as setting, editing and deleting breakpoints and viewing variables and memory contents.

Instruments also contains various features to help developers hunt down bugs, performance bottlenecks, and memory management problems.

As the name implies, the iOS Simulator is just that – a simulator and as such it has different runtime characteristics than actual iOS devices. Thus, a number of issues that will appear on real-world devices simply won't surface when testing apps in the simulator. Fortunately, all supported tests can be executed on devices too, and Apple allows developers to provision up to 100 iOS devices to run their apps for testing and demonstration purposes.

Recruiting and managing third-party software testing (if not available locally in-house) are made easy through TestFlight[17] and HockeyApp[18], both of which offer various helpful features, such as automatic code signing, crash report collection and in-app updating for beta testers.

[17] testflightapp.com/

[18] hockeyapp.net

Learn More

Online

Over the past few years, Dave Verwer's "iOS Dev Weekly"[19] mailing list has evolved to become probably the most important resource for any serious iOS developer. Dave compiles the most interesting news items of the week and sends them out each Friday.

Quite recently, a number of iOS developers, led by Chris Eidhof, Daniel Eggert and Florian Kugler have banded together to make objc.io[20], an online magazine for iOS developers which is published monthly for free on their website, or for a small subscription fee as a Newsstand app on the App Store. The magazine explores a different topic each month with a handful of articles covering technologies around that topic.

Over the past year, Mattt Thompson has built up an excellent repository of articles on relatively unknown or even overlooked APIs in Objective-C and Cocoa at "NSHipster"[21]. He continues to publish interesting articles on a weekly basis and recently released the first year of articles in the form of a book.

Ole Begemann regularly writes a blog in which he shares his insight on in-depth iOS development topics that usually cannot be found anywhere else. You can subscribe to his RSS feed at oleb.net/blog.

Probably the biggest collection of high-quality iOS development tutorials comes to you courtesy of Ray Wenderlich[22] and his team. They cover a wide width of interests and topics, from beginner-level tutorials to full-blown 3D game development.

Many other developers regularly post valuable information about current developments on their blogs. One notable

[19] iosdevweekly.com

[20] www.objc.io

[21] nshipster.com

[22] raywenderlich.com

blog is Mike Ash's[23], on which he posts a very interesting series of Q&As about Objective-C and Cocoa development. A number of other great other blogs can be found through *www.planetcocoa.org*.

Events

Owing to the growing popularity of iOS, there are numerous iOS-centered conferences around the world every year, way too many to list here. There are two notable conferences though that deserve a mention:

- Every year in June, Apple holds their Worldwide Developer Conference (WWDC)[24]. The full-week conference in San Francisco includes many simultaneous tracks about Mac OS X and iOS development with sessions by Apple engineers as well as on-hands labs, where attendees can ask Apple engineers for advice about problems they are facing while developing their apps.
- The biggest and most successful European conference around Mac OS X and iOS development is NSConference[25], held every year around March in England.

Both of these conferences usually sell out in a matter of days, if not hours – so plan well ahead and subscribe to alerts about the tickets going on sale if you are planning to go to either of those conferences.

If you are looking to connect with like-minded developers, you can probably find a CocoaHeads[26] meeting fairly close to you. Most CocoaHeads chapters are an informal group of Mac and iOS developers getting together, sharing their experiences and helping each other out.

[23] mikeash.com/pyblog
[24] developer.apple.com/wwdc
[25] ideveloper.tv/nsconference/
[26] cocoaheads.org

BY Ovidiu Iliescu

Java ME (J2ME)

The Ecosystem

J2ME (officially "Java ME") is the oldest mobile application platform still widely used. Developed by Sun Microsystems, which has since been bought by Oracle, J2ME is designed to run primarily on feature phones. Most feature phones on the market today support J2ME.

Due to its age and primary market segment, J2ME does not fare well compared to modern smartphone platforms in terms of APIs, hardware power and income generation. As a consequence, J2ME's popularity has declined significantly in recent years.

So why would you want to develop for J2ME? Mainly for one reason: market reach. Although global smartphone sales exceeded feature phone sales for the first time in Q2 2013[1], this still means that most of the mobile phones in use are feature phones. And feature phones usually support Java ME. So if your business model relies on access to as many potential customers as possible, then J2ME might still be a great choice – especially when you are targeting markets like certain African countries or India.

However, if your business model relies on direct application sales, or if your application needs to make use of state-of-the-art features and hardware, smartphone platforms are the better choice.

[1] gartner.com/newsroom/id/2573415

Prerequisites

To develop a Java ME application, you will need:

- The Java SDK[2] and an IDE such as Eclipse Pulsar for Mobile Developers[3], NetBeans[4] with its Java ME plug-in or IntelliJ[5]. Beginners often chose NetBeans.
- An emulator, such as the Wireless Toolkit[6], the Micro Emulator[7] or a vendor specific SDK or emulator.
- Depending on your setup you may need an obfuscator like ProGuard[8]. For professional development, consider using a build tool such as Maven[9] or Ant[10].
- You may want to check out J2ME Polish[11], the open source framework for building your application for various devices.

Complete installation and setup instructions are beyond the scope of this guide, please refer to the respective tools' documentation.

Also download and read the JavaDocs for the most important technologies and APIs: you can download most Java-Docs from www.jcp.org. For manufacturer-specific APIs, documentation is usually available on the vendor's website (for example, the Nokia UI API[12]).

[2] oracle.com/technetwork/java/javame/downloads

[3] eclipse.org

[4] netbeans.org

[5] jetbrains.com

[6] oracle.com/technetwork/java/download-135801.html

[7] microemu.org

[8] proguard.sourceforge.net

[9] maven.apache.org

[10] ant.apache.org

[11] j2mepolish.org

[12] www.developer.nokia.com/Community/Wiki/Nokia_UI_API

Implementation

The Java ME platform comprises the Connected Limited Device Configuration (CLDC)[13] and the Mobile Internet Device Profile (MIDP)[14]. As both CLDC and MIDP were designed a decade ago, the default set of capabilities they provide is rudimentary by today's standards.

Manufacturers can supplement these rudimentary capabilities by implementing various optional Java Specification Requests (JSRs), for example user data and file system access (JSR 75) or GPS support (JSR 179). For a comprehensive list of JSRs related to Java ME development, visit the Java Community Process' List by JCP Technology[15].

It is very important to know that the JSRs you want to use may not be available for all devices; so capabilities available on one device might not be available on another device.

The Runtime Environment

J2ME applications are called MIDlets. A MIDlet's lifecycle is quite simple: it can only be started, paused and destroyed. On most devices, a MIDlet is automatically paused when minimized; it cannot run in the background. MIDlets also run in isolation from one another and are very limited in their interaction with the underlying operating system – these capabilities are provided strictly through optional JSRs (for example, JSR 75) and vendor-specific APIs.

[13] java.sun.com/products/cldc/overview.html

[14] java.sun.com/products/midp/overview.html

[15] jcp.org/en/jsr/tech?listBy=1&listByType=platform

Creating UIs

You can create the UI of your app in several ways:

1. Highlevel LCDUI components: you use standard UI components, such as Form and List
2. Lowlevel LCDUI: you manually control every pixel of your UI using low-level graphics functions
3. SVG: you draw the UI in scalable vector graphics then use the APIs of JSR 226[16] or JSR 287[17].

In addition, some manufacturers provide additional UI extensions. For example, Nokia's latest feature phone series (Nokia Asha) employ either the Full Touch[18] or the Touch and Type[19] user interface paradigms, depending on device model. Samsung provide pinch zoom features in their latest Java ME APIs[20].

There are also tools that can help you with the UI development:

1. **J2ME Polish**[21]: This tool separates the design in CSS and you can use HTML for the user interface. It is backward compatible with the highlevel LCDUI framework
2. **LWUIT**[22]: A Swing inspired UI framework
3. **Mewt**[23]: Uses XML to define the UI

[16] www.jcp.org/en/jsr/detail?id=226

[17] jcp.org/en/jsr/detail?id=287

[18] www.developer.nokia.com/Resources/Library/Full_Touch/

[19] www.developer.nokia.com/Community/Wiki/Nokia_UI_API_1.1b

[20] developer.samsung.com/java/technical-docs/Multi-Touch-in-Samsung-Devices

[21] j2mepolish.org

[22] lwuit.java.net/

[23] mewt.sourceforge.net

Screen resolutions for Java ME range from 176x208/220 to 360x640, with the most popular being 240x320. Handling so many different resolutions can be a challenge, but using the above tools you can create UI layouts that scale automatically. Creating custom UIs for each resolution is possible, but not recommended: it is time consuming, error prone and expensive.

Graphical assets should always be optimized. A great free tool for this is PNGGauntlet[24].

Testing

Because of device fragmentation, testing applications is vital. Test as early and as often as you can on a mix of devices. Some emulators are quite good, but there are some things that have to be tested on devices. Vendors like Nokia[25] and Samsung[26] provide subsidized or even free remote access to selected devices.

Automated Testing

There are various unit testing frameworks available for Java ME, including J2MEUnit[27], MoMEUnit[28] and CLDC Unit[29]; Advanced tools like JInjector[30] provide code-coverage and UI testing support.

[24] pnggauntlet.com
[25] forum.nokia.com/rda
[26] developer.samsung.com
[27] j2meunit.sourceforge.net
[28] momeunit.sourceforge.net
[29] snapshot.pyx4me.com/pyx4me-cldcunit
[30] code.google.com/p/jinjector

Porting

At its core, Java ME is a set of standards and specifications, which vendors sometimes interpret differently. This results in all kinds of bugs and non-standard behavior. In the following sections we outline different strategies for porting your applications to different Java ME handsets and platforms.

Lowest Common Denominator

You can prevent many porting issues by limiting the functionality of your application to the lowest common denominator. This usually means CLDC 1.0 and MIDP 1.0, or CLDC 1.1 and MIDP 2.0 if you only plan to release your application in more developed countries / regions.

While this approach is good for simple applications, comprehensive and feature-rich applications will be limited by it. In this case, you might want to consider using Java Technology for the Wireless Industry (JTWI, JSR 185) or the Mobile Service Architecture (MSA, JSR 248) as your baseline, but be aware that these have more limited support in the market.

Porting Frameworks

Porting frameworks help you deal with fragmentation by automatically adapting your application to different devices and platforms. To achieve this, they provide specialized run-time client libraries and built-time tools (such as cross compilers) that work together to make the process almost seamless.

Good porting frameworks enable you to use platform and device specific code in your projects. In other words: a good framework does not hide device fragmentation, but makes it more manageable.

For Java ME one of your options is J2ME Polish from Enough

Software[31] (available under both the GPL Open Source license and a commercial license). Porting from C++ to Java ME is also possible with the open source MoSync SDK[32].

For more information about cross-platform development and the available toolsets, please see the “Going Cross-Platform” chapter.

Signing

The Java standard for mobile devices differentiates between signed and unsigned applications. Some handset functionality is available to trusted applications only.

Applications signed by the manufacturer or carrier of a device enjoy the highest security level and can access every Java API available on the handset.

Applications signed by JavaVerified[33], Verisign[34] or Thawte[35] are on a lower security level, while unsigned applications are on the lowest security level.

Which features are affected and what happens if the application is not signed is largely dependent on the implementation. Furthermore, not every phone carries all the necessary root certificates. The result is something of a mess, so consider signing your application only when required. In some cases an app store may offer to undertake the signing for you, as Nokia Store does.

Another option is to consider using a testing and certification service provider and leaving the complexity to them. Intertek[36] is probably the largest such supplier.

[31] enough.de

[32] mosync.com

[33] javaverified.com

[34] verisign.com

[35] thawte.com

[36] intertek.com/wireless-mobile

Distribution

App stores are probably the most efficient way to distribute your apps. Some of the most effective stores include:

- Handmark[37] and Mobile Rated[38] provide carrier and vendor independent application stores.
- GetJar[39] is one of the oldest distributors for free mobile applications – not only Java applications.
- Nokia Store[40] targets Nokia users worldwide and provides a revenue share to the developer at 70% from credit card billing and 60% from operator billing
- Carriers are in the game also, such as Orange[41] and O2[42].

An overview of the available app stores (not those selling J2ME apps alone) can be found in the WIP App Store Catalogue[43]. Also see the separate chapter on Appstores in this guide to learn more.

Learn More

If you want to learn more about Java ME development, below are a few resources that might help you.

Online

As Java ME is one of the oldest mobile platforms still used, it is easy to find resources related to it. For example:

[37] store.handmark.com
[38] mobilerated.com
[39] getjar.com
[40] publish.ovi.com
[41] www.orangepartner.com/distribute
[42] mobileapps.o2online.de
[43] wipconnector.com/appstores/

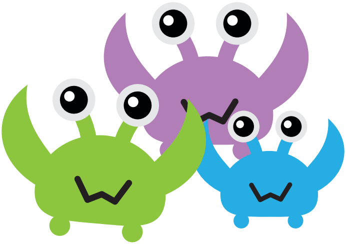
- Tutorials from sites such as J2ME Salsa[44].
- Resource archives from sites such as *billday.com/javame*
- Interesting projects via the blog at *opensource.ngphone.com* or on the Mobile and Embedded page of *java.net*[45], for example the Bluetooth project Marge[46].

Books

Over the years, a number of good Java ME books have been written, for example:

- **Beginning J2ME: From Novice to Professional** by Jonathan Knudsen and Sing Li
- **Pro Java Me Apps: Building Commercial Quality Java ME Apps** by Ovidiu Iliescu
- **Pro J2ME Polish: Open Source Wireless Java Tools Suite** by Robert Virkus, dealing with J2ME Polish development.
- **LWUIT 1.1 for Java ME Developers** by Biswajit Sarkar, dealing with LWUIT development

Unfortunately, due to Java ME's decreasing popularity, very few Java ME books have been written in recent years.



[44] j2mesalsa.com
[45] community.java.net/mobileandembedded/
[46] marge.java.net/

BY Marco Büttner & Patrick Mortara

Tizen

The Ecosystem

Tizen is an open source, Linux based operating system designed to run on smartphones, netbooks, In-Vehicle-Infotainment (IVI) systems and other smart devices. It can be viewed as a successor to Nokia/Intel's Meego and Samsung's LiMo; earlier smartphone operating systems based on Linux. Samsung also merged the remains of their abandoned bada OS into Tizen, providing a framework for native apps. Tizen, as a brand of the Linux Foundation, was first announced by the Tizen Association in December 2011 and version 1.0 (codename 'Larkspur') was released in April 2012. Since then, the system has been under continuous development, with Tizen 3.0 due for releases later in 2014. The main drivers of Tizen are Samsung for the mobile branch and Intel for the IVI focus. Examples of other contributing companies are Fujitsu, NTT Docomo, Huawei, Vodafone and Orange.

The first Tizen smartphones are expected to be released during 2014; at the time of writing Samsung only has a number of digital camera models running Tizen.

Development

The main focus of Tizen is to enable a standards-based operating system for running apps written in HTML5. However, Tizen enables you to write native apps in C++ as well, giving you the power to max out the capabilities of the hardware. Both development paths are supported by a variety of popular frameworks and libraries, such as JQuery, to give you a good start with your first Tizen app. If you have written native bada apps before, you can migrate them quite easily. Very few code-changes should be needed to make baba apps run on Tizen, there even is a migration tool available to make this process easier.

Web app developers can use a comprehensive list of HTML5 features, Tizen device APIs and libraries — such as JQuery and JQuery Mobile — to create beautiful apps. If you have created web apps for bada, you can use the most of the original code on Tizen. As with native apps, very few code changes should be needed.

The official Tizen SDK contains an Eclipse-based IDE, which can be used for both web and native app development. Former bada developers will probably recognize the roots of this SDK: Samsung's bada SDK. A code editor, UI designer, device emulator ... it is all in there, ready to go. For web-based applications you can also use Intel XDK, if you prefer.

Testing your apps

Of course, the best tests are those that can be done on a device. At the time of writing your only option would be trying to obtain of one of Samsung's rare Tizen test devices, the RD-PQ and RD-210. However, these devices are very hard to get. Therefore you will probably have to use the simulator and emulator included in the Tizen SDK, while waiting for the first mass market Tizen phones to become available. The simulator provides a simple approach to testing web-based apps, but is very limited in its features and cannot be compared to a real-life testing on a device. The emulator is much nearer to the device experience and can be used to test native apps as well. It is a virtual machine based on QEMU and running an image of a current Tizen mobile installation.

For those cases where a real device is needed, you can use Samsung Remote Test Lab[1]. These labs are situated around the world and give you the opportunity to remotely connect to devices from within your Tizen SDK.

Distribution

Tizen apps are distributed in Tizen package files (*.tpk) and widget files (*.wgt) created by the Tizen SDK. Very similar to Androids *.apk files, these can be installed by copying them to a Tizen device and tapping them in the file explorer on that device. The main hub for distributing apps will be TizenStore[2]. In contrast to Apple Appstore, Google Play Store and Microsoft Windows Phone Market you do not need to pay to become a registered developer at the Seller Office[3]. All submissions will be reviewed according to the Tizen Store guidelines. Usually, certification takes about 2 to 3 days, depending on the complexity of your app. Once the app is published, you will get 80% of the generated revenue.

Learn More

As a developer, your first stop should be developer.tizen.org. This site hosts all the documentation, tools and support services for Tizen development. There you can also find a forum with a very active and friendly community of Tizen app developers.

[1] developer.samsung.com/remotetestlab

[2] www.tizenstore.com

[3] seller.tizenstore.com

BY Robert Virkus

Windows Phone & Windows RT

Windows Phone and Windows RT are on a road to convergence. While they have not converged yet, it is just a matter of time – as they already share a significant number of details such as design paradigm, APIs and tooling. While developing for both platforms is not as straightforward as coding for the different form factors in Android or iOS, it is already much easier to share code and UI constructs. So, starting in this edition we are handling Windows Phone and Windows 8/RT in one chapter.

The Ecosystem

In the last edition of the guide we had a couple of complaints about Windows Phone:

- the lack of serious market traction.
- that Microsoft had not implemented their enthusiast program, which promised developers and power users the ability to install new versions of the OS ahead of carrier approval.

With much delight we have seen results in both cases: Microsoft has now started an early access program[1]. Windows Phone has also significantly improved its market share, now passing 10% in the big five European countries (UK, Germany,

[1] neowin.net/news/developers-finally-get-early-access-to-windows-phone-updates

France, Italy and Spain) according to Kantar World Panel[2]. In Italy and Latin America Windows Phone even outsells the iPhone significantly[3]. The only caveat is that growth is coming mainly from lower end phones, such as the Nokia Lumia 520.

So it would seem our criticisms were noted – that is how it should be! So, let us point out the some shortcomings in this release: Firstly, APIs need to grow considerably and should start to overtake other platforms, instead of slowly replicating features and APIs; secondly Windows Phone must improve its support for background apps; thirdly Windows Phone must offer better customisation options to manufacturers, so that for example Chinese companies can adapt their offering to local tastes better.

Active Windows Phone vendors are Nokia, Samsung, HTC, ZTE and Huawei. HTC is rumored to drop support for Windows Phone. The Nokia Lumia device range owns around 80% of this market[4]. Microsoft is realizing its 'services and devices' vision by not only bringing out their Surface Pro and Surface tablets but especially by buying Nokia[5].

The Windows Phone Store now contains more than 200,000 apps[6]; and the average Windows Phone user now installs 51 apps[7].

On Windows 8 and Windows RT side, Microsoft released the second iteration of its Surface tablets in Q4 2013. RT tablets, such as the original Surface RT, were not that successful, however the Surface 2 seems to be considerably more success-

[2] kantarworldpanel.com/global/News/news-articles/Apple-iPhone-5S-outsells-5C-three-to-one-in-Great-Britain

[3] microsoft.com/en-us/news/press/2013/aug13/08-21wplatampr.aspx

[4] blog.gsmarena.com/over-80-of-windows-phone-devices-made-by-nokia

[5] microsoft.com/nokia

[6] neowin.net/news/microsoft-windows-phone-store-now-has-over-200000-apps

[7] phonearena.com/news/Are-the-apps-in-the-Windows-Phone-Marketplace-of-a-higher-quality-than-the-ones-in-Google-Play-Store_id32045

ful. Most vendors have declined to release further Windows RT hardware with the notable exception of Nokia, releasing the Nokia Lumia 2520 tablet before it was acquired by Microsoft. Windows 8.1 received more positive feedback from users, Windows 8.1 and Windows 8 market share has now surpassed the share of all Mac OS X versions combined according to Net Applications[8].

One of the major criticisms of Windows 8 is its dual personality split between the desktop and the metro 'Windows Store Apps' world.

Languages and Tooling

Windows Phone development is undertaken in C/C++, C# or VB.NET, using Microsoft Visual Studio IDE or Expression Blend[9]. Applications are created using Silverlight, principally for event-driven applications, and DirectX, principally for games driven by a “game loop”, although both technologies can be used in a single application. Additionally you can create HTML 5 based apps using PhoneGap[10], however web development is not covered in this chapter. Last but not least you can create simple Windows Phone apps without coding using the Windows Phone App Studio[11].

Windows RT development can be done using the same languages and tools, with the additionally option to use JavaScript and HTML5 for the development of native WinRT apps. Microsoft also released the Project Siena[12] app, that provides a simple environment for creating business apps.

[8] netmarketshare.com/operating-system-market-share.aspx netmarketshare.com/operating-system-market-share.aspx

[9] dev.windowsphone.com

[10] phonegap.com

[11] apps.windowsstore.com

[12] microsoft.com/en-us/projectsiena

It is important to consider which platform you should leverage when building your application.

	C/C++	C#/VB.NET	JavaScript
WinRT	yes	yes	yes
Windows Phone	yes	yes	no (only within webview)
Silverlight/ XAML	yes	yes	no
HTML	no (but webview available)	no (but webview available)	yes
DirectX	yes	yes (with SharpDX)	no
Codesharing	Legacy native Windows Apps, professional Xbox, other platforms, ...	Legacy .NET Windows Apps, indie Xbox, Windows Phone apps, ...	Websites, HTML5 apps, ...

If you want to use DirectX with C#, you can use SharpDX.org, anxframework.codeplex.com or game libraries based on that, such as monogame.codeplex.com.

While the most common scenario is to use Silverlight for apps and DirectX for games, you can also create Silverlight games and DirectX apps, depending on your needs. It is also possible to host Direct3D inside your Silverlight application. This could be used to display a 3D model inside an event-driven Silverlight application, or to easily create stylish Silverlight-based menus around a full DirectX game.

Metro Design Paradigm

Windows Phone's most obvious specific characteristic is the unique, simple-to-use interface that focuses on typography and content. This UI paradigm called Metro or Modern UI[13] has been extended to the Xbox and Windows 8 as well. This UI paradigm contains the following principles:

- **Content not Chrome** removes unnecessary ornaments and lets the content itself be the main focus. You should also refrain from using every available pixel, as whitespace gives balance and emphasis to content.
- **Alive in motion** adds depths to the otherwise flattened out design with rich animations
- **Typography is beautiful** moves fonts to first class citizens within Metro. The Helvetica inspired Segoe font of Windows Phone matches the modernist approach.
- **Authentically digital** design does not try to mimic real world object but instead focuses on the interactions that are available to digital solutions.

While Microsoft abandoned the 'Metro' name for its design paradigm due to legal worries, alternative names such as 'Modern UI' never really caught on.

You should embrace the Metro UI design principles in your application, especially when porting over existing apps. Designers will find many inspirations and information at dev.windowsphone.com/design as well as design.windows.com.

Important for the overall experience are the 'live tiles', small widgets that reside on the start screen. You can update them programmatically or even remotely using push notifications.

[13] wikipedia.org/wiki/Metro_(design_language)

Codesharing between Windows RT and Windows Phone

As both Windows Phone 8 and Windows 8 share the same kernel, there are many APIs present on both operating systems. Note that some APIs are present but not (fully) implemented on Windows Phone. Refer to the API documentation for details. There are some Nuget compatibility packages available for popular APIs such as the HttpClient[14]. The BCL package[15] even allows you to use the async and await pattern on the old Windows Phone 7 platform.

You can create libraries that are shared between several platforms using a Portable Class Library project. Such projects can be created in Visual Studio Professional or higher, the free Expression editions do not support that.

You can also share code directly between projects in Visual Studio by linking sources between projects.

Integrating into the Platform

Both Windows Phone and Windows RT support a deep integration into the platform beyond the list of apps.

Both systems support 'live tiles' that show additional arbitrary information that can also be updated via push messages. Then you can use lockscreen apps that control the image of the lock screen on both Windows Phone and Windows RT.

On Windows Phone you can for example create camera extension apps that are called lenses[16], extend the music hub[17] or integrate into the search experience[18].

[14] nuget.org/packages/Microsoft.Net.Http

[15] blogs.msdn.com/b/bclteam/archive/2012/10/22/using-async-await-without-net-framework-4-5

[16] msdn.microsoft.com/library/windowsphone/develop/jj206990

[17] msdn.microsoft.com/library/windowsphone/develop/ff769558

[18] msdn.microsoft.com/library/windowsphone/develop/hh202957

For a complete overview visit the integration documentation[19].

On Windows RT there are contracts[20] that serve the same purpose, you can handle specific file extensions, take part in sharing content and more.

MVVM

For app developers coming from other platforms the data binding concepts of XAML will be new. For each page there should be a view model that includes the data for that page. The view itself only describes the UI, the actually displayed data is populated with the data from the view model. Model classes contain the actual data. This concept of a Model, a View and a ViewModel (MVVM) ease the development of complex apps considerably.

Game Engines

Thanks to the native app capabilities there are some game engines available for Windows Phone 8 and Windows RT, for example:

- Cocos2d-x[21]
- Havok[22]
- Marmalade[23]
- OGRE[24]
- Unity 3D[25]

[19] msdn.microsoft.com/library/windowsphone/develop/hh202969
[20] msdn.microsoft.com/library/windows/apps/hh464906
[21] cocos2d-x.org/wiki/Windows_Phone_8_Environment_Setup
[22] havok.com/products/havok-windows-ecosystem
[23] developer.madewithmarmalade.com/develop/supported-platforms
[24] ogre3d.org/2012/10/30/ogre-now-supports-windows-phone-8
[25] unity3d.com/pages/windows

Services

Push notifications[26] are available that can also update the live tiles of your app. You can also consider using the freely available SkyDrive cloud space and integrate with other Windows Live services[27] in your app. There are many third party offerings[28] available as well.

Multitasking and Application Lifecycle

Windows Phone has a limited form of multitasking that suspends applications in the background and allows for fast application switching. The only processes that can be run in the background, after an application has been left, are audio playback, location tracking and file transfer. Applications can also schedule to run arbitrary code in the background at an interval (code which is known as Background Agents). Background Agents are allowed limited use of resources and may be stopped or skipped if the OS determines that the phone needs to conserve resources.

Applications suspended in the background may be closed automatically if the OS determines resources are needed elsewhere.

To create the appearance of an application that was never closed, Windows Phone has a well-documented application lifecycle called Tombstoning[29]. To make Tombstoning possible, the Windows Phone framework provides the hooks needed to perform actions during different stages of the application lifecycle (such as caching and restoring data and UI states). With Windows Phone 8 there is also a new "fast app resume" feature available to developers.

Windows Store apps have a similar life cycle[30].

[26] msdn.microsoft.com/library/windowsphone/develop/ff402558

[27] msdn.microsoft.com/live

[28] dev.windowsphone.com/en-us/featured/partners

[29] msdn.microsoft.com/library/windowsphone/develop/ff817008

[30] msdn.microsoft.com/library/windows/apps/hh464925

Testing and Analytics

Unit tests are integrated into Visual Studio, just create an additional Unit Test project in your solution and reference the projects you would like to test.

For behavior-driven development, the Windows Phone Test Framework by Expensify[31] is available.

For developers wishing to collect runtime data and analytics, there are several options. Localytics[32] and Flurry[33] provide analytics tools and services that are compatible with Windows Phone and Windows RT. Developers can also use the Silverlight Analytics Framework[34] to connect to a variety of third-party tracking services such as Google Analytics. There are robust performance monitoring tools available in Visual Studio.

Distribution

Distribute your apps through the Microsoft Windows Store. While application content is reviewed and restricted in a way similar to the Apple App Store, Microsoft provides fairly comprehensive guidelines for submission, available at Dev Center[35]. Although developer tools are provided free of charge, a paid Store account is necessary to deploy software to devices through the Windows Store. Currently, a developer account costs 19 USD for individuals and 99 USD for companies per year. The fee is waived for students in the DreamSpark[36] program. The Store also provides for time-limited beta distribution and

[31] github.com/Expensify/WindowsPhoneTestFramework

[32] localytics.com/docs/sdks-integration-guides/#winphone7

[33] flurry.com/flurry-analytics.html

[34] msaf.codeplex.com

[35] dev.windowsphone.com

[36] www.dreamspark.com

offers a company hub for enterprises[37]. You can use the Windows Phone Store Test Kit or the Windows Certification Kit that are both integrated with Visual Studio to test your application locally before you submit them.

The standard revenue share of 70% is increased to 80% when your app makes more than 25,000 USD. The Windows Store supports over 200 countries and regions and more than 100 languages, so you can have a global reach.

Apps are managed by customer, not by device. So a user can use your app across a variety of platforms, such as a desktop PC and a tablet.

For paid applications, the Windows Phone framework provides the ability to determine if your application is in "trial mode" or not and limit usage accordingly. Microsoft specifically recommends against limiting trials by time (such as a thirty-minute trial) and instead suggests limiting features instead[38].

For ad-based monetization, there are several options. Microsoft has their own Microsoft Advertising Ad Control[39] (currently available in 18 countries), Smaato[40], inneractive[41], AdDuplex[42] and Google[43] all offer alternative advertising solutions. For more general information about monetization, please the dedicated chapter in this guide.

[37] msdn.microsoft.com/library/windowsphone/develop/jj206943
[38] msdn.microsoft.com/library/windowsphone/develop/ff967558
[39] advertising.microsoft.com/mobile-apps
[40] smaato.com
[41] inner-active.com
[42] adduplex.com
[43] developers.google.com/mobile-ads-sdk/

Learn More

Visit *dev.windowsphone.com* and *dev.windows.com* for news, developer tools and forums.

The development team posts on their blog at blogs.*windows.com/windows_phone/b/wpdev* and *blogs.windows.com/windows/b/appbuilder* or their Twitter account @wpdev. For a large collection of developer and designer resources, visit *windowsphonegeek.com* and *reddit.com/r/wpdev*.

There are currently several built-in OS controls that are not included in the Windows Phone SDK, such as context menu, date picker, and others. Those controls are available as part of the Phone Toolkit for Windows Phone, available at phone.codeplex.com. Other popular Windows Phone projects include coding4fun.codeplex.com, cimbalino.org and mvvmlight.codeplex.com. For inspecting the visual tree, bindings and properties of XAML-based user interfaces at runtime, xamlspy.com is available.

Find many video tutorials about all things Windows and Windows Phone at *channel9.msdn.com*.

Find sample code on *code.msdn.microsoft.com/windowsapps*, *code.msdn.microsoft.com/wpapps*, in various *codeplex.com* projects and in the end to end samples available at *msdn.microsoft.com/library/windows/apps/br211375*. The roadmap for app developers provides an good overview about planning, designing and developing Windows RT apps at *msdn.microsoft.com/library/windows/apps/xaml/br229583*.

If you are migrating from iOS or Android you can find help at *dev.windowsphone.com/en-us/featured/porting* and *msdn.microsoft.com/library/windows/apps/dn436165*.

BY Robert Virkus

Going Cross-Platform

So many platforms, so little time: This accurately sums up the situation that we have in the mobile space. There are more than enough platforms to choose from: Android, BlackBerry 10, Firefox OS, iOS, Tizen, Windows 8, and Windows Phone are or will likely be among the most important smartphone and tablet platforms while Brew MP and Java ME dominate on feature phones (arranged not by importance but rather alphabetically).

Most application sponsors, to quote Queen's famous lyrics, will tell the developer: "I want it all, I want it all, I want it all ...and I want it now!" So the choice may be between throwing money at multiple parallel development teams, or adopting a cross-platform strategy.

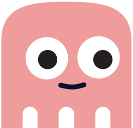
Key Differences Between Mobile Platforms

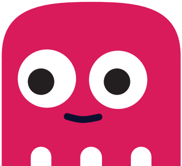
If you want to deliver your app across different platforms you have to overcome some obstacles. Some challenges are easier to overcome than others:

Programming Language

By now you will have noticed that most mobile platforms release their own SDKs, which enable you to develop apps in the platforms' supported programming languages.

However, these languages tend to belong to one of a few families of root languages and the following table provides an overview of these and the platforms they are supported on:





Language	1st class citizen[1]	2nd class citizen[2]
ActionScript	BlackBerry 10, BlackBerry PlayBook OS (QNX)	none
C, C++	BlackBerry 10, Brew MP, Sailfish OS, Windows RT, Windows Phone 8	Android (partially, using the NDK), iOS (partially)
C#	Windows 8, Windows Phone	none
Java	Android, BlackBerry, Java ME devices	none
JavaScript	BlackBerry PlayBook OS, Firefox OS, Tizen, Windows 8	BlackBerry (WebWorks), Nokia (WRT)
Objective-C	iOS	none

[1] Supported natively by the platform, for example either the primary or only language for creating applications

[2] Supported natively by the platform, for example either the primary or only language for creating applications

Cross platform frameworks can overcome the programming language barriers in different ways:

- **Web Technologies**
- **Interpretation**
- **Cross Compilation**

Most frameworks also provide a set of cross platform APIs that enable you to access certain platform or device features, such as a device's geolocation capabilities, in a common way. For features such as SMS messaging you can also use network APIs that are device-independent[1].

OS Versions

Platforms evolve and sooner or later they will be version specific features that you want to leverage. This adds another layer of complexity to your app and also a challenge for cross-platform tools: sometimes they lag behind when a new OS version is released.

UI and UX

A difficult hurdle for the cross platform approach is created by the different User Interface (UI) and User eXperience (UX) patterns that prevail on individual platforms.

It is relatively easy to create a nice looking UI that works the same on several platforms. Such an approach, however, might miss important UI subtleties that are available on a single platform only and could improve the user experience drastically. It would also ignore the differences when it comes to the platforms' design philosophies: While many platforms strive for a realistic design in which apps look like their real world counterparts, Windows Phone's Metro interface strives for an "authentically digital" experience, in which the content is emphasized not the chrome around it. Another key challenge with a uniform cross-platform UI is that it can behave differently to the native UI users are familiar with, resulting in your application failing to “work” for users. A simple example is not to support a hardware key such as the back key on a given platform correctly. Another challenge is the **uncanny valley** that results from mimicking native UI elements that look but do not work the same. Instead of mimicking native controls you

[1] www.developergarden.com/apis/

should either use non-native looking ones or just use the 'real deal'.

When you target end consumers directly (B2C), you often need to take platform specific user experience much more into account than in cases when you target business users (B2B). In any case you should be aware that customizing and tailoring the UI and UX to each platform can be a large part of your application development effort and is arguably the most challenging aspect of a cross platform strategy.

Desktop Integration Support

Integration of your application into devices' desktops varies a lot between the platforms; on iOS you can only add a badge with a number to your app's icon, on Windows Phone you can create live tiles that add structured information to the desktop, while on Android you can add a full-blown desktop widget that may display arbitrary data and use any visuals.

Using desktop integration might improve the interaction with your users drastically.

Multitasking Support

Multitasking enables background services and several apps to run at the same time. Multitasking is another feature that is realized differently among operating systems. On Android, BlackBerry and Sailfish OS there are background services and you can run several apps at the same time; on Android it is not possible for the user to exit apps as this is handled automatically by the OS when resources run low. On iOS and Windows Phone we have a limited selection of background tasks that may continue to run after the app's exit. So if background services can improve your app's offering, you should evaluate cross platform strategies carefully to ensure it enables full access to the phone's capabilities in this regard.

Battery Consumption And Performance

Closely related to multitasking is the battery usage of your application.

While CPU power is roughly doubled every two years (Moore's law says that the number of transistors is doubled every 18 months), by contrast battery capacity is doubling only every seven years. This is why smartphones like to spend so much time on their charger. The closer you are to the platform in a crossplatform abstraction layer, the better you can control the battery consumption and performance of your app. As a rule of thumb, the longer your application needs to run in one go, the less abstraction you can afford.

Also some platforms have a great variety of performance, most notably Android – Android devices range from painfully slow to über-fast.

Push Services

Push services are a great way to give the appearance that your application is alive even when it is not running. In a chat application you can, for example, send incoming chat messages to the user using a push mechanism. The way push services work and the protocols they use, again, can be realized differently on each platform. The available data size, for example, ranges between 256 bytes on iOS and 8kb on BlackBerry. Service providers such as Urban Airship[2] support the delivery across a variety of platforms.

In App Purchase

In app purchase mechanisms enable you to sell services or goods from within your app. Needless to say that this works differently across platforms. See the monetization chapter for details.

[2] urbanairship.com/

In App Advertisement

There are different options for displaying advertisements within mobile apps, some are vendor independent third-party solutions. Platform specific advertisement services, however, offer better revenues and a better user experience. Again, these vendor services work differently between the platforms. The monetization chapter in this guide provides more information on this topic as well.

Cross-Platform Strategies

This section outlines some of the strategies you can employ to implement your apps on different platforms.

Direct Support

You can support several platforms by having a specialized team for each and every target platform. While this can be resource intensive, it will most likely give you the best integration and user experience on each system. An easy entry route is to start with one platform and then progress to further platforms once your application proves itself in the real world.

Component libraries can help you to speed up native development, popular examples are listed in the following table.

Component Library	Target Platforms
cocoacontrols.com	iOS
chupamobile.com	Android, iOS
verious.com	Android, iOS, HTML5, Windows Phone
windowsphonegeek.com/Marketplace	Windows Phone

Asset Sharing

When you maintain several teams for different platforms you can still save a lot of effort when you share some application constructs:

- **Concept and assets**: Mostly you will do this automatically: share the ideas and concepts of the application, the UI flow, the input and output and the design and design assets of the app (but be aware of the need to support platform specific UI constructs).
- **Data structures and algorithms**: Go one step further by sharing data structures and algorithms among platforms.
- **Code sharing of the business model:** Using cross platform compilers you can also share the business model between the platforms. Alternatively you can use an interpreter or a virtual machine and one common language across a variety of platforms.
- **Complete abstraction**: Some cross platform tools enable you to completely abstract the business model, view and control of your application for different platforms.

Player And Virtual Machines

Player concepts typically provide a common set of APIs across different platforms. Famous examples include Flash, Java ME and Lua. This approach makes development very easy. You are dependent, however, on the platform provider for new features and the challenge here is when those features are available on one platform only. Often player concepts tend to use a “least common denominator” approach to the offered features, to maintain commonality among implementations for various platforms. Generator concepts like Applause[3] carry the player concept a step further, they are often domain specific and enable you to generate apps out of given data. They often lack flexibility compared to programmable solutions.

[3] applause.github.com

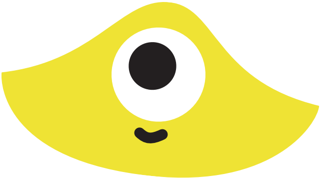
Cross Language Compilation

Cross language compilation enables coding in one language that is then transformed into a different, platform specific language. In terms of performance this is often the best cross platform solution, however there might be performance differences when compared to native apps. This can be the case, for example, when certain programming constructs cannot be translated from the source to the target language optimally.

There are three common approaches to cross language compilation: direct source to source translation, indirectly by translating the source code into an intermediate abstract language and direct compilation into a platform's binary format. The indirect approach typically produces less readable code. This is a potential issue when you would like to continue the development on the target platform and use the translated source code as a starting point.

(Hybrid) Web Apps

Some of the available web application frameworks are listed in the following table. With these frameworks you can create web apps that behave almost like real apps, including offline capabilities. However, be aware that the technologies have limitations when it comes to platform integration, performance, and other aspects. Read the web chapter to learn more about mobile web development.



Web App Solution	License	Target Platforms
Chrome Apps developer.chrome.com/apps	BSD	Android, Mac, Windows
jQuery Mobile www.jquerymobile.com	MIT and GPL	Android, BlackBerry, Firefox, iOS, Windows Phone
Sencha Touch www.sencha.com/products/touch	GPL	Android, BlackBerry, iOS, Windows Phone
The M Project the-m-project.org	MIT and GPL	Android, BlackBerry, Firefox, iOS, Windows

Typically you have no access to hardware features and native UI elements, so in our opinion they do not count as "real" cross platform solutions: these solutions are therefore not listed in the table at the end of this chapter.

Hybrid web development means to embed a webview within a native app. This approach allows you to access native functionality from within the web parts of your apps and you can also use native code for performance or user experience critical aspects of your app. Hybrid apps allow you to reuse the web development parts across your chosen platforms – a well known example for a hybrid web framework is PhoneGap.

ANSI C

While HTML and web programming starts from a very high abstraction you can choose the opposite route using ANSI C. You can run ANSI C code on all important platforms like Android, BlackBerry 10, iOS and Windows 8/Windows Phone. The main problem with this approach is that you cannot access platform specific APIs or even UI controls from within ANSI C. Using C is mostly relevant for complex algorithms such as audio encoders. The corresponding libraries can then be used in each app project for a platform.

Cross-Platform App Frameworks

There are many cross-platform solutions available, so it is hard to provide a complete overview. You may call this fragmentation, we call it competition. A word of warning: we do not know about all solutions here, if you happen to have a solution on your own that is publicly available, please let us know about it at *mdgg@enough.de*. A framework needs to support at least two mobile platforms to be listed.

Here are some questions that you should ask when evaluating cross platform tools. Not all of them might be relevant to you, so weight the options appropriately. First have a detailed look at your application idea, the content, your target audience and target platforms. You should also take the competition on the various platforms, your marketing budget and the know-how of your development team into account.

Research 2 Guidance also released a report on various cross-platform tools[4].

- How does your cross platform tool chain work? What programming language and what API can I use?
- Can I access platform specific functionality? If so, how?
- Can I use native UI components? If so, how?
- Can I use a platform specific build as the basis for my own ongoing development? What does the translated/generated source code look like?
- Is there desktop integration available?
- Can I control multitasking? Are there background services?
- How does the solution work with push services?
- How can I use in app purchasing and in-app advertisement?
- How does the framework keep up with new OS releases?

[4] research2guidance.com/cross-platform-tool-benchmarking-2013

Solution	License	Input	Output
Akula verivo.com	Commercial	(Visual)	Android, BlackBerry, iOS, Windows Phone
Application Craft applicationcraft.com	Commercial	HTML, CSS, JavaScript	Android, BlackBerry 10, iOS, Windows Phone, mobile sites
Codename One codenameone.com	Commercial	Java	Android, BlackBerry, iOS, J2ME, Windows Phone
Corona coronalabs.com (Corona Labs)	Commercial	JavaScript	Android, iOS, Kindle, Nook, Windows Phone, Windows RT
J2ME Polish j2mepolish.org (Enough Software)	Open Source + Commercial	Java ME, HTML, CSS	Android, BlackBerry, J2ME, PC
Flash Builder adobe.com/devnet/devices.html (Adobe)	Commercial	Flash	Android, BlackBerry Tablet OS, iOS, PC
Feedhenry feedhenry.com	Commercial	HTML, CSS, JavaScript	Android, iOS, HTML5, Windows Phone
Kony One kony.com/products/develop/studio	Commercial	HTML, CSS, JavaScript, RSS	Android, BlackBerry, iOS, J2ME, Windows Phone, PC, Web

Solution	License	Input	Output
LiveCode runrev.com (RunRev)	Commercial	English-like	Android, iOS, PC and Web
M2Active service2media.com (Service2Media)	Commercial	Drag and Drop + Lua	Android, BlackBerry, iOS, Windows Phone
MobiForms mobiforms.com (MobiForms)	Commercial	Drag and Drop + MobiScript	Android, iOS, PC, Windows Mobile
MoSync mosync.com	Open Source + Commercial	C/C++, HTML5/ JS	Android, BlackBerry, iOS, J2ME, Windows Phone
NeoMAD neomades.com	Commercial	Java	Android, BlackBerry, iOS, J2ME, Windows Phone
Orubase orubase.com	Commercial	ASP .NET MCV	Android, iOS, Windows Phone, Windows RT
PhoneGap/ Cordova phonegap.com (Adobe/Apache)	Open Source	HTML, CSS , JavaScript	Android, BlackBerry 10, iOS, Windows Phone
Qt qt.digia.com (Digia)	Open Source + Commercial	C++	Android, BlackBerry 10, iOS, Sailfish OS, Windows RT

Solution	License	Input	Output
Rhodes motorolasolutions.com/US-EN/RhoMobile+Suite/Rhodes (Motorola)	Open Source + Commercial	Ruby, HTML, CSS, JavaScript	Android, BlackBerry, iOS, Windows Phone
Titanium appcelerator.com (Appcelerator)	Open Source	JavaScript	Android, iOS, Tizen, Mobile Web
trigger.io trigger.io (Triggger Corp)	Commercial	HTML5, JavaScript	Android, iOS, Windows Phone
webinos webinos.org	Open Source	JavaScript	Android, BlackBerry, iOS, PC, TV
webMethods Mobile Designer (formerly Metismo Bedrock) metismo.com (Software AG)	Commercial	Java ME	Android, BlackBerry, brew, Consoles, iOS, PC, Windows Phone, Windows Mobile
Xamarin xamarin.com	Commercial	C#	iOS, Android, Windows Phone, PC
XDK xdk.intel.com (Intel)	Free to use	HTML, CSS, JavaScript	Android, iOS, Windows Phone, Windows RT
XML VM xmlvm.org	Open Source + Commercial	Java, .NET, Ruby	C++, Java, JavaScript, .NET, Objective-C, Python

Cross-Platform Game Engines

Games are very much content centric and often do not need to integrate deeply into the platform. So cross-platform development is often more attractive for games than for apps.

Solution	License	Input	Output
Cocos 2D cocos2d-x.org	Open Source	C++, HTML5, JavaScript	Android, BlackBerry, iOS, Windows 8, Windows Phone
Corona coronalabs.com (Corona Labs)	Commercial	Lua	Android, iOS, Kindle, nook, Windows Phone, Windows RT
EDGELIB edgelib.com (elements interactive)	Commercial	C++	Android, iOS, PC
Esenthel esenthel.com (elements interactive)	Commercial	C++	Android, iOS, PC
GameSalad gamesalad.com	Commercial	Drag and drop	Android, iOS, Tizen, Windows Phone, PC, web
Gideros Mobile giderosmobile.com	Commercial	Lua	Android, iOS
id Tech 5 idsoftware.com (id)	Commercial	C++	Consoles, iOS, PC
Irrlicht irrlicht.sourceforge.net	Open Source	C++	Android & iOS with OpenGL-ES version, PC

Solution	License	Input	Output
IwGame drmop.com/index.php/iwgame-engine	Open Source	C++	Android, BlackBerry Playbook OS, iOS, PC
Marmalade madewithmarmalade.com (Ideaworks3D)	Commercial	C++, HTML5, JavaScript	Android, BlackBerry 10, iOS, LG Smart TV, Windows Phone, Windows RT
Moai getmoai.com (Zipline Games)	Commercial	Lua	Android, iOS, PC, Web
MonoGame monogame.codeplex.com	Open Source	C#, XNA	Android, iOS, PC, Windows Phone, Windows RT
Ogre 3D ogre3d.org	Open Source	C++	Windows Phone, Window RT, PC
orx orx-project.org	Open Source	C, C++, Objective-C	Android, iOS, PC
ShiVa 3D stonetrip.comShiVa 3D stonetrip.com	Commercial	C++	Android, BlackBerry 10, iOS, PC, Consoles
SIO2 sio2interactive.com (sio2interactive)	Commercial	C, Lua	Android, iOS, PC
Unigine unigine.com (Unigine corp.)	Commercial	C++, UnigineScript	Android, iOS, PC, PS3
Unity3D unity3d.com (Unity Technologies)	Commercial	C#, JavaScript, Boo	Android, BlackBerry 10, iOS, Windows Phone, Windows RT, PC, consoles, web

BY Daniel Kranz

Mobile Sites & Web Technologies

The continuous development of web technology coupled with an increase in Internet-capable devices promises a great future for those catering to the ever-increasing mobile web audience. Global mobile Internet traffic is growing rapidly and has already surpassed 20%[1]. The share of time spent browsing the Internet by device (mobile, tablet, desktop and Smart TV) varies greatly across the globe. While users in the USA spend 16% of their time browsing the Internet on a mobile device, the Chinese spend 34% and Indians 79%. Most regions where mobile Internet traffic has already surpassed desktop Internet traffic are developing and emerging markets. While smartphone use is growing around the globe, there is still a huge opportunity to cater to a feature phone audience in developing and emerging markets. Take India as an example. Only 15% of Indians own a smartphone. Nevertheless this equates to a whopping 180 million users. At the same time 68% own a feature phone of which 9.5% are Internet-enabled feature phones[2]. Accordingly, a basic mobile site could help you reach another 116 million users.

The most obvious use of web technologies is to build mobile sites and this is also the key focus of this chapter. Nevertheless, it is worth pointing out that web technologies are also heavily used within web and hybrid mobile apps, cross-platform solutions and most recently native app development (Firefox OS). For more information on cross-platform development and the new Firefox OS, check out the respective chapters in this guide.

[1] gs.statcounter.com

[2] discovermobilelife.com

One big advantage of web technologies is that they offer the easiest route into mobile development. Web technologies, such as HTML, CSS and JavaScript have been well developed for many years; however they remain, and will continue to be, the main drivers of mobile site development. Additionally, they are arguably easier to learn than some of the rather complex languages needed for native app development. Mobile websites and web apps make content accessible on almost any platform with less effort in comparison to native development for a number of platforms. This means mobile websites automatically have a wider reach. Accordingly mobile web development not only saves development time and cost, but furthermore provides a time and cost-effective alternative when it comes to maintenance. And being independent of app stores allows you to offer any content you want quickly, and without having to align it to the app store's approval policy.

Nevertheless there are shortcomings. Web technologies struggle to match the level of deep platform integration and direct access to hardware features native app development can provide. Furthermore performance of web technologies is highly dependent on connectivity, large sites such as Facebook and LinkedIn experience memory issues and there is a lack of existing developer tools in comparison to developer tools available for native app development.

Monetization of mobile sites can prove tricky as well, since users expect to access mobile sites free of charge. The most common monetization tool for mobile sites is ad integration. Payment solutions for mobile sites are still in their early stages and tend to be rather challenging to implement. Existing app store monetization tools by contrast offer easy set-up and a high level of security for the end-user.

If monetization is one of the key requirements, a hybrid or web app strategy could prove to be a good compromise. In that case the key challenge is to combine the unique capabilities of native and web technologies to create a truly user-friendly product. In the cross-platform chapter of this book you will find a list of available frameworks to create hybrid apps.

As a guiding principle users should not be left frustrated and disappointed by being directed to a site which takes forever to load, triggers high data charges or does not work at all. Instead the worst-case scenario should be that a user is taken to a site that is basic but still provides all relevant content. Key criteria to consider prior to any development are your target audience's device capabilities, browsing habits and bandwidth/data plans.

From a UX perspective, Google offers 10 best practices to drive conversion for SMBs[3]:

- **Be thumb friendly** – design your site so even large hands can easily interact with it
- **Design for visibility** – ensure your content can be read at arm's length
- **Simplify navigation** – clear navigation, hierarchy and vertical scrolling aid access to information
- **Make it accessible** – ideally, your mobile site should work across all mobile devices and all handset orientations
- **Make it easy to convert** – focus on information that will aid conversion
- **Make it local** – including functionality that helps people find and get to you

[3] www.dudamobile.com/webinar/Google_DudaMobile_Webinar.pdf

- **Use mobile site redirects** – give users a choice to go back to the desktop site, but make it easy to return to the mobile site
- **Keep it quick** – help mobile users, design your site to load faster and make the copy easy to scan
- **Make it seamless** – bring as much of the functionality of your desktop site to mobile
- **Learn, listen and iterate** – good mobile sites are user-centric, meaning they're built with input from your audience.

Google has already rolled out changes to its mobile search results and has announced that it will penalize sites that are not in line with its recommendations. Have a look at Google's developer site[4] for more up-to-date information on how to optimize your mobile site.

HTML5

The fifth version of the HTML standard promises to reproduce features previously available only with the help of proprietary technology. HTML5 is one of the key drivers that makes coders and decision-makers consider developing mobile sites and web apps instead of native applications. A look-and-feel close to that of apps combined with a single code base for a number of popular devices, the ability to access device hardware such as the camera and microphone, local data storage for offline availability and optimization based on screen size make HTML5 an appealing alternative to native app development.

However HTML5 relies on universal browser support which is currently lacking.

Ex-Facebook CTO Brent Taylor describes the situation as fol-

[4] developers.google.com/webmasters/smartphone-sites

lows: ‘There is rampant technology fragmentation across mobile browsers, so developers do not know which part of HTML5 they can use. HTML5 is promoted as a single standard, but it comes in different versions for every mobile device. Issues such as hardware acceleration and digital rights management are implemented inconsistently. That makes it hard for developers to write software that works on many different phone platforms and to reach a wide audience.’

Most recently LinkedIn traded its mobile-web based apps in for a new set of native applications. Kiran Prasad, LinkedIn's senior director for mobile engineering decided to build both a HTML5 site and rich native apps. His reasoning is that HTML5 is ready and important for the business, but not supported by the ecosystem as it should be. 'There are tools, but they are at the beginning. People are just figuring out the basics.'[5]

For more info on browser compatibility, check out the HTML5Test online[6]. The site provides both an overview and in-depth analysis of which HTML5 features are supported by which browser. Facebook has also launched ringmark[7] which tests web browsers for 3 rings, or levels, of support for HTML5 features which helps developers to quickly check the level of support of various mobile (and desktop) web browsers.

To wrap it up: Almost everyone in the mobile business agrees that HTML5 will succeed in the long run. ABI research estimates that mobile devices with HTML-compatible browsers will total 1.4 billion worldwide by the end of 2013.[8]. Operating systems will gradually increase support for HTML5 features and browsers to increase overall adoption and speed. Open-source

[5] venturebeat.com/2013/04/17/linkedin-mobile-web-breakup

[6] html5test.com/results/mobile.html

[7] rng.io

[8] www.abiresearch.com/press/14-billion-html5-capable-mobile-devices-in-2013-bu

platforms such as the Firefox OS, Sailfish, Tizen and Ubuntu should also help to speed up adoption. Furthermore, the World-wide Web Consortium (W3C) has finally declared HTML5 feature complete and envisions that HTML5 will become an official web standard in 2014[9]. The Developer Economics 2014 research[10] already ranks HTML5 as the third most popular mobile app platform (after Android and iOS of course) with 52% mindshare.

Fragmentation Needs Adaptation

The biggest challenge of mobile site development is fragmentation. In theory all internet-enabled devices can access any mobile site via a browser. The reality however is that developers need to adapt and optimize mobile site content to cater to the ever increasing number of browsers and devices with varying levels of software and hardware capabilities.

Broadly speaking there are two approaches to optimizing content for mobile devices: Client-Side and Server-Side Adaptation.

- Client-Side Adaptation makes use of a combination of CSS and JavaScript running on the device to deliver a mobile-friendly experience.
- Server-Side Adaptation makes use of the server to execute logic before content is passed on to the client.

The following section provides an overview of client-side and server-side techniques used to make mobile sites accessible for the majority of current and future Internet-enabled devices.

[9] www.w3.org

[10] DeveloperEconomics.com

Client Side Adaptation

Responsive Web Design

Responsive Web Design has been a buzzword amongst marketers and web developers alike. In its simplest form responsive design consists of a flexible grid, flexible images and CSS media queries to cater to a number of screen resolutions or types of devices.

On its own this results in a device-sensitive experience for a limited range of devices and lacks sophisticated content adaptation. The same content is served to all devices. It is not advisable as a technique to deliver complex desktop and mobile sites.

Pro

- Pure client side adaptation ensures no impact on the existing infrastructure
- Automatic adjustment of content and layout possible

Con

- The same content available on the website will also be available on the mobile version (whether visible or not).
- The page weight of the site can have a significant impact in terms of performance on mobile devices
- It is a general approach instead of actual mobile-friendly device optimization (e.g. Top 5)

Progressive Enhancement

Progressive Enhancement has the capability to cater to the full spectrum of mobile devices. A single HTML page is sent to every device. JavaScript code is then used to progressively build up functionality to an optimal level for the particular device. As a mobile only solution the main drawback is performance. The progressive build-up takes time to execute and varies according to the device and network. As a desktop and mobile solution its main drawback is that a single HTML document is sent to all devices. A well-known framework that makes use of progressive enhancement is jQuery Mobile[11].

Pro

- Pure client side adaptation ensures no impact on the existing infrastructure
- Progressive adjustment of content, function and layout possible

Con

- A loss of control, since detection is handled by the browser
- Browser detection is still far from perfect
- Detection done on the client-side impacts overall performance of the site
- The same HTML page is served to all devices

Server-Side Adaptation

Device Databases

Device databases detect each device accessing the website and return a list of device capabilities to the server. This information is then used to serve a mobile site that caters to the device's capabilities. Server-side adaptation is one of the oldest and most reliable solutions. Popular device databases

[11] jquerymobile.com

include WURFL[12] and DeviceAtlas[13]. The main drawback of device databases is that the majority is only available as part of a commercial license.

Pro

- Most commonly used solution (Google, Facebook, Amazon and alike)
- Maximum control
- Device optimization possible (for example to iPhone, Samsung Galaxy and alike)

Con

- Device Description Repositories are hardware focused
- Besides the data, a detection mechanism is needed (a simple 'User-Agent' matching does not work)

Hybrid Adaptation

Truly the best of both worlds, the combination of client and server-side adaptation ensures high performance thanks to server-side adaptation and means that the capabilities sourced can be used to enrich the mobile experience on subsequent visits.

Hybrid adaptation solutions are available commercially from companies such as Sevenval[14] and Netbiscuits[15], and as community-backed cloud solutions, for example FITML[16].

[12] wurfl.sourceforge.net

[13] deviceatlas.com

[14] sevenval.com

[15] netbiscuits.com

[16] fitml.com

Better Data Input

With small, often on-screen, keyboards entering text can be cumbersome and time-consuming, particularly if the user has to enter numbers, email addresses or similar text. Thankfully developers can easily specify the expected type of input and smartphones will then display the most appropriate on-screen keyboard. mobileinputtypes.com provides various clear and concise examples.

Better Performance

Mobile users expect sites to load within 2-5 seconds. This is a currently a challenging task, especially for complex mobile sites. The following sections provides tips to reduce transfer size, content and HTTP requests to minimize load time and improve performance.

Reduce Transfer Size

Make use of image resizing and adjust the image quality according to network quality.

Reduce Content

Both site and asset loading becomes more and more important. Minifying assets such as JavaScript and CSS files can help to reduce overall asset load times. Multiple files of the same type are compressed into one and all whitespace is removed. Code becomes shorter, but still behaves in the same way. All

this can result in a lower number of requests and ultimately a faster loading time.

At the same time it is important that the user understands what is going on. Accordingly, if content is loading its important that the user is aware of this and is not presented with a blank box or page. A smooth experience is paramount to any mobile experience and this includes the journey from site to content loading in the site and any animation surrounding it.

Reduce HTTP Requests

Inline images, scripts and styles, and add JavaScript pipe and Application Caching. Key benefits are that scripts are delivered in a single request per page, HTTP round-trips are minimized and core scripts are stored in the application cache. The implementation will not affect reload and scripts are still cacheable publicly (CDN).

For more detailed tips around mobile web performance check out Roland Guelle's presentation for DWX[17].

Testing Web Technologies

How web technologies work in various mobile phones can be tested in several ways. The simplest way is to test the web site or web app in a variety of web browsers on mobile devices. These would include a mix of the most popular mobile web browsers, based for example on public data[18]. The set of devices can be refined by analyzing data from existing web logs and similar sources. Also, testing on various form-factors helps to expose layout and formatting issues.

[17] www.slideshare.net/sevenval/mobile-web-performance-dwx13

[18] gs.statcounter.com/#mobile_browser-ww-monthly-201207-201306

In terms of automated testing, WebDriver[19] is the predominant framework. There are two complementary approaches:

1. Automated testing using embedded WebView controls in Android and iOS
2. User-agent spoofing using Google Chrome or Mozilla Firefox configured to emulate various mobile web browsers

Both approaches have pros and cons:

- Embedded WebViews run on the target platform OS. They are likely to find many behavioral bugs. However the configuration is more involved and some platform OSs are not supported.
- Spoofing can fool web servers to treat the browser as if it came from any of a wide range of devices, including mobile browsers not available with the embedded WebView such as the Nokia Asha 201 phone. However the behavior and rendering is not realistic so many bugs will remain undetected, while other 'false positive' bugs will be found that do not occur on devices.

Learn More

Online

- **W3Schools and CSS Tricks** (good resource to understand basic HTML, CSS and JavaScript): *w3schools.com, css-tricks.com*
- **HTML5 Rocks** (great resource about HTML5 including tutorials, slideshows, articles and more): *html5rocks.com*
- **Breaking the Mobile Web** (Max Firtman, the author of

[19] seleniumhq.org/projects/webdriver/

several books about mobile web programming, provides up-to-date news in his dedicated mobile blog): *mobilexweb.com*
- **Mobi Thinking** (DotMobi's resource for marketers with insights, analysis and opinions from mobile marketing experts): *mobithinking.com*
- **Testing (Mobile) Web Apps:** *docs.webplatform.org/wiki/tutorials/Testing_web_apps*
- **Investigate what features work across all areas of the web:** *caniuse.com* and *beta.theexpressiveweb.com*
- **WHATWG** (The HTML community's homepage): *whatwg.org*
- **Word Wide Web Consortium** (The organization that defines web standards): *w3.org*

Books

- **Mobile First** by Luke Wroblewski
- **Adaptive Web Design: Crafting Rich Experiences** with Progessive Enhancement by Aaron Gustafson and Jeffrey Zeldman
- **Responsive Web Design** by Ethan Marcotte
- **Programming the Mobile Web** by Max Firtman
- **jQuery Mobile: Up and Running** by Max Firtman

Accessibility

Nearly 20% of the world's population have some form of disability. Accessibility has to do with creating a stellar user experience for all of your users and enabling your app to be used by as many people as possible.

Reasons you will want to make your apps accessible include but are not limited to:

- Implementing accessibility can often improve overall usability. For instance, including speech in your app can help blind people **and** also enable in-car use for drivers.
- Your app may be able to tap into government funded market sectors such as education where legislation, such as section 508 of the Rehabilitation Act in the US, may mandate an accessible solution.
- Mobile platforms from Apple, Google and Microsoft leverage their accessibility APIs for UI automation testing; so making your app accessible can make automated testing easier.

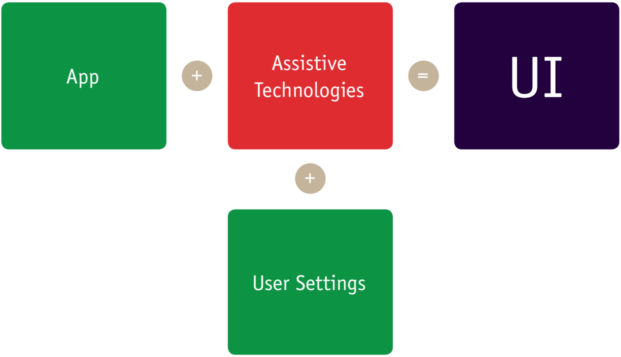
Many of your potential users may have a disability which makes it more difficult for them to use mobile technology. These disabilities include, but are not limited to, various levels of sight or hearing impairment, cognitive disabilities, dexterity issues, technophobia and the like. Many of these users rely on third-party utility software to assist them in using their device. This software is sometimes called Assistive Technology, and includes such utilities as screen reading and magnification apps. iOS includes VoiceOver[1] which is the front-runner in terms of providing an accessible interface on mobile phones. Android has a plug-in approach for Accessibility Software, the most

[1] apple.com/accessibility/iphone/vision.html

common plug-in is TalkBack. Several mobile platforms include screen magnification and other settings to make the user interface more accessible.

For these users, their overall experience is affected by how well an app works with the assistive technology.

UI: the sum of various factors



The Accessibility APIs look for text in specific attributes of standard UI elements. Screenreaders such as VoiceOver and TalkBack transform the text into spoken audio which the user listens to. The screenreader software may also determine the type of control and related attributes to help provide the user with more contextual information, particularly if no text is available.

Gestures may be affected when screenreaders are enabled. Several screenreaders, including VoiceOver and Android's Explore-By-Touch, enable the user to explore the screen to find what an element is by touching it, before they decide to interact with it. The changes may adversely affect how users can interact with your app. By testing your app with these screenreaders you can catch these problems early, before they affect end users. Also, you may be able to redesign the app and pick suitable gestures that work well with and without screenreaders being enabled.

To make your software accessible for users with disabilities, you should follow some general guidelines. If you stick to them, you will also give your app the best chance of interoperating with assistive technology that the user may be running in conjunction with your software:

- **Find out what accessibility features and APIs your platform has** and follow best practice in leveraging those APIs if they exist.
- **Use standard rather than custom UI elements** where possible. This will ensure that if your platform has an accessibility infrastructure or acquires one in the future, your app is likely to be rendered accessibly to your users
- **Follow the standard UI guidelines on your platform.** This enhances consistency and may mean a more accessible design by default
- **Label all images with a short description** of what the image is, such as “Play” for a play button.
- **Avoid using colour as the only means of differentiating an action.** For example a colour-blind user will not be able to identify errors if they are asked to correct the fields which are highlighted in red.
- **Ensure good colour contrast** throughout your app.
- **Use the Accessibility API for your platform,** if there is one. This will enable you to make custom UI elements more accessible and will mean less work on your part across your whole app.
- **Support programmatic navigation of your UI.** This will not only enable your apps to be used with an external

keyboard but will enhance the accessibility of your app on platforms such as Android where navigation may be performed by a trackball or virtual d-pad.

- **Test your app** on the target device with assistive technology such as VoiceOver on the iPhone.

You can find a more comprehensive list of guidelines online[2]. The BBC has also published a detailed set of guidelines[3] for developing accessible mobile apps.

Apple and more recently Google and Microsoft, have increased the importance of their respective Accessibility support by using the Accessibility interface to underpin their GUI test automation frameworks. This provides another incentive for developers to consider designing their apps to be more accessible, which is 'a good thing'.

Looking at the different mobile platforms more closely, it becomes obvious that they differ largely regarding their accessibility features and APIs.

Custom Controls and Elements

If you are using custom UI elements in your app, then, those platforms that have an Accessibility API enable you to make your custom controlls accessible. You do this by exposing the control to assistive technology running on the device so that it can interrogate the properties of the control and render it accessibly.

[2] slideshare.net/berryaccess/designing-accessible-usable-application-user-interfaces-for-mobile-phones

[3] bbc.co.uk/guidelines/futuremedia/accessibility/mobile_access.shtml

You can get more information about this process on Android from the Google IO 2012 presentations[4]

If you are a member of the Apple developer program, then take a look at their accessibility video presentations from WWDC 2012 and 2013 available in the iOS Developer Center[5].

Accessible Android Apps

The latest major version of Android, Version 4, brings a raft of accessibility improvements. These include the accessibility focus, Braille support and more. The developer documentation has also been enhanced. Subsequent minor versions, including 4.4, continue to improve the support for accessibility.

These include new capabilities such as accessible live regions and closed captioning support. Android Services[6]

To maximise the reach of your app, including to people using previous versions of Android:

- Use standard UI controls where practical
- Consider using the Support Library[7] which also includes ways to improve the accessibility of custom views.
- Make sure users can navigate your app via a trackball or D-pad, which will give your app the best chance of being rendered accessibly by the likes of TalkBack and other assistive technology applications.

For specifics on how to use the Android accessibility API along with details of best practice in Android accessibility,

[4] youtube.com/watch?v=q3HliaMjL38 *and* youtube.com/watch?v=ld7kZRpMGb8

[5] developer.apple.com/wwdc/videos

[6] developer.android.com/about/versions/android-4.3.html#A11y

[7] developer.android.com/tools/support-library/index.html

please see Google's document entitled Making Applications Accessible[8].

You will also find more examples in the training area of the developer documentation in a section entitled Implementing Accessibility[9]. Testing the Accessibility is also covered online[10].

For more information about Android accessibility including how to use the text to speech API, see the Eyes-Free project[11].

Accessible BlackBerry Apps

If you are targetting BB OS 7.1 you will find extensive information about the use of their accessibility API and many hints on accessible UI design on their website for developers[12].

In May 2012 Blackberry Released the BlackBerry Screen Reader[13] for various BlackBerry Curve phones. This is available as a free download which you may wish to use in the testing of the accessibility of your apps.

Blackberry 10 provides various accessibility settings to enable users to tailor their device. These include Magnify Mode, with gestures control magnification and navigation around the screen, and the ability to vary font sizes. Try using your app with magnification and by varying the font size from small to large to see how well your apps appear and how easily they can be used.

Blackberry 10.2 now ships with a screen reader. Documentation on creating accessible Blackberry 10 apps can be found at

[8] developer.android.com/guide/topics/ui/accessibility/apps.html

[9] developer.android.com/training/accessibility/index.html

[10] developer.android.com/tools/testing/testing_accessibility.html

[11] code.google.com/p/eyes-free

[12] developer.blackberry.com/java/documentation/intro_accessibility_1984611_11.html

[13] blackberry.com/screenreader

Accessibility features and best practices – BlackBerry Native[14]

Accessible iOS Apps

iOS has good support for accessibility. For example, iOS devices include:

- **VoiceOver**, a screen reader. It speaks the objects and text on screen, enabling your app to be used by people who may not be able to see the screen clearly
- **Zoom**: This magnifies the entire contents of the screen
- **White on Black**: This inverts the colors on the display, which helps many people who need the contrast of black and white but find a white background emits too much light
- **Captioning and subtitles** for people with hearing loss
- **Audible, visible and vibrating alerts** to enable people to choose what works best for them
- **Voice Control and Siri**: This enables users to make phone calls and operate various other features of their phone by using voice commands.

iOS 7 brings with it several new user-configurable accessibility settings as well as introducing Dynamic Type and Guided Access APIs.

If you are working on iOS, make sure to follow Apple's accessibility guidelines[15]. These guidelines detail the API and

[14] developer.blackberry.com/native/documentation/cascades/best_practices/accessibility/accessibility_features_best_practices.html

[15] developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/iPhoneAccessibility

provide an excellent source of hints and tips for maximising the user experience with your apps.

Accessible Windows Phone & Windows 8 Apps

There are 2 development paths for writing accessible apps on the Windows Phone/Windows 8 platform. They are XAML and HTML 5/JavaScript.

If your app is written in C# C++ or Visual Basic, you will find comprehensive information on making your app accessible in the document Accessibility in Metro style apps using C++, C#, or Visual Basic[16].

If you have chosen to use HTML 5 and JavaScript, then you will need Accessibility in Metro style apps using JavaScript[17].

Once you have tested the accessibility of your app[18], Microsoft uniquely allows you to declare your app as accessible[19] in the Windows store, allowing it to be discovered by those who who are filtering for accessibility in their searches.

Windows Phone 8 includes various ease-of-access settings including: high contrast, screen magnification, and text size. They apply to varying degrees to apps we develop, therefore it's worth testing our app to make sure they are as usable as practical with these various settings. Windows Phone 8.1 also offers a separate screen reader. It provides a limited set of in-built features and does not support other apps.

[16] msdn.microsoft.com/en-us/library/windows/apps/xaml/hh452680.aspx

[17] msdn.microsoft.com/en-us/library/windows/apps/hh452702.aspx

[18] msdn.microsoft.com/en-us/library/windows/apps/xaml/hh994937.aspx

[19] msdn.microsoft.com/en-us/library/windows/apps/xaml/jj161016.aspx

Accessible Mobile Web Apps

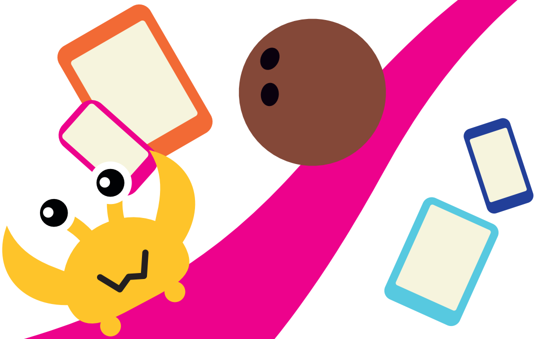
Much has been written on the subject of web accessibility, however, at the time of writing, there is no standard which embodies best practice for accessible mobile web development.

If your app is intended to mimic a native app look and feel, then you should follow the above guidelines in this chapter.

If you are a web content developer, then you should take a look at the Web Content Accessibility Guidelines (WCAG) Overview[20].

As support of HTML 5 is increasingly adopted on the various mobile platforms, consider reading Mobile Web Application Best Practices[21] as this is likely to form the foundation of any mobile web application accessibility standard that emerges in the future.

You will also find Relationship between Mobile Web Best Practices (MWBP) and Web Content Accessibility Guidelines (WCAG)[22] a helpful resource.



[20] w3.org/WAI/intro/wcag

[21] w3.org/TR/mwabp

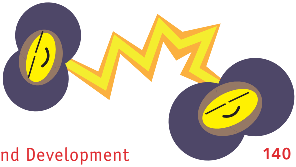
[22] w3.org/TR/mwbp-wcag/

BY Ian Thain & Davoc Bradley

Enterprise Apps: Strategy And Development

Corporate decision makers now view mobile enterprise apps as a strategic factor, a necessity, rather than an item on an accountant's spreadsheet. Internal enterprise apps are able to reduce the latency of information transfer within a company. They increase the agility of the worker by making competitive data available at any time and anywhere. Apps can also allow companies to engage with its customers, suppliers, and end consumers etc. Examples of enterprise apps include field & sales staff software, emergency response, inventory management, supply chain management but also B2C marketing.

It may seem an obvious thing to say, but the major risk at the moment, is **not** having an enterprise mobile strategy. Business is now looking at **Mobile for All** rather than limiting it to senior management, as it may have been in the past. To enable this the traditional IT approach of buying devices and distributing them throughout the management structure is no longer the only enabling strategy being used; Bring Your Own Device (BYOD) is taking hold, enabling staff to use their personal devices to connect to the IT infrastructure, download secure content and use enterprise apps. With the advent of BYOD, a company exposes itself to risks which traditionally have never been part of the corporate IT strategy. Early adoption of a well thought out and implemented enterprise mobile strategy is key to ensuring data is secured at all times.



Key points for Mobile Apps in Shaping the new Business Enterprise

- Cost reduction compared to existing systems
- Streamlining business processes
- Competitive advantage with up-to-date data immediately at hand
- Increase employee satisfaction and effectiveness
- Rapid response compared to existing processes

Enterprise Strategy

Many companies nowadays have a Chief Mobile Officer (CMoO) or have extended their CIO position. It is their job to coordinate mobile trends and directions and to bridge the gap between business and IT. Depending on the size and main focus of the company, his/her job is also to either build up an internal mobile software development team or coordinate the cooperation with an external development agency. To make sure that the mobile software delivers what the employees / users want, that this is technically achievable and that everything fits the overall company strategy, the leader might consider setting up a Mobile Innovation Council (MIC) or Center of Excellence (COE). This group should contain key members such as: skilled representatives from the mobile development team, stakeholders for mobile within the company, and most importantly end users from various departments with expertise in the relevant business processes.

Topics that the CMoO/CIO needs to focus on together with the MIC/COE include:

- Strategy – vision and direction for the general mobile strategy and for the apps.
- Governance policies – Bring Your Own Device (BYOD) vs. Chose Your Own Device (CYOD) which is essentially the difference between a Mobile Application Management (MAM) policy (BYOD) and a Mobile Device Management & Security (MDM) policy (CYOD)
- App specifications
- App roadmap
- Budget planning
- Acceptance – signing off the apps into production.
- App deployment – early feedback on demos and prototypes, testing, mass deployment
- Incentives – how to promote the adoption of mobile.

In commercial adoption terms Enterprise app development is still in its infancy, and as such one of the main hurdles a company writing third party enterprise apps, or a development manager keen to adopt an internal enterprise strategy will face is the requirement for a business need. The most common question is likely to be “This all sounds great, but why do we need it?”, so you must be prepared to give compelling reasons for a company to adopt a mobile strategy.

Key points when building the business case for Mobile Enterprise Apps

- **Create a Visionary Plan** for more mobile Apps and know how they will aid and shape your enterprise
- **Create an ADS (Application Definition Statement)** for each App, specifying purpose and intended audience.
- **Create a Budget** for devices.

- Create a plan for an **Application & Device Management Strategy & Security Infrastructure.**
- Create a plan for an **App Dev Team** using a future proof Development Platform – such as a MEAP/MADP

Mobilizing Existing Systems

If you are already providing a system to customers which has not yet been mobilized, you will have various decisions to make. It is critical to fully understand the impact of adding a mobile offering to your system before you start implementation of the solution. Common reasons to mobilize your product can include using phone features, such as camera and GPS, or just the ability to capture information on the move, without being connected to the internet. You must ensure you go mobile for the right reasons, as the ongoing support, maintenance and development of a mobile offering will become a separate product roadmap to your original system and will carry an on-going cost.

Key points when deciding how to mobilize an existing system

- Clearly define the reasons for going mobile and ensure that those reasons are strong enough to take the step into mobile
- Understand the difference between mobile and desktop. Do not just copy your existing system, so for instance, instead of a form to capture information, you could capture audio and upload that into your system, allowing a user to quickly makes notes without the need to type into a small device

- Do not try and implement all the features of your existing system; implement the important features in a way which suits mobile
- Ensure you understand which devices your clients use and which features of your system are most required to be mobilized
- Have a clearly defined mobile testing strategy which covers cross platform testing and multiple device types and operating systems

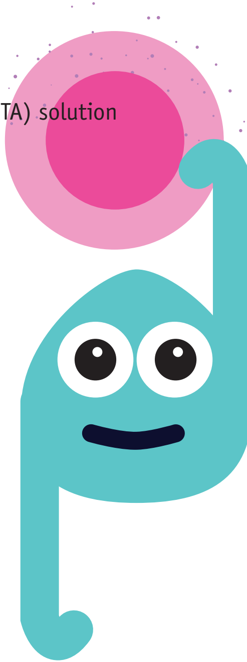
Device And Application Management In The Enterprise

When developing an enterprise app, you should always keep in mind that the hardware containing sensitive company data can get lost or stolen. There are now two approaches for securing devices, content and apps. Mobile Device Management (MDM) and Mobile Application Management (MAM).

MDM gives an enterprise ultimate control over a device, so when a device is lost, stolen or an employee leaves, taking the device, the enterprise can wipe the device and essentially stop the device from working. This approach is usually taken when an enterprise owns the device so all the data and apps on the device are owned by the company; any personal data stored on the device is stored at the employee‘s risk.

MAM enables an enterprise to adopt BYOD as it allows an enterprise to secure apps and content downloaded to a device without taking ultimate control away from the owner of the device. When an employee leaves a business, taking their device with them, the business can disable the enterprise apps and wipe any content downloaded to the device without affecting personal data, such as photos and consumer bought apps. Most MDM and MAM solutions are cross platform, supporting Apple, Android, Windows and BlackBerry devices, and this should always be taken into consideration when deciding upon an MDM or MAM provider.

Various security features are available through both these management solutions, including



- Device monitoring
- License control
- Distribution via an internal Over-The-Air (OTA) solution
- Software inventory
- Asset control
- Remote control
- Connection management
- Application support & distribution

Security measurements include

- Password protection
- On-device data encryption
- OTA data encryption
- Remotely lock devices
- Remotely wipe data
- Re-provision devices
- Back-up data on devices

Examples of MDM and MAM providers are:

- Airwatch[1]
- App47[2]
- Apperian[3]

[1] air-watch.com

[2] app47.com

[3] apperian.com

- Good[4]
- Microsoft[5]
- MobileIron[6]
- Mocana[7]

Mobile Enterprise Application Platforms (MEAP/MADP)

Usually, one key element of enterprise applications is data synchronization. The mobile devices have to be refreshed with relevant or up to date data from the company's servers and the updated or collected data has to be sent back. The scope of data access is determined by the job responsibilities of the user as well as by confidentiality policy. In any case synchronization has to be secure, as corporate data is one of your most prized assets. Furthermore, a company-wide accepted app will be multi-platform.
To compensate the shortcomings of the native SDKs as well as the common multi-platform solutions in these regards, you

[4] good.com
[5] microsoft.com/en-us/windows/windowsintune/explore.aspx
[6] mobileiron.com
[7] mocana.com

might want to consider evaluating Mobile Enterprise Application Platform or Mobile Application Development Platform (MEAP/MADP) solutions. MEAPs/MADPs are mobile development environments that provide the middleware and tools for developing, testing, deploying and managing enterprise apps running on multiple mobile platforms with various existing back-end datasources. Their aim is to simplify development and reduce development costs, where skills must be maintained for multiple platforms, tools and complexities, such as authentication and data synchronization.
Available solutions include:

- Amp Chroma by Antenna[8]
- IBM MobileFirst Platform[9]
- Kony KonyOne[10]
- SAP Mobile Platform[11]
- Spring Mobile Solutions[12]

Security In Enterprise Apps

One of the main functions of any IT department is to ensure that all aspects of the company infrastructure is secured against attack so that there are no data leaks and no data is compromised or stolen. As mobile devices are an extension of a company's IT infrastructure, all Enterprise apps must be designed to ensure that they cannot be used to illegally gain access to a company's internal network. As an Enterprise app writer you will usually be asked to conform to standards which a company has laid out in their security policies, so be

[8] www.antennasoftware.com
[9] www.ibm.com/mobilefirst/us/en/why-ibm-for-mobile/platform.html
[10] www.kony.com/products
[11] www.sap.com/mobileplatform
[12] www.springmobilesolutions.com

prepared to answer questions about securing your app, such as data encryption, network communication and dealing with jail broken or rooted devices.

Many MDM and MAM providers actually enhance app security, using techniques such as app wrapping or providing an SDK which app writers can use. These features, and regular updates of these platforms, allow an enterprise to lock down their apps remotely and also keep up with the changing security landscape without needing to invest as much time and effort into security.

Key points for securing Enterprise Apps

- If using an MDM or MAM provider ensure they have the security required security features to meet your enterprise standards.
- When storing any data on the device ensure it is encrypted
- When communicating with web services, always use https
- In addition to using https, when communicating with web services ensure you perform end point checking in both the app and the web service to confirm that the server/device you are connecting with is valid
- Always check that any settings your app is packaged with have a checksum to ensure that the values cannot be changed once shipped to the device
- Do not allow the app to run on jail broken or rooted devices.
- Have a method for disabling the app if the app detects that it has been compromised.
- Ensure that all use of encryption complies to export regulations and any laws relevant to the region/s the app is being used in.

BY Julian Harty

Mobile Analytics

Our apps are used remotely by people we may never meet. Mobile Analytics can help us to discover how your app is being used so we can improve future releases of the app. Over half of the top mobile apps already include mobile analytics[1].

We are spoilt for choice, at least 20 companies offer a smorgasbord of mobile analytics solutions with multiple flavours ranging from campaign tracking to improving software quality. BlackBerry even promote mobile analytics for both the older Java-based platform[2] and in version 10.2 of their newer platform[3]. Many include extra features such as crash reporting, customer and revenue tracking. Nearly half offer opensource implementations of their libraries, possibly to allay fears of how their libraries behave?[4]

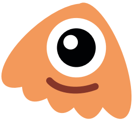
Read on for tips and guidance to help you understand how mobile analytics can help you discover how your app is being used. You will learn how to pick an appropriate solution and to implement it into your app.

[1] blog.velti.com/mobclix-index-the-when-where-what-of-apps *and* static.usenix.org/event/sec11/tech/slides/enck.pdf

[2] github.com/blackberry/WebWorks-Community-APIs/tree/master/Smartphone/Analytics

[3] devblog.blackberry.com/2013/08/discover-whats-coming-in-the-blackberry-10-2-sdk/

[4] readwrite.com/2013/12/05/why-mobile-developers-need-open-source-analytics-embedded-in-their-applications



Getting Started

Many providers of mobile analytics solutions offer a 'Getting Started' section where you learn how to take your first steps with their products. Examples include Flurry[5] and KISSmetrics[6]. You often need to register before you can usefully use the products, as many need configuring with a unique 'key' for your app.

Consider several of the potential solutions before committing to any of them. Read the documentation and example code to see how easily you can implement them into your app. And check the legal agreements, including privacy. Then pick at least one of them so you can experiment with implementing mobile analytics into your app. By integrating their code, you are likely to learn much more about what you would like to achieve by using mobile analytics in your app, and how mobile analytics works in practice.

For multi-platform apps you may want consistency across each platform; otherwise you may be trying to compare dissimilar, or even disparate, data sets; particularly if different mobile analytics solutions are used for the various platforms. Consider picking a common solution that supports every platform you want to launch your app on.

Two providers are well worth studying. Segment.io[7] abstracts a wide range of other mobile analytics offerings; and they provide their code as opensource at github.com/segmentio. They demonstrate ways to implement tracking in ways that reduce the effort needed to adapt to different analytics providers. Count.ly[8] provide opensource implementations of their server

[5] support.flurry.com

[6] support.kissmetrics.com/getting-started/overview

[7] segment.io

[8] count.ly

as well as of their client libraries and they encourage you to create a complete test environment to evaluate their product.

Be aware, some mobile analytics solution providers may use data reported by your app and they may provide and sell it to others. They may control the life of that data, which means they could make it inaccessible to you; conversely they may preserve and use it long after you have retired your app. Also if there is personally identifiable information in the data, there may be additional legal and privacy implications. So it is worth considering how third-parties will use and share the data reported via their software and APIs.

Deciding What To Measure

What would you like to measure, to understand how the app is being used? Some suggestions for you are:

- **Key usage events:** Usage; for instance of the new search option, or when they launch social networking from your app.
- **Business-centric events:** Any interaction of the user that generates revenue for you: How often do your users purchase the premium version of your app or other items offered within your software? When do they cancel orders or discard their shopping cart before checking out?
- **Usability metrics:** Where do your users get stuck in the usage flow? Do they quickly reach their goal when using your software?

Once you defined your main areas of interest, you will need to design the analytics measures, for instance, what data elements need to be reported.

Defining How To Measure

Create meaningful names for your interaction events so you can easily and correctly remember what they measure. For each event you want to record, decide what elements it needs to include. Consider how the data will be used once it has been gathered, for instance sketch out typical reports and graphs and map how the various data elements will be processed to generate each report and graph.

Also remember to address globalization issues such as the timestamp of each element. Does the app detect the time of an event according to the device's location, the device's settings or does it use a global time like UTC time[9]?

Some of the Mobile Analytics solutions will automatically record and report data elements to the server. It is worth checking what these elements are, how and when they are reported, and how they are formatted. Then you can decide whether you want to use and rely on these automatically-reported elements.

Custom event tags augment predefined events, and many mobile analytics solutions provide ways for your app to generate them. You may need to format the custom event messages. If so, pay attention to encoding of the elements and separators; for instance they may need to be URL encoded[10] when they are sent as REST messages[11] .

You may want to consider how often the app should report events to reduce the risk of flooding the available capacity of the analytics system, which might affect the reliability and accuracy of the delivered analytics data. One method to reduce the volumes of data processed by the analytics solutions is

[9] en.wikipedia.org/wiki/Coordinated_Universal_Time

[10] en.wikipedia.org/wiki/Percent-encoding

[11] msdn.microsoft.com/en-us/library/live/hh243648

called sampling. Adam Cassar published an interesting blog post on this topic at periscopix.co.uk/blog/should-you-be-worried-about-sampling/.

Adjusting Your Code

You may need to declare additional capabilities required in order for the mobile analytics to function correctly when integrated with your app.

For Android these are known as permissions. The analytics probably need Internet permissions so the events can be reported online, and location-centric permissions if the solution records the location of the phone. If your app already uses the permissions, you do not need to specify their use again.

For iOS, `UIRequiredDeviceCapabilities` tells iTunes and the App Store what device-related features the app needs. It is implemented as a dictionary where the elements are specified using keys. Keys include wifi, location-services and gps.

For Windows Phone, capabilities are used to decide what the app uses. Localytics has a quickstart guide online[12] that includes an example of setting the `ID_CAP_IDENTITY_DEVICE` capability.

Handling the results

There is a lag from when an app sends an analytics event to when the information is processed and made available to you. The lag, or latency, varies from near 'real-time' to many hours. You, and your business sponsors, need to decide how long you can afford to lag real-time events.

Some analytics solutions provide an API to allow you to access the data. This may give you more greater scope to create

[12] localytics.com/docs/windows-phone-7-integration/

custom reports. Several allow you to host the servers which provides you greater control of the data and how it is used.

To evaluate the quality of the results some organizations invest the extra effort of incorporating several analytics solutions into their app and cross-reference the results. However, two conflicting results do not make reconciliation easy, so it may be necessary to use three sets of results to diagnose the differences by triangulation[13].

If you have decided to work with KISSmetrics, check out their article on ways to test your metrics at support.kissmetrics.com/getting-started/testing-km.

Privacy

Remember to explain to the end-users that the app is designed to record and share information about how the app is being used, ideally in your terms and conditions. You may need or want to enable users to decide if they want their use of the app to be tracked. If so, make it easy for the user to control the settings; and consider providing the user a way to access the recorded data, delete it, or contact the analytics solution provider.

Providers of third-party libraries seem to have a range of attitudes to privacy. Some claim the privacy of users is paramount and stresses the importance of not tracking users. Google Analytics clearly prohibit tracking personally identifiable information in their terms of service[14]. Others provide examples, including snippets of source code, that demonstrates how to record clearly personally identifiable data.
For instance, KISSmetrics provides the following code snippet[15]:

[13] en.wikipedia.org/wiki/Triangulation_(social_science)

[14] google.com/analytics/terms/us.html

[15] support.kissmetrics.com/apis/objective-c

```
[[KISSMetricsAPI sharedAPI] identify:
@”name@email.com”];.
```

Mixpanel provides an example of how to update a user's People Analytics record[16], as well.

There are several places to learn more about privacy and ethics of working with data related to users, e.g.:

- **Jeff Northrop's blog post** on Mobile Analytics[17]
- Kord Davis' book **Ethics of Big Data**
- The retail industry has released a code of conduct with **useful general guidelines for mobile apps**[18]

Learn More

We hope this chapter has whetted your appetite to learn more about Mobile Analytics. Here are some places to start your ongoing research:

- **The Mobile Developer's Guide to the Parallel Universe,** a sister book to this one, covers Mobile Analytics from a marketing perspective[19].
- **TheNextWeb.com**[20] is another useful springboard into the topic from a developer's perspective.
- **Kontagent**[21] provides a wide range of whitepapers, on-demand webinars and other materials on mobile analytics, probably worth reading regardless of which implementation you choose.

[16] mixpanel.com/docs/people-analytics/android
[17] jnorthrop.me/2012/07/2/privacy-considerations-mixpanel-people-analytics/
[18] futureofprivacy.org/issues/smart-stores
[19] www.wipconnector.com
[20] thenextweb.com/dd/2013/08/11/9-tools-to-help-you-measure-mobile-analytics
[21] kontagent.com/resources

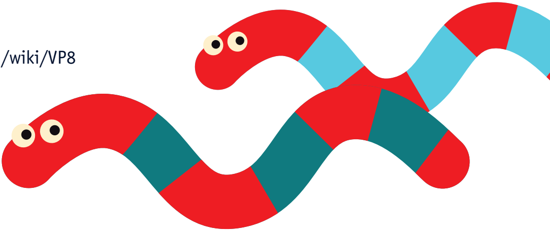
BY Michel Shuqair

Implementing Rich Media

“As many standards as handsets” is a truism when it comes to the list of supported media formats on mobile phones. In contrast to PCs, where most audio and video formats are supported or a codec can easily be installed to support one, mobiles are a different story. To allow optimization for screen size and bandwidth, specific mobile formats and protocols have been developed over the past few years. Small variations in resolution, bit rate, container, protocol or codec can easily cause playback to fail, so always test on real devices.

That said, most of today's smartphones support MP4 h.264 640x480 AAC-LC, however multiple variations are possible among handsets, even within one vendor or firmware version. New formats are still added every year, such as WebM/vp8[1], an open video standard running on Android 4+, however it lacks support from most software developers claiming it is just another variation of h.264. The trend is that screen resolutions become larger and larger, pushing Full HD TV resolutions onto mobile screens at the expense of consuming more battery and memory.

[1] en.wikipedia.org/wiki/VP8



Below are the recommended full screen formats for highest compatibility:

Container	mp4, 3gp, avi (BlackBerry only), wmv (Windows Phone + BB10 only)
Protocol	HTTP (progressive or download) or RTSP (streaming)
Video	H.264, H.263
Audio	AAC-LC, MP3, AAC+
Classic Resolutions	176x144 (Older phones), 320x240, 480x320 (J2ME)
Common Resolutions	480x800, 640x480 (Blackberry), 960x640 (iPhone), 1024x768 (iPad 1+2), 2048x1536 (iPad 3+4)
HD Resolutions	1280x720 (BB10, Samsung, Windows Phone 8), 1136x640 (iPhone 5)
Full HD / aka 1080p	1080x1920 (HTC One, Samsung Galaxy S4)
Quad HD / 2K aka 1440p	1440x2560 (Samsung Galaxy S5, Vivo xPlay)

For a detailed list of devices supporting HD resolutions visit Wikipedia[2]

[2] en.wikipedia.org/wiki/List_of_mobile_phones_with_HD_display

Streaming vs. Local Storage

There are two options to bring media content to mobile devices: Playing it locally or streaming it in real time from a server.

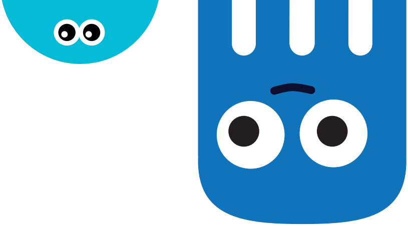
To stream content through relatively unstable mobile networks, a specific protocol called RTSP was developed that solves latency and buffering issues. Typical frame rates are 15 fps for MP4 and 25 fps for 3gp, with data rate up to 48 kbps for GPRS (audio only), 200 kbps for Edge, 300 kbps for 3G/UMTS/WCMDA and 500 kbps for HSDPA, Wi-Fi and 4G. HD-video starts at 2Mbps and is not recommended for streaming yet.

When targeting Windows Mobile/Phone, Windows Media Services[3] is preferred to support HTTP streaming. Android 3.0 upwards also supports HTTP streaming. Note that atomic hinting is required (see Progressive Download) and mp4 files are very strict in encoding (use H.264 15 fps AAC-LC 48khz stereo). Only HTC Android devices and Android 4.0 devices are less strict in streaming formats and will play much more encoding variations than other brands.

When streaming is not available on the phone, blocked by the carrier or you want to enable the user to display the media without establishing a connection each time, you can of course simply link and download the file. This is as easy as linking to a download on the regular web, but mobile phones might be stricter in checking for correct mime types. Use audio/3gp or video/3gp for 3gp files and video/mp4 for mp4 files.

Some handsets simply use the file extensions for data type detection, so when using a script — such as download.php — a well-known trick is to add a parameter such as `download.php?dummy=.3gp` to ensure correct processing of the media. Some phones cannot play 3gp audio without video,

[3] technet.microsoft.com/en-us/windowsserver/dd448620.aspx



but a workaround is to include an empty video track in the file or a still image of the album cover.

Depending on the extension and protocol, different players might handle the request. On some phones, like Android, multiple media players can be available and a popup is displayed to allow the user to select one. To improve the user experience, use embedded player objects with limited functionality.

Finally you can simply include media files in your mobile app as a resource. On Android devices pay attention to support media located on the SD-Cards (Android 3.1 and up) which requires the `android.permission.READ_EXTERNAL_STORAGE` permission.

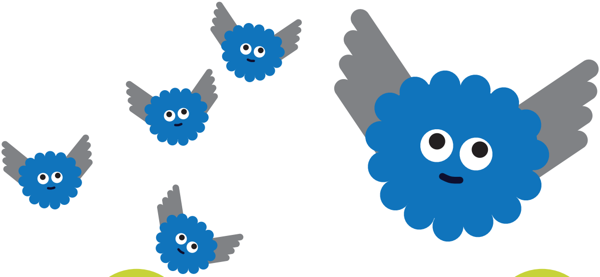
Progressive Download

To avoid configuring a streaming server, a good alternative is to offer progressive downloads, for which your media files can be served from any web server. To do this, you have to **hint** your files. Hinting is the process of marking several locations in the media, so a mobile player can start playing the file as soon as it has downloaded a small part of it (typically the first 15 seconds). With current handset capabiliyies and data networks this has become the prefered way to handle media files. Note: an mp3 file does not need hints, and cannot be hinted.

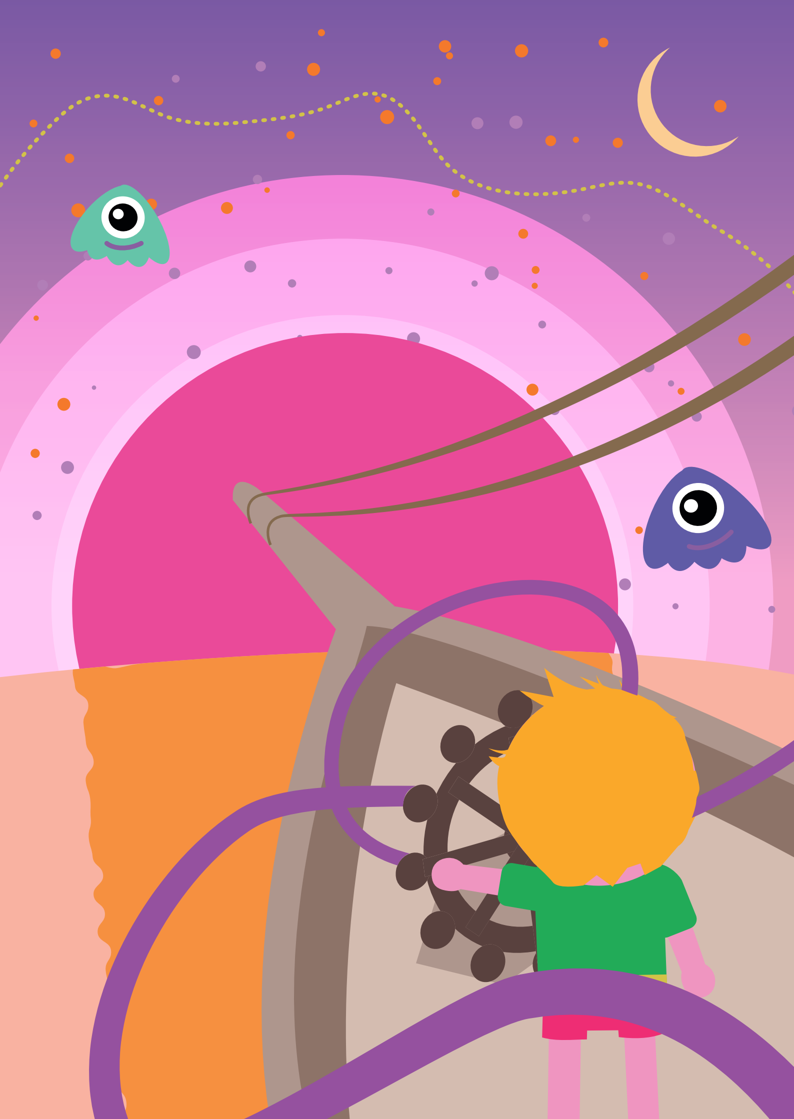
Possibly the most reliable open source hinting software available is Mp4box[4], however more and more media tools have built-in hinting options.

Media Converters

To convert a wide variety of existing media to mobile phone compatible formats FFMPEG is a must have (open source) media format converter. It can adjust the frame rate, bit rate and channels at the same time. Make sure you build or get the binary with H263, H264, AAC and AMR encoder support included. There are good converters available based on FFMPEG, such as “Super” from eRightSoft[5]. For MAC users, QuickTime pro (paid version) is a good alternative to encode and hint 3gp and mp4 files. If you are looking for a complete server solution with a Java / opensource background, check out Alembik[6].



[4] gpac.wp.institut-telecom.fr/mp4box/
[5] www.erightsoft.com/super
[6] www.alembik.sourceforge.net



BY Alex Jonsson

Implementing Location-Based Services

Location based services continue to be one of the hot areas for mobile applications.

Knowing a user's location means you can deliver them more relevant information; helping them to find a nearby veggie restaurant, review the local weather forecast, find friends at a convention, or find the most scenic local bike routes from information crowdsourced from other bikers. Getting location data is only half the story, providing the user with a meaningful representation is a key factor. This usually implies delivering a graphical representation overlaid with routes, points-of-interest and other relevant information. But don't overlook simplicity, a comprehensive list of resources sorted by proximity may be more appropriate than a scrollable, slow map view (particularly when the user may be roaming at five Euro per megabyte).

How To Obtain Positioning Data

Location-based applications can acquire location information from several sources; one of the phone's available network connections, GPS satellites, short range systems based on visible tags or local short range radio, or old-school by inputing data through the screen or keyboard.

- **Network positioning:** Each GSM or UMTS base station carries a unique ID, containing its country code, network id, five-digit Location Area and two-digit Routing Area. The coordinates of a base station can then be obtained by looking up the operator's declaration in a database. The resulting location is not particularly accurate, depending on the cell size (base station coverage): in urban areas densely packed cells should provide greater accuracy than the sparse cells in rural areas. Techniques, such as measuring the difference in the time-of-arrival of signals from several nearby base stations (known as multilateration) can help improve accuracy, however mobile operator's may charge for these premium network services. For phones with WiFi capabilities, details about wireless LAN access points can be used as an alternative or supplement to mobile network positioning. This technique is used by several companies, including Google.

- **GPS positioning**: An on-board GPS module (or an external one) typically gives you a 50% accuracy ranging from 5 to 50 meters, depending on the quality of the hardware and how many satellites the GPS module can see when making a fix. Accuracy is also affected by the terrain, canopy and wall materials; any of these may obscure the satellite signals: in cities, urban canyons created by clusters of tall buildings can distort the signal, giving false or inaccurate readings. Combining GPS with network positioning is increasingly common: Assisted GPS, or A-GPS, uses an intermediary, called an Assistance Server, in order to minimize the delay to the first GPS fix. The server uses orbital data, accurate network timing and network-side analysis of GPS information. However, A-GPS does not mean a more accurate position, but rather a faster result when the GPS is initially enabled, or when GPS satellite coverage is poor. This shortens the time needed for a location lock. Note: most A-GPS solutions require an active cellphone network connection.
- **Short range positioning:** Systems based on sensors; near field communication (NFC), Bluetooth (including Bluetooth

Low Energy) and other radio-based tag systems — use active or passive sensors in proximity to points of interest, such as exhibits in a museum or stores in a shopping mall. Low-tech solutions include bar codes and other visual tags (such as QR codes) that can be photographed and analyzed on a server or locally on the phone; such tags may contain an id from which a position can be looked up. The user can specify their position by selecting a location on a map, inputting an area code or a physical address. This option is used typically for applications on feature phones, which may lack other means of determining a location.

Lately, implementations of Bluetooth 4.0, including Apple's iBeacon, combine radio IDs with a received signal strength indicator (RSSI) as a means of triangulation. However, the accuracy of these techniques is yet to be determined, although placing Bluetooth tags on equipment, valuable assets, pets and alike for the purpose of crude positioning and creating realtime inventory will prove increasingly popular in 2014 with many new gadgets and services to follow.

Mapping Services

In general, a map service takes a position and a collection of metadata as input parameters and returns a bitmap or vector map, layered with contextual metadata. Vector data has several advantages over bitmaps: vector representations consumes less bandwidth and enables arbitrary zooming of any map view. However it requires more processing on the client side. Bitmaps are often provided in discrete zoom levels, each with a fixed magnification, named after its coordinates and zoom level.

Free maps, both served as bitmaps and vectors, include Open Street Map[1] or CloudMade[2]. Commercial maps include Garmin[3] and Microsoft's Bing resources[4] to name just two.

Some solutions, such as Google Maps[5], are free when your application is made available at no cost, but require you to obtain a map key. Other map services, such as Google's static maps, are limited to serving a number of tiles to a map key or IP address. Several of the sources share similar map formats and are thus interchangeable.

Implementing Location Support

Location API for Java ME offers detail such as the latitude and longitude position, the accuracy, response time, and altitude derived from the on-board GPS as well as speed based on performing consecutive readings.

With iOS there is integrated support for location but with restrictions on how the location data can be generated by the supporting functions, and what map sources can be used. Currently, there is also an on-going debate on how location data is recorded and stored on the iOS devices and how Apple are planning to use this data for their own purposes. Android developers also have access to high-level libraries and these devices are more liberal with the choice of map sources, although they default to Google's map APIs. On Windows Phone 7 and Windows 8 devices, note that the all new Maps API in Windows Phone 8 is not the same as Bing Maps available in

[1] wiki.openstreetmap.org/wiki/Software

[2] www.cloudmade.com

[3] garmin.com

[4] www.microsoft.com/maps/developers

[5] code.google.com/apis/maps

Windows Phone 7. The Bing Maps control is still supported in Windows Phone 8, but is deprecated.

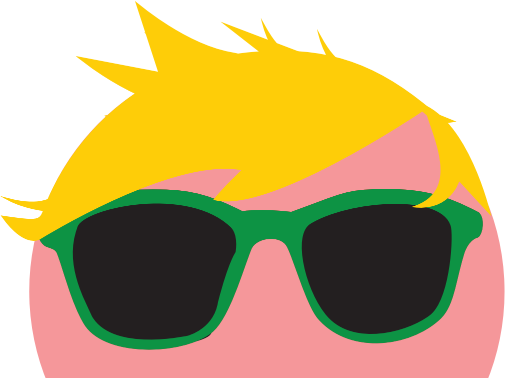
Ever since iOS 3.x and Android 2.0, Web app developers have been able to access geoinformation via the navigator.geoposition interface, for example calling `navigator.geolocation.getCurrentPosition(my_handle)`gives you the opportunity to fetch the `my_handle.coords.latitude` and `my_geo_handle.coords.latitude`, after getting permission from the user and if satellites are available. As an example, here's how an actual snippet example could look using JavaScript syntax:

```
function init_geolocation() {
// e.g. called from a ready() or onLoad function
   navigator.geolocation.getCurrentPosition(
   get_geolocation,process_errors);
}

function process_errors(error)  {
   switch(error.code)  {
     case error.PERMISSION_DENIED:
     alert(”user not sharing geolocation”);
     break;
     case error.POSITION_UNAVAILABLE:
   alert(”could not calculate current
position”);
     break;
     case error.TIMEOUT:
        alert(”retrieving position timeout”);
     break;
     default:
        alert(”misc. geolocation error”);
     break;
   }
}

function get_geolocation(position){
   alert(’Latitude: ’ + position.coords.latitude + ’
   Longitude: ’ + position.coords.longitude);
}
```

Instead of failing, an error can be combined with fallbacks to network lookups, entering zip codes or alike. Apart from the bare coordinates, geographical data often is presented with other information, available in a number of formats. One of the widely accepted standards is called geoRSS, and could look like this for a single point-of-interest:

```
<entry>
<title>Byviken's fortress</title>
<description>Swedish 1900-century army
installation, w. deep mote
</description>
<georss:point>18.425 59.401</georss:point>
</entry>
```

There are other formats for geodata, but the basic idea is similar; by harmonizing data streams and webservices, robust mashups can be created to run seamlessly in various user contexts. Other important formats for geoinformation include the Geography Markup Language (GML), an XML encoding specifically for the transport and storage of geographic information, and KML which is an elaborate geoformat used in Google Earth and related web services.



Tools For LBS Apps

Several companies provide developer-friendly tools and APIs as a value added service. Using these dramatically speeds up the development and deployment of location-aware services. Each tool normally focuses on one or a range of mobile platforms. Advertisement companies such as Admob offer developers stand-alone location aware advertisement programs, to better target their offerings. There are no map interfaces offers, just the coordinates sent and hopefully advertisements returned.

Below are more links to maps and location based service resources:

- **Android offline maps project:** *code.google.com/p/big-planet-tracks/*
- **BlackBerry:** *us.blackberry.com/developers/* (search for “map api”)
- **Garmin Mobile XT SDK:** *developer.garmin.com*
- **Google Map resources:** *developers.google.com/maps*
- **iBeacon:** *support.apple.com/kb/HT6048*
- **Nutiteq:** *www.nutiteq.com*
- **Nokia Maps:** *developer.here.net*
- **TeleAtlas:** *developerlink.teleatlas.com*
- **Windows Phone 8 Maps and Navigation:** *msdn.microsoft.com/en-us/library/windowsphone/develop/jj207045*

BY Richard Bloor

Near Field Communication (NFC)

While the first Near Field Communications (NFC)-enabled phone appeared in 2007 (the Nokia 6131 NFC) it has only been in the last few years the technology has gained popularity. The first significant moves came in 2010 with both Nokia and Samsung adding NFC to their smartphones so that today — with the exception of Apple — most Android, BlackBerry and Windows Phone devices carry an NFC chip.

The killer technology was supposed to be NFC's ability to securely replace a physical wallet. This was a convenience that people were ready to pay for, a study from Allied Market Research[1] predicts that the global mobile wallet market reaches $5,250 billion in 2020, growing at a compound annual growth rate of 127.5 percent from 2013 to 2020.

However, NFC has probably had its greatest impact to date by providing tap-and-share and tap-and-connect features.

Brief overview of NFC under the hood

NFC is an evolution of Radio Frequency Identification (RFID) technology, which has been in use since the early 90s[2]. NFC extends the capabilities of RFID and at the same time maintains backward compatibility with the older technology. For years RFID has been used mainly for tracking objects and for simple tap-to-pay payment applications.

NFC is an interface and protocol built on top of RFID that is targeted at mobile devices. The technology operates on unlicensed Industry Scientific Medical (ISM) radio bands at 13.56

[1] alliedmarketresearch.com/mobile-wallet-market

[2] nearfieldcommunication.org/history-nfc.html

MHz with a limit on the range of operation of 3 cm or less. It also provides mobile devices with a means of secure communication without any network configuration, an advantage over Bluetooth that requires pairing for communication to work.

NFC functions by electromagnetic induction: when the electromagnetic waves in a coil changes, a voltage is induced. From a hardware perspective, a series of high and low voltages represent bits. This is how packets are sent between devices. In 2006, the NFC Forum, which oversees the NFC ecosystem, defined the NFC Specifications. The Forum established a standard for the NFC Data Exchange Format (NDEF), a light-weight binary message format that encapsulates data. For instance, to encode a URI there is a 5 byte header followed by the URL, with a short URL consuming as little as 12 bytes. More information can be found online[3].

Advantages of NFC over Bluetooth[4]:

When compared to Bluetooth NFC offers the following advantages:

1. **Security**: Here NFC's short range of operation is an advantage, making communications hard to intercept. In addition, some smartphone implementations require the screen to be active and a PIN to be entered before access to NFC hardware is allowed.
2. **Low power consumption**: Due to the short range of operation, only a weak electromagnetic field has to be generated to read or write NFC tags.
3. **Quick pairing**: NFC devices are able to connect within a tenth of a second.

[3] www.developer.nokia.com/Community/Wiki/Understanding_NFC_Data_Exchange_Format_(NDEF)_messages *and* www.nfc-forum.org/specs/

[4] nearfieldcommunication.org/bluetooth.html

NFC modes of operation[5]

There are three modes of operation available in NFC: Reader/Writer mode, Peer-to-Peer mode and Card Emulation. This section describes these modes and their uses.

Reader/Writer mode

In this mode, the smartphone can read or write data from or to an NFC tag, with the phone's NFC chip generating an electromagnetic fields to interact with the NFC tags. NFC tags may be read only, so it is only possible to write to an NDEF message on a write enabled NFC tag.

Simple read applications may involve tags that hold general or app specific data. An example is smart posters where an NFC tag embedded in an advertising poster provides a link to a supplier's, product's or service's website. In this case the tag's data has a specific format and the phone knows, for example, to open a URL in the web browser. App specific uses are only limited by your imagination (and the amount of data an NFC tag can store). Read/write apps can be used to send messages or write tags for reading by apps.

[5] nfc.cc/technology/nfc

Peer-to-Peer mode

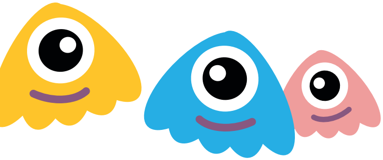
This is a major extension to RFID technology. In P2P mode, two smartphones are able to exchange small amounts of information, such as vCards, URLs or initiate a Bluetooth connection for large data transfers. Android Beam typically functions in P2P mode where a “user-invisible” pairing takes place with NFC, and data transfer takes place over the faster Bluetooth connection.

Using this mode apps can exchange data between two NFC-enabled smartphones. Many smartphones include apps for tasks such as Bluetooth pairing and the exchange of data, such as business cards. In your apps you could, for example, use this mode to exchange information on moves in a game.

Card Emulation Mode

In this mode, the smartphone acts as a passive NFC tag, which can be read by a retailers EFTPOS terminal. The key to this mode is the secure element, in which sensitive information is stored (more on secure elements shortly). Google Wallet and Microsoft Wallet are examples of applications that allow an NFC-equipped smartphone to be used for tap-to-pay application and rely on card emulation.

While more specialized, and usually requiring a relationship with the payment or service provider, this mode can be used for ticketing, payments, switching operations (such as opening a door), and replace physical cards (such as health insurance cards, credit cards, driving license, among others).



The Secure Element

At the heart of any card-emulated device lies the secure element (SE)[6], which contains the secure data (credit card information among others[7]). There are three locations where the SE can be stored:

1. On the SIM/UICC (via Single Wire Protocol, a specification that allows a connection between the SIM card and the NFC chip).
2. Inside the phone's NFC Chip.
3. On an SD Card.

As a side note, Google Wallet[8] stores credentials of a Google prepaid credit card only on the secure element of the phone's NFC chip. Only the Google credit card numbers are passed to merchants while actual credit card numbers are stored on secure Google servers. Microsoft Wallet on the other hand stores sensitive element on the secure element of the SIM card. According to Microsoft, such a method allows people to swap their wallets from one phone to another.

Around the world, a growing number of credit institutes and Mobile Network Operators (MNOs) are cooperating to deploy payment methods that use NFC. MasterCard PayPass and VISA PayWave are examples of such deployed solutions. NFC SIM cards are currently being issued by MNOs such as A1 (Austria), Orange France and China Mobile.

[6] mp-nfc.org/nfc_near_field_communication_architecture.html

[7] smartcardalliance.org

[8] google.com/wallet/faq.html

Current difficulties

NFC is an exciting technology that will bring about more economic transactions. However, before we see a widespread deployment of payment methods using NFC, a full understanding and cooperation among all banks, hardware manufacturers, MNOs and operating system developers is necessary.

Also, some major phone manufacturers, have not yet adopted NFC technology. Current Apple devices, for instance, do not support NFC. The company has decided to rely on the Passbook application which has a completely different mode of operation compared to NFC-centric applications.

Furthermore, the secure element has limited storage space. It is not clear how this space should be shared among all the players.

Lastly, because of the lack of accepted standards, some banks have deployed their own solutions and want to convince merchants to accept their new mode of payment.

NFC APIs

Android: From API Level 9 (Gingerbread 2.3), Android provides a set of high-level APIs that makes the use of NFC seamlessly easy. More information can be found on the Android developer page[9].

BlackBerry: The latest SDK provides high level APIs for NFC interaction. Example code for implementing an NFC tag reader and writer[10] and further information on how to use card emulation mode on BlackBerry[11] can be found on BlackBerry's websites.

Windows Phone 8: The Windows Phone 8.0 SDK includes the Proximity packages which provides a set of classes that provides the necessary APIs to enable P2P data sharing between Windows Phone 8 applications. It is also possible to transfer small packets of data from an Android device to a WP8 device and vice-versa. However, the implementation is still in its infancy and it is not possible to transfer large amount of data.

More implementation details can be found online[12]

[9] developer.android.com/reference/android/nfc/package-summary.html

[10] docs.blackberry.com/en/developers/deliverables/34480/Near_Field_Communication_1631111_11.jsp

[11] supportforums.blackberry.com/t5/Java-Development/NFC-Card-Emulation-Primer/ta-p/1596893

[12] msdn.microsoft.com/en-us/library/windowsphone/develop/jj207060

BY Bob Heubel

Implementing Haptic Vibration

Haptic Vibration Design Considerations

Why should you use Haptic vibration effects to your app? Your app will run just as well without the tactile feedback, right? Yes, possibly, but you will lose the one sensory element that makes your virtual environment more realistic and compelling. Margaret Atwood once wrote, “Touch comes before sight, before speech. It is the first language and the last, and it always tells the truth.” It is this sense of touch feedback, more than sight or hearing that teaches us what to expect from interactions in the real world.

It is our experiences in the real world that define a user's expectiations in the virtual world found in your apps.

Take a button press as an example. A real button press is a very tactile experience. It has a beginning and an end. You feel a satisfying confirmation of your action. In comparision, a virtual button press feels hollow without a Haptic effect to simulate that same confirmation of action. More than this, without a Haptic tactile confirmation you force the user to rely on visual/audio cues that are more stressful to process than simply using our sense of touch.

Haptic feedback is even more important in mobile video games. We know this from our experience with console games. Remember when the Sony PS3 launched without "DualShock" rumble pad motors? Gamers voiced their dissatisfaction and shortly after Sony brought the DualShock rumble feedback to the PS3. The same Haptic feedback satisfaction applies to mobile games. Using Haptic effects in your games will help to give your mobile users what they already expect from console platforms. And if you design well, your games will feel more realistic and compelling to your users.

When designing a Haptic experience, keep in mind the ultimate experience of the user. Spend some time planning before starting your Haptic implementation. Once the project is defined and taking shape in your mind, consider the following guidelines:

- Simple sensations are often the most effective. It is sometimes surprising to realize that something like a very simple Pop or Click sensation can enhance menu interactions and increase user confidence within the application.
- Sensations synchronized with audio and visual events, like a simple button click event, make the whole greater than the sum of its parts. Seeing, hearing, AND feeling an object or activity promotes sensory harmony in a way just seeing and hearing alone cannot.
- It is bad to annoy the user. Poorly chosen or designed touch sensations can be annoying and counterproductive. While a high-pitched buzz may be very effective as part of an alert, continuous reoccurring buzzing will eventually cause a user to leave an application annoyed.
- It is bad to confuse and overwhelm the user. Just as too many beautiful sounds played simultaneously become a cacophony, too many compelling touch sensations played together or too close to each other in time and space can become confusing and overwhelming.
- Familiarity eases the user experience. Haptic effects can relay important information to a user, which might not be available or practical to provide through graphics or sound. Standardization and consistency are important. Limiting

the Haptic effect language to a manageable, reused set of sensations makes the user's learning process easier because there are fewer Haptic effects to recognize.

Nearly all mobile platforms allow for some form of haptic vibration feedback control. This section will be your resource for understanding the classes and methods between these platforms.

Android

Android is unique for vibration control. It provides native support and has more vibration control than iOS. Furthermore, there are ways to extend this Android vibration control for developers so they can create more console-like X-Box or PlayStation feedback experiences. But whether you use the basic or extended methods below, please note that a user may have enabled haptic effects for better accessibility. For instance the KickBack Accessibility Service provides haptic feedback and is available as part of the eyes-free[1] open source project. So, consider how haptic effects generated by your application may interact with, or disturb, such services.

For basic vibration control in Android, you must first grant permission `android.permission.VIBRATE` to allow your application to vibrate. Next you use the `Vibrator` Class[2] with `getSystemService` function and the `Context.Vibrator_Service` to call the vibration service.

Within the above method you can vary the duration of the vibration event in milliseconds and set vibration patterns by setting up as many of start and sleep events as you like. The basic Android vibrate control method only lets you control the duration of vibration events.

[1] code.google.com/p/eyes-free

[2] developer.android.com/index.html#q=Vibrator

Extended Android Vibration Control

Because the Android platform is open source, there is at least one company that offers free methods to extend Android's vibration control. Immersion Corporation's Haptic SDK[3] allows full vibration motor control of duration, amplitude and pulsing frequency with a library of 124 pre-defined Haptic vibration feedback effects. With this type of control, application developers have the capability of designing vibration effects rivaling console gaming vibration experiences while also conserving battery power.

For Android developers using Unity3D, Marmalade or YoYo Games' GameMaker Studio, Immersion offers this same extended method through plugin support, also found on their main SDK webpage. Developers interested in this extended vibration control can download the company's Quick Start Guide[4] that explains how to set-up your Eclipse environment and use the `Launcher` method to call Haptic effects from the pre-defined library. On Google Play you can also download a few free showcase apps to feel the pre-designed Haptic effects before using them in code. One app is called "Haptic Muse" and the other "Haptic Effect Preview" app.

One other bonus of the pre-designed effect library is a hardware abstraction layer compensating for differences in motor types between mobile devices, so the feeling you create remains consistent.

BlackBerry 10

BlackBerry gives you the same basic on/off vibration control that Android does, but without an extended method. For BlackBerry you use the `VibrationController`

[3] www.immersion.com/haptic/sdk

[4] www.immersion.com/haptic/guide

Class[5] with `startVibrate(int duration)` and `stopVibrate(int duration)`

In addition, Blackberry now has an `intensity` (1-100) parameter for developers to play with.

Firefox OS

Mozilla's Firefox Mobile uses a web view API that allows for single or patterned vibration arrays in the form of a `window.navigator.vibrate` function call[6] with millisecond duration parameters. Patterned vibration arrays use alternating on and off vibration durations. For example, the patterned array `window.navigator.vibrate()` would playback vibration for 1 second, pause for 200 milliseconds and then play vibration for an additional 2 seconds. Zero value duration or empty array vibration events cancel any currently running vibration.

iOS

The iOS platform gives developers little vibration control directly from their mobile devices. The iOS vibration method below applies to iPhones only. iPads and iPods currently do not support on-device vibration playback.

Use the `SysSoundViewController` Class[7] with the `AudioServicesPlaySystemSound` function and the `kSystemSoundID_Vibrate` constant to trigger vibration on your iPhone device. Calling this constant will turn your motor on for a set duration of 400 milliseconds. Additionally, there is an iOS AudioToolbox framework using `AudioServicesPlaySystemSound` method[8] to call a

[5] developer.blackberry.com/search/?search=VibrationController

[6] developer.mozilla.org/en-US/docs/WebAPI/Vibration

[7] developer.apple.com/search/index.php?q=SysSoundViewController

[8] developer.apple.com/search/index.php?q=AudioServicesPlaySystemSound

kSystemSoundID_Vibrate constant that will also trigger a 400 millisecond vibration.

Supplemental iOS vibration capability may come from mobile controller peripheral makers like Logitech (PowerShell) and BDA (Moga Ace Power) now that Apple has officially added a "Made for iPhone/iPad/iPod" (MFi) peripheral architecture to the iOS platform.

PhoneGap

PhoneGap is a web view API owned by Adobe Systems (see the cross-platform chapter to learn more). The PhoneGap vibrate method uses the navigator.notification.vibrate function call[9] with millisecond duration parameters. Since PhoneGap works across a number of operating systems, you will need to be sure to set vibrate permissions according to each platform.

Additionally, on iOS the PhoneGap duration parameter is ignored and will vibrate using the iOS constant.

Tizen

Tizen, the fairly new OS supported by Samsung and Intel, has two primary development environments[10]; one for native app development and one for web app development using the WC3 Vibration API.

— Native app developers use the Vibrator Class[11] start and stop methods for vibration control. The developer can specify intensity and pulsing, by setting values in a pattern array that is passed as an argument to start(). Each element in the pattern array has two fields. The duration

[9] docs.phonegap.com/en/2.1.0/cordoba_notification_notification.md.html

[10] developer.tizen.org/documentation/dev-guide

[11] developer.tizen.org/help/topic/org.tizen.native.appprogramming/html/guide/system/vibrator_mgmt.htm

field says how long the vibration will play in milliseconds; the intensity field sets how strong the vibration will be. Intensity values are 1-100 (min to max), -1 for "System Default," 0 for "Silent." If you wish to render more than one haptic effect, you add additional elements beyond the first in the pattern array. The `stop()` method is only needed if you wish to terminate an effect prematurely (e.g., because a phone call comes in).

— Web app developers use the `navigator.vibrate()` function call.[12] The method has two variations: one that takes a single duration argument in milliseconds `singleVibration(int duration)`, and one that takes an array of duration times. The second function allows control of pulsing effects `patternVibration(int duration on, int duration off)`, but there is no support for the intensity control found in the native environment.

Web/ HTML5

Like Tizen, HTML5 vibration control for web app development relies on using the WC3 Vibration API. The same function calls apply.

Windows 8

Windows offers a basic method for vibration control, but no extended method at this time. Use the `VibrateController` Class[13]with `Start` & `Stop` Methods to vibrate your device motor from 0-5 seconds. For finer duration control you will need to set a `TimeSpan` method in order to use millisecond values.

The Windows 8 Class listed above is the same as the previous Windows 7 Class.

BY Mostafa Akbari

Implementing Augmented Reality

Augmented Reality (AR) is something you may have read and heard a lot about over the last few years. It seems to be everywhere: AR games, AR catalogues, AR posters, AR tattoos, AR in cars, AR apps, AR advertising and many more. According to ‘Research and Markets’ the AR market is growing exponentially: by 2020 the number of consumers using AR apps will climb to one billion. During last year's TED Conference Tomi Ahonen declared AR to be the 8th mass medium. He stated that, with the spread of smart glasses, the demand for AR apps will increase remarkably. This opinion is supported by studies from various companies, such as the Semico report 'Augmented Reality: Envision a More Intelligent World'[1] that predict revenues related to this technology will approach $600 billion by 2016. Additionally, ‘Research and Markets’ predicts an annual growth rate of 95.35%, with revenues of 5,155 million USD by 2016 generated by AR apps[2].

But what exactly is AR? It is a technology that enhances the real world by adding virtual elements. Visual augmented reality is the most common form, usually involving three-dimensional objects or a two-dimensional overlay containing text or images added to the user's real-life view.

[1] www.semico.com/content/augmented-reality-envision-more-intelligent-world

[2] www.researchandmarkets.com/reports/1963197

AR Scenarios in Mobile Context

Mobile Augmented Reality is used in situations where additional information can increase the efficiency, effectiveness and joy of use while on the move. Mobile AR is especially suitable for applications where people are confronted with a lot of data and the need to process it in a short period of time. By integrating information into a live-stream visible through the display of the mobile device, the user's attention no longer needs to switch between the mobile device and the environment. Mobile Augmented Reality solutions have a range of applications, such as in enterprises, marketing, education or entertainment, here are some examples:

- **Augmented Reality Browsers:** AR-browsers, such as Layar[3] and Wikitude[4], superimpose location-based data over a live view of the real-world. The user's location is determined by GPS and information about nearby points of interest (POIs) is displayed on the screen of the smartphone. Wikitude additionally offers a connection to Wikipedia for more information. Blippar[5] is another example that uses AR to bring adverts to life.
- **IKEA Catalogue App:** Using the IKEA AR App the user scans specially marked content from the Ikea catalogue with their mobile phone to display additional product information, such as customization options and further product images. The app relies on image recognition software from Metaio[6] rather than using the more common QR codes. The app also enables the user to place virtual furniture in their home or office.

[3] layar.com/products/app/

[4] wikitude.com/app/

[5] blippar.com

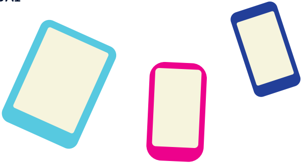
[6] metaio.com

- **AR Jump n’ Run:** AR Jump n’ Run is a location based app for Android smartphones developed with DroidAR[7]. The game can be played both indoors and outdoors using GPS, step recognition, or both. The player experiences the game by walking through a virtually-enriched world with the device acting as the viewport. They try to collect or avoid 3D items that are helpful or damaging. The game also features an in-game map editor that enables players to create and edit maps, for example, by placing new 3D items.
- **Ingress:** “The world around you is not what it seems. Our future is at stake and you must choose a side.” This is the introduction of the AR game Ingress[8] developed by Niantic Labs, an internal startup from Google. The game is based on portals placed all over the world. Users join a team and then the mission is to destroy portals of the opponents' team and of course to protect their own.
- **Audi eKurzinfo:** Also built by Metaio, Audi is offering a manual for their A3 car in the form of an augmented reality app for iOS[9]. It covers more than 300 different elements of the car – all of which are easy to identify with the phone's camera. If for example, a warning symbol comes on inside the vehicle, the user can scan it with the app to find out how to deal with the problem.

[7] code.google.com/p/droidar/

[8] www.ingress.com

[9] youtube.com/watch?v=TDTWOlbWBXI



AR Developing 101

This section provides a general introduction to the key concepts needed to create AR applications. Once you understand these concepts you should be able to choose the right framework for your project.

The Real And The Virtual World

AR involves placing artificial objects into the real world by using a virtual layer. This virtual layer, or virtual world, and its coordinate system is tied to the real world by reference points. This reference can be a GPS position, a visual marker, or an image. You can for example place a virtual object at a specific GPS location, then the object is bound to these coordinates. With visual markers or images you tie the augmentation to an real world object. For example, a game printed on a cornflakes box could be played anywhere in the world as long as you have the cornflakes box as a visual reference point.

The reference points are determined by tracking. Common tracking technologies include GPS, optical sensors, compass, accelerometer, gyroscope and step detection. Other concepts are marker-based, markerless and hybrid tracking. Markers are a simple, inexpensive and accurate solution to identify objects. By processing the image of the marker and the actual size of it an image processing system calculates the distance between the device and the marker. Markerless tracking on the other hand uses natural features instead of markers. These features can be two-dimensional patterns (e.g. advertisement posters) or even three-dimensional surroundings (e.g. buildings). The recorded images are compared with a database to detect a match. This requires complex algorithms and high processing power. Hybrid tracking technology combines the different sources of position data, such as GPS, 3D feature detection, marker detection and step detection. This enables higher degrees of positioning and motion detection accuracy.

Mapping Between The Two Worlds

For location-based AR apps a mutual mapping between the constantly changing position in the real world and the position in the virtual world is needed. Rendering engines, such as OpenGL, reduce the complexity of this process and increase speed and real-time accuracy. The engine also takes care of matching the camera's virtual and real-world position and guarantees fluent camera transitions when these positions change. To make your life easier as a developer you can use extended engines like gameplay[10] or Unity[11]

The core element, the camera data, can also be passed to other components. This means that, for example, a collision component can easily calculate the distance between virtual objects and the image captured by the camera.

Creating Virtual Elements

Virtual items are represented as 3D or 2D objects that might exhibit various behavior: Some objects might be collectible, others follow the user, or they may remain static and allow no interaction at all. With software such as Metaio Creator, Layar Creator or Wikitude Studio it is now possible to create own AR content very easily. With a web application you can place augmentations on a target image. When you have chosen and placed all augmentations you can export your project as an AR app. With this app the augmentations will appear when the target image is recognized by the app.

The tools from metaio, Wikitude and Layar only provide for augmentation of images. If you want to create other Augment-

[10] www.gameplay3d.org

[11] unity3d.com

ed Reality scenarios you need a technique called Simultaneous Localization And Mapping (SLAM). This is a very powerful Computer Vision technique based on features that are uniquely identifiable areas in an image. The 3D features are connected to the real-world through which the device moves. Together these features build a 3D cloud. This 3D cloud can be used for real-world object detection or shape detection, as well as the augmentation of the object. Another part of Computer Vision is image recognition. The recognized image position can be used as the reference position for the virtual world (compare to the example with the cornflakes box). SLAM and Computer Vision technique require immense computational powers to run in real time. The limitations and the capabilities of the hardware have to be taken into account.

Combining Application Layers

An important element of visual AR is to place something over parts of the image received from the camera. Depending on your app requirements you will want to place 2D or 3D graphics and use the respective APIs. A 2D overlay is usually sufficient for simple POI browsers.

We strongly recommend using a rendering framework such as OpenGL, rather than re-inventing the wheel. Such frameworks use the user's position, device orientation, other sensor information or the image analysis data and translate it to display your content accordingly. If the rendering component is decoupled from the rest of the app's code it can be exchanged in future, for instance to switch to more advanced rendering solutions.

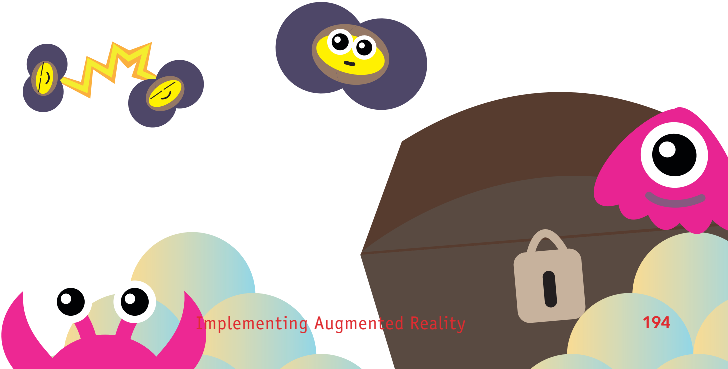
Maybe you want to add things like a small radar UI to visualize the position of the virtual objects or some simple buttons to interact with the virtual world. Then make sure to stick to platform-specific patterns and designs. Furthermore, implement these elements on a separate layer to remain flexible.

Composing A Virtual World With Multiple Layers

The best practice for composing the virtual world is to use a tree structure and place virtual objects in different layers: One layer for fixed “background” objects which do not need to be updated and which do not interact with the user and other layers for movable objects or UI elements.

You should only update and render objects close to the user and make use of the quad tree structure. A quad tree is a data-structure that enables your app to efficiently obtain all objects in a bounding box. Depending on the device hardware a different radius of the view can be used to ensure the best app performance.

We also recommend using an update mechanism that triggers the updates and calls an update method, for example, every 20 ms. The nodes in the object tree individually decide to which children these update calls should be forwarded. A quad tree for example will only update the objects close to the user to keep the update procedure efficient. A basic list structure would update all its children and is more suitable for elements that do not have a virtual position (like logical game stats) or which have to be updated together. The exact object composition concept depends on the application scenario and cannot be defined in an universal way.



Augmented Reality SDKs

ARLab

ARLab provides separate SDKs for the different tracking methods. They offer one for 2D location and sensor based content like POIs, similar to the first versions of Layar and Wikitude, as well as SDKs for image matching, image tracking, object tracking, creating virtual buttons and a 3D engine SDK. It should be noted that the SDK licences have to be purchased separately for iOS and Android and each SDK has its own cost, ranging from 99€ to 299€.
www.arlab.com

ARPA SDK

The ARPA SDK provides tracking of images, markers, GPS and offers a module for face tracking as well. It provides a Unity Extension so that interactive AR scenarios are possible. There is no concrete information available about the costs and all applications using the SDK have to be licensed separately. Another important restriction to mention is that it only runs on Android devices with Android 4.0 or higher. So far we have seen good performances on AR Apps built by the company itself for Desktop and the iPad.
www.arpa-solutions.net/ARPA_SDK

D'Fusion

The D'Fusion SDK by Total Immersion offers 2D feature extraction to augment images and is very similar to other SDKs such as those from metaio, Qualcomm and Layar. Within some SDK bundles you get face tracking and movement detection libraries. One advantage over most other SDKs is the free license.
www.t-immersion.com

DroidAR

DroidAR SDK is built for the development of interactive location-based and marker based AR Android applications. It enables 3D position tracking via SLAM and 3D object detection and reconstruction. For tracking it uses a combination of location based tracking, marker tracking and a step detection algorithm to cope with large scale indoor scenarios. DroidAR v1 is freely extensible and one of the few completely open source AR SDKs. DroidAR is backed by a developer community of about 6,000 people and has been used in various apps such as the CHIO app 2013.
https://github.com/bitstars/droidar

Layar

In the beginning Layar was a pure location-based AR platform with layers that faded in, with very limited interaction possibilities for the user. Now, Layar has changed its focus from 2D feature extraction to augment images. The new Vision SDK and the Layar creator are designed for extending print media content. The Layar Player SDK for iOS makes it possible to build Layar Apps that do not need the Layar browser. The Vision SDK can be licensed for €2500/year, the Geo SDK for €7500/year. The SDKs come with a watermark, which can be removed for an extra fee of €7500/year.
layar.com

metaio

metaio GmbH was the first mover in the Augmented Reality market. The metaio SDKs main focus is on the augmentation of 2D images, for example, in magazine or catalogues. The pro version of the SDK also supports the recognition of 3D objects such as a product package, a statue or the facade of a building. The Metaio SDK comes standard with a powerful 3D rendering

engine. It is possible to use the SDK for free by always displaying a metaio watermark to the user. The watermark free basic version costs €2,950 and the pro version without watermarks €4,950. The SDK has been downloaded over 50,000 times and is used in over 1,000 apps.
www.metaio.com

PointCloud SDK

The PointCloud SDK specializes in small scale SLAM tracking for tabletop games and other small space scenarios. The Unity plug-in is available as a beta. The SDK is still in an early state but seems promising. The SDK can be used for free, then the PointCloud logo has to be permanently displayed in the app. If you want to remove the watermark you can contact 13th Lab for more details. The iOS version can be downloaded from the PointCloud website and on request the Android alpha can also be accessed.
developer.pointcloud.io/sdk

Vuforia

The Vuforia SDK from Qualcomm can be used for free and has a Unity extension, hence, it is a good choice for image detection and augmentation scenarios. It only supports 2D image detection and recognizes locally stored special images. Multiple 2D markers can be composed to one marker but arbitrary 3D objects cannot yet be detected. Moreover, it cannot be used to create location based apps which use movement data or geo-references. There are 80,000 registered developers and the Vuforia SDK has been downloaded over 100,000 times. A lot of big commercial projects are build with the Vuforia SDK.
www.vuforia.com

Wikitude SDK

Wikitude is a location-based Point of Interest (POI) Browser. A classic use case is where Wikitude offers an adequate solution is POI search (for example answering the question "Where is the nearest post office?"). Wikitude is designed for static content and does not enable interactive scenarios. If you use newer versions to build your own Wikitude browser app, you can also use HTML5, Titanium or Phonegap. Even BlackBerry 10 is supported. Wikitude is available in 32 languages and has over 13 million users worldwide. Besides the AR Browser, Wikitude offers with Wikitude Studio a tool to augment images. This works quite similar to the Metaio Creator and the Layar Creator.

www.wikitude.com/developer

BY Dean Churchill

Application Security

Readers of this guide know how widespread smart mobile devices have become and how useful mobile apps can be. Mobile devices are also much more personal than personal computers ever have been. People wake up with their phones, stay close to them all day, and sleep next to them at night. Over time they become our trusted 'partners'.

Many of these apps take advantage of this closeness and trust. For instance, your phone might be treated as part of the authentication for accessing your bank account. Or your tablet could get direct access to the online movies you have bought. The device might even store a wallet of real money for making payments with Near Field Communications (NFC), or virtual money like Bitcoins.

Mobile apps are attracting the attention of hackers and thieves whose interests extend well beyond getting a 99 cent app for free. Kaspersky Lab reports that in June 2013, they counted 100,000 malicious code samples in Android apps which consisted of 629 malware families[1]. The historical network and endpoint based defenses (like anti-virus tools) are not enough. Embedding security into the mobile application is critical.

The architecture of mobile apps continues to evolve. Some apps are native-only, and require distinctly different code bases for each different mobile operating system. Some are web-views, little more than a web site url wrapped in an icon. Others are hybrids, a combination of native app functionality with web views. Most mobile apps need to connect with backend services using web technologies to fetch or update information. Like web apps, classic application security needs to be used with mobile apps. Input needs to be validated for

[1] securelist.com/en/analysis/204792299/IT_Threat_Evolution_Q2_2013#16

size, type, and values allowed. Error handling needs to provide useful error messages that do not leak sensitive information. Penetration testing of applications is needed to assure that identification, authentication and authorization controls cannot be bypassed. Storage on the devices needs to be inspected and tested to assure that sensitive data and encryption keys are not stored in plain text. Log files must not capture passwords or other sensitive information. SSL configurations need to be tested.

Users want to use your applications safely; they do not want unwelcome surprises. Their mobile phone may expose them to increased vulnerabilities, for instance potentially their location could be tracked using an inbuilt GPS. The camera and microphone could be used to capture information they prefer to keep private, and so on. Applications can also be written to access sensitive information such as their contacts. And malicous applications can covertly make phone calls and send SMS messages to expensive numbers.

The application developer may be concerned about his/ her reputation, loss of revenue, and loss of intellectual property. Corporations want to protect business data which users may access from their mobile device, possibly using your application. Can their data be kept separate and secure from whatever else the user has installed?

Threats to Your Applications

On some platforms (iOS and Android in particular), disabling the built-in signature checks is a fairly common practice. You need to consider whether or not it would matter to you if someone could modify your code and run it on a jail-broken or rooted device. An obvious concern would be the removal of a license check, which could lead to your app being stolen and used for free. A less obvious, but more serious, threat is the insertion of malicious code (malware)that could steal your users' data, or inject illicit content and destroy your brand's reputation.

Reverse-engineering your app can give a hacker access to a lot of sensitive data, such as the cryptographic keys for DRM-protected movies, the secret protocol for talking to your online game server, or the way to access credits stored on the phone for your mobile payment system. It only takes one hacker and one jail-broken phone to exploit any of these threats.

If your application handles real money or valuable content you need to take every feasible step to protect it from Man-At-The-End (MATE) attacks. And if you are implementing a DRM standard you will have to follow robustness rules that make self-protection mandatory.

Protecting Your Application

Hiding the Map of Your Code

Some mobile platforms are programmed using managed code (Java or .NET), comprised of byte code executed by a virtual machine rather than directly on the CPU. The binary formats for these platforms include metadata that lays out the class hierarchy and gives the name and type of every class, variable, method and parameter. Metadata helps the virtual machine to implement some of the language features (e.g. reflection). However, metadata is also very helpful to a hacker trying to reverse engineer the code. Decompiler programs, freely available, regenerate the source code from the byte code, and make reverse engineering easy.

The Android platform has the option of using the Java Native Interface (JNI) to access functions written in C and compiled as native code. Native code is much more difficult to reverse engineer than Java and is recommended for any part of the application where security is of prime importance.

"gcc" is the compiler normally used to build native code for Android, its twin-sister "clang" is used for iOS. The default setting for these compilers is to prepare every function to be exported from a shared object, and add it to the dynamic symbol table in the binary. The dynamic symbol table is different to the symbol table used for debugging and is much harder to strip after compilation. Dumping the dynamic symbols can give a hacker a very helpful index of every function in the native code. Using the `–f visibility` compiler switch[2] correctly is an easy way to make it harder to understand the code.

Compiled Objective-C code contains machine code and a lot of metadata which can provide an attacker with information about names and the call structure of the application. Currently, there are tools and scripts to read this metadata and guide hackers, but there are no tools to hide it. The most common way to build a GUI for iOS is by using Objective-C, but the most secure approach is to minimize its use and switch to plain C or C++ for everything beyond the GUI.

Hiding Control-Flow

Even if all the names are hidden, a good hacker can still figure out how the software works. Commercial managed-code protection tools are able to deliberately obfuscate the path through the code by re-coding operations and breaking up blocks of instructions, which makes de-compilation much more difficult. With a good protection tool in place, an attempt to de-compile a protected binary will end in either a crashed de-compiler or invalid source code.

De-compiling native code is more difficult but can still be done. Even without a tool, it does not take much practice to be

[2] gcc.gnu.org/wiki/Visibility

able to follow the control-flow in the assembler code generated by a compiler. Applications with a strong security requirement will need an obfuscation tool for the native code as well as the managed code.

Protecting Network Communications

Network communications are also vulnerable, particularly when apps can be installed in emulators or simulators, where network analyzers are freely available and able to monitor and intercept network traffic. Consider protecting sensitive network communications, for instance by using SSL for HTTP traffic between your app and servers. SSL protects data in transit, but only from the app to the web server where the SSL session ends. Even then MATE attacks, especially over WiFi networks, may disclose sensitive data. One way to step up transport security is to use asymmetric encryption between the server and the mobile app (using public/private key pairs) to provide end-to-end security. For sensitive corporate data and applications, install Virtual Private Network (VPN) servers, and install VPN clients on the mobile devices. VPNs generally provide strong authentication, and secure transport above and beyond SSL.

Protect Against Tampering

You can protect the code base further by actively detecting attempts to tamper with the application and respond to those attacks. Cryptography code should always use standard, relatively secure cipher algorithms (e.g. AES, RSA, ECC), but what happens if an attacker can find the encryption keys in your binary or in memory at runtime? That might result in the attacker unlocking the door to something valuable. Even if you use public key cryptography and only half of the key-pair is exposed, you still need to consider what would happen if an attacker swapped that key for one where he already knew the other half. You need a technique to detect when your code has been tampered. Tools are available that encrypt/decrypt code on the fly, run checksums against the code to detect tampering, and react when the code has changed.

Communications can be monitored and hacked between the mobile app and backend services. Even when using SSL, an intercepting web proxy (like Paros) can be setup on a WiFi connection that will inspect SSL traffic. Attackers can then tamper with the data in transit, for profit or fun. So if really sensitive data is being sent via HTTPS, consider encrypting/decrypting application data between the mobile application and the server, so that network sniffers will only ever see encrypted data.

Protecting Cryptographic Algorithms

An active anti-tampering tool can help detect or prevent some attacks on crypto keys, but it will not allow the keys to remain hidden permanently. White-box cryptography aims to implement the standard cipher algorithms in a way that allows the keys to remain hidden. Some versions of white-box cryptography use complex mathematical approaches to obtain the same numerical results in a way that is difficult to reverse engineer. Others embed keys into look-up tables and state machines that are difficult to reverse engineer. White-box cryptography will definitely be needed if you are going to write DRM code or need highly-secure data storage.

Best Practices

Do Not Store Secrets or Private Info

Minimize the amount of sensitive information stored on the device. Do not store credentials or encryption keys, unless secure storage is used protected by a complex password.

Instead, store authentication tokens that have limited lifetime and functionality.

Log files are useful for diagnosing system errors and tracking the use of applications. But be careful not to violate the privacy of users by storing location information, or logging personally identifiable information of the users. Some countries have laws restricting the tracking information that can be collected – so be sure to check the laws in the countries in which your app will be used.

Do not print stack traces or system diagnostics that hackers can leverage to penetrate further.

Do Not Trust The Device

When you design an application, assume that the device will be owned by an attacker trying to abuse the app. Perform the same secure software development life cycle when building mobile apps as you would for backend services. Do not trust even the databases you create for your mobile apps – a hacker may change the schema. Do not trust the operating system to provide protection – many OS protections can be bypassed trivially by jailbreaking the device. Do not trust that native keystores will keep data secret – some keystores can be broken by bruteforce guessing unless the user protects the device with a long complex password.

Minimize Permissions

Android has the concept of permissions, iOS has entitlements, which allow the application access to sensors such as the GPS and to sensitive content. On Android these permissions need to be specified as part of creating the application in the

AndroidManifest.xml file. They are presented to the user when they choose to install the application on their device.

Each permission increases the potential for your application to do nefarious things and may scare off some users from even downloading your application. So aim to minimize the number of permissions or features your application needs.

Tools

Protection

Basic Java code renaming can be done using Proguard[3], an open-source tool and Arxan's GuardIT[4].

Two vendors for managed-code (Java and .NET) protection tools are Arxan Technologies[5] and PreEmptive Solutions[6].

The main vendors for native code protection tools and white-box cryptography libraries are Arxan and Irdeto[7].

Techniques for protecting Android code against tampering are documented at androidcracking.blogspot.com. Arxan's EnsureIT allows you to insert extra code at build time that will detect debuggers, use checksums to spot changes to the code in memory and allow code to be decrypted or repaired on-the-fly.

Sniffing

A standard free web proxy tool is Paros[8]. A standard network sniffing tool available on common platform is Wireshark[9].

[3] www.proguard.sourceforge.net

[4] arxan.com

[5] arxan.com

[6] preemptive.com

[7] www.irdeto.com

[8] sourceforge.net/projects/paros

[9] sourceforge.net/projects/wireshark

De-Compiling

See the Hex Rays de-compiler[10].

Learn More

Here are some useful resources and references which may help you:

- Apple provides a general guide to software security[11]. It also includes several links to more detailed topics for their platform.
- Commercial training courses are available for iOS and Android. Lancelot Institute[12] provides secure coding courses covering iOS and Android.
- O'Reilly published a book on Android security **Jeff Six (2011): Application Security For The Android Platform. Processes, Permissions and Other Safeguards**[13] and another for iOS, **Jonathan Zdziarski (2012): Hacking and Securing iOS Applications**[14] .
- Charlie Miller et al. (2012) published **iOS Hackers Handbook**[15], which demonstrates how easy it is to steal code and data from iOS devices.
- Academic researchers demonstrate how much information can be gleaned from public Android apps at USENIX 2011[16].
- A free SSL tester is provided by Qualsys Labs[17].

[10] www.hex-rays.com

[11] developer.apple.com/library/mac/navigation/#section=Topics&topic=Security

[12] www.lancelotinstitute.com

[13] shop.oreilly.com/product/0636920022596.do

[14] shop.oreilly.com/product/0636920023234.do

[15] www.wiley.com/WileyCDA/WileyTitle/productCd-1118204123.html

[16] static.usenix.org/event/sec11/tech/slides/enck.pdf

[17] www.ssllabs.com/ssltest

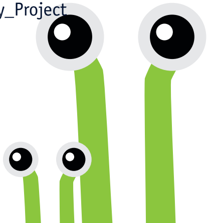
- Extensive free application security guidance and testing tools are provided by OWASP[18], including the OWASP Mobile Security Project[19].
- An open-source mobile application performance monitoring tool for Android is provided by AT&T's Application Resource Optimization tool[20].

The Bottom Line

Mobile apps are becoming ever more trusted, but they are exposed to many who would like to take advantage of that trust. The appropriate level of application security is something that needs to be considered for every app. In the end, your app will be in-the-wild on its own and will need to defend itself against hackers and other malicious threats, wherever it goes.

Invest the time to learn about the security features and capabilities of the mobile platforms you want to target. Use techniques such as threat modelling to identify potential threats relevant to your application. Perform code reviews and strip out non-essential logging and debugging methods. Consider how a hacker would analyze your code, then use similar techniques, in a safe and secure environment, against your application to discover vulnerabilities and mitigate these vulnerabilities before releasing your application.

[18] www.owasp.org
[19] www.owasp.org/index.php/OWASP_Mobile_Security_Project
[20] developer.att.com



BY Julian Harty

Testing

After all your hard work creating your application how about testing it before unleashing it on the world? Testing might be seen as an impediment but failures in your app can be all too public. And recovering your credibility is hard when your app has a poor score in the app store. Testing mobile applications effectively can be complex and challenging where you need to combine automated and interactive testing across a range of devices. Thankfully, several of the major mobile development platforms include test automation in the core tools, including Android and iOS. And cross-platform test automation tools are available for popular platforms; some are free-of-charge and open-source, others are commercial.

Continuous delivery needs continuous testing. Viable apps need to be updated on an ongoing basis in production. Updates may include fixes for new platform versions or device models, new functionality and other improvements. Therefore testing is not a one-off task; high quality apps befit ongoing, optimized testing, including testing in production. Production testing includes testing engagement and validation as well as early detection of potential problems before they mushroom.

This chapter covers the general topics; testing for specific platforms is covered in the relevant chapter.

Testability: The Biggest Single Win

If you want to find ways to test your application effectively and efficiently then start designing and implementing ways to test it; this applies especially for automated testing. For example, using techniques such as Dependency Injection in your code enables you to replace real servers (slow and flaky) with mock servers (controllable and fast). Use unique, clear identifiers for key UI elements. If you keep identifiers unchanged your tests require less maintenance.

Separate your code into testable modules. Several years ago, when mobile devices and software tools were very limited, developers chose to ‘optimize’ their mobile code into monolithic blobs of code, however the current devices and mobile platforms mean this form of ‘optimization’ is unnecessary and possibly counter-productive.

Provide ways to query the state of the application, possibly through a custom debug interface. You, or your testers, might otherwise spend lots of time trying to fathom out what the problems are when the application does not work as hoped.

Test-Driven Development

There are several ways to design and implement software. Test-Driven Development (TDD) is an approach where developers write automated tests in parallel with writing the main code for the app. The automated tests will include unit-tests, these are covered in the next topic.

TDD is both a mindset and a practice. It requires a certain amount of discipline to write the tests even when the going gets tough. And by practising TDD diligently developers are likely to write better-quality, simpler, cleaner code which is also easier to maintain in future (as they are protected and supported by a set of automated tests which can be run when maintaining and revising the source code of the app).

The pure approach is when the tests are written first, and run, before new application code is written. The new tests are expected to fail, that is they should report failures in the behavior of the app. The failures express the mismatch between what the app needs to do and what it currently does. Now the developer has a simple, automated way to test their modifications to the source code for app. Once just enough software has been written to get all the tests to pass you now have confidence the app meets the requirements specified by these tests.

While you may have met the business requirements you may decide your work is not 'done' yet. For instance there may be duplication, unnecessary complexity, and other known flaws in the implementation. You now have an opportunity to revise and improve the source code through a process known as 'refactoring'. Refactoring is where developers improve the implementation where the automated tests continue to pass when run against the improved code.

Although TDD is a struggle when using the current Mobile Test Automation tools several people have provided examples of using TDD successfully, for instance Graham Lee's book Test-Driven iOS Development[1]. You can also consider using TDD for the generic aspects of the app.

Unit Testing

Unit testing involves writing automated tests that test small chunks of code, typically only a few lines of source code. Generally they should be written by the same developer who writes the source code for the app as they reflect how those individual chunks are expected to behave.

Unit tests have a long pedigree in software development, where JUnit[2] has spawned similar frameworks for virtually all of the programming languages used to develop mobile apps.

Unit tests are only one aspect of automated testing, they are not sufficient to prove the app works. They help develop-

[1] informit.com/store/product.aspx?isbn=0321774183

[2] en.wikipedia.org/wiki/JUnit

ers to understand what individual pieces of the software is expected to do. Additional testing, including other forms of automated tests can help to increase your confidence in the app.

Effective Testing Practices

You need to test effectively in order to find problems before they are found by your users. Larger teams may include specialist 'testers'. Effective testing includes knowing the devices, platforms, and how similar apps behave. You need efficient ways to configure devices, for instance to test multilingual apps. You also need to test under realistic conditions, as your users are likely to use your app. Testdroid have a good checklist[3] on getting the right testing expertise into your team.

Here are various topics to get you started.

Interactive Testing

Variety and movement can help expose bugs which remain dormant when testing on a small set of devices in a fixed location such as your workplace. Learn from your users – how do they use your app, or similar apps? Then devise tests that mimic the ways they use apps and devices.

The guidelines at appqualityalliance.org/resources are worth considering when devising your test cases. For instance they include testing the app to see what happens when an incoming phone call is received; and when the user switches the phone to 'flight mode'.

The next few sections will describe three different approaches to interactive testing.

[3] testdroid.com/testdroid/6336/get-the-superb-expertise-in-your-testingqa-team

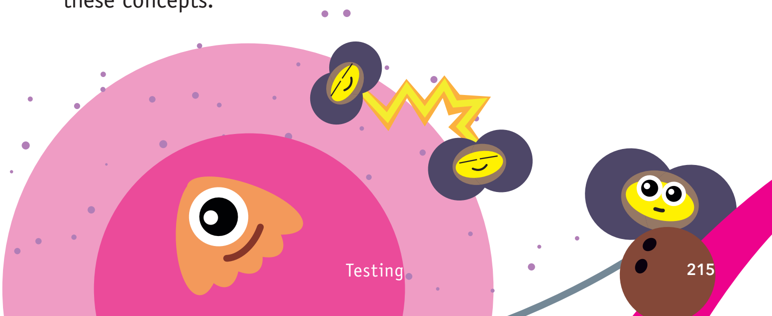
- **Physical devices:** why testing with real phones is important.
- **Remote control:** a way you can test using phones that are not physically in your hands, where they may be thousands of miles away and even on another continent.
- **Crowd sourced testing:** where other testers perform testing on your behalf.

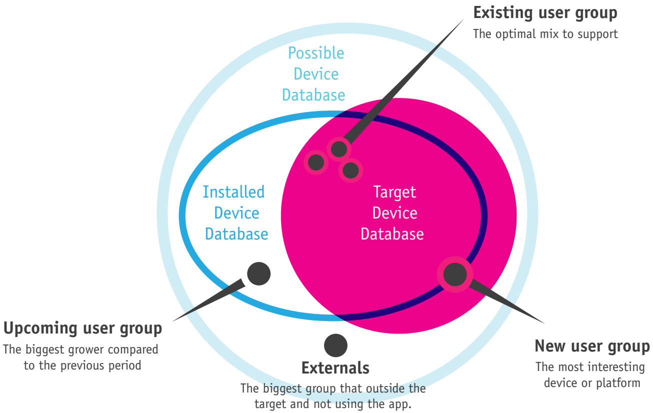
Physical and Virtual Devices

Physical devices are real, you can hold them in your hands. Virtual devices run as software, inside another computer. Both are useful hosts for testing mobile apps.

Virtual devices are generally free and immediately available to install and use. Some platforms, including Android, allows you to create custom devices, for instance with a new screen resolution, which you can use for testing your apps even before suitable hardware is available. They can provide rough-and-ready testing of your applications. Key differences include: performance, security, and how we interact with them compared to physical devices. These differences may affect the validity of some test results.

The set of test devices to use needs to be reviewed on an ongoing basis as the app and the ecosystem evolve. Also you may identify new devices, that your app currently does not support, during your reviews. The following figure illustrates these concepts.




Existing user group
The optimal mix to support
Possible Device Database
Installed Device Database
Target Device Database
Upcoming user group
The biggest grower compared to the previous period
Externals
The biggest group that outside the target and not using the app.
New user group
The most interesting device or platform

Ultimately your software needs to run on real, physical, phones, as used by your intended users. The performance characteristics of various phone models vary tremendously from each other, and from virtual devices on your computer. So: buy, beg, borrow phones to test on. A good start is to pick a mix of popular, new, and models that include specific characteristics or features such as: touch screen, physical keyboard, screen resolution, networking chipset, et cetera. Try your software on at least one low-end or old device as you want users with these devices to be happy too.

Here are some examples of areas to test on physical devices:

- **Navigating the UI:** for instance, can users use your application with one hand? Effects of different lighting conditions: the experience of the user interface can differ in real sunlight when you are out and about. It is a mobile device – most users will be on the move. Rotate the screen and make sure the app is equally attractive and functional.

- **Location:** if you use location information within your app: move – both quickly and slowly. Go to locations with patchy network and GPS coverage to see how your app behaves.
- **Multimedia:** support for audio, video playback and recording facilities can differ dramatically between devices and their respective emulators.
- **Internet connectivity:** establishing an internet connection can take an incredible amount of time. Connection delay and bandwidth depend on the network, its current strength and the number of simultaneous connections. Test the effects of intermittent connectivity and how the app responds.

Remote Devices

If you do not have physical devices at hand or if you need to test your application on other networks, especially abroad and for other locales, then one of the 'remote device services' might help you. They can help extend the breadth and depth of your testing at little or no cost.

Several manufacturers provide this service free-of-charge for a subset of their phone models to registered software developers. Both Nokia[4] for their platforms; and Samsung[5] (for Android and Tizen) provide restricted but free daily access.

You can also use commercial services of companies such as SauceLabs, testdroid, PerfectoMobile or DeviceAnywhere for similar testing across a range of devices and platforms. Some manufacturers brand and promote these services however you often have to pay for them after a short trial period. Some of

[4] developer.nokia.com/Devices/Remote_device_access/

[5] developer.samsung.com/remotetestlab/rtlDeviceList.action

the commercial services provide APIs to enable you to create automated tests.

You can even create a private repository of remote devices, e.g. by hosting them in remote offices and locations.

Beware of privacy and confidentiality when using shared devices.

Crowd-Sourced Testing

There are billions of users with mobile phones across the world. Some of them are professional software testers, and of these, some work for professional out-sourced testing service companies such as utest.com, testhub.com and mob4hire.com. They can test your application quickly and relatively inexpensively, compared to maintaining a larger dedicated software testing team.

These services can augment your other testing, we do not recommend using them as your only formal testing. To get good results you will need to devote some of your time and effort to defining the tests you want them to run, and to working with the company to review the results, et cetera.

Beware Of Specifics

Platforms, networks, devices, and even firmware, are all specific. Any could cause problems for your applications. Test these manually first, provided you have the time and budget to get fast and early feedback.

Test Automation

Automated tests can help you maintain and improve your velocity, your speed of delivering features, by providing early feedback of problems. To do so, they need to be well-designed and implemented. Otherwise you risk doubling your workload to maintain a mess of broken and unreliable automated tests as well as a broken and an unreliable app. Good automated tests mimic good software development practices, for instance using Design Patterns[6], modularity, performing code reviews, et cetera.

It is important to assess the longevity and vitality of the test automation tools you plan to use, otherwise you may be saddled with unsupported test automation code. Test automation tools provided as part of the development SDK are worth considering. They are generally free, inherently available for the particular platform, and are supported by massive companies.

BDD Test Automation

BDD stands for Behavior-Driven Development[7] where the behavior is described in formatted text files that can be run as automated tests. The format of the tests are intended to be readable and understandable by anyone involved with the software project. They can be written in virtually any human language, for instance Japanese[8], and they use a consistent, simple structure with statements such as **Given, When, Then** to structure the test scripts.

There are various BDD frameworks available to test mobile apps. These include:

- **Calabash for Android and iOS:** *http://github.com/calabash*
- **Frank for iOS:** *www.testingwithfrank.com*
- **RoboGerk for Android:** *http://github.com/leandog/RoboGherk*
- **Zucchini for iOS:** *www.zucchiniframework.org*

[6] en.wikipedia.org/wiki/Design_Patterns

[7] en.wikipedia.org/wiki/Behavior-driven_development

[8] github.com/cucumber/cucumber/tree/master/examples/i18n/ja

and various implementations that integrate with Selenium-WebDriver for testing web apps, including web apps on iOS and Android.

Often, custom 'step-definitions' (small scripts that interact with the app being tested) need to be written by someone with coding skills.

GUI Test Automation

GUI test automation is where automated tests interact with the app via the Graphical User Interface (GUI). It is one of the elixirs of the testing industry, many have tried but few have succeeded in creating useful and viable GUI test automation for mobile applications. One of the main reasons why GUI test automation is so challenging is that the User Interface is subject to significant changes which may break the way automated tests interact with the app.

For the tests to be effective in the longer term, and as the app changes, developers need to design, implement and support the labels and other hooks used by the automated GUI tests. Both Apple, with UI Automation[9], and more recently Android[10] use the Accessibility label assigned to UI elements as the de-facto interface for UI automation.

Some commercial companies have opensourced their tools GorillaLogic's MonkeyTalk[11] and Xamarin's Calabash[12]. These tools aim to provide cross-platform support particularly for

[9] developer.apple.com/library/ios/documentation/DeveloperTools/Conceptual/InstrumentsUserGuide/UsingtheAutomationInstrument/UsingtheAutomationInstrument.html

[10] developer.android.com/tools/testing/testing_ui.html

[11] gorillalogic.com/testing-tools/monkeytalk

[12] github.com/calabash

Android and iOS. Other successful opensource frameworks include Robotium[13] and Frank[14].

Headless Client

The user-interface (UI) of a modern mobile application can constitute over 50% of the entire codebase. If your testing is limited to using the GUI designed for users you may needlessly complicate your testing and debugging efforts. One approach is to create a very basic UI, a thin wrapper, around the rest of the core code (typically this includes the networking and business layers). This 'headless' client may help you to quickly isolate and identify bugs e.g. related to the device, carrier, and other environmental issues.

Another benefit of creating a headless client is that it may be simpler to automate some of the testing e.g. to exercise all the key business functions and/or to automate the capture and reporting of test results.

You can also consider creating skeletal programs that 'probe' for essential features and capabilities across a range of phone models e.g. for a J2ME application to test the File Handling where the user may be prompted (many times) for permission to allow file IO operations. Given the fragmentation and quirks of mature platforms such probes can quickly repay the investment you make to create and run them.

Testing Through The Five Phases of an App's Lifecycle

The complete lifecycle of a mobile app fits into 5 phases: implementation, verification, launch, engagement and validation. Testing applies to each phase. Some of the decisions made

[13] code.google.com/p/robotium/

[14] testingwithfrank.com/

for earlier stages can affect your testing in later stages. For instance, if you decide you want automated system tests in the first phase they will be easier to implement in subsequent phases.

Phase 1: Implementation

This includes design, code, unit tests, and build tasks. Traditionally testers are not involved in these tasks; however good testing here can materially improve the quality and success of the app by helping us to make sure the implementation is done well.

In terms of testing, you should decide the following questions:

- Do you use test-driven development (TDD)?
- Do you write unit tests even if we are not using TDD?
- Will you have automated system tests? If so, how will you facilitate these automated system tests? For instance by adding suitable labels to key objects in the UI.
- How will you validate your apps? For instance, through the use of Mobile Analytics? Crash reporting? Feedback from users?

Question the design. You want to make sure it fulfills the intended purposes; you also want to avoid making serious mistakes. Phillip Armour's paper on five orders of ignorance[15] is a great resource to help structure your approach.

Also consider how to improve the testability of your app at this stage so you can make your app easier to test effectively and efficiently. Practices, including unit tests and Test-Driven-Development (TDD) apply to the implementation phase. Remember to test your build process and build scripts to ensure

[15] www-plan.cs.colorado.edu/diwan/3308-07/p17-armour.pdf

they are effective, reliable and efficient, otherwise you are likely to suffer the effects of poor builds throughout the life of the app.

Phase 2: Verification

This includes reviewing unit tests, internal installation, and system tests.

Review your unit tests and assess their potency: Are they really useful and trustworthy? Note: they should also be reviewed as part of the implementation phase, however this is a good time to address material shortcomings before the development is considered 'complete' for the current code base.

For apps that need installing you need ways to deploy them to specific devices for pre-release testing. Some platforms (including Android, iOS and Windows Phone) need phones to be configured so development apps can be installed. You also need to decide which phones to test the app on. For instance, it is wise to test the app on each suitable version of the mobile platform. For iOS this may only include the latest releases. For Android you also need to consider low end devices that are still being sold with version 2.x of Android and will never be updated to 4.x

You will also want to test different form-factors of devices; for instance where the ratio of the screen dimensions differ. The iPhone 5's new screen dimensions exposed lots of UI bugs. Android developers are well aware of the many issues different screen sizes can trigger.

System tests are often performed interactively, by testers. Consider evaluating test automation tools and frameworks for some of your system tests. We will go into more detail later in this section.

You also want to consider how to make sure the app meets:

- Usability, user experience and aesthetics requirements
- Performance, particularly as perceived by end users
- Internationalization and localization testing

Phase 3: Launch

This includes pre-publication and publication.

For those of you who have yet to work with major app stores be prepared for a challenging experience where most aspects are outside your control, including the timescales for approval of your app. Also, on some app stores, you are unable to revert a new release. So if your current release has major flaws you have to create a new release that fixes the flaws, then wait until it has been approved by the app store, before your users can receive a working version of your app.

Given these constraints it is worth extending your testing to include pre-publication checks of the app such as whether it is suitable for the set of targeted devices. The providers of the main platforms now publish guidelines to help you test your app will meet their submission criteria. These guidelines may help you even if you target other app stores.

Android	developer.android.com/distribute/googleplay/publish/preparing.html#core-app-quality
Apple	developer.apple.com/appstore/resources/approval/guidelines.html (Apple account needed for access)
BlackBerry	developer.blackberry.com/devzone/appworld/tips_for_app_approval.html
Windows Phone	msdn.microsoft.com/en-us/library/windowsphone/develop/hh394032(v=vs.105).aspx

Phase 4: Engagement

This includes search, trust, download and installation. Once your app is publicly available users need to find, trust, download and install it. You can test each aspect of this phase in production. Try searching for your app on the relevant app store, and in mainstream search engines. How many different ways can it be found by your target users? What about users outside the target groups – do you want them to find it? How will users trust your app sufficiently to download and try it? Does your app really need so many permissions? How large is the download, and how practical is it to download over the mobile network? Will it fit on the user's phone, particularly if there is little free storage available on their device? And does the app install correctly – there may be signing issues which cause the app to be rejected by some phones.

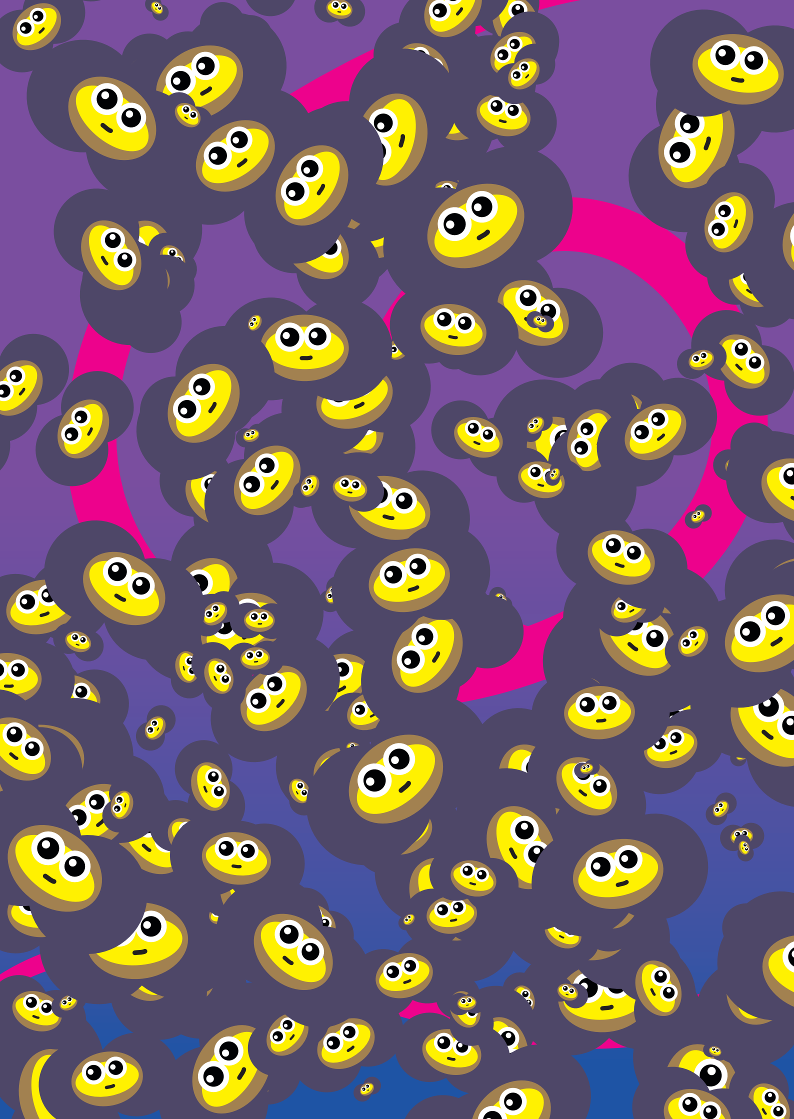
With a continuous flow of new releases to production, a proactive approach is needed to monitor the engagement aspects.

Phase 5: Validation

This includes payment, use and feedback. As you may already know, a mobile app with poor feedback is unlikely to succeed. Furthermore many apps have a very short active life on a user's phone. If the app does not please and engage them within a few minutes it is likely to be discarded or ignored. And for those of you who are seeking payment, it is worth testing the various forms of payment, especially for in-app payments.

Consider finding ways to test the following as soon as practical:

- **Problem detection and reporting:** These may include your own code, third-party utilities, and online services.
- **Mobile Analytics:** Does the data being collected make sense? What anomalies are there in the reported data? What is the latency in getting the results, et cetera?



BY Michel Shugair & Carlo Longino

Monetization

Finally you have finished your app or mobile website and polished it as a result of beta testing feedback. Assuming you are not developing as a hobby, for branding exposure, et cetera, now it is time to make some money. But how do you do that, what are your options?

In general, you have the following monetization options:

1. **Pay per download:** Sell your app per download
2. **In-app payment:** Add payment options into your app
3. **Mobile advertising:** Earn money from advertising
4. **Sponsorships:** Receive money for each user signing up to your sponsor
5. **Revenue sharing:** Earn revenue from operator services originating in your app
6. **Indirect sales:** Affiliates, data reporting and physical goods among others
7. **Component marketplace:** Sell components or a white-label version of your app to other developers

When you come to planning your own development, determining the monetization business model should be one of the key elements of your early design as it might affect the functional and technical behavior of the app.

Pay Per Download

Using pay per download (PPD) your app is sold once to each user as they download and install it on their phone. Payment can be handled by an app store, mobile operator, or you can setup a mechanism yourself.

When your app is distributed in an app store, the store will handle the payment mechanism for you. In return the store takes a revenue share (typically 30%) on all sales. In most cases stores offer a matrix of fixed price points by country and currency ($0.99, EUR 0.79, $3 etc) to choose from when pricing your app.

Payment for downloaded apps is generally handled in one of two ways: operator billing or credit-card payments.

Operator billing enables your customers to pay for your app by just confirming that the sale will be charged to their mobile phone bill or by sending a Premium SMS. In some cases, operator billing is handled by an app store (such as Google Play, which supports operator billing for a number of carriers around the world). In other cases, it can be implemented directly by the developer.
Each operator will take a revenue share of the sale price (typically 30% to 65%, but some operators can take up to 95%), and, if you use one, an aggregator will take its share too. Security (how you prevent the copying of your app) and manageability are common issues with the PPD model, but in some scenarios it might be the only monetization option. Operator billing can be quite difficult to handle on your own, particularly if you want to sell in several countries, as you need to sign contracts with each operator in each country. For unknown reasons some operators, like Vodafone, seem to remove operator billing as an option for Android Play in some key markets, like UK and Germany. Possibly because better alternatives like local mobile bank payments become available.

It is worth noting that most of the vendor app stores are pursuing operator billing agreements, with Nokia Store having good coverage with operator billing available in 60 countries for both their legacy Nokia Store and Microsoft's Windows Phone Marketplace. Google and Blackberry have similar options. The principal reason they are doing this is that typically, when users have a choice of credit card and operator billing methods users show a significant preference for operator billing (Nokia says its research has shown up to a 10x increase in revenues over credit card payments). Nokia, at least, also insulates developers from the variation in operator share, offering developers a fixed 70% of billing revenue.

Credit-card billing is used by Apple, Google (in some cases), Amazon and other stores. Apple has required iPhone users to provide credit-card data at registration for many years, and Google now requires this as well for Android users. Having this information entered before purchase is, according to analysts, a key differentiator for higher monthly per app revenue.

The last payment option is to create your own website and implement a payment mechanism through that, such as PayPal mobile, Dutch initiative èM! Payment[1], dial-in to premium landlines[2] or others.

Using PPD can typically be implemented with no special design or coding requirements for your app and for starters we would recommend using the app store billing options as it involves minimal setup costs and minor administrative overhead.

For each form of payment it is important to determine price elasticity of demand PED[3]. Increasing the price does not necessarily mean higher total revenue (and vice versa), your price needs to match expectations of your user base.

[1] empayment.com

[2] daopay.com

[3] en.wikipedia.org/wiki/Price_elasticity_of_demand

In-App Payment

In-app payment is a way to charge for specific actions or assets within your application. A very basic use might be to enable the one-off purchase of your application after a trial period — which may garner more sales than PPD if you feel the features of your application justify a higher price point. Alternatively, you can offer the basic features of your application for free, but charge for premium content (videos, virtual credits, premium information, additional features, removing ads and alike). Most app stores offer an in-app purchase option or you can implement your own payment mechanism. If you want to look at anything more than a one-off “full license” payment you have to think carefully about how, when and what your users will be willing to pay for and design your app accordingly.

In-app purchases have become the leading monetization model in many markets, particularly among "freemium" games that use free distribution to get users hooked before turning them into buyers. In 2013 92% of global iOS app revenues and 98% of Android app revenues come from in-app purchasing, according to Distimo[4].

This type of payment is particularly popular in games (for features such as buying extra power, extra levels, virtual credits and alike) and can help achieve a larger install base as you can offer the basic application for free. Note, however, that some app stores (such as Apple's) do not allow third-party payment options to be implemented inside your app. This is done to prevent you from using the app store for free distribution while avoiding payment of the store’s revenue share.

It should also be obvious that you will need to design and develop your application to incorporate the in-app payment

[4] www.distimo.com/publications, download “2013 Year in review” from December 2013

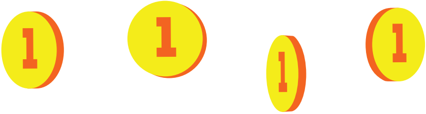
method. If your application is implemented across various platforms, you may need to implement a different mechanism for each platform (in addition to each app store, potentially).

As with PPD we would recommend that you start with the in-app purchasing mechanism offered by an app store, particularly as some of these can leverage operator billing services (such as Google Play) or utilize pre-existing credit-card information (such as Apple or Amazon) , or with in-app payment offered directly by operators. From a user's perspective, this is the easiest and most convenient way to pay (one or two clicks, no need to enter credit card numbers, user names or other credentials), so developers can expect the highest user acceptance and conversion rates.

Mobile Advertising

As is common on websites, you could decide to earn money by displaying advertisements. There are a number of players who offer tools to display mobile ads and it is the easiest way to make money on mobile browser applications. Admob.com, Buzzcity.com and inmobi.com are a few of the parties that offer mobile advertising. However because of the wide range of devices, countries and capabilities there are currently over 50 large mobile ad networks. Each network offers slightly different approaches and finding the one that monetizes your app's audience best may not be straightforward. There is no golden rule; you may have to experiment with a few to find the one that works best. However, for a quick start you might consider using a mobile ad aggregator, such as Madgic[5], smaato[6] or inneractive[7] as they tend to bring you better earnings by combining

[5] madgic.com
[6] smaato.net
[7] inner-active.com



and optimizing ads from 40+ mobile ad networks. Most ad networks take a 30% to 50% share of advertising revenue and aggregators another 15% to 20% on top of that.

If your app is doing really well and has a large volume in a specific country you might consider selling ads directly to advertising agencies or brands (Premium advertising) or hire a media agency to do that for you.

Again many of the device vendors offer mobile advertising services as part of their app store offering and these mechanisms are also worth exploring. In some cases you may have to use the vendor's offering to be able to include your application in their store.

In-application advertising will require you to design and code your application carefully. Not only the display location of ads within your app needs to be considered with care, also the variations and opt-out mechanism. If adverts become too intrusive, users may abandon your app, while making the advertising too subtle will mean you gain little or no revenue. Relatively new compared to traditional banner advertising is interstitial advertising: This term is generally used to describe an ad that takes up the entire screen and typically has a "skip screen" button at the bottom. It may require some experimentation to find the right level and positions in which to place adverts.

Sponsorships

The German startup Apponsor[8] offers a new way of earning money without the need to display advertising or charge a download fee: The user gets your app for free and is prompted to sign-up for a newsletter of your sponsor. The sponsor will in return pay the developer an amount for each newsletter registration.

[8] apponsor.com

Indirect Sales

Another option is to use your application to drive sales elsewhere.

Here you usually offer your app or website for free and then use mechanisms such as:

1. Affiliate programs: Promote third party or your own paid apps within a free app. See also MobPartner[9]. This can be considered a variation on mobile advertising
2. Data reporting: Track behavior and sell data to interested parties. Note that for privacy reasons you should not reveal any personal information, ensure all data is provided in anonymous, consolidated reports
3. Virtual vs. real world: Use your app as a marketing tool to sell goods in the real world. Typical examples are car apps, magazine apps and large brands such as McDonald's and Starbucks. Also coupon applications as Groupon often use this business model

There is nothing to stop you from combining this option with any of the other revenue generation options if you wish, but take care that you do not give the impression of overly-intrusive promotions.

Component Marketplace

A Component Marketplace (CMP) provides another opportunity for developers to monetize their products to other developers and earn money by selling software components or white-labelled apps. A software component is a building block piece

[9] mobpartner.com

of software, which provides a defined functionality, that is to be used by higher level software.

The typical question that comes up at this point is on how CMPs contrast to open source. As a user, open source is often free-of-charge. Source code must be provided and users have the right to modify the source code and distribute the derived work.

Some component providers require a license fee. They may provide full source code which enables the developer to debug into lower level code. Some CMPs support all models: Paid components with or without source code as well as free components with or without source code.

If you are a developer searching for a component, CMPs offer two major advantages: First, you do not have to open source your code just because you use software components. All open source comes with a license. Some licenses like the Apache are commercially friendly; others, such as AGPL and OSL, require you to open source your code that integrates with theirs. You might not want this. Secondly, CMPs provide an easy way to find and download components. You can spend days browsing open source repositories to find the right thing to use.

Component marketplaces have existed for decades now. The most prominent marketplace is for components for Visual Basic and .NET in the Windows community. Marketplaces such as componentOne and suppliers like Infragistics are well known in their domain. The idea of component marketplaces within the mobile arena is quite new. Deutsche Telekom's Developer Garden[10], ChupaMobile[11] and Verious[12] are relevant players in this domain.

[10] www.developergarden.com/component-marketplace/

[11] www.chupamobile.com

[12] www.verious.com

Choosing your Monetization Model

So with all these options what should your strategy be? It depends on your goals, let us look at a few:

- Do you want a large user base? Consider distributing your application for free with in-app purchases, or with mobile advertising (you could even offer a premium ad-free version)
- Are you convinced users will be willing to buy your app immediately? Then sell it as PPD for $0.99, but beware while you might cash several thousand dollars per day it could easily be no more than a few hundred dollars per week if your assessment of your app is misplaced or the competition fierce
- Are you offering premium features at a premium price? Consider a time or feature limited trial application then use in-app purchasing to enable the purchase of a full version either permanently or for a period of time
- Are you developing a game? Consider offering the app for free with in-app advertising or a basic version then use in-app purchasing to allow user to unlock new features, more levels, different vehicles or any extendable game asset
- Is your mobile app an extension to your existing PC web shop or physical store? Offer the app for free and earn revenue from your products and services in the real world

Appstore Strategies

The flip side of revenue generation is marketing and promotion. The need might be obvious if you sell your application through your own website, but it is equally important when using a vendor's app store. Appstores are the curse and the blessing of mobile developers. On the bright side they give developers extended reach and potential sales exposure that would otherwise be very difficult to achieve. On the dark side the more popular ones now contain hundreds of thousands of apps, decreasing the potential to stand out from the crowd and be successful, leading many to compare the chances of appstore success to the odds of winning the lottery.

So, here are a few tips and tricks to help your raise your odds.

Basic Strategies To Get High

The most important thing to understand about appstores is that they are distribution channels and not marketing machines. This means that while appstores are a great way to get your app onto users' devices, they are not going to market your app for you (unless you purchase premium positioning either through banners or list placings). You cannot rely on the app stores to pump up your downloads, unless you happen to get into a top-ten list. But do not play the lottery with your apps, have a strategy and plan to market your app.

We have asked many developers about the tactics that brought them the most attention and higher rankings in appstores.

Many answers came back and one common theme emerged: there is no silver bullet – you have to fire on all fronts! However it will help if you try to keep the following in mind:

- You need a kick ass app: it should be entertaining, easy to use and not buggy. Make sure you put it in the hands of users before you put it in a store.
- Polish your icons and images in the appstore, work on your app description, and carefully choose your keywords

and category. If unsure of or unsatisfied with the results, experiment.

- Getting reviewed by bloggers and magazines is one of the best ways to get attention. In return some will be asking for money, some for exclusivity, and some for early access.
- Get (positive) reviews as quickly as possible. Call your friends and ask your users regularly for a review.
- If you are going to do any advertising, use a burst of advertising over a couple of days. This is much more effective than spending the same amount of money over 2 weeks, as it will help create a big spike, rather than a slow, gradual push.
- Do not rely on the traffic generated by people browsing the appstore, make sure you drive traffic to your app through your website, SEO and social media.

Multi-Store vs Single Store

With 120+ appstores available to developers, there are clearly many application distribution options. But the 20 minutes needed on average to submit an app to an appstore means you could be spending a lot of time posting apps in obscure stores that achieve few downloads. This is why a majority of developers stick to only 1 or 2 stores, missing out on a potentially huge opportunity but getting a lot more time for the important things, like coding! So should you go multi-store or not?

Multi-store	Single store
The main platform appstores can have serious limitations, such as payment mechanisms, penetration in certain countries, content guidelines.	90%+ of smartphone users only use a single appstore, which tends to be the platform appstore shipping with the phone
Smaller stores give you more visibility options (featured app)	Your own website can bring you more traffic than appstores

Multi-store	Single store
Smaller stores are more social media friendly than large ones.	Many smaller appstores scrape data from large stores, so your app may already be there.
Operators' stores have notoriously strict content guidelines and can be difficult to get in, particularly for some types of apps.	For non-niche content, operator or platform stores may offer enough exposure to not justify the extra effort of a multi-store strategy.
Smaller stores may offer a wider range of payment or business model options, or be available in many countries.	Some operators' stores have easier billing processes – such as direct billing to a user's mobile account – leading to higher conversion rates.
Some developers report that 50% of their Android revenues come from outside of Android Market	iOS developers only need 1 appstore

The platform app stores should give you general coverage for users, but over time, it is in your interest to adapt your appstore strategy to match your targeted user base, and utilize the appstores that best reach it. This could mean using particular operator stores, stores popular in a specific country, or simply sticking with the platform stores. There are some third-party appstores with large audiences, such as the Amazon appstore for Android, which offers developers a number of ways to monetize their apps, such as PPD and in-app payments in several countries. Additionally, in some countries, there are locally popular appstores, such as AndroidPit[13] in Germany, or one of the many China-specific Android stores.

[13] androidpit.de

What Can You Earn?

One of the most common developer questions is about how much money they can make with a mobile app. It is clear that some apps have made their developer's millionaires, while others will not be giving up their day job anytime soon. According to a 2012 research by Forbes.com[14], most app developers are not generating enough revenue to break even with development costs and single platform developers confirmed it was not enough to support a standalone business. According to VisionMobile's Developer Economics 2013 research among over 6,000 mobile developers 67% of them are below the "app poverty line" of $500 per app per month[15].

Ultimately, what you can earn is about fulfilling a need and effective marketing. Experience suggests that apps which save the user money or time are most attractive (hotel discounts, coupons, free music and alike) followed by games (just look at the success of Angry Birds) and business tools (office document viewers, sync tools, backup tools and alike) but often the (revenue) success of a single app cannot be predicted. Success usually comes with a degree of experimentation and a lot of perseverance.

Learn More

If you want to dig deeper into the topic of app marketing, check out the "Mobile Developer's Guide To The Parallel Universe" published by WIP[16].

[14] www.forbes.com/sites/tristanlouis/2013/08/10/how-much-do-average-apps-make

[15] www.visionmobile.com/products/research

[16] wip.org

Epilogue

Thanks for reading this 14th edition of our Mobile Developer's Guide. We hope you have enjoyed reading it and that we helped you to clarify your options. Perhaps you are now ready to get involved in developing a mobile app or have discovered new options in the app business. We hope so. Please also get involved in the community and share your experiences and ideas with us and with others.

If you like to contribute to this guide or sponsor upcoming editions, please send your feedback to *mdgg@enough.de*.

If you are using Twitter, you are invited to follow us on *twitter.com/enoughsoftware* and spread the word about the project using the Twitter hashtag *#mdgg*.

You can of course also get this guide as an ebook: check *amazon.com* and *kobobooks.com*.

If you prefer the hardcopy version, you can order it at: *www.enough.de/mdgg*.

At the time of writing we are also working on making the content of this book available as a website at *www.mobiledevelopersguide.com*, where you will hopefully very soon find new ways to get involved and submit your feedback.

About the Authors

Mostafa Akbari / bitstars

Mo worked in software engineering and human interaction research the past few years. He is involved in green mobility projects. Now Mo has started a spinoff with Simon Heinen out of the RWTH Aachen University for augmented reality research and development. He focuses on AR interactions with personal location-based data and on computer vision. His passion for mix reality games is based on his passion for board games and geocaching.

🐦 **@mosworld** **www.bitstars.com**

Anna Alfut

Anna started her professional life as Creative Designer. After discovering her passion for interface design she co-authored an app for iOS and Android platforms and consulted on multiple projects both on the agency and client side. Currently she works in-house as UI and UX designer for consumer facing products on mobile and desktop. Apart from thinking through and drawing screens she also does illustration and enjoys living in London.

www.alfutka.net

Andrej Balaz / Enough Software

As a graduate of the University of the Arts Bremen, Andrej focuses on UI, UX and visual design for mobile applications and other interactive technologies. He is also in charge of the layout and design of this guide. When not involved with something mobile, he loves to experiment with digital art and illustration.

🐦 **@Designamyte** **www.enough.de** **www.balaz.de**

Richard Bloor / Sherpa Consulting Ltd

Richard has been writing about mobile applications development since 2000. He has contributed to popular websites, such as AllAboutSymbian.com, but now focuses on assisting companies in creating resources for developers. Richard brings a strong technical background to his work, having managed development and testing on a number of major IT projects, including the Land Information NZ integrated land ownership and survey system. When not writing about mobile development, Richard can be found regenerating the native bush on his property north of Wellington, New Zealand.

Davoc Bradley / Rivo Software

Davoc has been working as a software engineer since 1999 specializing in architecture and design of high usage web and mobile systems. Recently he was behind the architecture, design and development of a gold award winning Mobile Application Management system and is currently Rivo Software's Technical Architect. Davoc is also a keen musician, avid cricket fan and loves travelling.

🐦 **@davocbradley** **www.rivosoftware.co.uk**

Marco Büttner / SciDev

Marco is 25 years old and has been a mobile developer since 2011. He is studying computer science at the Humboldt University of Berlin and founded the mobile development project SciDev, which focuses on developing web apps for bada, Tizen and other emerging mobile platforms. He is well connected among the bada and Tizen community and always glad to share his know-how.

🐦 **@scionbln** **www.scidev.eu**

Dean Churchill / AT&T

Dean works on secure design, development and testing of applications at AT&T. Over the past several years he has focused on driving security requirements in mobile applications, for consumer applications as well as internal AT&T mobile applications. He has been busy supporting AT&T's emerging Mobile Health and Digital Life product lines. He lives in the Seattle area and enjoys downhill skiing and fly fishing.

Julian Harty / Commercetest

Julian was hired by Google in 2006 as their first Test Engineer outside the USA responsible for testing Google's mobile applications. He helped others, inside and outside Google, to learn how to do likewise; and he ended up writing the first book on the topic. He subsequently worked for eBay where his mission was to revamp testing globally. Currently he is working independently, writing mobile apps & suitable test automation tools, and helping others to improve their mobile apps. He is also writing a new book on testing and test automation for mobile apps.

🐦 **@julianharty**

Bob Heubel / Immersion Corp.

Bob Heubel is a haptic technology evangelist with Immersion Corporation specializing in helping developers implement what is known as force-feedback, tactile-feedback or rumble-feedback effects. He holds a number of patents in the field of Haptics and has spent more than fourteen years working with developers, carriers and hardware OEMs to design and program tactile sensations into game and interaction experiences. You may have felt some of Bob's work in Rockstar Games' Grand Theft Auto: Vice City & Max Payne for Android. Bob graduated from UC Berkeley in 1989 with a BA in English Literature.

🐦 **@bobheubel** **www.immersion.com**

Ovidiu Iliescu / Enough Software

After developing desktop and web-based applications for several years, Ovidiu decided mobile software was more to his liking. He is involved in Java ME and Blackberry development for Enough Software since 2009. He gets excited by anything related to efficient coding, algorithms and computer graphics.

🐦 **@ovvyblabla** **www.enough.de** **www.ovidiuiliescu.com**

Alex Jonsson / EvoThings

Alex likes anything mobile, both apps and web technologies as well as cleverly connecting physical stuff to digital stuff. He holds a PhD in Media Technology from the Royal Institute of Technology in Stockholm and freely shares his ideas and thought with both the industry and academia. Dr Jonsson also has an eclectic urge to investigate how apps and services can drive new business, by bringing novel values and ways to make things more connected, thereby binding the universe together in new, clever ways. Alex is founder and CTO of EvoThings because things simply are better connected.

@dr_alexj www.evothings.com

Matos Kapetanakis / VisionMobile

As marketing manager of VisionMobile his activities include managing the VisionMobile website and blog, as well as coming up with the concepts and marcoms for the illustrations and infographics published by the company. Matos is also the project manager of the Developer Economics research series, as well as other developer research projects.

www.visionmobile.com

Michael Koch / Enough Software

Michael has developed software since 1988 and joined the development team at Enough Software in 2005. He holds the position of CTO. He has led numerous mobile app development projects (mainly for Java ME, Android, Windows Mobile and BlackBerry) and he is also an expert in server technology. Michael is an open source enthusiast involved in many free projects, such as GNU classpath.

@linux_pinguin www.enough.de

Daniel Kranz / Joule

Daniel is a multi-channel strategist with consultancy, agency and tech background. Previously a technical project manager at one of the leading advertising agencies and a mobile solution consultant for a mobile and multi-channel web specialist, he now works in global strategic planning advising brands on how to integrate mobile as part of their overall strategy.

www.jouleww.com

Carlo Longino / WIP

Carlo has more than a decade of experience in the mobile industry, beginning just after the turn of the century when he worked for Nokia at its headquarters in Finland. Before joining the Wireless Industry Partnership (WIP) as director of developer marketing services in 2010, Carlo worked as a freelance consultant and writer while he completed an MBA. Prior to that, he was senior analyst for Floor64, a Silicon Valley-based analyst firm, where he covered the mobile and fixed telecom industries. He also helped launch and spent five years running TheFeature.com, a thought-leadership site owned by Nokia. Carlo has also been published in The Wall Street Journal, Business 2.0 and Dow Jones Newswires and has spoken at a number of industry events, including Mobile World Congress, SXSW, MobileBeat and CTIA, among others.

@caaarlo   www.wip.org

Tim Messerschmidt / PayPal

Tim has been developing Android applications since 2008. After studying business informatics, he joined the Berlin-based Neofonie Mobile as Mobile Software Developer in 2011 and has consulted for Samsung Germany as Developer Advocate for Android and bada since 2010. In 2012 he moved to PayPal as a Developer Evangelist. He is passionate about Mobile Payments, UI, UX and Android development in general. Furthermore he loves to speak at conferences, writing articles and all kind of social media.

@seraandroid & @PayPalDev www.timmesserschmidt.de

Patrick Mortara

Patrick studied computer sciences in Frankfurt. He has been developing desktop-based software since the mid-nineties, both as a freelancer and for his day job. He started mobile development in 2010 when Samsung released its first bada smartphone, the Samsung Wave I.

@pmortara   www.mortara.org

Gary Readfern-Gray / RNIB

Gary is an Accessibility specialist working for the Royal National Institute of the Blind. Located in the Innovation Unit, he has a passion for the mobile space and particularly for enabling accessible app development across a range of platforms by engaging with developer communities.

www.rnib.org.uk

Alexander Repty

Alexander has been developing software for Mac OS X since 2004. When the iPhone SDK was released in 2008, he was among the first registered developers for the program. As an employee of Enough Software, he worked on a number of apps, one of which was featured in an Apple TV commercial. He has written a series of articles on iPhone development. As of April 2011, he started his own business as an independent software developer and contractor.

@arepty **www.alexrepty.com**

Marcus Ross

Marcus is a freelance developer and trainer. After 10 years of being an employee in several companies, he is now doing SQL-BI Projects and everything mobile cross platform. He is a regular author in the German magazine "mobileWebDeveloper". In his spare time, he is often seen at conferences, speaking on mobile subjects and JavaScript. He also writes articles, books & tweets on mobile development.

@zahlenhelfer **www.zahlenhelfer-consulting.de**

Michel Shuqair / AppValley

Starting with black and white WAP applications, iMode and SMS games in the 1990's, Michel moved to lead the mobile social network m.wauwee.com. Serving almost 1,000,000 members, Michel was supported by a team of Symbian, iPhone, BlackBerry and Android specialists at headquarters in Amsterdam. m.wauwee.com was acquired by MobiLuck.

www.appvalley.nl

Marco Tabor / Enough Software

Marco is responsible for PR, sales and much more at Enough Software. He coordinates this project, taking, as well, the responsibility of finding sponsors and merging the input provided by the mobile community.

@enoughmarco   www.enough.de

Ian Thain / SAP

Ian is a Mobile Evangelist at SAP, though he started 13 years ago with Sybase Inc. He regularly addresses audiences all over the world providing mobile knowledge and experience for the Enterprise. He also writes articles, blogs & tweets on Enterprise Mobility and is passionate about the Developer & Mobile Experience in the Corporate/Business world.

@ithain   scn.sap.com/blogs/ithain/   www.sap.com

Marc van 't Veer / Polteq

Marc is a test consultant at Polteq with over 7 years working as a coordinator and system tester. He has a lot of experience in testing in a technically oriented context, such as telecom, SOA, test automation, development of stubs and drivers and testing API's. In his current job he coordinates all the mobile app testing for a big Dutch supermarket.

www.polteq.com

Robert Virkus / Enough Software

Robert has been working in the mobile space since 1998. He experienced Java fragmentation first hand when developing and porting a mobile client on the Siemens SL42i, the first mass market phone with an embedded Java VM. After this experience he launched the Open Source J2ME Polish project in 2004. J2ME Polish helps developers overcome device fragmentation. He is the founder and CEO of Enough Software, the company behind J2ME Polish, many mobile apps, and this book.

@robert_virkus   www.enough.de   www.j2mepolish.org

an initiative by

ENOUGH SOFTWARE
www.enough.de

wip wireless industry partnership
www.wip.org

sponsored by

SAP

twilio

PayPal | Developer

A NON-COMMERCIAL, COMMUNITY-DRIVEN OVERVIEW ON MOBILE TECHNOLOGIES FOR DEVELOPERS AND DECISION-MAKERS.

Daniel Hudson, www.webtechman.com
A spectacular piece of work! You will be astonished by how incredibly fast you can establish your presence in the mobile market with the simple steps explained in this guide.

Monika Lischke, Community Manager, Intel AppUp developer program
Extremely helpful content, also for non-developers.
And the design is nothing but fantastic!