

Don't Panic

MOBILE DEVELOPER'S GUIDE TO THE GALAXY



11th



published by:



ENOUGH
SOFTWARE

Services and Tools for All Mobile Platforms

Enough Software GmbH + Co. KG
Sögestrasse 70
28195 Bremen
Germany
www.enough.de

Please send your feedback,
questions or sponsorship requests to:
developers@enough.de
Follow us on Twitter: [@enoughsoftware](https://twitter.com/enoughsoftware)

11th Edition October 2012

This Developer Guide is licensed under the
Creative Commons Some Rights Reserved License.

Art Direction and Design by
Andrej Balaz
(Enough Software)

Mobile Developer's Guide

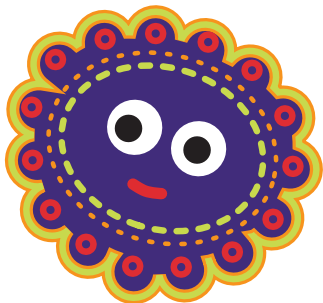
Contents

- 1 **Prologue**

- 1 **An Introduction To Mobile Development**
 - 1 Form Factors and Usage Patterns
 - 2 Mobile Service Options
 - 5 Lost in the Jungle

- 7 **Android**
 - 9 Prerequisites
 - 10 Implementation
 - 12 Testing
 - 14 Distribution

- 16 **bada**
 - 17 Getting Started
 - 17 Implementation
 - 21 Resources
 - 21 Testing
 - 22 Distribution

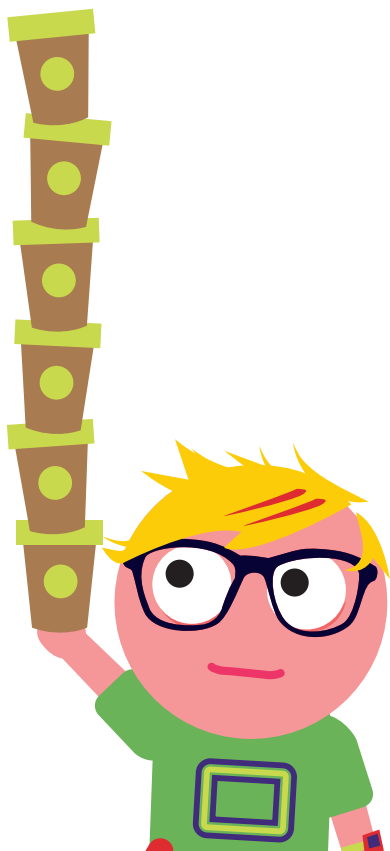


- 24 **BlackBerry Java Apps**
- 25 Prerequisites
- 26 Java SDK
- 26 IDE
- 26 Desktop Manager
- 27 Coding Your Application
- 28 Services
- 29 Testing
- 29 Porting
- 30 Signing
- 30 Distribution

- 32 **BlackBerry 10**
- 32 The Alpha Device
- 33 Development
- 39 Testing
- 40 Signing
- 40 Distribution

- 42 **iOS**
- 43 Prerequisites
- 45 Implementation
- 46 Testing
- 48 Distribution
- 49 Books
- 51 Community

- 53 **Java ME (J2ME)**
- 54 Prerequisites
- 55 Implementation
- 58 Testing
- 59 Porting
- 61 Signing
- 62 Distribution



| | |
|-----|---|
| 65 | Qt |
| 66 | Prerequisites |
| 67 | Creating Your Application |
| 69 | Testing |
| 69 | Packaging |
| 70 | Signing |
| 71 | Distribution |
| 73 | Windows Phone |
| 73 | UI Design |
| 74 | Development |
| 77 | Functions And Services |
| 77 | Multitasking And Application Lifecycle |
| 78 | Native Code |
| 78 | Distribution |
| 79 | Testing And Analytics |
| 80 | Monetization |
| 80 | Resources |
| 81 | Windows Phone 8 |
| 84 | Windows 8 |
| 84 | The Artist Formerly Known As Metro |
| 84 | Prerequisites |
| 85 | Developing Metro Style Apps |
| 90 | Distribution |
| 91 | Resources |
| 93 | Going Cross-Platform |
| 94 | App Development Process |
| 94 | Limitations And Challenges Of Cross Platform Approaches |
| 99 | Cross-Platform Strategies |
| 103 | Cross-Platform App Frameworks |
| 108 | Cross-Platform Game Engines |

111 **Web Technologies**

- 112 Usability
- 113 Performance
- 113 HTML5
- 114 WebApps
- 115 Adaptation
- 118 Technical Limits of Web Technologies
- 119 HTML without Browsers
- 122 Test & Debugging
- 123 Summary

125 **Accessibility**

- 127 Developing Accessible Android Apps
- 128 Developing Accessible BlackBerry Apps
- 128 Developing Accessible iOS Apps
- 129 Developing Accessible Symbian / Qt Apps
- 129 Developing Accessible Windows Phone & Windows 8 Apps
- 130 Developing Accessible Mobile Web Apps

133 **Enterprise Apps**

- 135 Mobile Device Management In The Enterprise
- 136 Mobile Enterprise Application Platforms

138 **Implementing Rich Media**

- 139 Streaming vs. Local Storage
- 140 Progressive Download
- 141 Media Converters

143 **Implementing Location-Based Services**

- 143 How To Obtain Positioning Data
- 145 How To Obtain Mapping Services
- 146 Implementing Location Support On Different Platforms
- 149 Tools For LBS Apps

- 151 Implementing Near Field Communication (NFC)**
- 153 Support For NFC
- 153 Creating NFC Apps

- 155 Implementing Haptic Vibration**
- 155 The iOS platform
- 155 The Android Platform
- 157 The bada Platform
- 158 BlackBerry Platform
- 158 Windows 7 Platform
- 158 Haptic Vibration Design Considerations

- 161 Security**
- 162 General Concepts
- 163 The Threats to Your Applications
- 164 Hiding the Map of Your Code
- 166 Hiding Control-Flow
- 167 Protecting Network Communications
- 167 Active Protection That Stays With The Application
- 168 White-Box Cryptography
- 168 Best Practices
- 169 Protection Tools
- 169 Resources
- 171 The Bottom Line

- 173 Testing**
- 173 Testing Through The Five Phases of an App's Lifecycle
- 175 Interactive Testing
- 179 GUI Test Automation
- 179 Headless Client
- 180 Beware Of Specifics
- 181 Testability: The Biggest Single Win
- 181 Test-Driven Development
- 182 Web-Based Content And Applications

- 184 **Monetization**
- 184 Pay Per Download
- 187 In-App Payment
- 188 Mobile Advertising
- 189 Revenue Sharing
- 190 Indirect Sales
- 190 Component Marketplace
- 192 Marketing And Promotion
- 193 Strategy
- 194 What Can You Earn?
- 194 Learn More

- 196 **Appstores**
- 196 Top 5 Appstores
- 197 Basic Strategies To Get High
- 199 Multi-Store vs Single Store

- 202 **Now What – Which Environment Should I Use?**
- 202 The Business Perspective: Market Reach
- 205 The Developer’s Perspective: Technology
- 207 The Developer’s Perspective: Marketing
- 210 The Developer’s Perspective: The Final Choice

- 212 **Epilogue**

- 213 **About the Authors**

Prologue

Welcome to the 11th edition of the Mobile Developers Guide To The Galaxy. Since our first edition in 2009 we have continued to cover various mobile technologies and to react to the ongoing changes in our industry; an industry where the smartphone adoption outpaces PC adoption in the 80s by around tenfold, if we want to believe Flurry Analytics. And we have seen some more big changes since our last edition in February 2012:

Android became the most dominant smartphone OS and proved that its UI can be buttery smooth. The eerie-yet-cool Google Now service impressed many and a major content offensive makes Android more compelling, at least in the USA. The first appstores that offered hacked apps were closed down by the FBI. Malware keep troubling Android, even though Google introduced several security mechanisms.

iOS introduced its sixth incarnation of its operating system. And while iPhones cannot keep up in device numbers with Android handsets, they frequently outnumber Android in earnings for developers even though that gap seems to be closing somewhat now. The tablet market remains in the firm grip of the iPad. New iOS 6 features include turn-by-turn navigation, 3D maps and deep Facebook integration. And iOS has shown us the dark side of a cloud-enabled world when writer Mat Honan's iPhone, iPad and Mac were remotely wiped using iCloud.

While the iPhone 5 brought LTE support to Apple's handsets, Microsoft surprised us with its own Windows 8 powered Surface tablets and is now ready to launch Windows 8 and Windows Phone 8. When Xbox 8 (or however they will call it) comes out, Microsoft's 3-screen-vision will become an interesting reality. So far people seem reluctant to change, but it hasn't been the first

time that people first need to use a system for a while before they learn to love it.

Research In Motion got a new CEO and concentrates on BlackBerry OS 10. Perhaps they will emerge like a Phoenix - er, or like Apple did with their OS X - with that all new approach? Otherwise OS 7.1 has been released but its relative market share keeps plummeting.

While you can still create AIR based Flash apps, Flash on mobile is pretty much dead in the water - and so we removed the corresponding chapter. Samsung's bada OS has received little love in 2012; and while Tizen has released a 1.0 SDK devices have been pushed back to 2013. Firefox OS seems to be getting more traction with first devices also expected in 2013. Nokia released its 808 PureView device, but otherwise things got quieter around Symbian - and Nokia even sold the Qt technology to Digia. Nokia focuses on Windows Phone 8 and Series 40 now.

Stupid patent wars keep on stifling the innovation. All the big players seem to be in a constant war with each other instead of focusing on their customers' needs. Point in case is the the record penalty of 1 billion US dollar for Samsung's alleged copying of the iPhone design.

On a more positive note, we have some editorial changes: Next to the Flash chapter we have also removed the Symbian one, as new mobile developers are unlikely to likely start with native Symbian development anymore. Last but not least we have new chapters dealing with BlackBerry 10 and Enterprise Apps.

We hope you enjoy this guide. Please get involved and let us know what content might be missing, or if you would like to contribute, at developers@enough.de!

Robert + Marco / Enough Software
Bremen, September 2012





An Introduction To Mobile Development

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

Form Factors and Usage Patterns

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

Android usage patterns differ from iOS ones, which differ from Windows Phone apps, et cetera.

You should, therefore, refrain from bringing the very same experience on all form factors or even all your target smart-phone systems. Otherwise you risk delivering a mediocre service to various sections of your target user base.

Mobile Service Options

There are several ways how you can realize a mobile service:

- App
- Website
- SMS, USSD¹ and STK²

App

An app runs directly on the device. It can be developed natively, with cross-platform tools, or as an HTML5/web app. Apps are typically distributed through the app stores. Pure web apps can be installed directly, an option that may be necessary depending on the nature of the app and the rules of the app store in question. A hyped controversy circles around whether native or web apps are the future. Famous examples include the Financial Times app which left the app store in order to keep all subscriber revenue to themselves for the web world; conversely the Facebook iOS app which was recently revamped as a native app in order to dramatically improve the performance and usability. For most mobile app developers this controversy does not really exist, as a hybrid approach to app development is quite common. An app can use native code for best performance and to integrate the app with the platform; and use a webview together with HTML5 based content for other parts of the same app.

There are many different operating systems to choose from:

¹ en.wikipedia.org/wiki/USSD

² en.wikipedia.org/wiki/SIM_Application_Toolkit

| Platform | Language(s) | Form factor | URL |
|------------------------------|------------------------------------|------------------------|---------------------------------------|
| Aliyun | Java, C, C++ | Smartphone | developer.aliyun.com |
| Android | Java, C, C++ | Smartphone, Tablet, TV | developer.android.com |
| bada | C, C++ | Smartphone | developer.bada.com |
| BlackBerry | Java, Web Apps | Smartphone | developer.blackberry.com |
| BlackBerry Playbook OS (QNX) | ActionScript, C++, HTML | Tablet | developer.blackberry.com |
| Brew MP | C | Featurephone | brewmp.com |
| iOS | Objective-C, C | Smartphone, Tablet | developer.apple.com/devcenter/ios |
| Nokia OS | Java ME | Featurephone | developer.nokia.com/Develop/Series_40 |
| Symbian | C, C++, Java, Qt, Web Apps, others | Smartphone | forum.nokia.com/symbian |
| Windows 8 | C#/VB.NET, C++, JavaScript | Tablet, PC | dev.windows.com |
| Windows Phone | C#, VB.NET, C++ | Smartphone | dev.windowsphone.com |



Additional mobile operating systems include:

| Platform | Language(s) | Form factor | URL |
|---------------|-------------------|--------------------|--|
| BlackBerry 10 | C, C++, HTML, AIR | Smartphone, Tablet | developer.blackberry.com |
| Firefox OS | HTML | Smartphone | mozilla.org/en-US/b2g |
| Mer | HTML, C/Qt | Smartphone | merproject.org |
| Tizen | HTML | Smartphone | tizen.org |

Website

A website runs for the most part on your server but you can access various phone features on the device with JavaScript, e.g. to store data locally or to request the current location of the device.

SMS, USSD and STK

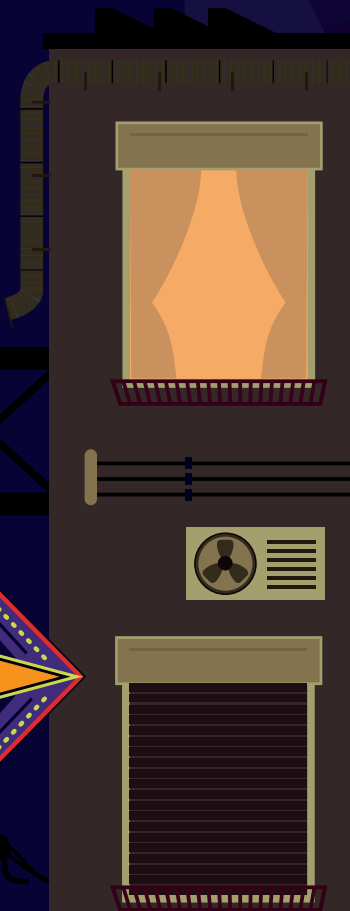
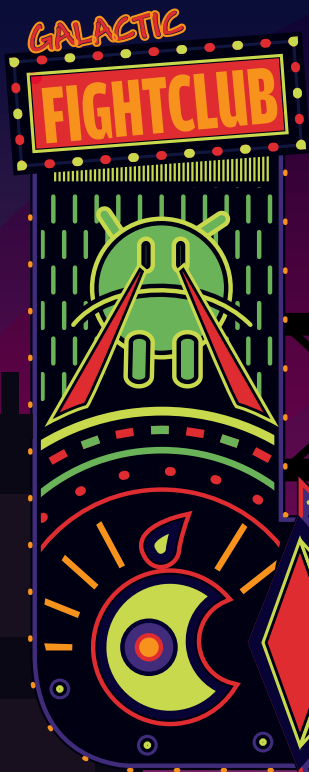
Simple services can be realized with SMS, USSD or STK. Everyone knows how SMS (Short Message Service) text messaging works and every phone supports SMS, but you need to convince your users to remember textual commands for more complex services. Some operators offer APIs for messaging services that work for WiFi-only devices, such as the network APIs of Deutsche Telekom³. USSD (Unstructured Supplementary Service Data) is a GSM protocol used for pushing simple text based menus, the range of support depends on the carrier and the device. STK (SIM Application Toolkit) allows to implement low-level but interactive apps directly on the SIM card of a phone.

³ www.developergarden.com/apis

Lost in the Jungle

If you are lost in the jungle of mobile development, don't 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 less well 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 already successful. This guide helps you to make an informed decision!





Android

The Android platform is developed by the Open Handset Alliance led by Google and has been publicly available since November 2007.

Android is an operating system, a collection of preinstalled applications and an application framework (Dalvik) supported by a comprehensive set of tools. Its use by many hardware manufacturers has made it the fastest growing smartphone operating system. According to Gartner¹, more than 64% of all smartphones sold in Q2 2012 worldwide were based on Android, and more than 600,000 apps are available in the Android Market². In September 2012 Google announced that there are a half billion Android devices activated and that this number grows by 1.3 million per day³. Android is also used in tablets, media players, set-top boxes, desktop phones and car entertainment systems. Some non-Android devices are also able to run Android applications with reduced functionality, such as RIM's Playbook with its virtual machine called App player⁴.

The platform continues to evolve rapidly, with the regular addition of new features every 6 months or so. At the last Google I/O in June 2012 Google announced the newest Android version called "Jelly Bean". Android 4.1 is intended to improve the experience created with Android 4.0, known as "Ice Cream Sandwich". Therefore a functionality called "Butter" was implemented which basically works as Triple Buffer and results in a much smoother navigation and a stable frame rate. Furthermore Google introduced an improved notification system with

1 www.gartner.com

2 www.zdnet.com/blog/burnette/live-from-google-io-2012/2597

3 mashable.com/2012/09/12/500-million-android-devices-activated/

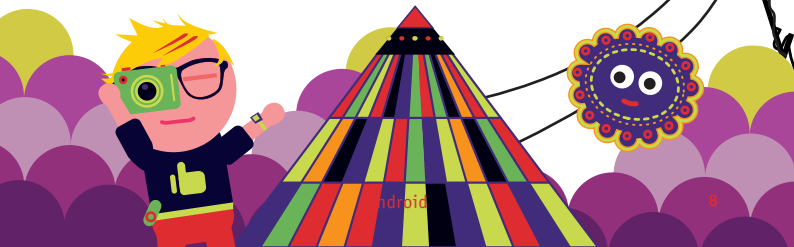
4 www.theregister.co.uk/2011/03/25/rim_playbook_android/

drop-downs, floating homescreen icons which automatically get positioned if you move icons towards them and USB-Audio support. There are many more tiny improvements and additions which show that Google has succeeded in getting Android to the state where it is a stable base for the future.

One of the most discussed issues when developing for Android is fragmentation: The multitude of different devices by different manufacturers and the fast progress of the platform itself leads to uncertainty over whether your Android application will run everywhere. In addition, only a very small number of phone and tablet models support the latest OS version. However, today you will reach 99.3% of the installation base if you decide to target Android 2.1 or above⁵. To reduce fragmentation issues caused by large differences in screen size, Android 3.2 (“Honeycomb”) introduced a new resource descriptor called “smallestWidth” which can be used to target phones and tablets with different layout depending on their dimensions⁶. To push solid user experience and consistent appearance of Android applications, Google published a design guide for Android apps available at developer.android.com/design/.

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

6 developer.android.com/guide/practices/screens_support.html#NewQualifiers



Prerequisites

The main programming language for Android is Java. But beware, only a subset of the Java libraries are supported and there are many platform specific APIs. You can find answers to your “What and Why” questions in the Dev Guide⁷ and to your “How” questions in the reference documentation⁸. Furthermore Google introduced a section in their documentation called “Android Training” which targets new developers who want to learn about best practices⁹, where developers can learn about basics like navigation and inter-app communication or more advanced features like intelligent Bitmap downloads and optimizing for less battery drainage.

To get started, you need the Android SDK¹⁰, which is available for Windows, Mac OS X and Linux. It contains the tools needed to build, test, debug and analyze applications. You will probably also want a good Java IDE. Eclipse or IntelliJ seem good choices. These IDEs offer good support for development, deployment and – importantly – library projects that enable the sharing of code and resources between projects.

Command line tools and Ant build scripts are also provided, so you can create almost any development and build process.

The Android Developer Tools for Eclipse bring nice features like Lint¹¹, which helps in improving applications by identifying common programming mistakes. Thankfully, the improvements are easy to implement.

⁷ developer.android.com/guide

⁸ developer.android.com/reference

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

¹⁰ developer.android.com/sdk

¹¹ tools.android.com/recent/lint

Implementation

An Android application is a mix of activities, services, message receivers and data providers, which are declared in the application 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, not tied directly to a visual representation.

A message receiver handles messages broadcast by the system or other applications. A data provider is an interface to the content of an application that abstracts from the underlying storage mechanisms. 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 done by intents.

An intent bundles data, such as the user's location or an URL, with an action. These intents trigger behaviors in the platform and can be used as messaging system in your application.

For instance, the intent of showing a web page will open the browser activity. The powerful thing about this building-block philosophy is that 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 user's preference and the applications installed: Any application that declares the sharing intent as their interface can be used.

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 such like.
- **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 or 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 and 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.

The user interface of an application 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 able to define layouts in code as well.

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 & IntelliJ in case you want to do some rapid prototyping by dragging around UI-elements in the editor.

If you are facing issues, such as exceptions being thrown, be sure to check the ddms log. It enables you to

check if you omitted to add all necessary permissions like `android.permission.INTERNET` in the `uses-permission` element¹².

If you are going to use Honeycomb, ICS or Jelly Bean related layout features – such as Fragments¹³ – for large screens, be sure to add the Android Compatibility package from Google. It is available through the SDK & AVD Manager and helps to develop for Android 3.0+ without causing problems with deployment to Android 1.6¹⁴ through to Android 2.3. Be sure to use the v4 packages in your application to provide maximum backwards support.

Developing your application against Android 3.1+, you will be able to make homescreen widgets resizable, and connect via USB to other devices, such as digital cameras, gamepads and many others.

Testing

The first step to test 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 might have changed pieces of the platform¹⁵. Therefore, testing on a physical device is essential.

¹² developer.android.com/reference/android/Manifest.permission.html

¹³ developer.android.com/guide/topics/fundamentals/fragments.html

¹⁴ android-developers.blogspot.com/2011/03/fragments-for-all.html

¹⁵ For an overview see androidfragmentation.com



Automated Testing

To automate testing, the Android SDK comes with some capable and useful testing instrumentation¹⁶ 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 application after these events have occurred. MonkeyRunner¹⁷ is a powerful and extensible test automation tool for testing the entire application. The test scripts are written in Python.

The automated tests can be run on virtual and physical devices. Open source testing frameworks, such as Robotium¹⁸ can complement your other automated tests. Robotium can even be used to test binary apk files, if the source is not available. A maven plugin¹⁹ and a helper for the continuous integration of a Hudson server may also assist your testing²⁰.

Signing

Your application is always be 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 application. Remember that you can use the same key for all your applications or create a new one for every app.

¹⁶ developer.android.com/guide/topics/testing/testing_android.html

¹⁷ developer.android.com/guide/developing/tools/monkeyrunner_concepts.html

¹⁸ code.google.com/p/robotium

¹⁹ code.google.com/p/maven-android-plugin/

²⁰ wiki.hudson-ci.org/display/HUDSON/Android+Emulator+Plugin

Distribution

After you have created the next killer application and tested it, you should place it in the Android Market. This is a good place to reach both customers and developers of the Android platform, to browse for exciting new apps and to sell your own apps. It is also used by other app portals as a source for app metadata. To upload your application to the Android Market, start at *market.android.com/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 application, add screenshots and descriptions and 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. As there are lots of competing applications in Android Market, you might want to use alternative application stores. They provide different payment methods and may target specific consumer groups²¹. One of those markets is the Amazon Appstore, which comes preinstalled on the Kindle Fire. For more information about appstores, please refer to the dedicated chapter in this guide.

Android 1.6 upwards also supports in-app purchase. This enables you to sell extra content, feature sets and such like from within your app, using the existing infrastructure of the Android Market²².

²¹ wipconnector.com/index.php/appstores/tag/android

²² developer.android.com/guide/market/billing/index.html

AWESOME



bada

bada is Samsung's proprietary smartphone platform and is based on open-source tools and software. bada was introduced in late 2009 and the first version of the SDK was released to the public in June 2010. Samsung's main reason for introducing bada was to accommodate the anticipated need for smartphone features at the low-end of the market. Because bada can run on top of a Linux kernel for high-end devices or a real-time OS kernels for low-end devices, all market segments can be served.

Samsung provides a documentation at *developer.bada.com*. This site offers a forum, premium support and direct access to Samsung bada experts as well.

Currently there are ten bada-based devices available with the Wave 3 being the current flagship device and Wave 578, the first device with NFC hardware running bada 2.0. According to Gartner, the overall market share of bada among smartphone sales has been 2.7% in Q2 2012 – exactly the same than Windows Phone¹.

bada 2.0 – released in September 2011 – introduced multitasking, which makes real background services possible. In addition, bada 2.0 supports Near Field Communication as well as the possibility for easy ad-hoc WiFi-P2P network setup from within the SDK. Other interesting features include enhancements to the UX with speech-to-text (STT) and text-to-speech (TTS), as well as support for 3D sound with OpenAL. Furthermore, the support for web-based applications is extended, with more JavaScript frameworks, HTML5 and a lot of APIs from the WAC 2.0 standard within the Webcontrol. Another interesting new feature is the MIME-type registration for applications, so

¹ gartner.com/it/page.jsp?id=2120015

that you can register applications to the system for handling specific file or media types, such as MP3.

Getting Started

You get start with developing for bada by registering at *developer.bada.com*, there is no charge for this. Next, download the bada SDK, which is available for Microsoft Windows based computers only. The SDK includes the bada IDE (based on Eclipse CDT), an emulator and a GNU toolchain.

Before starting to program you should be familiar with the application manifest, which is a unique application profile. This profile is needed to enable debugging and testing of applications on devices and distribution of apps through the store. A manifest can be generated and managed on *developer.bada.com* under the menu item “My Application”.

You can create feature-rich bada apps with the C++ framework for C++/ Flash-based applications or the Web framework for developing applications based on standard web technologies, such as HTML, CSS and JavaScript. The bada project templates, included in the IDE, provide a good starting point for bada application development for all of these technologies.

Implementation

After creating an application manifest you can start with app development using the bada SDK/IDE. The IDE has a plentiful library of example code and this code can be copied with one click into your own workspace. These examples are a great way to get familiar with the features of bada and its programming paradigm.

C++ based applications

Native bada apps are developed in C++. Some restrictions apply however, for example, the language does not use exceptions. Instead, it returns values and a combination of macros is used for error handling and RTTI is turned off, so that `dynamic_cast` does not work on bada.

When creating bada apps, you need to understand memory management basics, because the language often leaves this up to you. For example, the app will have to delete any pointer returned by a method ending in 'N'. You should also make sure that each new variable has a delete method:

```
MyType var = new MyType(); // call delete
MyType array = new MyType// call delete[ MyType
type, stackarray[];
// variable on stack will be destroyed by scope,
no delete
```

The API uses some parts of STL, so while Samsung says that STL can be used in code, be aware that the current STL implementation shipping with bada is missing some components. This can be addressed by using STLPort for full STL support. Similarly you can port modern C++ Libraries, such as Boost, to work on bada, but the lack of RTTI and exceptions can make it challenging work.

The bada API itself is wrapped in a number of namespaces. The API offers UI Control and Container classes, but there are no UI Layout management classes, so the UI elements must be positioned by hand or within the code. An UI layout for the landscape and/or the portrait mode is also your responsibility. The API provides most standard classes for XML, SQL or Network and a pretty complete framework. You should make use of the

callbacks for important phone events in the application class, such as low battery level or incoming calls.

When writing games for bada, the SDK supports OpenGL ES 1.1 and 2.0. The SDK wraps parts of OpenGL for use in its own classes, making it easy to port existing OpenGL code to bada.

Flash based applications

Flash based applications make use of the

`Osp::Ui::Controls::Flash` control of the c++ framework: so a Flash based app is simply a Flash movie played within a C++ bada app. This means you are able to take advantage of the entire C++ API feature set, such as geolocations and accelerometer. bada 2.0 is able to handle Adobe Flash Lite Version 4 and ActionScript 3.0. To create an interface with the Flash app, use the `fscommand2` and `SendDataEventToActionScript()` API calls. Following examples from the bada documentation will show this roundtrip:

```
fscommand2("Set","SendDataEvent", "/");
// Implement this callback method from the
// Osp::Ui::IFlashEventListener Interface to
respond to the ActionScript
// "Set" fscommand2 that we defined for the
PushButton Flash object.
void FlashForm::OnFlashDataReceived
(const Osp::Ui::Control &source, const
Osp::Base::Collection::IList &mList)
{
}
```

More information on using Flash as application platform can be found in the bada online help system under the menu entry

“bada Flash App Programming” (bada help > Flash Application Programming).

Web based applications

The bada web framework uses HTML, JavaScript and CSS. These standards are enriched by additional APIs (such as those defined by WAC) and other concepts. The web framework extends the object-oriented programming concepts of JavaScript with following paradigm:

- **Class:** Adopts object-oriented class concept with its definition, members and types
- **Inheritance:** Adds the inheritance feature of object-oriented programming by using the extend key to a drive a new class.
- **Mixins:** Mixins provides a way to extend the functionality of existing classes without using inheritance.

In addition you should be aware of following system limitations:

- Selected fonts from the Settings menu are not supported with web applications
- Only the bada 2.0 theme is supported in web applications
- Content links for download external content are not supported and queried sites from a link (`a href`) will be loaded into the same document page.

For basic information on mobile web programming, please see the respective chapter in this guide.

Resources

The central resource for bada developers is *developer.bada.com*.

The biggest independent bada website and forum is currently *BadaDev.com*, which has a good library of great tutorials about coding for bada. There is an IRC channel #bada at *irc.freenode.net*, and of course there are groups for bada developers on most social networks.

Testing

The bada API offers its own logging class and an AppLog method; you should make extensive use of logging in debug builds.

The AppLog will show up in the IDE. The IDE provides for testing and debugging in the simulator or on a device. As mentioned earlier in this guide, we strongly recommend testing on devices. Without device testing you cannot be sure how the app will perform and, in rare cases, code that worked perfectly on the simulator will not do so on the handset.

Samsung provides the bada Remote Test Lab (RTL), which is available for all registered developers, and can be installed as an Eclipse-plugin.

Tools and frameworks for unit testing are available within the IDE/SDK. For details about these tools, check out the “bada Tutorial Development *Environment.pdf*” included in the documents folder in the SDK base directory.

Another new tool that was introduced with bada 2.0 SDK is a code coverage and performance-monitoring tool, which enables code optimizations.

Distribution

There is only one option for distributing bada apps, Samsung's own appstore. Samsung's application store gives developers a route to market and includes features such as sales and download statistics, advertising and a direct feedback channel for customers. The store is accessible to customers through a website², a client application on bada smartphones and a PC client called Samsung's Kies. Applications can be offered as paid apps or free. In the case of a purchased app, you will receive 70% of sales, which is the same offer as most other popular mobile application stores.

As with Apple's AppStore, there are quite strict acceptance rules for apps submitted. You can find out more in the "Samsung Apps Publisher Guide," which can be downloadable after registering at the Samsung Apps Seller Office.

For advertising Samsung own advertising service – AdHub – but also allows the inclusion of third party ad network contents

2 samsungapps.com





BlackBerry Java Apps

The BlackBerry platform is developed by Canadian company Research In Motion (RIM)¹ and was launched in 1999. BlackBerry devices became extremely popular because they were equipped with a full keyboard for comfortable text input (which spawned a condition named BlackBerry Thumb²), offered long battery life and included BlackBerry Messenger, their mobile social network offering. Add PDA applications such as address book, secure email, calendar, tasks and memopad to these features and you will understand why the platform is very popular among business and mainstream users alike.

The overall market share of BlackBerry phones has continued to decline in 2012³, but it is still an important smartphone platform. Furthermore, RIM's offering in the tablet market – the PlayBook – has received an important upgrade in the form of BlackBerry Tablet OS 2.0. The new OS comes with a slew of compelling features and improvements, such as support for Android apps, built-in Documents to Go mobile office suite and improved web browsing.

In this chapter we concentrate on Java development for BlackBerry smartphones running the current Blackberry OS (up to version 7.1). RIM's next-generation OS, Blackberry 10, is covered in a separate chapter of the guide.

¹ rim.com

² wikipedia.org/wiki/Blackberry_thumb

³ gs.statcounter.com

Prerequisites

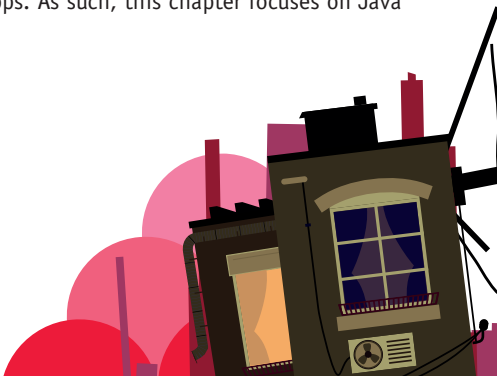
BlackBerry OS is the operating system found on all current BlackBerry smartphones. Its latest iteration released in January 2012 (BlackBerry OS 7.1) offers some notable improvements over its predecessor (BlackBerry OS 7): tethering, WiFi calling support, NFC Tag support and FM radio.

The most relevant API additions in OS 7.1 are:

- **NFC Peer-to-Peer API**, which offers the ability to initiate data transfers between two devices via NFC, then complete the transfer via Bluetooth
- **FM Radio API**
- **Profiles API**, which allows read/write access to the user's current profile

For BlackBerry OS, two development approaches are available depending on the type and nature of your planned project. For mid-sized to large applications native Java development is the best choice; while small apps could be developed with the BlackBerry WebWorks SDK.

Although it will be phased out in the future, currently the BlackBerry Java API is the most commonly used method to develop BlackBerry apps. As such, this chapter focuses on Java development.



Java SDK

As for all Java-driven application development, you need the Java SDK⁴ (not the Java Runtime Edition).

IDE

For native Java development, you first need to decide which IDE to use. The modern option is to use Eclipse and the BlackBerry plugin⁵, for previous BlackBerry OS versions you can also use the BlackBerry Java Development Environments (JDEs)⁶.

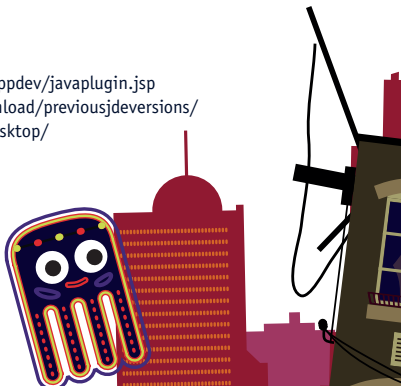
These JDEs are complete environments enabling you to write, compile, package and sign your applications. Device simulators are included as well.

Desktop Manager

The BlackBerry Desktop Manager⁷ should be downloaded and installed.

It enables you to deploy your app package on to a device for testing. For faster deployment, you might also use a tool called javaloader that comes with the JDE.

- 4 oracle.com/technetwork/java
- 5 us.blackberry.com/developers/javaappdev/javaplugin.jsp
- 6 developer.blackberry.com/java/download/previousjdeversions/
- 7 us.blackberry.com/apps-software/desktop/



Coding Your Application

The BlackBerry JDE is partly based on Java ME and some of its JSR extensions: Integrated into the SDK is the MIDP 2.0 standard with popular JSR extensions that provide APIs for UI, audio, video, and location services among others⁸. 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.

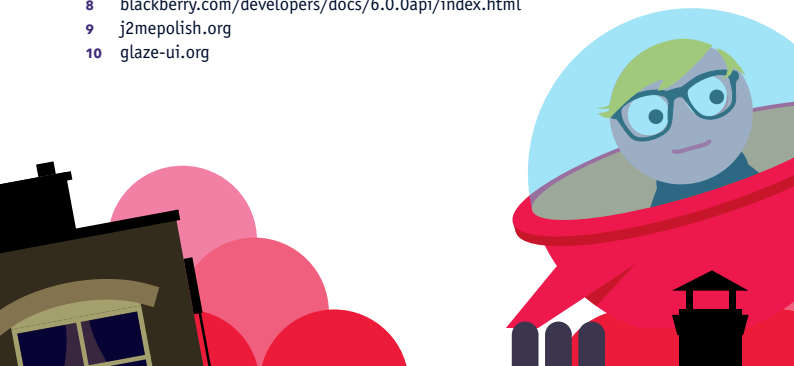
Native UI components can be styled to an extent, but they inherit their look from the current theme. This can be prevented in code, by overriding the `Field.applyTheme()` method for each component/field.

From OpenGL-ES to homescreen interaction and cryptography, the BlackBerry APIs provide you with everything you need 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 Polish⁹ or Glaze¹⁰ which enable you to design and animate your UI using CSS.

⁸ blackberry.com/developers/docs/6.0.0api/index.html

⁹ j2mepolish.org

¹⁰ glaze-ui.org



Services

BlackBerry offers many services that can be useful in developing your applications including advertising, mapping, payment and push services¹¹.

The push service 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, instead of the device continuously polling the server to find out if updates are available and then pulling the updates from the server. This reduces network traffic, battery usage and, for users on metered data plans or roaming, lowers costs.

The push service¹² works as follows: Your server sends a data package of up to 8KB to the BlackBerry push infrastructure. The infrastructure then broadcasts the message to all or a group of clients (for content such as a news report) or to one specific client (for content such as a chat message). The device client then receives the message through BlackBerry's Push API and may confirm message receipt back to the infrastructure. Your server can then check if the message was delivered. BlackBerry offers the push mechanism as a limited free service, with a premium paid extension that enables you to send more push messages.

¹¹ developer.blackberry.com/services/#platform

¹² us.blackberry.com/developers/platform/pushapi.jsp

Testing

BlackBerry provides simulators for various handsets in the JDE and plug-ins or as separate downloads. These simulators enable you to run an app on a PC in the same way it would be run on a device. To assist with testing, the simulators include features such as simulating incoming calls and setting the signal strength enabling you to check how your application reacts if a device is outside network coverage. Applications running on the emulators are fully debuggable with breakpoints.

As a great plus, BlackBerry devices provide the capability to perform on-device debugging with all the features that you enjoy from the simulators.

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 and functionality are only available on specific OS versions. For example the FilePicker that is used to choose a file is only available from OS 5.0 onwards.
- You need to handle different screen resolutions and orientation modes (landscape and portrait).
- You need to handle touch and non-touch devices. In addition, the Storm devices use a touchscreen that is physically clickable, so there is a distinction between a touch and a click on these devices. BlackBerry's more recent touch devices do not use this technology anymore.

Porting to other Java platforms such as Java ME and Android is complicated as it is not possible to port the BlackBerry UI.

Code written for server communication or storage might be reused on Java ME and Android if you avoid native BlackBerry API calls. In general, cross-platform portability strongly depends on how frequently your app uses native BlackBerry components.

For example it is not possible to reuse BlackBerry push services classes on other platforms.

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 BlackBerry¹³. The signing itself is undertaken using the `rapc` tool, which also packages the application.

Distribution

BlackBerry's own distribution channel is called App World¹⁴ where you can publish your apps. For paid applications, you'll get a 70% revenue share. In addition, GetJar¹⁵ is a well-known independent website that also publishes BlackBerry apps.

¹³ blackberry.com/SignedKeys/

¹⁴ appworld.blackberry.com

¹⁵ getjar.com



BlackBerry 10

The BlackBerry 10 platform (BB10) is a general relaunch from RIM. It will only be available on next-gen devices - there are no upgrade plans for current generation devices. RIM has taken this approach in order to catch-up with competing mobile operating systems: iOS, Android and Windows Phone 8. Although the OS is entirely new, the other parts of the BlackBerry ecosystem, like the App World or the push-service, will not change. Security-wise, BB10 will be built to the same high standards for which the BlackBerry platform is well known. One of BB10's major goals is to harmonize mobile phones and tablets by having them run the same operating system and the same apps, something not currently possible in the BlackBerry ecosystem. BB10 is scheduled to arrive in Q1 2013.

The Alpha Device

RIM offers developers a physical device called Dev Alpha¹ in order to convince developers to adopt the new platform and to update their existing apps.

Developers can get this device on the Blackberry JAM Tour², a series of conferences held all over the world. The DevAlpha is not a final product, but it does showcase how the new OS will work and what hardware and software features developers can expect from the new BB platform. This helps developers write and test their apps on real hardware as well as on simulators. The BlackBerry 10 Dev Alpha has a 4.2-inch 1280x768 HD LCD (356dpi) touch screen, internal storage of 16GB, two cameras, a Micro HDMI port for video output, Bluetooth and WiFi, a

¹ developer.blackberry.com/blackberry10devalpha

² blackberryjamconference.com

network unit capable of quad band HSPA+ (with no LTE support as of this writing) and a Micro USB port for debugging and charging.

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 (Android 2.3.3 compatibility layer)

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 current BlackBerry devices need to re-orient themselves to one of the technologies previously mentioned. As not all developers will be 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.

As RIM is fully aware of these concerns, they provide a rich set of resources for developers: the Dev Alpha device, simulators, sample projects on GitHub³ and frequently updated documentation⁴.

³ github.com/blackberry

⁴ developer.blackberry.com/platforms/bb10

C Native SDK

The BlackBerry NDK supports many open standards that allow developers to bring their existing apps to the platform. To get started, there is a Native Dev Site⁵ for this. With the C Native approach 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 SDK⁶
- Install and configure the simulator⁷
- Configure your environment for development and deployment
- Create your first project
- Run sample applications

As a new addition, Blackberry added Scoreloop⁸ 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.

⁵ developer.blackberry.com/native/beta

⁶ developer.blackberry.com/native/beta/download

⁷ developer.blackberry.com/native/beta/download

⁸ developer.blackberry.com/native/beta/documentation/scoreloop

Some of the features currently available include:

- User profile
- Leaderboards
- Challenges
- Awards and achievements

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 implementation. 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 a Markup-Language called QML⁹. 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

⁹ en.wikipedia.org/wiki/QML



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 QSql, 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 with Core Animation.

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 let 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.

To get further information, there is a Cascades Dev Site¹⁰ 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 SDK¹¹. Among other features, it allows you to write regular webpages and then package them as native BlackBerry apps with ease. If you want to mimic the Blackberry-UI style in HTML, there is a project on GitHub to

¹⁰ developer.blackberry.com/cascades

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

help you. It's called *BBUI.js*¹². It provides extensive CSS to make your regular webpage look like a native BlackBerry-UI application.

The second tool is the Ripple Emulator¹³. It is a Chrome Browser extension that acts as a BlackBerry 10 device simulator for WebWorks apps. It even emulates hardware-specific features, such as the accelerometer and the GPS sensor.

To get more information about developing with WebWorks there is a HTML5 Dev micro-site¹⁴ with more information.

Adobe Air

If you are an existing AIR developer 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, 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

¹² github.com/blackberry/bbUI.js

¹³ developer.blackberry.com/html5/download/ripple

¹⁴ developer.blackberry.com/html5

- 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 website¹⁵.

Android Runtime

You can use the BlackBerry Runtime for Android apps to run Android 2.3.3 platform applications on BlackBerry 10. 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.

¹⁵ developer.blackberry.com/air/beta

— **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 App World 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 website¹⁶.

Testing

BlackBerry continues to provide simulators for BB10 handsets as separate downloads¹⁷. These simulators enable 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 simulators include features such as simulating incoming calls and setting the signal strength, thus enabling you to check how your application reacts in real-world scenarios.

¹⁶ developer.blackberry.com/android

¹⁷ us.blackberry.com/sites/developers/resources/simulators.html

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. To achieve this, you need to obtain a signing key directly from BlackBerry¹⁸. The signing itself is undertaken using the rapc tool, which also packages the application. There is no difference between signing apps for BB10 signing and signing for previous OS versions.

Distribution

As with all previous OS versions, BB10 apps are distributed via App World¹⁹. For paid applications, developers get a 70% revenue share.

¹⁸ developer.blackberry.com/CodeSigningHelp/codesignhelp.html

¹⁹ appworld.blackberry.com





iOS

iOS, running on the iPhone, iPod touch and iPad, is a very interesting and very popular development platform, one commonly stated reason for this being the App Store. When it was introduced in July 2008, the App Store took off like no other marketplace had before. Now there are more than 725,000 applications in the App Store, and the number is growing daily. This reflects the success of the concept, but it means that it is getting ever harder to stand out in this mass of applications.

In March 2012, Apple announced that users have downloaded more than 25 billion iOS apps¹. Nearly every quarter year, device sales are reaching new all-time highs and there is no sign of a slowdown in the billion downloads per month. Over 400 million iOS devices have been sold to users willing to try apps and pay for content, making the App Store one of the most economically interesting targets for mobile app development.²

The iOS SDK offers high-level APIs for a wide range of tasks, which helps to cut down on development time. New APIs are added in every major update of iOS for iPhone, such as MapKit in iOS 3.0, (limited) multitasking in iOS 4.0, and Game Center in iOS 4.1.

The iPad, which went on sale in April 2010, uses the same operating system and APIs as the iPhone, therefore the skills acquired in iPhone development can be used in iPad development too. A single binary can even contain different versions for both platforms with large parts of the code being shared.

- 1 www.apple.com/pr/library/2012/03/05Apples-App-Store-Downloads-Top-25-Billion.html
- 2 techcrunch.com/2012/09/12/apple-has-sold-over-400-million-ios-devices-9-5-growth-since-march

Since the release of iOS 4.2, in November 2010, all iOS devices sold have used a common firmware version. This absence of fragmentation makes it possible to develop universal apps for multiple device classes much more easily than on other mobile platforms.

iOS 5.0, released in Q4 2011 includes various new features and over 1,500 new APIs for developers. One of the most interesting is iCloud, which provides for easy cloud storage of application-specific data, documents and easy-to-implement Twitter functionality. At WWDC 2012, Apple announced iOS 6.0, which is estimated to be released in Q4 2012 and will introduce a number of new features, such as Facebook integration, Passbook and turn-by-turn navigation in Maps.

Prerequisites

Apple's iOS SDK

In order to develop iPhone (and iPod Touch and iPad) apps, you will need the iOS SDK, which can be downloaded at developer.apple.com/devcenter/ios/index.action. This requires a membership, which starts at USD 99/year. If you do not plan on distributing your apps in the App Store and do not wish to test your apps on an actual device, you can also download Xcode from the Mac App Store for free.

The iOS SDK contains various applications that will allow you to implement, test, and debug your apps. The most important applications are:

- **Xcode**, the IDE for the iOS SDK
- **Interface Builder**, to build user interfaces for iPhone app (integrated into Xcode as of Xcode 4.0)
- **Instruments**, which offers various tools to monitor app execution

- **iOS Simulator**, which enables you to test apps quickly, rather than deploying them to a device.

The iOS SDK will work on any Intel-based Mac running Mac OS X 10.7 (Lion) or 10.8 (Mountain Lion).

A guide to get you started and introduce you to the tools is included in the SDK, as is a viewer application for API documentation and sample code. References and guides are also available online at developer.apple.com/library/ios/navigation/.

The SDK includes a large number of high-level APIs separated into a number of frameworks and libraries, which include:

- **Cocoa Touch**, which consists of the UI libraries, various input methods such as multi-touch and accelerometer
- **Media frameworks**, such as OpenAL, OpenGL ES, Quartz, Core Animation and various audio and video libraries
- **Core Services**, such as networking, SQLite, threading and various other lower level APIs.

The list of available frameworks grows with each major release of the iOS firmware.

Alternative Third-Party Development Environments

Since Apple relaxed their App Store distribution guidelines, development using tools other than Objective-C, Cocoa Touch and Xcode is officially permitted again and most commonly used in game development, for example using the Unreal Development Kit³, which Epic released for iOS to much fanfare in December 2010.

Using third party development environments and languages for iOS development offers a number of advantages and disadvantages.

The major advantage being that it is easy to support multiple platforms from a single code base without having too much of a maintenance burden. However, as experience with desktop software has shown, cross-platform software development rarely produces apps of outstanding quality. In most cases the cross-platform tool concentrates on the lowest common denominator and the resulting product does not feel like it really belongs on any of the targeted platforms.

For an overview on cross-platform technologies in general, please see the corresponding chapter in this guide.

There are, however, third party development environments that focus solely on iOS development, such as MonoTouch⁴.

This platform enables developers to build iOS apps using C# and .NET while taking advantage of iOS APIs. This makes it the alternative that comes closest to the original SDK, while still allowing code re-use, for example when creating similar Windows Phone 7 apps.

Some alternative IDEs carry additional fees, which are in addition to Apple's yearly development program charge and their 30% cut of all sales. Given the drawbacks of cross-platform development mentioned earlier, using third party IDEs makes the most sense for games, which can share almost all their code between different platforms. Java IDE makers JetBrains recently released an Objective-C IDE of their own, called AppCode⁵.

Implementation

Usually, you will want to use Apple's high-level Cocoa Touch APIs when developing for the iPhone. This means that you will write Objective-C code and create your user interfaces in Interface Builder, which uses the proprietary XIB file format.

⁴ monotouch.net

⁵ www.jetbrains.com/objc/

Objective-C is, as the name suggests, a C-based object-oriented programming language. As a strict superset of C, it is fully compatible with C, which means that you can use straight C source code in your Objective-C files.

If you are used to other object-oriented languages such as C++ or Java, Objective-C's syntax might take some time getting used to, but is explained in detail at developer.apple.com. What separates Objective-C most from these languages is its dynamic nature, lack of namespace support and the concept of message passing vs. method calls.

A great way to get you started is Apple's guide "Your First iPhone Application", which will explain various concepts and common tasks in an iPhone developer's workflow⁶. Also check out some of the sample code that Apple provides online⁷ to find out more about various APIs available to you.

Testing

As performance in the iPhone Simulator can be superior to the performance on a device, it is absolutely vital that testing is carried out on devices. It is highly recommended that you have at least one example of each class of device you want to deploy your apps on.

For example, an iPhone-only app shouldn't need to be tested separately on an iPad. However, it cannot hurt to have several classes of device, including older models, since problems such as excessive memory consumption sometimes will not present themselves on newer hardware.

Testing on real devices is also important because touch-based input is completely different from the pointer-driven UI model.

⁶ developer.apple.com/iphone/manage/distribution/distribution.action

⁷ developer.apple.com/iphone/library/navigation/SampleCode.htm

End-user testing can be achieved by distributing builds of the application to as many as 100 testers, through Ad-Hoc Provisioning, which you can set up in the Program Portal⁸. Each iPhone (and iPad/ iPod touch) has a unique identifier (UDID – universal device identifier), which is a string of 40 hex characters based on various hardware parts of the device.

If you choose to test using Ad-Hoc-Provisioning, simply follow Apple’s detailed set-up instructions⁹. Every single step is vital to success, so make sure that you execute them all correctly.

With iOS 4.0, Apple has introduced the option for developers to deploy Over-The-Air (OTA) Ad-Hoc builds of their apps to beta testers. There are open source projects¹⁰ to facilitate this new feature, as well as commercial services such as Testflight¹¹ and HockeyApp¹².

Google Toolbox for Mac¹³ runs test cases using a shell script during the build phase, while GHUnit¹⁴ runs the tests on the device (or in the simulator), allowing the developer to attach a debugger to investigate possible bugs. Version 2.2 of the SDK Apple included OUnit; an example of how to create the unit tests is available online¹⁵.

In iOS 4.0 Apple introduced a new tool, UIAutomation that aims to automate the testing of your application by scripting touch events. UIAutomation tests are written in JavaScript

- 8 developer.apple.com/iphone/library/referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/index.html
- 9 developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhone101/Articles/00_Introduction.html
- 10 github.com/therealkerni/hockeykit
- 11 www.testflightapp.com
- 12 www.hockeyapp.net
- 13 code.google.com/p/google-toolbox-for-mac
- 14 github.com/gabriel/gh-unit
- 15 www.mobileorchard.com/ocunit-integrated-unit-testing-in-xcode

and a full reference is available in the iOS Reference Library¹⁶. Several other third party testing automation tools for iPhone applications are available, including FoneMonkey¹⁷ and Squish¹⁸.

Distribution

In order to reach the broadest possible audience, you should consider distributing your app on the App Store. There are other means, such as the Cydia Store for jailbroken iOS devices, but the potential reach is not nearly as large as the App Store's.

To prepare your app for the App Store, you will need a 512x512 version of your app's icon, up to five screen shots of your app, and a properly signed build of your app. Log in to iTunes Connect and upload your app according to the onscreen instructions.

After Apple has approved your application, which usually should not take more than 2 weeks, your app will be available to customers in the App Store. Due to past app submission rejections, the approval process receives more complaints than any other aspect of the iPhone ecosystem. Recently, Apple has released their full App Store testing guidelines in order to give developers a better chance to evaluate their app's likelihood of being approved. Also, the restrictions have been relaxed and apps that were previously rejected have been approved after being resubmitted.

Approximate review times as experienced recently by other developers are gathered at *reviewtimes.shinydevelopment.com*¹⁹

¹⁶ developer.apple.com/library/ios/#documentation/DeveloperTools/Reference/UIAutomationRef/_index.html

¹⁷ www.gorillalogic.com/fonemonkey

¹⁸ www.froglogic.com/products

¹⁹ reviewtimes.shinydevelopment.com

for your information. However, there is no guarantee that an app will be approved in the timeframe specified on the site. This should be used as a guideline only.

Books

A number of great books have been written on iOS development. Here is a short list, which is by no means complete, of good tutorials and references:

Beginner books

These books are best for someone looking into getting started with iOS development.

- **iPhone SDK Development** by Bill Dudney and Chris Adamson
- **Beginning iPhone 3 Development** by Dave Mark & Jeff LaMarche



Intermediate books

Books suited for those who have had some exposure to the iOS SDK and are looking to deepen their knowledge of the platform.

- **More iPhone 3 Development** by Dave Mark and Jeff LaMarche
- **Programming in Objective-C 2.0** by Stephen Kochan

Professional books

If you already have a good knowledge of the iOS SDK, one of these books is sure to increase your skill set.

- **Cocoa Design Patterns** by Erik M. Buck and Donald A. Yacktman
- **Core Data** by Marcus Zarra

Companion books

Books that every aspiring iOS developer should call their own, because they impart knowledge beyond programming; such as the importance of user experience, using case studies and personal experiences.

- **Tapworthy** by Josh Clark
- **App Savvy** by Ken Yarmosh

Community

One of the most important aspects of iOS development is the community. Many iOS developers are very forthcoming and open about what they do, and how they did certain things.

This activity has become even more visible as Twitter and Github have gained momentum and become widely-known.

Search for iPhone, iPad or any other related search terms on *Github.com* and you will find a lot of source code, frameworks, tutorials, code snippets and complete applications – most of them with very liberal licenses that even allow for commercial use.

Practically all of the most important and most experienced iOS developers use Twitter to share their thoughts about the platform. There are many comprehensive lists of iOS developers available, a notable and well-curated one being Robert Scoble's list²⁰. Following such a list helps you stay up to date on current issues and interesting information about iOS development generally. What makes the community especially interesting is that many iOS developers pride themselves on taking an exceptional interest in usability, great user experience and beautiful user interfaces. You can usually find out about the most interesting trends on blog aggregators such as *CocoaHub.de* and *PlanetCocoa.org*.

²⁰ www.twitter.com/Scobleizer/iphone-and-ipad



Colossal

café



Java ME (J2ME)

J2ME (or Java ME as it is officially called) 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. It has been very successful in this market segment, with an overwhelming majority of feature phones supporting it. J2ME is also supported natively on Symbian and current BlackBerry smartphones.

J2ME's major drawback is that, due to its age and primary market segment, it does not fare all that well compared to more modern smartphone platforms, such as Android, iPhone, BlackBerry and Windows Phone: it offers a less powerful set of APIs, often runs on less powerful hardware and tends to generate less money for the developer. As a consequence, J2ME's popularity in the developer community has declined significantly in recent years.

So why would you want to develop for J2ME? Mainly for one reason: market reach. In Q2 2012, smartphone sales still accounted only for 36.7% of total mobile phone sales worldwide¹. The majority of devices are still feature phones which usually support Java ME. So if your business model relies on access to as many potential customers as possible, or on providing extra value to existing customers via a mobile application, then J2ME might still be a great choice.

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, you might want to consider targeting a different platform (such as Android, BlackBerry, iPhone or Windows Phone).

¹ gartner.com/it/page.jsp?id=2120015

Prerequisites

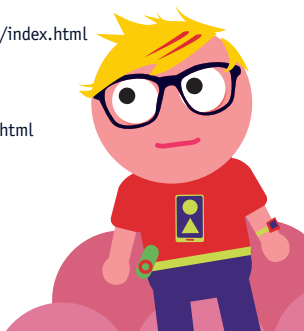
To develop a Java ME application, you will need:

- The Java SDK² (not the Java Runtime Environment) and an IDE of your choice, such as Eclipse Pulsar for Mobile Developers³, NetBeans⁴ with its Java ME plug-in or IntelliJ⁵.
- An emulator, such as the Wireless Toolkit⁶, the Micro Emulator⁷ or a vendor specific SDK or emulator.
- Depending on your setup you may need an obfuscator like ProGuard⁸. If you build applications professionally you will probably want to use a build tool such as Maven⁹ or Ant¹⁰ also.
- You may want to check out J2ME Polish, the open source framework for building your application for various devices¹¹.

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

Beginners often like to use NetBeans, with the Java ME plug-in installed. Also download and read the JavaDocs for the most important technologies and APIs: You can download most Java-Docs from *www.jcp.org*. There are a couple of useful vendor

- 2 oracle.com/technetwork/java/javame/downloads/index.html
- 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



specific APIs that should be tracked down manually from the vendor's pages (such as the Nokia UI API and Samsung APIs).

Implementation

The Java ME platform is fairly straight-forward: it comprises the Connected Limited Device Configuration (CLDC)¹² and the Mobile Internet Device Profile (MIDP)¹³, both are quite easy to understand. These form the basis of any J2ME environment and provide a standardized set of capabilities to all J2ME devices. 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). JSRs exist for everything from accessing the device's built-in calendar, address book and file system (JSR 75); to using the GPS (JSR 179) and Near Field Communication (JSR 257). For a comprehensive list of JSRs related to Java ME development, visit the Java Community Process' "List by JCP Technology"¹⁴.

It is very important to remember that not all JSRs are available on all devices, so capabilities available on one device might not be available on another device, even if the two devices have similar hardware.

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 mini-

¹² java.sun.com/products/cldc/overview.html

¹³ java.sun.com/products/midp/overview.html

¹⁴ jcp.org/en/jsr/tech?listBy=1&listByType=platform

mized; it cannot run in the background. Some devices support concurrent application execution, so it is possible for applications to run in the background. However, this usually requires the use of vendor-specific APIs and/or relies on device-specific behavior, which can cause fragmentation issues.

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.

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 or JSR 287¹⁵.

In addition, you will find that some manufacturers provide additional UI features. For example, Nokia recently introduced the Touch and Type UI to its Series 40 platform. To enable developers to make best use of this UI in their applications, the Nokia UI API was extended to provide features to capture screen gestures and provide controlling data for UI animations. Similarly Samsung provide pinch zoom features in their latest Java ME APIs.

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

All of them use low-level graphics to create better looking



¹⁵ jcp.org/en/jsr/detail?id=287

and more powerful UIs than are possible with the standard highlevel LCDUI components.

1. J2ME Polish¹⁶: 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. LWUIT¹⁷: A Swing inspired UI framework
3. Mewt¹⁸: Uses XML to define the UI

One very important aspect to consider when designing your UI is the typical screen resolution for Java ME devices. The vast majority of Java ME devices have one of the following resolutions: 240x320, 176x208, 176x220, 128x160, 128x128 or 360x640 pixels. By far the most popular is 240x320, while 360x640 is a common resolution for high-end Java ME devices (typically those running Symbian or Blackberry) and 176 x 208/220 is a common resolution for low-end devices. You will also encounter devices that have these resolutions in landscape, for example 320x240 instead of 240x320 pixels.

Handling so many different resolutions can be a challenge.

Your best approach is to create UI layouts that can scale well across all of them, in the same way that web pages scale well across different browser window sizes. You can also create custom UIs for each resolution, though this is not recommended because it is time consuming, error prone and expensive.

Another aspect worth considering is the size of your application's assets, especially its graphical assets. Whenever possible, your assets should be optimized, in order to keep your application's size as small as possible. This results in cheaper downloads for your users (as less data traffic is needed) and

¹⁶ j2mepolish.org

¹⁷ lwuit.java.net/

¹⁸ mewt.sourceforge.net

greater market reach (as some devices have a limit on the maximum application size). A great free tool for this is PNG-Gauntlet¹⁹, which can optimize your graphical assets without compromising quality.

Despite the platform's limitations, it is quite possible to create great looking and easy to use Java ME user interfaces, particularly if one of the tools mentioned above is used.

Open Source

There is a rich open source scene in the J2ME sector. Interesting projects can be found via the blog on opensource.ngphone.com.

You will also find fascinating projects on the Mobile and Embedded page of java.net²⁰, for example the Bluetooth project Marge²¹.

Testing

Because of the fragmentation in the various implementations of Java ME, testing applications is vital. Test as early and as often as you can on a mix of devices. Some emulators are quite good (personal favorites are BlackBerry and Symbian), but there are some things that have to be tested on devices.

Thankfully, vendors like Nokia²² and Samsung²³ provide subsidized or even free remote access to selected devices.

¹⁹ pnggauntlet.com

²⁰ community.java.net/mobileandembedded/

²¹ marge.java.net/

²² forum.nokia.com/rda

²³ innovator.samsungmobile.com



Automated Testing

There are various unit testing frameworks available for Java ME, including J2MEUnit²⁴, MoMEUnit²⁵ and CLDC Unit²⁶; System and UI testing is more complex given the security model of J2ME, however JInjector²⁷ is a flexible byte-code injection framework that supports system and UI testing. Code coverage can also be gathered with JInjector.

Porting

One of the strengths of the Java environment for mobile devices is that it is backed by a standard, so it can be implemented by competing vendors. The downside is that the standard has to be interpreted, and this interpretation process can cause differences in individual implementations. This results in all kinds of bugs and non-standard behavior. In the following sections we outline different strategies for porting your applications to all 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. In the J2ME world this usually means CLDC 1.0 and MIDP 1.0. If you only plan to release your application in more developed countries/ regions, you may consider targeting CLDC 1.1 and MIDP 2.0 as the lowest common denominator (without any additional APIs or JSR support).

Depending on the target region for the application you might also consider using Java Technology for the Wireless

²⁴ j2meunit.sourceforge.net

²⁵ momeunit.sourceforge.net

²⁶ snapshot.pyx4me.com/pyx4me-cldcunit

²⁷ code.google.com/p/jinjector

Industry (JTWI, JSR 185) or the Mobile Service Architecture (MSA, JSR 248) as your baseline. Both extensions are designed to ensure a common implementation of the most popular JSRs. They are supported by many modern devices and provide many more capabilities to your applications. However, in some regions such as Africa, South America or India you should be aware that using these standards may limit the number of your potential users, because the more common handsets in these regions do not implement those extensions.

Using the lowest common denominator approach is typically easy: There is less functionality to consider. However, the user experience may suffer if your application is limited in this way, especially if you want to port your application to smartphone platforms later. So this approach is a good choice for simple applications – for comprehensive, feature-rich applications it may not be the way to go.

Porting Frameworks

Porting frameworks help you deal with fragmentation by automatically adapting your application to different devices and platforms. Such frameworks typically feature the following components:

- Client libraries that simplify development
- Build tool chains that convert code and resources to application bundles
- Device databases that provide information about devices
- Cross compilers to port your application to different platforms

For Java ME some of the options you can choose from are: Celsius from Mobile Distillery²⁸ that is licensed per month,

Bedrock from Metismo²⁹ that provides a suite of cross compilers on a yearly license fee and J2ME Polish from Enough Software³⁰ that is available under both the GPL Open Source license and a commercial license. Going in the other direction (from C++ to Java ME) is also possible with the open source MoSync SDK³¹.

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

Good frameworks enable you to use platform and device specific code in your projects, so that you can provide the best user experience. In other words: a good porting framework does not hide device fragmentation, but makes the fragmentation more manageable.

Signing

The Java standard for mobile devices differentiates between signed and unsigned applications. Some handset functionality is available to trusted applications only. Which features are affected and what happens if the application is not signed but uses one of those features, is largely dependent on the implementation.

On one phone the user might be asked once to enable the functionality, on another they will be asked every time the feature is used and on a third device they will not be able to use the feature at all without signing. Most implementations also differentiate between the certification authorities who have signed an application.

Applications signed by the manufacturer of a device enjoy the highest security level and can access every Java API

²⁹ metismo.com

³⁰ enough.de

³¹ mosync.com

available on the handset. Applications signed with a carrier certificate are similarly trusted.

Applications signed by JavaVerified³², Verisign³³ or Thawte³⁴ are on the lowest security level. To make matters worse, not every phone carries all the necessary root certificates. And, in the past, some well known device vendors have even stripped away all root certificates. The result is something of a mess, so consider signing your application only when required, that is when deploying to an app store or when you absolutely need access to security constrained features. However, 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.

Intertek³⁵ is probably the largest such supplier.

Distribution

J2ME applications can be installed directly onto a phone in a variety of ways; the most commonly used methods are over a Bluetooth connection, via a direct cable connection or Over-the-Air (OTA). However, app stores are probably the most efficient way to distribute your apps.: They manage the payment, hosting and advertisements, taking a revenue share for those services. Some of the most effective stores include:

32 javaverified.com

33 verisign.com

34 thawte.com

35 intertek.com/wireless-mobile



- Handmark³⁶ and Mobile Rated³⁷ provide carrier and vendor independent application stores.
- GetJar³⁸ is one of the oldest distributors for free mobile applications - not only Java applications.
- Nokia Store³⁹ 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 Orange⁴⁰ and O2⁴¹.

Basically almost everyone in the mobile arena has announced an app store. An overview of the available app stores (not those selling J2ME apps alone) can be found in the WIP App Store Catalogue⁴².

Furthermore there are various vendors who provide solutions for provisioning of Java applications over a Bluetooth connection, including Waymedia⁴³ and Futurlink⁴⁴.

³⁶ store.handmark.com

³⁷ mobilerated.com

³⁸ getjar.com

³⁹ publish.ovi.com

⁴⁰ orangepartner.com/site/enuk/mobile/application_shop/app_shop_client/p_app_shop_client.jsp

⁴¹ mobileapps.o2online.de

⁴² wipconnector.com/appstores/

⁴³ waymedia.it

⁴⁴ www.futurlink.com



Qt

Pronounced “cute”, not “que-tee”. Qt is an application framework that is used to create desktop applications and even a whole desktop environment for Linux, the KDE Software Compilation. The reason many developers have used Qt for desktop apps, is that it frees them from having to consider the underlying platform a single Qt codeline can be compiled to run on Microsoft Windows, Apple Mac, and Linux.

When Nokia acquired Trolltech the company behind Qt it was with the goal of bringing this same ease of development for multiple platforms to Nokia mobile phones. At the time of writing, Nokia was in the process of selling Qt to Digia. When the acquisition was announced, Digia indicated that it plans to “quickly enable Qt on Android, iOS and Windows 8 platforms”¹.

With Nokia winding down its Symbian portfolio, choosing Qt is not an easy decision. While there is still a significant number of Symbian phones in use and a continued demand for apps on these phones, the opportunities will decline. At the same time the opportunities that will be created by Digia’s plans is hard to quantify, however for those willing to make the investment in Digia’s vision the rewards could be significant.

The challenge when developing with C and C++ is that these languages place all the responsibility on you, the developer. For example, if you make use of memory to store some data in your application, you have to remove that data and free the memory when it is no longer needed (if this is not done, the dreaded memory leak occurs).

Qt uses standard C++ but makes extensive use of a special preprocessor (called the Meta Object Compiler, or moc) to deal with many of the challenges faced in standard C++ develop-

¹ www.digia.com/en/Home/Company/News/Digia-to-acquire-Qt-from-Nokia/

ment. As a consequence Qt is able to offer powerful features that are not burdened by the usual C++ housekeeping. For example, instead of callbacks, a paradigm of signals and slots is used to simplify communication between objects²; the output from one object is a “signal” that has a receiving “slot” function in the same or another object.

Adding Qt features to an object is simply a case of including QObject (which is achieved by adding the Q_OBJECT macro to the beginning of your class). This meta-object adds all the Qt specific features to an object. Qt then provides a range of objects for realizing GUIs created using Qt Quick, building complex graphical views (the QGraphicsView object), managing network connections and communications, using SVG, parsing XML, and using scripts among others.

Many developers who have used Qt report that applications can be written with fewer lines of code and with greater in-built reliability when compared to coding from scratch in C++. As a result less time is needed to create an application and less time is spent in testing and debugging.

For mobile developers using Qt is free of cost. It benefits from being open source also, with a large community of developers contributing to the content and quality of the Qt APIs. Should you wish to get involved in developing Qt, you can do so through the Qt Project³.

Prerequisites

Qt SDK⁴ installs everything you need to create, test, and debug applications for Symbian phones and the Nokia N9 smartphone, as well as desktop applications, from a single package.

² doc.qt.nokia.com/4.7-snapshot/signalsandslots.html

³ qt-project.org/

⁴ developer.nokia.com/Develop/Qt/Tools

All versions offer tools for compiling Symbian and Nokia N9 apps, with Symbian apps being compiled in the Linux and Apple Mac versions using the Remote Compiler service.

Creating Your Application

Qt SDK is built around the Qt Creator development tool. Using Qt Creator you define most of your application visually and then add the specific program logic through a code editor, which offers full code completion support and integrated help. One of the neat features of Qt is Qt Quick⁵, a comprehensive solution for declarative UI definition. Qt Quick uses QML, a language similar to JavaScript⁷ to define the UI; but also provides Qt Quick Components a set of predefined UI components that match the UI style seen on the latest Symbian phone and the Nokia N9 that further speed up UI development.

While Qt Quick generally simplifies UI development, one of its biggest advantages is that the tools within Qt Creator enable the UI to be defined by graphic designers who do not have to be aware of the technical programming aspects.

In the past, one of the challenges with cross platform applications for mobile has been accessing platform features: Anytime you want to find the phone's location or read a contact record it has been necessary to revert back to the platform's native APIs⁶.

This is where the Qt Mobility APIs come in. The APIs provided by Qt Mobility offer a common interface to phone data such as contacts, location, messages, NFC, and several others.

This means that if you, for example, need the phone's location the same API will obtain the location information on

⁵ qt.nokia.com/qtquick/

⁶ qt.nokia.com/products/qt-addons/mobility/

both a Symbian phone and the Nokia N9. (The Qt SDK enables you to work with the native APIs if you want to, as it includes the Symbian and Nokia N9's MeeGo 1.2 Harmattan APIs too.) As with Qt in general, working with the mobility APIs is quite straightforward. The following code, for example, shows that only a few lines are needed to access a phone's current location:

```
void positionUpdated(const QGeoPositionInfo
&gpsPos) {
}
```

However, do be aware that Qt does not yet insulate you from all the differences between platforms. For example, the X and Y axes reported from the phone accelerometers are transposed between Symbian phones and the Nokia N9. A simple enough issue to address with a `#IFDEF`, but still an issue to be aware of.

If you are already familiar with C++ development for the desktop, creating Qt applications for Symbian phones and the Nokia N9 is straightforward.

Once you have mastered the Qt APIs you should find you can code much faster and with fewer of the usual C++ frustrations particularly if you take advantage of Qt Quick to create your UI. Qt has many interesting features, such as WebKit integration enabling you to include web content into your app and scripting that can be used to add functionality quickly during development or change runtime functionality. It is also worth pointing out that, because Qt applications are compiled to the platform they will run on, they deliver very good performance, too. For most applications the levels of performance will be comparable to that previously achieved by hardcore native applications only.

Testing

Qt SDK includes a lightweight simulator enabling applications to be tested and debugged on the development computer (Qt SDK runs under Microsoft Windows, Ubuntu Linux and Apple Mac OS X). The simulator includes tools that enable phone data, such as location or contacts records, to be defined so that the application's functionality can be tested fully. The simulator does not, however, eliminate the need for on phone testing.

In addition, the Qt SDK includes tools to perform on-phone debugging on Symbian phones and the Nokia N9. This feature can be handy to track down bugs that come to light only when the application is running on a phone. Such bugs are rare and tend to surface in areas such as comms, where the Qt simulator uses the desktop computer's hardware, hardware that differs from the equivalent technology on a mobile phone.

Packaging

For a Qt application to run on a mobile phone the Qt API framework has to be present. The Nokia N9 smartphone has the Qt APIs built in. In addition, it provides a built-in update mechanism that will install the necessary framework components, should there be newer or additional versions needed by the app.

For Symbian phones the situation is a little different.

Symbian^3 (including Symbian Anna and Belle) phones have the APIs built in. However, Symbian does not include a built-in mechanism to add the APIs to earlier phones or load new or updated APIs to Symbian^3 phones. The solution is Smart Installer, which is included automatically in Symbian apps built with Qt SDK. As an app is installed on a Symbian

phone, Smart Installer checks for the presence of the necessary Qt packages and, if they are not there, downloads and installs them. Using this mechanism, Qt apps can be easily targeted at almost all recent S60 and Symbian phones.

Signing

As Qt applications install as native applications on Symbian phones and the Nokia N9, they need to comply with each platform's signing requirements.

Qt apps for the Nokia N9 need to be signed, but this is done for you during the Nokia Publisher process. To enable testing the Nokia N9 smartphone has a "developer" capability that enables unsigned apps to be installed and run for testing purposes. For applications to be installed on Symbian phones, signing is necessary even during testing. If you choose to use Nokia Store to distribute your apps, Nokia will organize for your Symbian app to be Symbian Signed, at no cost.

Unlike the Nokia N9, Symbian phones do not have a 'developer' mode. To enable apps to be tested they have to be signed with a "developer certificate". The process you have to follow is straightforward and described in full in the Distribute section of the Nokia Developer website⁷, but in summary:

- You sign up as a Nokia Publisher⁸
- You provide up to five phone IMEIs and request a UID for your application
- The Nokia Publisher team provides you with a "developer certificate" and a UID for your app
- You create your app with the UID provided, sign your app during development using the "developer certificate" to en-

⁷ www.developer.nokia.com/Distribute/Packaging_and_signing.xhtml

⁸ publish.nokia.com

able it to run the five phones elected and test it to ensure it complies with the Symbian Signed Test Criteria⁹

- Once tested, you submit an unsigned copy of the app to the Nokia publishing portal

Distribution

Nokia Store is the latest iteration of the Nokia app store solution, with a history stretching back to 2003. The store now delivers in excess of 10 million downloads a day, and the store's traffic is increasing steadily.

Importantly, once an application has met the store's quality requirements beyond removing indecent or illegal applications there are no restrictions on the types of applications that can be distributed.

So you will find many applications in Nokia Store that compete directly against offerings from Nokia, such as alternative browsers, music players, and email applications.

To use Nokia Store you need to register and pay a onetime €1 fee registration is open to both companies and individuals. When your application starts selling you receive 70% of the sale price of revenue after any applicable taxes and costs.

One significant advantage of Nokia Store is that consumers in 53 countries can use operator billing. This is because operator billing is universal and trusted, but it is also available in countries where credit card ownership is low. As a result, for each \$1 in credit card revenue you can expect to receive over \$10 from operator billing purchases – making operator billing the most lucrative option for generating revenue.

⁹ www.developer.nokia.com/Community/Wiki/Symbian_Signed_Test_Criteria_V4_Wiki_version



Windows Phone

Microsoft made a fresh start with the Windows Phone platform. The Windows Mobile operating system was declining in both user acceptance and market share, so Windows Phone was created as Microsoft's response to competing platforms, particularly in the consumer market. However, Windows Phone is designed for business users as well as consumers, and offers a simple-to-use interface that focuses on typography and content.

In early 2011 Nokia announced that Windows Phone would become its smartphone platform. This partnership was expected to push the market share significantly¹. Apparently the high hopes have not been fulfilled by now: According to Gartner, only 2.7% of all smartphones sold in Q2 2012 were based on Windows Phone².

UI Design

Windows Phone introduced a new UI paradigm called Modern UI³ that now has been extended to the Xbox 360 and Windows 8 as well. This UI paradigm contains following principles:

- **Content not Chrome** removes unnecessary ornaments and lets the content itself be the main focus. You should also restrain 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

1 microsoft.com/Presspass/Features/2010/dec10/12-21AchimBergQA.aspx

2 gartner.com/it/page.jsp?id=2120015

3 wikipedia.org/wiki/Metro_%28design_language%29

- **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.

You should embrace the Modern UI design principles in your application, especially when porting over existing apps. Designers will find many inspirations and information in the Microsoft Design Toolbox⁴. You should also use one of the available grid solutions. A grid helps you to align your content with the Modern UI style. One solution that you can also use in the emulator is MetroGridHelper⁵. Important for the overall experience are also the ‘live tiles’, small widgets that reside on the start screen. You can update them programmatically or even remotely using push notifications.

Development

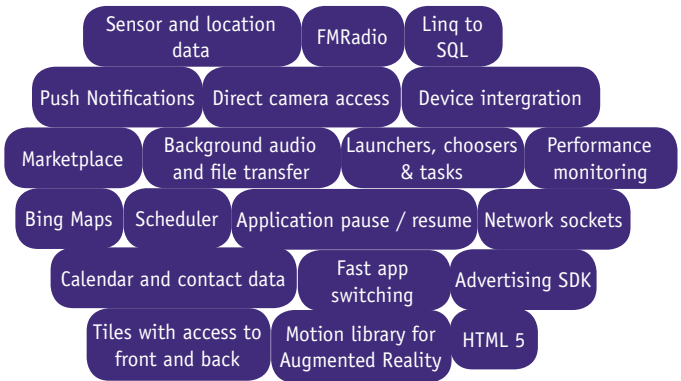
Windows Phone development is undertaken in C# or VB.NET, using the Microsoft Visual Studio IDE or Expression Blend. Applications are created using Silverlight, principally for event-driven applications, and XNA, principally for games driven by a “game loop”, although both technologies can be used in a single application. The user interface for Silverlight applications can be created either in Microsoft Visual Studio or Microsoft Expression Blend. Additionally you can create HTML 5 based apps using PhoneGap⁶, however web development is not covered in this chapter.

4 microsoft.com/design/toolbox/

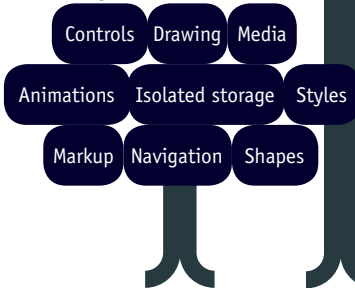
5 jeff.wilcox.name/2011/10/metrogridhelper

6 phonegap.com

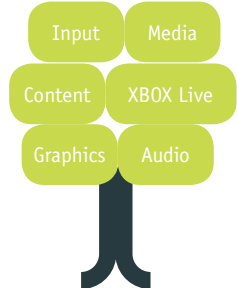
The Windows Phone 7.1 SDK



Silverlight Presentation and Media



XNA Frameworks



Common Class Library



The Windows Phone SDK is free of charge and includes “Express” editions of both Visual Studio 2010 and Expression Blend. While the Express editions support everything necessary to develop for Windows Phone, many extra features found in the commercial editions are not available. The SDK also includes a device emulator to run code against. The device emulator uses hardware acceleration and performs reasonably well when running 3D XNA games. In addition to basic functionality, the emulator has advanced features for location input (using Bing Maps) and accelerometer simulation.

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

| Use Silverlight if... | Use XNA if... |
|---|--|
| ...you want to create an event-driven application or a casual game. | ...you want to create a 2D or 3D game. |
| ...you want to use standard Windows Phone controls. | ...you want to manage art assets such as models, meshes, sprites, textures and animations. |
| ...you want to target Windows Phone, Windows and the web; re-using some code. | ...you want to target Windows Phone, Windows, and Xbox 360; re-using lots of code. |

While the most common scenario is to use Silverlight for apps and XNA for games, you can also create Silverlight games and XNA apps, depending on your needs. It is also possible to host XNA 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 XNA game.

Functions And Services

Windows Phone applications have access to input data such as location, multi-touch screen, accelerometer, gyroscope, and microphone.

Available services include FM radio, media playback, raw camera feed and push notifications⁷ that can also update the live tiles of your app. You can also consider using the freely available SkyDrive cloud space for you app⁸.

Multitasking And Application Lifecycle

Windows Phone has a limited form of multitasking that suspends applications in the background and allows for fast application switching. Currently, the only processes that can be run in the background, after an application has been left, are audio playback 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 Tombstoning⁹. To make Tombstoning possible,

7 msdn.microsoft.com/en-us/library/ff402558%28v=vs.92%29.aspx

8 windowsteamblog.com/windows_live/b/windowslive/archive/2011/12/07/skydrive-apis-for-docs-and-photos-now-ready-to-cloud-enable-apps-on-windows-8-windows-phone-and-more.aspx

9 [msdn.microsoft.com/en-us/library/ff817008\(v=vs.92\).aspx](http://msdn.microsoft.com/en-us/library/ff817008(v=vs.92).aspx)

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).

Native Code

In contrast to some other platforms, developers cannot execute native code or access the device hardware directly in Windows Phone. While this restricts the extensibility of the platform and arguably limits the type of applications that can be developed, it also ensures applications are sandboxed and cannot do anything that will permanently affect the usability of the phone.

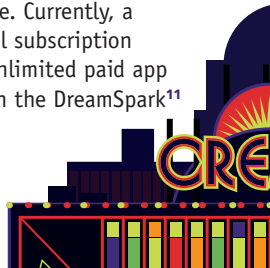
This means that core platform features – such as the dialer and onscreen keyboard – cannot be replaced or extended, and low-level access to Wi-Fi or Bluetooth radios is not possible. However, for most applications you are unlikely to encounter any restrictions that will affect your ability to deliver user features.

Distribution

Applications for Windows Phone are distributed through the Microsoft Marketplace service. 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 App Hub¹⁰. Although developer tools are provided free of charge, a paid App Hub account is necessary to deploy software to devices and Marketplace. Currently, a developer account costs 99 USD for an annual subscription and includes 100 free app submissions and unlimited paid app submissions. The fee is waived for students in the DreamSpark¹¹

¹⁰ dev.windowsphone.com

¹¹ www.dreamspark.com



and for the first year for Nokia Publish developers. The Marketplace also provides for time-limited beta distribution and offers a private distribution channel for enterprises. You can use the Windows Phone Marketplace Test Kit¹² to test your application locally before you submit them.

Testing And Analytics

You can unit test applications using the Windows Phone Test Framework¹³ or the Silverlight Unit Test Framework¹⁴.

For behavior-driven development, the Windows Phone Test Framework by Expensify¹⁵ is available. This will allow you to execute business requirements as end-to-end tests, driving automation of the emulator.

For developers wishing to collect runtime data and analytics, there are several options. Localytics¹⁶, PreEmptive Solutions¹⁷ and Flurry¹⁸ all provide analytics tools and services that are compatible with Windows Phone 7. Developers can also use the Silverlight Analytics Framework¹⁹ to connect to a variety of third-party tracking services such as Google Analytics. Starting with the Windows Phone SDK 7.1 update, there are robust performance monitoring tools available in Visual Studio.

¹² msdn.microsoft.com/en-us/library/hh394032%28v=VS.92%29.aspx

¹³ wptestlib.codeplex.com

¹⁴ jeff.wilcox.name/2011/06/updated-ut-mango-bits/

¹⁵ github.com/Expensify/WindowsPhoneTestFramework/

¹⁶ localytics.com/docs/windows-phone-7-integration/

¹⁷ preemptive.com/windowsphone7.html

¹⁸ flurry.com

¹⁹ msaf.codeplex.com/

Monetization

There are two primary methods of monetizing your Windows Phone apps, as paid for applications and as ad-serving applications. 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 instead²⁰. Notably there is no in-app purchase mechanism, although it will be available in Windows Phone 8.

For ad-based monetization, there are several options. Microsoft has their own Microsoft Advertising Ad Control²¹ (currently available in 18 countries), while Smaato²², inneractive²³, AdDuplex²⁴ and Google²⁵ all offer alternative advertising solutions. For more general information about monetization, please see the dedicated chapter in this guide.

Resources

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

The development team posts on their blog at *windowsteam-blog*²⁶ or their Twitter account *@wpdev*. For a large collection of developer and designer resources, visit *windowsphonegeek.com* and *reddit*²⁷.

²⁰ [msdn.microsoft.com/en-us/library/ff967558\(v=vs.92\).aspx#Best_Practices](http://msdn.microsoft.com/en-us/library/ff967558(v=vs.92).aspx#Best_Practices)

²¹ advertising.microsoft.com/mobile-apps

²² smaato.com

²³ inner-active.com

²⁴ adduplex.com

²⁵ developers.google.com/mobile-ads-sdk/

²⁶ windowsteamblog.com/windows_phone

²⁷ reddit.com/r/wp7dev

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 Silverlight Toolkit for Windows Phone, available at silverlight.codeplex.com. Other popular Windows Phone projects include coding4fun.codeplex.com and mvvmlight.codeplex.com. For inspecting the visual tree, bindings and properties of XAML-based user interfaces at runtime, XAMLSpy²⁸ is available.

There are several eBooks available for free, for example Windows Phone Programming in C# (Windows Phone Version 7.5)²⁹ or Silverlight for Windows Phone Toolkit In Depth³⁰.

Windows Phone 8

Microsoft has recently unveiled³¹ the next version of Windows Phone, called Windows Phone 8. The next version of Windows Phone has been re-hauled to share a common core with Windows 8. Along with these changes comes a new programming paradigm. More details and the initial SDK release should occur in late summer 2012³².

Upgrade Path

Microsoft has revealed that existing Windows Phone 7 apps will continue to run on Windows Phone 8, but will not be able to access new features and hardware capabilities.

²⁸ xamlspy.com/

²⁹ blogs.msdn.com/b/uk_faculty_connection/archive/2011/11/23/windows-phone-free-ebook-amp-demos.aspx

³⁰ windowsphonegeek.com/WPToolkitBook

³¹ channel9.msdn.com/Events/Windows-Phone/Summit

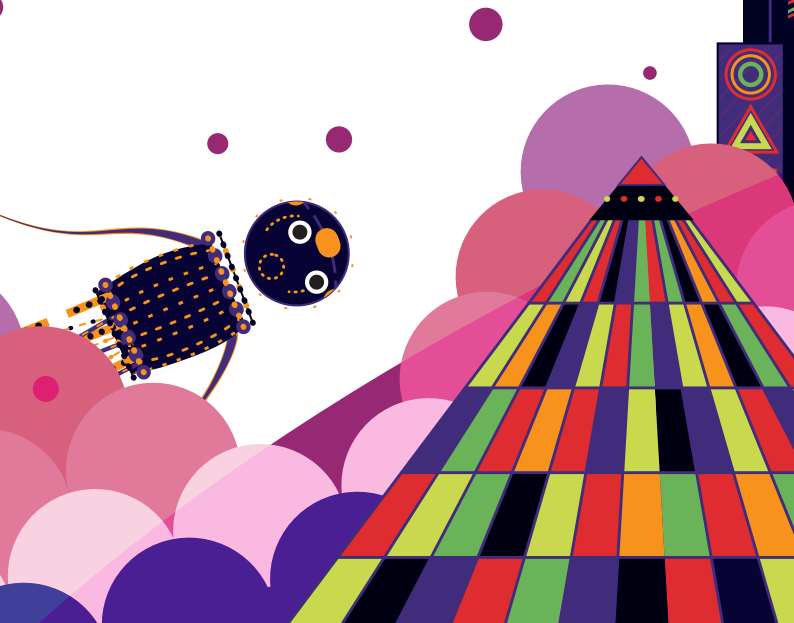
³² social.msdn.microsoft.com/Forums/en-US/wpdevelop/thread/3b1e975e-7687-4530-881c-356893f23eb9

Development

Development in Windows Phone 8 will be done in C# or C++, using the DirectX API or the WinRT API. Because of this, it should be very easy to share code between Windows 8 and Windows Phone 8 apps and games. It has been speculated that the inclusion of C++ / DirectX as an option will encourage more games and middleware to be ported to the Windows Phone 8 platform. In fact, Unity³³ has already announced support³⁴ for Windows 8 and Windows Phone 8.

³³ unity3d.com/

³⁴ edge-online.com/news/unity-announces-windows-phone-8-support





Windows 8

Windows 8 is Microsoft's first OS that runs on tablets and PCs alike. On PCs, Windows 8 is backwards compatible, meaning that any Windows 7 compliant apps can run on Windows 8 as well. In addition, specific Windows 8 apps run on Intel and ARM based machines alike. Windows 8 apps (formerly Metro style apps) can be developed in C++, C#/VB.NET or JavaScript. They are all first class citizens in the ecosystem as all programming languages have equal access to the Windows Runtime (WinRT) APIs. As PCs are outside of the scope of this guide, we concentrate on Windows 8 apps. The design language Metro/Modern UI is discussed in the Windows Phone chapter.

The Artist Formerly Known As Metro

Metro, Modern UI, Windows 8 style - are you confused? So are we. Apparently one European sales company called Metro AG was not happy about the name of the design language for Windows 8, so Microsoft backed down and started to rephrase Metro. It seems that "Modern UI" is used for developers whereas "Windows 8 apps" is the term used for consumers. However, Metro is still being used and since we do not see any reason how anyone could confuse a sales chain with a design paradigm, we grumpily stick with Metro.

Prerequisites

To develop Metro style apps you require Visual Studio 12 and Blend, the Express versions are available for free. You can install Windows 8 within a virtual machine, side by side with your existing OS or as your main OS. Having a touch enabled monitor helps to fine tune the user experience for tablets, but

Metro style apps and Windows 8 work equally well when using a mouse.

Technically you also need a developer license, however this license is automatically acquired and free of charge. Only the Windows Store account costs money, compare the Distribution section below.

Developing Metro Style Apps

In this section, read about the basics of developing Windows 8 Metro style apps.

What Language Should I Use?

While you may simply chose the language that matches the know-how of you or your team, it is worth understanding the differences in capabilities offered by the various options:

| | C/C++ | C#/VB.NET | JavaScript |
|----------------------|---|---|---------------------------------|
| WinRT | yes | yes | yes |
| Silverlight/ XAML | yes | yes | no |
| HTML | no | no | 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* or game libraries based on that like *monogame.codeplex.com*. As Windows Phone 8, the rumored Xbox 8 and Windows 8 all share the same kernel, it will be simpler to share code between these platforms in the future. The easiest way for that seems to use *shared .NET* based libraries between these platforms.

App Parts

Each Metro style app consists of several parts:

- **App tile** represents the app on the splash screen and can show relevant content to the user, even when your app is not running;
- **Splash screen** is optionally shown when you app starts;
- **App bar** contains the context relevant actions and commands;
- **Content area** displays your app in different view states such as full screen or snapped, compare the 'Views and Form Factors' section;
- **Charms** allow the user to start interactions with the application, compare the 'Application Contracts' section.

The Windows Runtime APIs

The WinRT APIs are documented on [msdn¹](#), they contain the usual suspects like JSON/XML parsing over geolocation, sensors, media handling and networking APIs. But WinRT has some more rather interesting concepts, for example:

```
Windows.Security.Authentication.Live  
Windows.Security.Authentication.Web  
Windows.Security.Credentials  
Windows.ApplicationModel.Contacts
```

¹ msdn.microsoft.com/en-us/library/windows/apps/br211369.aspx

Application Contracts

Windows 8 features charms in each Metro style app that you can access by sliding in from the right hand side. There are five charms: search, share, start, devices and settings. By implementing contracts you can plug into these charms and share information between apps. Declare contracts in your `Package.appx` manifest file, then implement the required functionality.

There are following contracts²:

- **Search** searches for content in your app. You can optionally provide search suggestions while the user types. The relevant functionality is found in the `Windows.ApplicationModel.Search` namespace.
- **Share** provides a way to share data between a source and target app, by declaring a corresponding contract. If you want your app to share data, you should make it available in as many data formats as possible to increase the number of potential target apps. You can use standard formats such as text, HTML, images or create your own formats. The share target app can optionally return a quicklink that points to the consumed data. For example, you can share an image with Facebook or Flickr and get back a link. Relevant classes are in `Windows.ApplicationModel.DataTransfer.ShareTarget`.
- **Play To** plays data with connected devices, for example by streaming a video to your DLNA enabled TV. Start with the `Windows.ApplicationModel.PlayTo` namespace.
- **Settings** allows the user to adjust context dependent settings from anywhere within your app. Define you settings with the help of `Windows.UI.ApplicationSettings`.

² msdn.microsoft.com/en-us/library/windows/apps/hh464906.aspx

- **App to App Picking** allows you to open or save app files from within another app. Find classes in the `Windows.Storage.Pickers` namespace.

Do not duplicate charms functionality elsewhere in your app. That will just confuse your users. You should, for example, not include a specific search field, unless searching is the main task of your app.

Windows 8 has many more extensions and contracts, a full list is available at msdn.microsoft.com/library/windows/apps/hh464906.aspx.

Views and Form Factors

Metro style apps can run in different layout modes³:

- **full screen** is the default mode, either in **landscape** or **portrait** orientation. Your app will use all the available screen real estate to immerse the user completely in `ApplicationLayoutState.FullScreen`.
- **snapped** and **filled** are modes in which apps are shown side by side. You should change your layout accordingly but maintain the state of your app and keep at least the main functions easily accessible, this applies to both `ApplicationLayoutState.Filled` and `ApplicationLayoutState.Snapped`.

To get notified about layout changes, listen to the `Windows.UI.ViewManagement.ApplicationLayout.GetForCurrentView().LayoutChanged` event. There you can even change the state programmatically: When your app is in snapped mode and your user selects a

³ msdn.microsoft.com/en-us/library/windows/apps/hh465371.aspx

function that demands a different mode, you can call `ApplicationLayout.TryUnsnap()`.

Autoscaling

Windows 8 runs on devices with different screen resolutions and pixel densities. Depending on the resolution Metro style apps are scaled automatically to:

- 1366 x 768 (100%)
- 1920 x 1080 (140%)
- 2560 x 1440 (180%)

Web developers should use SVG graphics and CSS media queries, when possible. XAML developers can use naming schemes for resources, so that the best fitting resource is chosen automatically (such as `image.scale-100.jpg`, `image.scale-140.jpg` and `image.scale-180.jpg`). You should also use resources with dimensions that are multiples of 5px, so that no pixel shifting occurs when autoscaling.

Push

You can send data and even images to your Metro style apps using the Windows Notification Service (WNS)⁴. This also enables you to update the live-tiles of your app. Using WNS is free of charge. You can use the Windows Azure Toolkit for Windows 8⁵ to simplify the implementation of a push server.

⁴ msdn.microsoft.com/en-us/library/windows/apps/hh465460.aspx

⁵ watwindows8.codeplex.com

Single Sign On

Windows 8 provide user credential management services⁶ and using Microsoft Account for its user authentication. You can leverage this to provide single sign to your apps, enabling you to identify the user directly without further authentication⁷.

Distribution

Metro style apps can be distributed through Windows Store⁸ only. The standard revenue share of 70% is increased to 80% when your app makes more than 25,000 USD. The Windows Store will support over 200 countries and regions and more than 100 languages, so you can have a global reach. You also can distribute feature- or time-limited trial versions of your app, use in app purchasing or integrate adverts. Third-party payment providers are allowed as well.

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.

Before you sell apps, you need to obtain a Windows Store account that costs 49 USD per year for individuals and 99 USD for companies.

⁶ msdn.microsoft.com/en-us/library/windows/apps/br211367.aspx

⁷ msdn.microsoft.com/en-us/library/windows/apps/hh465098.aspx

⁸ msdn.microsoft.com/en-us/library/windows/apps/hh694084.aspx



Resources

Your starting point for Windows 8 development is dev.windows.com.

You can follow the Windows 8 development team on twitter.com/buildwindows8 and blogs.msdn.com/b/b8/.

Discuss development problems on social.msdn.microsoft.com/Forums/en-US/category/windowsapps.

Find sample code on code.msdn.microsoft.com/windowsapps, in various codeplex.com projects and in the end to end samples available at msdn.microsoft.com/en-us/library/windows/apps/br211375.aspx. The roadmap for app developers provides an good overview about planing, designing and developing Windows 8 apps at msdn.microsoft.com/library/windows/apps/xaml/br229583.aspx.





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, bada, BlackBerry, 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 (all in alphabetical order).

Before embarking on a mobile apps project one of the key decisions to make is which platforms to target. In making this decision – by looking at the market potential and cost of development for each platform – it is well worth reviewing the option of a cross platform framework. In considering a cross-platform approach do not confuse the market size of a platform with the market potential for your application – while Android and iPhone appear to have the biggest market places, you will also need the biggest marketing effort to get noticed. So concentrating on several seemingly smaller platforms, might be a smart choice for some apps.

Another challenge is that 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 the development and adopting a cross-platform strategy.

By the way, we are not talking about app stores here; this is a different market fragmentation problem. The more than 120 app stores, from operators, manufacturers and independent companies create challenges of their own, outlined in the “Appstores” chapter.

App Development Process

When talking about cross-platform development you should be aware about the overall app development phases:

1. Planing & specification
2. Prototyping & design
3. Implementation
4. Testing
5. Deployment

Cross-platform development can speed up the implementation phase, but it does not help with the other phases. On device testing on all supported platforms and their respective versions is really crucial for the success of any app, for example.

Limitations And Challenges Of Cross Platform Approaches

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

Native Programming Languages

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 | 1 st class citizen ¹ | 2 nd class citizen ² |
|--------------|---|---|
| ActionScript | BlackBerry 10, BlackBerry PlayBook OS (QNX) | none |
| C, C++ | bada, BlackBerry 10, BlackBerry PlayBook OS, Brew MP, Symbian, Windows 8, Windows | Android (partially, using the NDK), iOS (partially) |
| C# | Windows 8, Windows Phone and Windows Mobile | none |
| Java | Android, BlackBerry, Java ME devices | Symbian, Windows Mobile |
| 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:** This approach exploits the fact that most platforms provide direct support for web technologies through embedded ‘webviews’ in native applications. Along with HTML and CSS, this approach supports JavaScript also.
- **Interpretation:** Here the framework delivers an engine for each platform that interprets a common or framework

specific language. For example, a popular option for games development is Lua scripting.

- **Cross Compilation:** The holy grail of cross platform frameworks is cross compilation, but it is also the most complex technical solution. It enables you to write an app in one language and have it transcoded into each platform’s native language, offering native runtime speed.

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-independent¹.

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. The other challenge with a 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.

Some platforms also have different design philosophies. While iOS strives 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.

¹ www.developergarden.com/apis/

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

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 and Symbian 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

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 Symbian 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.

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 Airship² 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.

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. And again these vendor services work differently between the platforms.

² urbanairship.com/

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.

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 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.

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

While websites are inherently cross platform, they have some big disadvantages:

1. Websites are not listed in the app stores, so users do not find them and monetization is difficult. (Although you could create a simple application or widget that opens your website and submit that to a store, but this will not help with monetization.)
2. Websites only work online (although the increasing availability of HTML5 is slowly eliminating this disadvantage).
3. Websites have an inferior user experience compared to native 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.

| Web App Solution | License | Target Platforms |
|--|-------------|---|
| jQuery Mobile www.jquerymobile.com | MIT and GPL | Android, bada, BlackBerry, iOS, Symbian, webOS, Windows Phone |
| JQTouch www.jqtouch.com | MIT | iOS |
| iWebKit iwebkit.net | LGPL | iOS |
| iUI code.google.com/p/iui | BSD | iOS |
| Sencha Touch www.sencha.com/products/touch | GPL | Android, iOS |
| The M Project the-m-project.org | MIT and GPL | Android, BlackBerry, iOS, webOS |

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. Web apps have some advantages over traditional websites:

1. You can put a web app in an app store. Even when not directly supported by the vendor, you can use web based tools such as PhoneGap in combination with a web app solution to make web apps available in app stores.
2. Web apps can work offline.

3. Web apps can look and behave in a similar fashion to native apps, however there are often slight – albeit annoying – differences compared with their native counterparts.

A step further towards native applications is provided by hybrid web apps, in which you create a native application that uses a webview to display a website.

With this approach you can have access to any native functionality that you require while keeping most of the functionality on the server side.

This approach is easier than creating native apps for every platform while enabling you to extend the native parts of your app as required in an incremental fashion.

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, I 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 developers@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 answers 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.

- How does your cross platform tool chain work? What programming language and what API can I use?

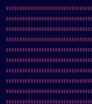
- Can I access platform specific functionality? If so, how?
- Can I use native UI components? If so, how?
- Can I use a platform specific build as the basis for my own ongoing development? What does the translated/generated source code look like?
- Is there desktop integration available?
- Can I control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

| Solution | License | Input | Output |
|--|-----------------------------|--------------------------|--|
| Application Craft applicationcraft.com | Commercial | HTML, CSS, JavaScript | Android, BlackBerry, iOS, Symbian, Windows Phone, mobile sites |
| appMobi appmobi.com | Commercial | HTML, CSS, JavaScript | Android, iOS, Kindle Fire, Nook, web |
| Codename One codenameone.com | Commercial | Java | Android, BlackBerry, iOS, J2ME, Windows Phone |
| Corona coronalabs.com (Corona Labs) | Commercial | JavaScript | Android, iOS |
| J2ME Polish j2mepolish.org (Enough Software) | Open Source + Commercial | Java ME, HTML, CSS | Android, BlackBerry, J2ME, PC |



| Solution | License | Input | Output |
|---|-------------|----------------------------|--|
| Flash Builder adobe.com/devnet/devices.html (Adobe) | Commercial | Flash | Android, BlackBerry Playbook OS, iOS, PC |
| Feedhenry feedhenry.com | Commercial | HTML, CSS, JavaScript | Android, BlackBerry, iOS, Windows Phone |
| Kirin/JS kirinjs.org/ | Open Source | JavaScript | Android, iOS |
| Kony One kony.com/node/4 | Commercial | HTML, CSS, JavaScript, RSS | Android, BlackBerry, iOS, J2ME, Symbian, Windows Phone |
| LiveCode runrev.com (RunRev) | Commercial | English-like | Android, iOS, PC and Web |
| MobiForms www.mobiforms.com (MobiForms) | Commercial | Drag and Drop + MobiScript | Android, iOS, PC, Windows Mobile |
| Mono for Android xamarin.com/monoforandroid (Xamarin) | Commercial | C# | Android (share code with iOS and Windows Phone) |
| Mono Touch xamarin.com/monotouch (Xamarin) | Commercial | C# | iOS (share code with Android and Windows Phone) |

| Solution | License | Input | Output |
|--|--------------------------|-----------------------------|--|
| MoSync mosync.com | Open Source + Commercial | C/C++, HTML5/JS | Android, BlackBerry, iOS, J2ME, Symbian, Windows Phone 7, Windows Mobile |
| NeoMAD neomades.com | Commercial | Java | Android, bada, BlackBerry, iOS, J2ME, Symbian, Windows Phone 7 |
| PhoneGap/ Cordova www.phonegap.com (Adobe/Apache) | Open Source | HTML, CSS , JavaScript | Android, BlackBerry, iOS, Symbian, Windows Phone |
| Qt qt.nokia.com (Nokia) | Open Source + Commercial | C++ | PC, Symbian, MeeGo and Windows Mobile, desktop Windows, Apple & Linux OS |
| Rhodes rhone.com/ products/rhodes (Motorola) | Open Source + Commercial | Ruby, HTML, CSS, JavaScript | Android, BlackBerry, iOS, Symbian, Windows Mobile, Windows Phone |
| Spot Specific www.spotspecific.com | Commercial | Drag and Drop, JavaScript | Android, iOS |
| Titanium www.appcelerator.com | Open Source | JavaScript | Android, Consoles, iOS, PC |



| Solution | License | Input | Output |
|--|--------------------------|------------------|---|
| Verivo verivo.com | Commercial | (Visual) | Android, BlackBerry, iOS |
| webMethods MobileDesigner (formerly Metismo Bedrock) www.metismo.com (Software AG) | Commercial | Java ME | Android, bada, BlackBerry, brew, Consoles, iOS, PC, Windows Phone, Windows Mobile |
| XML VM xmlvm.org | Open Source + Commercial | Java, .NET, Ruby | C++, Java, JavaScript, .NET, Objective-C, Python |

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).

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 |
|--|-------------|---------------------------|--|
| Corona cocos2d-x.org | Open Source | C++, HTML5, JavaScript | Android, iOS |
| Corona coronalabs.com (Corona Labs) | Commercial | JavaScript | Android, iOS |
| EDGELIB www.edgelib.com (elements interactive) | Commercial | C++ | Android, iOS, PC, Symbian |
| Esenthel esenthel.com (elements interactive) | Commercial | C++ | Android, iOS, PC |
| GameSalad gamesalad.com | Commercial | Drag and drop | Android, iOS, PC, web |
| id Tech 5 www.idsoftware.com (id) | Commercial | C++ | Consoles, iOS, PC |
| Irrlicht irrlicht.sourceforge.net | Open Source | C++ | Android & iOS with OpenGL-ES version, PC |
| IwGame drmop.com/index.php/ iwgame-engine | Open Source | C++ | Android, bada, BlackBerry Playbook OS, iOS, PC |

| Solution | License | Input | Output |
|---|-------------|---------------------------|--|
| ORX orx-projet.org | Open Source | C, C++, Objective-C | Android, iOS, PC |
| Marmelade madewithmarmalade.com (Ideaworks3D) | Commercial | C++, HTML5, JavaScript | Android, bada, BlackBerry PlayBook OS, iOS, LG Smart TV |
| Moai getmoai.com (Zipline Games) | Commercial | Lua | Android, iOS, PC, Web |
| MonoGame monogame.codeplex.com | Open Source | C#, XNA | Android, iOS, PC, Windows 8 |
| SIO2 sio2interactive.com (sio2interactive) | Commercial | C, Lua | Android, bada, iOS, PC |
| Unigine www.unigine.com (Unigine corp.) | Commercial | C++, Unigine- Script | Android, iOS, PC, PS3 |
| Unity3 unity3d.com (Unity Technologies) | Commercial | C#, JavaScript, Boo | Android, iOS, PC, consoles, web |



Web Technologies

One big advantage of web technologies is that they offer the easiest route into mobile development. For a web developer mobile is simply the web ;) Equally, for anyone taking the first steps into mobile development – or first steps into programming – HTML, CSS and the JavaScript language are a lot easier to learn than relatively complex native languages.

This article does not teach you how to create websites. There are a lot of fantastic resources on the web:

- HTML5Rocks.com¹
- Mozilla.org²

This may come as a surprise, but there is no mobile web. There is only one web³. What is unique about The Web is that it enables everyone to access information and services from anywhere on any device, Long Live the Web⁴!

Simply enter a URL in any browser and off you go. Websites that have been around for many moons are still compatible and accessible with the latest browsers. In comparison, it is pretty much impossible to find native applications that (without alterations) manage to run for a number of years on any modern operating system.

The web is accessible for everyone on any browser/device, but people are different⁵ and browsers are different, too. Diversity⁶ is great and illustrates evolution. Unfortunately it is

1 www.html5rocks.com/en/

2 developer.mozilla.org/en-US/learn/html

3 www.the-haystack.com/2011/01/07/there-is-no-mobile-web/

4 www.scientificamerican.com/article.cfm?id=long-live-the-web

5 www.thinkwithgoogle.com/mobileplanet

6 html5test.com/results-mobile.html

not always that easy to create a cross-device, cross-platform, cross-browser, cross-markup, "cross-blahblah" mobile website. Dealing with many different devices also creates an annoying fragmentation jungle. Some devices use their own implementation of device APIs (such as MSIE 6 or Blackberry OS 4.6), or simply cannot support certain features. As a rule of thumb, when developing on the mobile or multi-device web expect the unexpected⁷.

Why create a website instead of an application? My brain hurts⁸. The entire App vs. Web discussion is bullshit⁹. Both technologies are fundamentally different. "Apps are like Songs"¹⁰ and "Cool URIs don't change"¹¹. Both technologies coexist and, taking the desktop PC as an example, have done so for years. One advantage of a website is that it runs on any device. Just type or share the URL. If you want to be native, be native. Native is not bad, native means being closer to the device, the OS and ultimately closer to the user. But at the same time native excludes universality.

Usability

Content should come first¹². Do not think about features. Think about functions and how to access and interact best with your content. If your content/service is not interesting, why should anyone use it?

Less is more. If you do not create it for mobile – why on a desktop?

⁷ futurefriend.ly/

⁸ www.youtube.com/watch?v=IILKiRPSNGA

⁹ bradfrostweb.com/blog/notes/native-vs-web-is-total-bullshit/

¹⁰ torgo.com/blog/2009/06/apps-are-like-songs.html

¹¹ w3.org/Provider/Style/URI

¹² lukew.com/ff/entry.asp?1598

Performance

Performance is not optional¹³. But how do you speed up your pages? „80-90% of the end-user response time is spend on the front-end.“¹⁴ Especially your front-end can waste crucial time.

- Loading unnecessary resources¹⁵
- Using JavaScript Frameworks – give VaporJS¹⁶ a try ;)
- Complex CSS rendering

HTML5

Oh wait, isn't there HTML5¹⁷!? HTML5 has been hyped up to be the ultimate tool¹⁸, the holy grail, or the solution to pretty much anything. We are not saying HTML5 is bad, but the hype is getting on our nerves¹⁹. It is important to factor in that the web is not just evolving on devices, but in accordance with technology and standards. There are organizations safeguarding HTML, such as the WHATWG²⁰ and W3C²¹. What is referred to as HTML5 by W3C is a part of HTML's living standards²² as outlined by WHATWG. As with any collaboration, it is not always easy for two organizations to work together²³. As an observer of

¹³ velocityconf.com/velocity2009/public/schedule/detail/7709

¹⁴ stevesouders.com/blog/2012/02/10/the-performance-golden-rule/

¹⁵ guypo.com/mobile/performance-implications-of-responsive-design-book-contribution/

¹⁶ vaporjs.com/

¹⁷ w3.org/TR/html5/

¹⁸ brucelawson.co.uk/2011/html5-notes-for-analysts-and-journalists/

¹⁹ slideshare.net/sevenval/why-html5-is-getting-on-my-nerve

²⁰ whatwg.org

²¹ w3c.org

²² blog.whatwg.org/html-is-the-new-html5

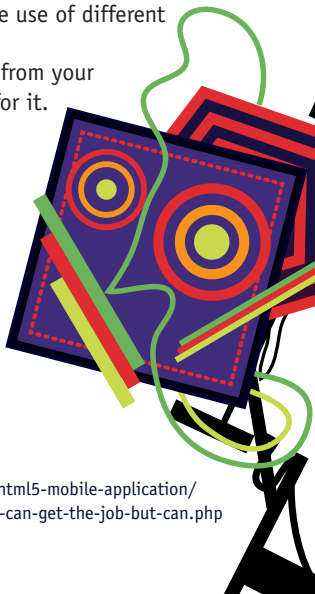
²³ zeldman.com/2012/07/24/whtmlyaduck/

standards, the best quote to describe standards comes from Andrew S. Tanenbaum: "The nice thing about standards is that you have so many to choose from."²⁴.

WebApps

Do you still create lame web sites or new fancy HTML5 WebApps? Where is the difference? When we use the term WebApp, we are not referring to a traditional Web Application²⁵, but a JavaScript application that is running client side. There are numerous frameworks readily available to do so. One of the most famous is Sencha Touch²⁶. Even though web technologies are used, WebApps should not be described as websites²⁷. The web is full of myths²⁸ about HTML5 and WebApps are no exceptions. A WebApp does not have access to exclusive APIs (i.e. camera, offline mode). Websites and WebApps use the same browser, but make use of different concepts.

If you prefer to move the complexity from your (dumb) server to the smart browser, go for it. But be aware: The browser is the known unknown and harder to control. The server is yours, the browser is unknown.



²⁴ en.wikiquote.org/wiki/Andrew_S._Tanenbaum

²⁵ en.wikipedia.org/wiki/Web_application

²⁶ sencha.com/products/touch/

²⁷ w3cubes.com/blog/2011/10/26/anatomy-of-a-html5-mobile-application/

²⁸ www.readwriteweb.com/mobile/2011/08/html5-can-get-the-job-but-can.php

Adaptation

A web resource displayed as a URL is first and foremost a message (document) to all people. To understand that message every language needs a universal translation to be understood by different target groups, age groups and cultures. Web technologies achieve this thanks to browsers, standards, devices (CPUs), images, videos... How to translate your message/document? There are different approaches:

Device & Browser Detection

Device and browser detection violates a universal or future-friendly approach. Nevertheless, if you want to finish a project, or adapt an aspect for a specific case that only affects the device/ browser, then that is the exception of the rule.

In most instances this is managed by the User-Agent.

Example²⁹:

Client Side (JavaScript):

```
if((navigator.userAgent.match(/iPhone/i)) ||
(navigator.userAgent.match(/iPod/i))) {
}
```

Server Side (PHP):

```
if(strpos($_SERVER['HTTP_USER_AGENT'], 'iPod')) {
}
```

²⁹ davidwalsh.name/detect-iphone

Feature Detection

Feature Detection is always device neutral, universal and future-friendly. The Modernizr³⁰ JavaScript library is a good option for client-side feature detection. But sometimes browsers lie and aspects are undetectable³¹. Detector³² is a solution for server-side feature detection.

Client Side Adaptation

Client side adaptation is a process whereby adaptations to the device/browser are handled within the browser. Options include Media Queries³³, Feature Detection or User-Agent sniffing with JavaScript. Client side adaptation includes device specific adaptations. Based on this philosophy, no infrastructure adaptations such as additional server components are needed. The client (browser) handles (required) adaptations. This approach is not necessarily better or worse - why not form your own conclusions after evaluating it?

With responsive design³⁴ you can create a layout that can adapt to conditions using one URL. In many cases, this includes handling of screen size (width/height) and alignment. This is done with Media Queries³⁵ and fluid layout and flexible images.

Progressive Enhancement³⁶ makes use of Client-Side Feature Detection. Generally speaking, this involves the use of JavaScript to detect whether a certain capability is supported on a device. Client side feature detection generally involves the use of JavaScript to detect whether a certain capability is supported on a device. The foundation is plainHTML and is

30 modernizr.com/

31 github.com/Modernizr/Modernizr/wiki/Undetectables

32 detector.dmolsen.com/

33 w3.org/TR/css3-mediaqueries/

34 alistapart.com/articles/responsive-web-design/

35 w3.org/TR/css3-mediaqueries/

36 accessites.org/site/2007/02/graceful-degradation-progressive-enhancement/

supported by any browser. Features that need to be detected are optional. A well-known framework that makes use of this approach is jQuery Mobile³⁷.

Server Side Adaptation

Server Side Adaptation is a process in which the server performs something specific for the Client. Server Side Technologies are often coupled with user agent sniffing, but to actively manage a piece of `<noscript>...</noscript>` tag, or the transfer and handling of features submitted Client Side via JavaScript is Server Side Adaption.

- **Device Templates:** The example³⁸ illustrates that Server Side Adaptation works best when (multiple) tailored templates are developed and its selection is handles by the user agent. Accordingly, you could create the latest feature version to serve the iPhone, but serve a generic version to every other device. Sadly it is not as easy as it sounds. For example the firmware of the phone also comes with a multitude of features. A clear and preferred differentiation between feature phones and smartphone is therefore problematic.
- **Device Detection:** Server Side Device Detection³⁹ is the most popular approach to perform adaptations. The big challenge with this approach is the rate at which new devices are released, making it hard to keep a device database up to date. There are device databases (such as WURFL⁴⁰ or DeviceAtlas⁴¹). These services are well worth

37 jquerymobile.com/

38 davidwalsh.name/detect-iphone

39 mobiforge.com/designing/blog/server-side-device-detection-used-82-alexa-top-100-sites

40 wurfl.sourceforge.net

41 deviceatlas.com

considering, as they have full time employees (or a community) actively maintaining their databases and do a lot of work that would be impractical for you to do.

Client & Server Side Adaptation

Another approach is to combine responsive design + server side components⁴². To create a mix of server & client side and feature & device detection has been cited as the evolution of responsive web design⁴³. The clear advantage is that everything that a server has already adapted for a specific client is easier for the client to handle.

These evolutions of responsive web design are available as commercial solutions (e.g. Sevenval⁴⁴) or cloud solutions⁴⁵ backed by a community.

Summary

Deciding between multi-device adaptation and optimization⁴⁶ need to be decided dependent on a number of factors. Universal feature detection addresses the future, but in every day life specific hacks are needed to address browsers/devices.

Technical Limits of Web Technologies

Browser's technical limitations are changing⁴⁷. Take for example the previous issue of this guide itself. It stated that access to the camera from within the browser is impossible. And now the impossible is already possible thanks to a number of modern

⁴² netmagazine.com/tutorials/getting-started-ress

⁴³ slideshare.net/dmolsenwvu/ress-an-evolution-of-responsive-web-design

⁴⁴ sevenval.com

⁴⁵ fitml.com

⁴⁶ lukew.com/ff/entry.asp?1562

⁴⁷ w3.org/2009/dap/#roadmap

browsers. To stay ahead of the game, have a look at compatibility tables⁴⁸.

HTML without Browsers

Many more recent SDKs use web technologies outside of the browser to enable developers faster entry routes. Other packages, feature detection/ adaptation solutions, or new APIs do not stop at browser level.

Hybrid Apps

The best of two worlds – or the worst? These solutions use web and native technologies. The challenge is to ensure that you combine the unique capabilities of the web with a native single platform in a way that creates synergy. That way you truly get the best of both worlds. If you want (or need) to publish your mobile app in an app store, you can create a hybrid app. This approach enables you to create your app with common web technologies (HTML, CSS, JavaScript) to compile everything within a native app. There are several hybrid app frameworks⁴⁹ that will help to make your life easier. For example, a single HTML5 web app codebase reduces your development and maintenance costs. These frameworks work by creating a native platform wrapper app that embeds your web app in a web view. They may also provide some platform features you cannot use in the browser – such as vibration, access to the camera, or the address book. It is important to note though that they are rarely as comprehensive as the features for native apps. And you still have to develop your own UI, which probably turns out to be slower than a native one.

⁴⁸ caniuse.com/

⁴⁹ See the cross-platform chapter for a comprehensive list

Note: A large number of platforms use different options to render embedded web views or the real browser.

Widgets

Widgets are created using the scripting and mark-up languages used for websites (HTML, CSS, JavaScript) and bundle this web content into a zip archive that is installed on a device and runs just like any other application.

Widgets, just like websites, are created entirely in plain text. These text files are then packaged as a zip archive. This makes it possible to create widgets using a text editor, zip application, and a graphics application (to create an icon and graphics for the widget). If you already have a web development tool then use it for widget development. The primary advantage of a web editor is the support these tools provide to compose HTML, CSS and JavaScript. There are a number of tools specifically designed to develop widgets. These may be delivered as plug-ins or add-ons to web authoring tools. Have a look at the BlackBerry Widget SDK⁵⁰, which works in conjunction with Adobe Air, or standalone tools such as Nokia Web Tools⁵¹. These tools generally provide project templates, a preview environment, validation, packaging, and deployment features.

⁵⁰ us.blackberry.com/developers/browserdev/widgetsdk.jsp

⁵¹ forum.nokia.com/Develop/Web/Tools#NWT

If there is a challenge in creating widgets, it is the lack of universal support for a common standard. W3C, together with Wholesale Application Community (WAC) and Joint Innovation Lab (JIL), is pushing forward with the definition of standards. This standardization is still underway and information on its progress can be found in the W3C Wiki⁵². Since standards are not complete, it is important to note that each widget technology has slightly different ways to implement the draft specifications and not all environments implement all of the draft standards. In general, a widget that follows the specifications given by W3C will enable you to target these widget environments:

- BlackBerry (v5.0 or later)⁵³
- Nokia WRT (on selected S60 3rd Edition, Feature Pack 2 devices and all S60 5th Edition and Symbian^3 devices)⁵⁴
- Nokia Browser for Series 40 (selected Nokia Series 40 devices)⁵⁵
- Vodafone360⁵⁶
- WAC/ JIL:⁵⁷
- Windows Mobile (v6.5)⁵⁸

⁵² w3.org/2008/webapps/wiki/WidgetSpecs

⁵³ bit.ly/blackberry-widgets

⁵⁴ bit.ly/nokia-wrt

⁵⁵ forum.nokia.com/webapps

⁵⁶ bit.ly/vf-widgets

⁵⁷ jil.org

⁵⁸ bit.ly/winmo-widgets

Distribution via a website is not the only option. Many application stores welcome widgets. At this point in time, the only store that supports W3C widgets is the Vodafone Widget store⁵⁹, but by packaging your widgets appropriately you can upload them into Nokia Store⁶⁰, the Windows Marketplace⁶¹, or RIM BlackBerry AppWorld⁶².

Test & Debugging

Firstly, create a test plan to assess how a WebApp actually behaves on a device. Decide what is needed to truly develop, test and debug. Should I buy all devices, or rent them⁶³? Decide on the browsers you want your app to run in. Based on the type of implementation, it is crucial to test the features and browser detection that influence the behavior of the app. You can use emulators⁶⁴. But be warned, emulators do not provide accurate information in terms of loading and rendering performance. Remote Debugging⁶⁵ can be performed using multiple tools.

⁵⁹ widget.vodafone.com

⁶⁰ store.ovi.com

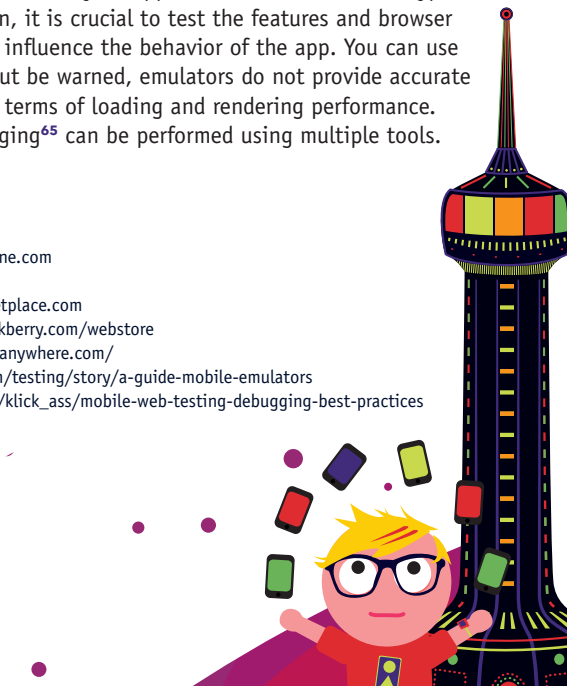
⁶¹ windowsmarketplace.com

⁶² appworld.blackberry.com/webstore

⁶³ keynotedeviceanywhere.com/

⁶⁴ mobiforge.com/testing/story/a-guide-mobile-emulators

⁶⁵ slideshare.net/klick_ass/mobile-web-testing-debugging-best-practices



Summary

The gap between native apps and web apps is rapidly decreasing. Browser vendors have done a great job rolling out new features. Creating mobile websites or mobile web apps makes your content accessible on almost any platform with much less effort than native development for several platforms. This does not only save initial development time and cost, but also reduces time and cost of future maintenance. Hybrid app frameworks can publish your apps in app stores. However, you still need to intelligently optimize content, because of the wide variety of browsers used on mobile devices. Deciding on a combined approach of server- and client-side optimization could be seen as the best option. While web apps are getting closer to native app capabilities, you need to create your own UI – but look out for templates that help simplify this task.

Some takeaways:

- Diversity is great
- The world is imperfect
- The Web is alive
- Browser/Device fragmentation is reality
- Web is not print
- Every screen/device and browser is different
- Content & Service should come first
- Add JavaScript & CSS
- Setup your project general
- Add the Special iPhone OMG Stuff – browser/device adaptations
- Speed displays the quality of your website
- Content preparation on the server side is much more comfortable for your clients



Accessibility

Regardless of the technology you choose to develop your apps, you will want to ensure that your app can be used by as many people in as many different markets as possible.

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 such like. Many of these users rely on third-party applications such as TalkBack on the Android platform or Talks from Nuance for the Symbian platform, which provides screen reading and screen magnification features. iOS now includes VoiceOver¹ which is the front-runner in terms of providing an accessible interface on mobile phones.

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 any third-party access software 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

¹ www.apple.com/accessibility/iphone/vision.html

- 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 online².

Some of the mobile platforms have built-in accessibility features that can help make it easier for people to use your apps.

In addition to accessibility features for users, some of the platforms include Accessibility APIs that help developers in two ways. Firstly, they can enable your app to be accessible with little or sometimes no extra work on your part. Secondly, they make it easier to develop assistive apps such as screen readers.

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

Developing Accessible Android Apps

The latest version of Android, Jelly Bean, brings a raft of accessibility improvements, these include the accessibility focus, Braille support and more. The developer documentation has also been enhanced. However, To maximise the reach of your app to those using previous versions of Android, you should use standard UI controls where possible and make sure users can navigate your app via a trackball or D-pad. This 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, please see Google's document entitled Making Applications Accessible³.

You will also find more examples in the training area of the developer documentation in a section entitled Implementing Accessibility⁴.

For more information about Android accessibility including how to use the text to speech API, see the Eyes-Free project⁵.

3 developer.android.com/guide/topics/ui/accessibility/apps.html

4 developer.android.com/training/accessibility/index.html

5 code.google.com/p/eyes-free



Developing Accessible BlackBerry Apps

BlackBerry also provides good and extensive information about the use of their accessibility API and many hints on accessible UI design on their website for developers⁶.

In May 2012 BlackBerry Released BlackBerry Screen Reader⁷ for the BlackBerry® Curve™ 9350, 9360 and 9370 smartphones. This is available as a free download which you may wish to use in the testing of the accessibility of your apps.

Developing 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.

⁶ https://developer.blackberry.com/java/documentation/intro_accessibility_1984611_11.html

⁷ www.blackberry.com/screenreader

If you are working on iOS, make sure to follow Apple’s accessibility guidelines⁸. These guidelines detail the API and provide an excellent source of hints and tips for maximising the user experience with your apps.

Developing Accessible Symbian / Qt Apps

At the time of writing, there is no “accessibility API” for the Symbian platform, however there are several third party apps that provide good access to many Symbian phones along with many of the apps they use.

When developing native Symbian apps your best chance of developing an accessible app is to use the standard UI controls where possible. If you are developing using Qt, then please check the web for details of their accessibility API⁹.

Developing Accessible Windows Phone & Windows 8 Apps

As with the other major mobile platforms, the accessibility features of Windows Phone are being enhanced with every successive release. Though, at the time of writing, Windows Phone 8 has not yet been fully unveiled.

You essentially have two choices when writing apps for the platform.

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 Basic¹⁰.

8 developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/iPhoneAccessibility

9 doc.qt.nokia.com/qq/qq24-accessibility.html

10 msdn.microsoft.com/en-us/library/windows/apps/xaml/hh452680.aspx

If you have chosen to use HTML 5 and JavaScript, then you will need Accessibility in Metro style apps using JavaScript¹¹.

Once you have tested the accessibility of your app¹², Microsoft uniquely allow you to declare your app as accessible¹³ in the Windows store, allowing it to be discovered by those who who are filtering for accessibility in their searches.

Developing 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) Overview¹⁴.

As support of HTML 5 is increasingly adopted on the various mobile platforms, you might find it useful to take a look at the document entitled Mobile Web Application Best Practices¹⁵ as this is likely to form the foundation of any mobile web application accessibility standard that emerges in the future.

¹¹ msdn.microsoft.com/en-us/library/windows/apps/hh452702.aspx

¹² msdn.microsoft.com/en-us/library/windows/apps/xaml/hh994937.aspx

¹³ msdn.microsoft.com/en-us/library/windows/apps/xaml/jj161016.aspx

¹⁴ w3.org/WAI/intro/wcag

¹⁵ w3.org/TR/mwabp

You will also find Relationship between Mobile Web Best Practices (MWBP) and Web Content Accessibility Guidelines (WCAG)¹⁶ a helpful resource.

16 www.w3.org/TR/mwbp-wcag/





Enterprise Apps

Corporate decision makers now view mobile enterprise apps as a strategic factor, a necessity, rather than an item on an accountant's spreadsheet. 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.

Internal enterprise apps are capable 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.

Many companies nowadays have a Chief Mobile Officer (CMoO) or equivalent position. It is the CMoO's 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 CMoO might consider setting up a Mobile Innovation Council (MIC). The MIC 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 needs to focus on together with the MIC 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) and Mobile Device Management & Security (MDM).
- **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.



Mobile Device 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. Mobile Device Management (MDM) has to include secure management of data, devices and applications.

This includes:

- 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
- Reprovision devices
- Back-up data on devices



Enterprise Apps



135

Mobile Enterprise Application Platforms

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 might want to consider evaluating Mobile Enterprise Application Platform (MEAP) solutions. MEAPs 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 Antenna¹
- Kony²
- SpringWireless³
- Sybase Unwired Platform⁴
- Syclo⁵

1 www.antennasoftware.com

2 www.kony.com

3 www.springwireless.com

4 www.sybase.com/products/mobileenterprise/sybaseunwiredplatform

5 www.syclo.com



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 320x240 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/vp8¹, an open video standard running on Android 4+ in an attempt to become the html5 standard (But not supported by Apple yet).

Here 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+ |
| Resolution | 176x144 (Older phones), 320x240, 480x320, 480x800, 640x480 (Blackberry), 960x640 (iPhone), 1024x768 (iPad), 1280x720 (BB10, Samsung) |

1 en.wikipedia.org/wiki/VP8

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/WCMA and 500 kbps for Wi-Fi and 4G. HD-video starts at 2Mbps and is not recommended for streaming (yet).

Apple's open source Darwin streaming server² can serve streaming video and audio with the highest level of compatibility and reliable RTSP combined with FFMPEG³ and is always a good choice to stream 3gp or mp4 files.

When targeting Windows Mobile/Phone, Windows Media Server⁴ 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

² dss.macosforge.org

³ www.ffmpeg.org

⁴ www.microsoft.com/windows/windowsmedia/forpros/server/server.aspx

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, 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.

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).

Possibly the most reliable open source hinting software available is Mp4box⁵. Note that an mp3 file does not need and cannot be hinted.

⁵ gpac.wp.institut-telecom.fr/mp4box/

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 eRightSoft⁶. 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/ open source background, check out Alembik⁷.

⁶ www.erightsoft.com/super

⁷ www.alembik.sourceforge.net



MDGG



SMART

Implementing Location-Based Services

Location based services are one of the hot areas for mobile applications. While nobody has yet proven that offering position and heading information in itself is very lucrative, apps which contain a geographically aware component, can provide more relevant services, which may lead to greater revenue. Knowing a user's location means you can deliver more relevant information; helping them find a nearby restaurant taking into account the local weather forecast, finding where friends are, or helping users find most scenic bike routes as crowdsourced by other bikers. Yet getting location data is only half the story, providing the user with a meaningful representation is a key factor in many apps, which usually means delivering a graphical representation overlaid with routes, points-of-interest et cetera. Yet, a comprehensive list of resources assorted by proximity, can many times be more fruitful than a scrollable, slow map view.

How To Obtain Positioning Data

Location-based applications can acquire location information from several sources; via 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 inputting data via 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. This information is not particularly accurate and depends on the cell size (base station coverage): Higher accuracy is obtained in urban areas than 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. For phones with WiFi capabilities, known wireless LAN access points can also be used. Several companies provide such WiFi data.

— **GPS positioning:**

An on-board GPS module (or an external one) gives you an accuracy ranging from 5 to 50 meters, depending on quality of the hardware and how many satellites the GPS module finds. Accuracy is also affected by the terrain, canopy and wall materials which 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 – such as near field communication (NFC), Bluetooth 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 the phone; such tags may contain an id from which a position can be looked up, or latitude and longitude.

— **Manual input:**

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.

How To Obtain Mapping Services

A map service takes a position as parameters and returns a map, often with metadata. The map itself can be in the form of one or more image bitmaps, vector data or a combination of both. Vector data has several advantages over bitmaps: it consumes less bandwidth and enables arbitrary zooming. However it requires more processing on the client side. Bitmaps are often provided in discrete zoom levels, each with a fixed magnification.

Free maps, both served as bitmaps and vectors, include Open Street Map¹ or CloudMate². Commercial maps include Garmin³, Microsoft's Bing resources⁴ to name a few.

1 wiki.openstreetmap.org/wiki/Software

2 developers.cloudmade.com/projects

3 garmin.com

4 microsoft.com/maps/developers

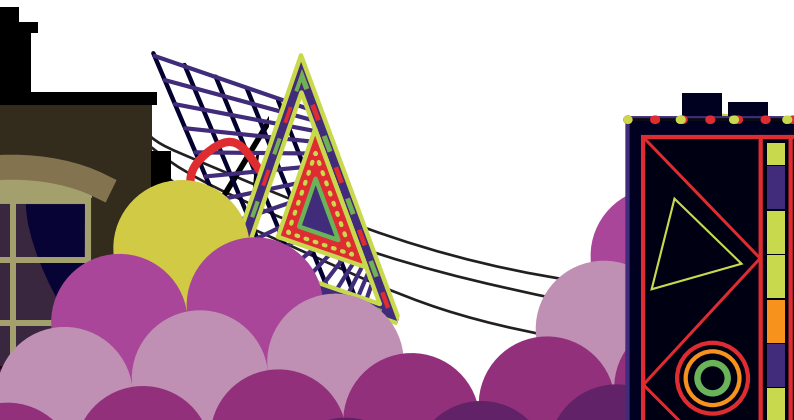
Some solutions, such as Google Maps⁵, are free when your application is made available at no cost, but require you to obtain a map key. Some map services, such as Google's static maps, are limited to serving a number of tiles to a map key or IP address. Several of the sources share similar map formats and are thus interchangeable.

Implementing Location Support On Different Platforms

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. 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

⁵ code.google.com/apis/maps



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 Symbian devices, Nokia Maps can be used free of charge including commercial use.

Microsoft's MSDN has gathered location-aware resources for Windows Phone 7 and beyond, under Location for Windows Phone⁶

Since iOS 3.x and Android 2.0, Web app developers have been able to access geoinformation via the `navigator.geolocation` interface, e.g calling `navigator.geolocation.getCurrentPosition(my_geo_handle)` gives you the opportunity to fetch the `my_geo_handle.coords.latitude` and

⁶ msdn.microsoft.com/en-us/library/windowsphone/develop/ff431803

`my_geo_handle.coords.latitude`, after given permission from the user and satellites are available. Via clever scripting, this action can be combined with fallbacks to network lookups.

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 webservice, 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 geofmt used in Google Earth and related web services.

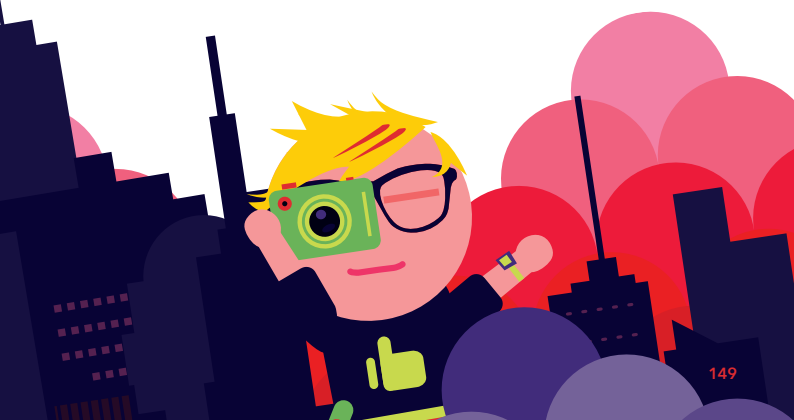
Tools For LBS Apps

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 lesser range of mobile platforms.

Advertisement companies like Admob offer developers a stand-alone location aware advertisement program, to better target their offerings, while there are no map interfaces to be seen, just the coordinates.

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

- **Garmin Mobile XT SDK:** developer.garmin.com
- **Android offline maps project:** code.google.com/p/big-planet-tracks/
- **TeleAtlas:** developerlink.teleatlas.com
- **Nutiteq:** www.nutiteq.com
- **RIM:** us.blackberry.com/developers/ (search for “map api”)
- **Nokia Maps:** api.maps.nokia.com





Implementing Near Field Communication (NFC)

Near Field Communication (NFC) is one of the latest technologies to come to mobile devices. It is a very-short range radio technology, typically operating in a 0 to 4cm range, that relies on a tag – that stores data – and a reader to read and write a tag's data. NFC enabled mobile phones are typically able to act as either a tag or a reader.

The appeal of NFC as a technology for mobile applications is the simplicity of operation, the user needs only to place their phone in close proximity to a NFC tag or reader – there is no setup or configuration to be done. The challenge with NFC will be educating users about the technology, as its use will be a new experience to many and does not have a direct analogy in current behavior. For example, how many users will see the action of touching a poster as an obvious way of opening a related website? However, there is an entire industry poised to educate users on the technology, so there are many opportunities for early adopters.

The NFC standards¹ provide for three modes of operation that can be used in mobile devices:

- **Read/Write:** where a phone can read or write data to a tag
- **Peer to Peer:** where two NFC enabled phones can exchange data, from information for creation of a Bluetooth connection through to business cards and digital photos

¹ www.nfc-forum.org/specs/spec_list

- **Card Emulation:** where a phone can act as a tag or contactless card

Types of use cases envisaged for NFC in mobile phones include:

- **Service Initiation:** here a phone can read a tag embedded or attached to everyday objects, the tag would provide a URL, phone number or application specific string that can be used to open a website, dial a number or initiate application specific functionality. A practical application might involve embedding a tag in a product's packaging to provide a way of opening the product's website
- **Sharing:** here two NFC enabled phones could share a piece of information, a business card for example.
- **Connecting devices:** an NFC enabled phone could read connection settings from another phone or peripheral. For example, a Bluetooth headset could include a tag that provides the information for pairing the headset with a phone
- **Ticketing:** the NFC phone could be delivered a ticket which is then "redeemed" by being read from the phone
- **Rechargeable or cashless payment cards:** here the phone can act as a replacement for a credit card or bank card, travel cards such as Oyster² or payments cards such as Snapper³.

2 oyster.tfl.gov.uk/oyster

3 www.snapper.co.nz



Support For NFC

Support for NFC in mobile devices is still relatively new. However, the technology is arriving in the mainstream with Apple, BlackBerry, Google, Microsoft and Nokia⁴ all having announced NFC support in their platforms and manufacturers such as Google, BlackBerry, Nokia and Samsung having announced or already started shipping smartphones with NFC capabilities⁵.

Creating NFC Apps

One challenge in creating NFC applications is that there is no single standardized API. While Contactless Communication API (JSR-257) provides a standard, it is not universally available (Apple and Google for example certainly will not provide support for it). Where it is offered, it can be supplemented with additional manufacturer specific APIs, as Nokia does for example.

Nokia provides support for NFC in the Qt Mobility APIs⁶, making it likely that a single set of APIs can be used for Symbian phones and the Nokia N9.

Otherwise it will essentially be one set of APIs per platform, such as the Google APIs for Android⁷.

However, conceptually NFC is not that complex so the number of APIs to master across multiple platforms should not be a hindrance.

4 www.forum.nokia.com/nfc

5 www.nearfieldcommunicationsworld.com/nfc-phones-list

6 labs.qt.nokia.com/2011/04/12/qt-mobility-1-2-beta-package-released

7 developer.android.com/reference/android/nfc/package-summary.html



Implementing Haptic Vibration

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.

The iOS platform

iOS may have the least amount of vibration documentation for developers as Apple currently gives developers little vibration control for their devices. The iOS vibration method below applies to iPhones only. iPads and iPods current have no motors for vibration support.

Use the `SysSoundViewController Class`¹ 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 about 2 seconds.

The Android Platform

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 this platform may be include existing accessibility haptic effects. For instance the

¹ developer.apple.com/search/index.php?q=SysSoundViewController

KickBack Accessibility Service² provides haptic feedback. 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`³ Class 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.

Extended Android Vibration Control

Because the Android platform is open source, there is at least one company that offers methods to extend Android's vibration control. Immersion Corporation's Haptic SDK⁴ allows full vibration motor control of duration, amplitude and pulsing frequency with a library of 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 conserving battery life through smart control of vibration amplitude and frequency parameters. The extended method also contains an abstraction layer in its effect libraries to compensate for the difference in motor types between hardware manufacturers. This is important for developers looking to program feedback effects once and have

² KickBack is available as part of the eyes-free open source project code.
google.com/p/eyes-free

³ developer.android.com/index.html#q=Vibrator

⁴ immersion.com/haptic/sdk

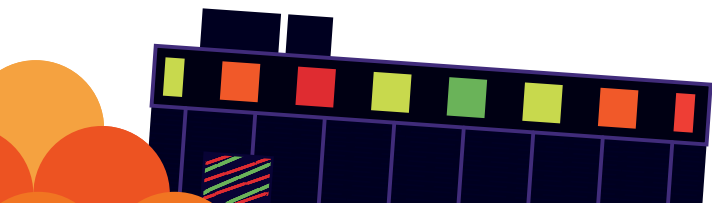
a consistent user experience across all Android devices. This extended method supports Android 1.6 and above.

The bada Platform

Like the Android platform, bada has both a basic and extended methods of vibration control. Bada uses three classes for different types of vibration control: `Vibrator`, `TouchEffect` and `Haptic`. The `Vibrator` class is the basic method used for notification alerts and uses `Start()` and `Stop()` methods. Within this structure you can set four types of parameters for vibration patterns. The `onPeriod` and `offPeriod` parameters allow you to designate in milliseconds the length of your vibration period. The `level` parameter allows you to set the intensity or magnitude of the vibration from 0 to 100, controlling the percentage of voltage to the vibration motor. Additionally, you can set a `count` parameter to repeat your vibration pattern.

The `TouchEffect` and `Haptic` classes are bada's extended vibration methods. The `TouchEffect` class is used to playback vibration and/or sound in respond user actions or application events but mostly for UI events. The `Haptic` class is used primary for applications development. Both the `TouchEffect` and `Haptic` classes are only for bada devices using the TouchSense Player API⁵ found in most high-end bada devices.

⁵ developer.bada.com/search/searchIndexList.do?searchValue=Vibrator



BlackBerry Platform

BlackBerry gives you the same basic on/off vibration control that Android does, but without an extended method. For BlackBerry you use the `VibrationController`⁶ Class with `startVibrate(int duration)` and `stopVibrate(int duration)`

Windows 7 Platform

Windows offers a basic method for vibration control, but no extended method at this time. Use the `VibrateController`⁷ Class 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.

At this time of this writing there was no officially documented method for vibration control from Microsoft. However, we expect the Windows 7 class listed above should be compatible.

Haptic Vibration Design Considerations

When designing a user interface, 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,

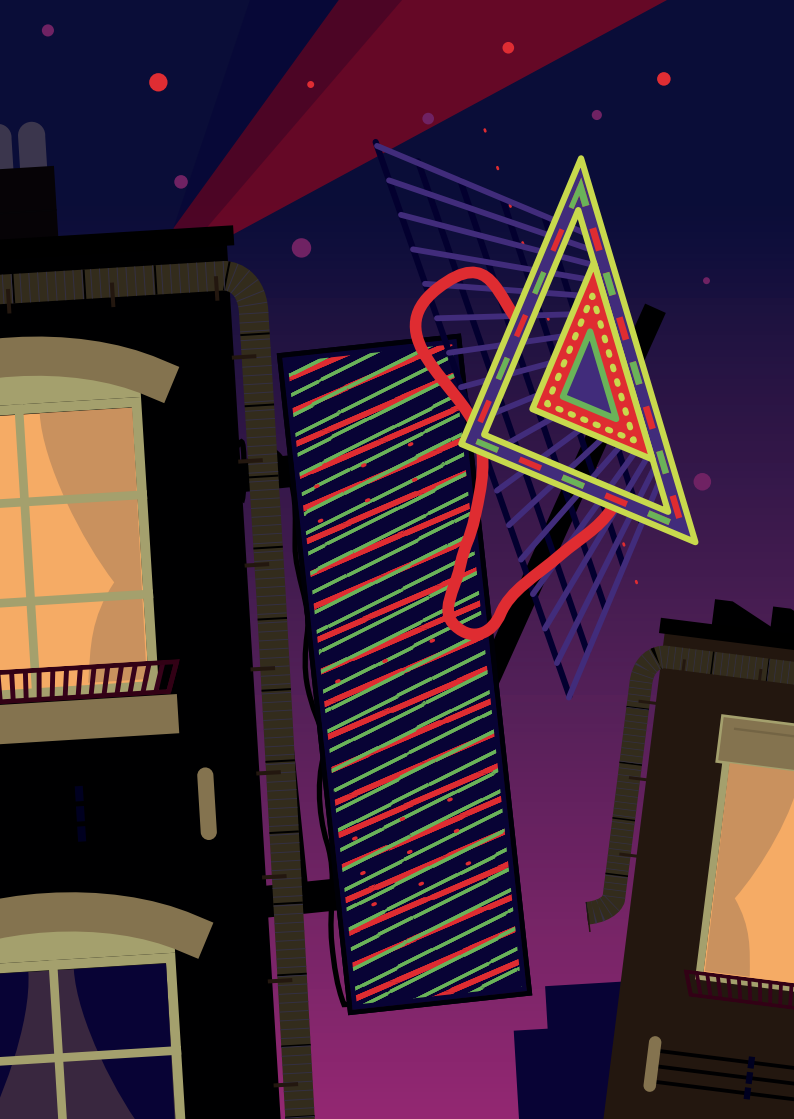
⁶ developer.blackberry.com/search/?search=VibrationController

⁷ social.msdn.microsoft.com/search/en-us?query=VibrateController

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.





Security

Readers of this guide know how widespread smart mobile devices have become and how useful mobile apps can be. We use these powerful, connected and mobile computers for a multitude of things every day. 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'.

Companies are now developing apps that 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).

Clearly mobile apps are going to attract the attention of hackers and thieves whose interests extend well beyond getting a 99 cent app for free. 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 size, type, and values allowed. Error handling needs to provide useful error

messages to users that do not leak sensitive information. Do not print stack traces or system diagnostics that hackers can leverage to penetrate further. Penetration testing of applications is needed to assure that identification, authentication and authorization controls cannot be by passed. 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 should be tested.

All the main mobile platforms require applications to be cryptographically signed, so that binaries can be traced back to their source and modified programs detected. Binary signing gives the mobile platforms a higher level of application security than used by many desktop platforms, but there are still plenty of security threats that need to be addressed.

General Concepts

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 applications can covertly make phone calls and send SMS messages to expensive numbers, which cost users lots of money.

Conversely, the application developer may be concerned about their reputation, loss of revenue, and loss of intellectual property. And 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?

Each mobile platform provides a distinct set of security features. For instance Android has the concept of permissions, which allow the application access to sensors such as the GPS and to sensitive content. 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 limit the number of permissions or features your application needs to an absolute minimum.

The 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 (or malware) that could steal your users' data 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.

The next step is to explore defenses against these threats, starting with protecting managed code.

Hiding the Map of Your Code

Most mobile platforms are programmed using managed code (Java or .NET), which is executed by a virtual machine rather than directly on the CPU. As well as the instructions for the virtual machine, the binary formats for these managed code 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 that programmers love and that some GUIs depend on (e.g. reflection). However, metadata is also very helpful to a hacker trying to reverse engineer the code. There is no need for the hacker to guess what every piece of code does when the metadata reveals the friendly, logical name that its developer chose.

Fortunately, there are several obfuscation tools, such as ProGuard¹ and Arxan's GuardIT², that can help by processing the compiled Java or .NET binary and give randomized names to most items. Adopting one of these tools and using it on every application will make it much more difficult to reverse engineer your software.

1 proguard.sourceforge.net

2 www.arxan.com

On the Android platform there is also 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.

That said, even native code is not immune to hackers finding interesting and useful function names.

“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 switch³ correctly is an easy way to make it harder to understand the code.

Finally, do not be fooled by the “C” in Objective-C. Along with the machine code, compiled Objective-C code contains a lot of metadata which can provide an attacker with a wealth of 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.

³ gcc.gnu.org/wiki/Visibility

Hiding Control-Flow

Even if all the names are hidden, a good hacker can still figure out how the software works. This is particularly true for platforms using Java and .NET code, which use high-level instruction sets and rely on the virtual machine to optimise execution. These factors make it much easier to create tools that will reverse the compilation process and create valid, understandable source code from the binary. There are many excellent de-compilation tools for both Java and .NET freely available to every hacker.

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, using a tool such as the Hex Rays de-compiler⁴. Even without a tool, it does not take much practice to be 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.

4 www.hex-rays.com



Protecting Network Communications

The network communications is 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.

Active Protection That Stays With The Application

The next step after passively hiding the functions within the code base is to actively detect attempts to tamper with the application and respond to those attacks.

You may be able to roll-your-own; for instance, several techniques for protecting Android code are documented at <http://androidcracking.blogspot.com/>. Commercial products are another option, and some native code protection products, such as Arxan's EnsureIT, allow 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. The use of code protection products can be implemented such that release schedules are not affected and the protection is tunable to a desired level of security as well as being resilient to attack.

Using one of these software protection tools to actively protect your software will give you the best chance of keeping your secrets secret.

White-Box Cryptography

Cryptography code always handles sensitive data and therefore needs special attention. Nearly all applications handling encrypted data use the same small list of cryptographic algorithms (mainly AES, RSA and ECC). These algorithms are relatively secure, but what if an attacker can find the 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.

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. Enter white-box cryptography. The aim of white-box cryptography is to implement the standard algorithms in a way that allows the keys to remain hidden. Some versions of white-box cryptography use complex mathematical approaches to getting the same results in a different way. Others embed keys into look-up tables and state machines that are difficult to reverse engineer.

Few application developers have the skills to write their own secure cryptography code, but some of the companies that offer software security tools also sell white-box cryptography libraries. 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 passwords or other sensitive data on devices, unless secure storage is used protected by a complex password. Instead, store authentication tokens that have limited lifetime

and functionality. Be aware of the privacy policies that apply in the country where you will be selling your application. 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 -- most OS protections can be bypassed trivially by jailbreaking the device. Do not trust that native keystores will keep data secret -- keystores can be broken by bruteforce guessing unless the user protects the device with a long complex password.

Protection Tools

Basic Java code renaming can be done using Proguard⁵, an open-source tool. For more durable software protection you will need to use a commercial tool.

Two vendors for managed-code (Java and .NET) protection tools are Arxan Technologies⁶ and PreEmptive Solutions⁷.

For native code protection tools and white-box cryptography libraries, the main vendors are Arxan and Irdeto⁸.

Resources

Here are some useful resources and references which may help you:

⁵ www.proguard.sourceforge.net

⁶ www.arxan.com

⁷ www.preemptive.com

⁸ www.irdeto.com

- Apple provides a general guide to software security⁹. It also includes several links to more detailed topics for their platform.
- Commercial training courses are available for iOS¹⁰, Android¹¹, and Lancelot Institute¹² provide secure coding courses covering iOS and Android.
- O'Reilly (2011) published a book on Android security Jeff Six: Application Security For The Android Platform. Processes, Permissions and Other Safeguards (Dec 2011)¹³ and another for iOS, Jonathan Zdziarski: Hacking and Securing iOS Applications¹⁴ .
- Charlie Miller et al. (2012) published iOS Hackers Handbook¹⁵, which demonstrates how easy it is to steal code and data from iOS devices.
- A free SSL tester is provided by Qualys Labs¹⁶.
- Extensive free application security guidance and testing tools are provided by OWASP¹⁷, including the OWASP Mobile Security Project¹⁸ .
- An open-source mobile application performance monitoring tool for Android is provided by AT&T's Application Resource Optimization tool¹⁹.

⁹ www.securecoding-iphoneapps.com/

¹⁰ developer.apple.com/library/mac/navigation/#section=Topics&topic=Security

¹¹ marakana.com/training/android/android_security.html

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 www.ssllabs.com/ssltest/

¹⁷ www.owasp.org

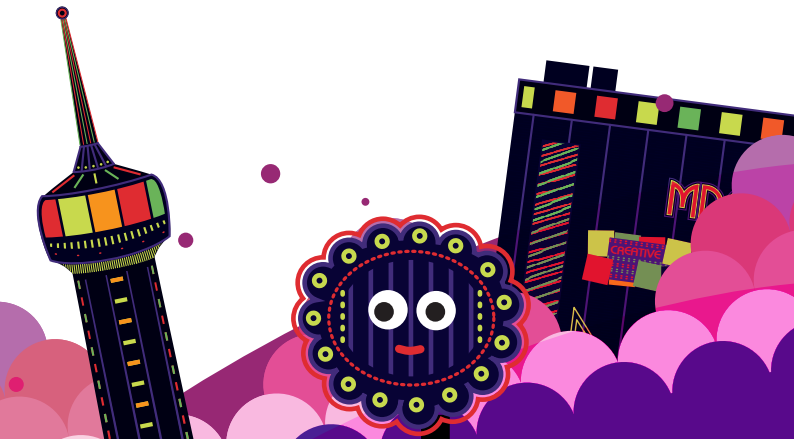
¹⁸ www.owasp.org/index.php/OWASP_Mobile_Security_Project

¹⁹ developer.att.com

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 the potential threats relevant to your application. Reduce the security footprints in terms of permissions (Android) or entitlements (iOS). Perform code reviews and strip out non-essential logging, debugging methods, and so on. Consider how a hacker would analyse your code and use similar techniques, in a safe and secure environment, against your app to discover some of the vulnerabilities before they do, so you can mitigate these vulnerabilities and make your app more secure.





Testing

After all your hard work creating your application how about testing it before unleashing it on the world? Testing mobile applications used to be almost entirely manual, thankfully automated testing is now viable for many of the mobile platforms. Several of the major mobile development platforms include test automation in the core tools, including Android and iOS.

Cross-platform test automation tools are available for popular platforms; some are free-of-charge and open-source, others are commercial. This chapter covers the general topics; testing for specific platforms is covered in the relevant chapter.

Testing Through The Five Phases of an App's Lifecycle

The complete lifecycle of a mobile app fits into 5 phases. Testing applies to each phase.

1. Implementation:

This includes design, code 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. Ask questions of the design to shape the design so it will fulfil the intended purposes, while avoiding making serious mistakes. Phillip Armour's paper on five orders of ignorance¹ 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 coding.

1 www-plan.cs.colorado.edu/diwan/3308-07/p17-armour.pdf

And test your build process and build scripts to ensure they are effective, reliable and efficient, otherwise you are likely to suffer the effects of poor builds throughout the life of the app.

2. **Verification:**

This includes unit tests, internal installation and system tests: We have already mentioned unit tests, remember to review them and assess their potency - how effective and useful they are. Are they really useful and trustworthy? For apps that need installing we need ways to deploy them to specific devices for pre-release testing. For some platforms including Android, iOS and Windows Phone; the phones need to be configured so the apps can be installed. System tests are often performed interactively, by testers. We also have a wide range of test automation tools and frameworks for mobile apps these days and should consider using these for some of our system tests. We will go into more detail later in this section.

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.

4. **Engagement:**

This includes search, trust, download and installation. Once your app is publicly available users need to find, trust, download and install it. We can test each aspect of this phase. 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.

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.

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. Buy, beg, borrow various

phones to test on. Select phones based on their popularity, capabilities, version of the operating system, and so on.

Test your app when on the move, for instance when you only have one hand free and are trying to enter text, make selections in the UI, et cetera. Test the effects of intermittent connectivity and how the app responds.

Rotate the screen and make sure the app is equally attractive and functional. Make sure it does not lose information. If your app relies on sensors e.g. GPS, accelerometers, et cetera, then find ways to control or affect these inputs to the app. Sometimes the development tools enable you to fake these inputs e.g. with mock locations.

The various guidelines at appqualityalliance.org/resources are worth considering when devising your test cases.

Physical Devices

Although emulators and simulators can provide rough-and-ready testing of your applications, and even allow tests to be fully automated in some cases, ultimately your software needs to run on real phones, as used by your intended users. The performance characteristics of various phone models vary tremendously from each other and from the virtual device on your computer.

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.



- **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.

For platforms such as Android and Java ME where there are so many manufacturers and models, it is particularly useful to test on a range of these devices. 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 we want users with these devices to be happy too.

Remote Control

If you have physical devices to hand, use them to test your application.

However when you do not, 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. For instance 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 develop-

ers. Both Nokia² (for MeeGo and Symbian) and Samsung³ (for Android and Bada) provide restricted but free daily access.

You can also use commercial services of companies such as PerfectoMobile⁴ or DeviceAnywhere⁵ 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 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-Sourcing

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 and mob4hire. 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.

- 2 apu.ndhub.net/devices
- 3 rtl.innovator.samsungmobile.com/
- 4 www.perfectomobile.com
- 5 www.deviceanywhere.com



GUI Test Automation

GUI test automation 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. 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.

2012 has seen lots of new contenders of test automation tools and services. Some commercial companies have open-sourced their tools GorillaLogic's MonkeyTalk⁶ and LessPainful's Calabash⁷. These tools aim to provide cross-platform support particularly for Android and iOS. The companies charge for consulting and other services, the software is free to use.

Sadly several opensource projects appear to be mothballed including several we mentioned in earlier editions. Others including Robotium⁸ and Frank⁹ are doing well and may even have been incorporated into other test automation tools.

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.

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

⁶ www.gorillalogic.com/testing-tools/monkeytalk

⁷ <https://github.com/calabash>

⁸ code.google.com/p/robotium/

⁹ testingwithfrank.com/

to create a very basic UI that is 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.

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.

Separate The Generic From Specific

Many mobile applications include algorithms, et cetera, unrelated to mobile technology. This generic code should be separated from the platform-specific code. For example, on Android or J2ME the business logic can generally be coded as standard Java, then you can write, and run, automated unit tests in your standard IDE using JUnit. Consider platform-specific test automation once the generic code has good automated tests.

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 even 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

Test-Driven Development (TDD) has become more popular and widespread in the general development communities, particularly when using Agile Development practices.

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*¹⁰. You can also consider using TDD for the generic aspects of the client code.

¹⁰ www.informit.com/store/product.aspx?isbn=0321774183

Web-Based Content And Applications

We can benefit from the extensive history of test automation tools for desktop web-based content and applications to automate aspects of our Mobile equivalents.

Tools such as WebDriver wrap web browsers, including, headless WebKit, Android, iPhone, Mobile Opera, and BlackBerry as well as the main desktop web browsers. Google published a useful blog post¹¹ on how to write automated tests using the AndroidDriver.

On the desktop the ability to wrap Firefox means it can crudely emulate most mobile browsers by programmatically changing browser parameters such as the user-agent string. There is an article on the Google Testing blog¹² that includes an example of how to emulate the iPhone browser¹³. Beware, the behavior of desktop browsers differs significantly from those on mobile devices so test again using mobile web browsers before launching your web app.

For interactive testing we can use the various emulators supplied for various mobile platforms; and Opera have released Opera Mobile Emulator, which allows us to quickly test how sites would look and behave on the various platforms supported by Opera Mobile¹⁴.

¹¹ google-opensource.blogspot.co.uk/2011/10/test-your-mobile-web-apps-with.html

¹² googletesting.blogspot.com

¹³ googletesting.blogspot.com/2009/05/survival-techniques-for-web-app.html

¹⁴ www.opera.com/developer/tools/



\$\$\$
monetize

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 or for a charity, 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. **Revenue sharing:** Earn revenue from operator services originating in your app
5. **Indirect sales:** Affiliates, data reporting and physical goods among others
6. **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 – in most cases it will be one offered by the target platform's owner, such as

Apple, Google, RIM, Microsoft or Nokia – 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.

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. Premium SMS is still very popular for mobile web applications, Java games, wallpaper and ringtones.

Other operator APIs enable you to include features such as MMS, Call Back and Multimedia Conference in your app and earn revenue from their use. However, operator billing has been quite difficult to handle particularly if you want to sell in several countries, as you needed to sign contracts with each operator in each country.

In 2011, 57 of the world's largest mobile phone network operators and manufacturers founded the Wholesale Applications Community (WAC)¹, a not for profit organization that helps to standardize the mobile applications ecosystem. One of their key products is the WAC Payment API, allowing a developer to easily interface with all connected operators. As recently announced², WAC will be integrated into GSMA. GSMA has the ability to make the technology available to their 800 operator members worldwide.

Each operator will take a revenue share typically 45% to 65% of the sale price, but some operators can take up to 95% of the sale price (and, if you use them, a mediator will take its share too). Security (how you prevent the copying of your app)

1 wacapps.net

2 www.gsma.com/newsroom/gsma-and-wac-join-forces-to-accelerate-mobile-applications-market/

and manageability are common issues with PPD but for some devices this might be the only option.

As of Android 4, Google decides to ask for Credit Card data at sign-up, something that Apple already required since 2008, which according to analysts is the key differentiator for higher monthly per app revenue. In addition to customers on monthly billing arrangements, pre-pay customers can use their pre-paid credits to purchase apps. Like BlueVia's in-app payment API, this is particularly significant for developers targeting emerging markets where credit cards ownership is low.

It is worth noting that most of the vendor app stores are pursuing operator billing agreements, with Nokia Store having by far the best coverage with operator billing available in 46 countries. In addition, Nokia will be bringing its expertise to Microsoft's Windows Phone Marketplace to rapidly expand its operator billing coverage. Google and RIM are actively recruiting operators too. 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, at least, also insulates developers from the variation in operator share, offering developers a fixed 70% of billing revenue.

The last option is to create your own website and implement a payment mechanism through that, such as PayPal mobile, Dutch initiative eM! Payment³, dial-in to premium landlines⁴ 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.

³ empayment.com

⁴ daopay.com

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 could 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.

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 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 method. If your application is implemented across various platforms, you may have a different mechanism to build into each platform’s version.

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, or with in-app payment offered directly by operators. In Germany Deutsche Telekom, Vodafone and Telefónica/02 have become

the first operators to launch an in-app payment APIs that work cross-operator and enable billing directly to the phone bill of the user. 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 *smaato*⁵, *Madgic*⁶ or *inneractive*⁷ as they tend to bring you better earnings by combining and optimizing ads from 30+ 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

⁵ smaato.net

⁶ www.madgic.com madgic.com

⁷ www.inner-active.com inner-active.com

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.

It may require some experimentation to find the right level and positions in which to place adverts.

Revenue Sharing

Revenue sharing with mobile operator for services built into your app is an emerging opportunity for developers, and one that is worth following. This monetization method lets developers build services such as SMS, MMS, location, advertising, customer profile and operator billing into their apps. With well-documented APIs that are free to use, revenue generated is split transparently between operator and app owner.

While BlueVia is currently the only developer community dedicated to this model, if its early adoption continues to grow, it may become a recognized business model for mobile operators.

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 mechanism such as:

1. **Affiliate programs:** Promote third party or your own paid apps within a free app. See also MobPartner⁸. 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 Mac Donald's and Starbucks. Also coupon applications often use this business model

There is nothing to stop you 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 overcharging.

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 of software, which provides a defined functionality, that is to be used by higher level software.

⁸ mobpartner.com

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. <http://opensource.org/docs/osd> provides a good definition of open source.

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 don't 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. Recently, Developer Garden and Verious started into this domain⁹. Also, the idea to open the marketplace to semi-professional suppliers is new. In Developer Garden, you can register yourself and offer your component to the public – for free or as a paid offer.

Now imagine you built a piece of code which could be seen as one or even several components. Next steps would

⁹ www.developergarden.com/component-marketplace/

include: providing extensive and detailed documentation. You might then also add several “hello-world” samples using your component. Combine the component in source, or precompiled with the documentation and the samples, and you are ready to offer it on the component marketplace. Since your customers will probably ask questions (the better the documentation is, the fewer questions you will get), it often also makes sense to offer an FAQ.

Marketing And Promotion

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 – even the smallest stores have application counts in the 10s of thousands, so there will be a lot of competition competing for users’ attention.

Some stores enable you to purchase premium positioning either through banners or list placings. But in most cases you will also need to think about other promotion mechanisms, such as social networks, reviews on technology websites, Twitter, Press releases. Nokia provides a useful Vendor independent page of information on marketing your apps¹⁰.

¹⁰ forum.nokia.com/Distribute/Public_relations_guidelines.xhtml



Strategy

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 at first, then start adding mobile advertising or split between a free and paid version, when you have more than 100 thousand users worldwide
- 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



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 given up their day job anytime soon. 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.

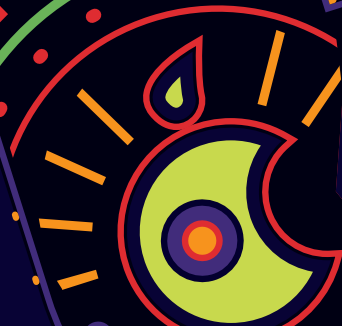
Learn More

If you want to dig deeper into the topic, check out the "Mobile Developer's Guide To The Parallel Universe" published by WIP. The 2nd edition has just been published and is available on their events and their website¹¹.

¹¹ wipconnector.com

GALACTIC

FIGHT CLUB



Appstores

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 standout 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 you raise your odds.

Top 5 Appstores

| Appstore | Platform | Daily Downloads | Alternatives |
|------------------------|--------------------------------|-----------------|--|
| Android Market | Android | >50 million | GetJar, Samsung, Motorola, Amazon and 50+ others |
| Apple iTunes App Store | iOS | >33 million | Cydia (Jailbroken iPhones) |
| Nokia Store | Symbian, Qt, Widget, Java | >11 million | GetJar |
| Blackberry App World | BlackBerry | >3 million | Crackberry |
| GetJar | Java, Symbian, Android, widget | >2 million | Appia, Handster, Nexva |

The volume of downloads makes some of these stores look promising. However, the stores with the highest volume attract the largest number of applications fighting for attention, so you might want to pick your niche carefully or spend more time marketing your app.

For example, a research4market's study showed that in Q2 2011, Apple's App Store and the Android Market were behind Nokia's Ovi Store, Windows Marketplace and BlackBerry's App World when it comes to the number of downloads any particular app might achieve¹. Also, it is worth noting that downloads do not necessarily equal profits. Despite having fewer downloads, apps in the Apple App Store generate four times the profit of apps in the Android Market, according to research from Distimo². Make sure that your platform choice, app store choice and business model all align with your revenue goals.

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. 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.

- 1 www.research2guidance.com/apps-on-nokia-s-ovi-store-had-2.5-times-higherdownload-numbers-in-q2-2011-compared-to-apps-on-apple-app-store/
- 2 techcrunch.com/2011/12/20/distimos-year-end-report-shows-why-developers-love-ios-iphone-4x-android-revenue-ipad-2x/

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 (especially if you have a well-known brand) |
| 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 | iPhone developers only need 1 appstore |



Now What – Which Environment Should I Use?

The answer to that questions is of course far from obvious. It depends on what you are trying to achieve, how much budget you have, what your strengths and weaknesses are and how well you have defined your strategy. In this chapter, we will try to give you a hand, presenting some key facts and figures that might help you make up your mind.

The Business Perspective: Market Reach

The vast growth of smartphone penetration, from 28% in Q2 2011 to nearly 40% in Q2 2012 has opened up new horizons for developers, as users are downloading more and more apps on their devices. But this growth is unevenly distributed amongst the major smartphone platforms. Android and iOS claim an increasingly bigger piece of the pie; with bada, BlackBerry, Symbian and Windows Phone getting the leftovers. According to a recent research report Developer Economics 2012¹, a platform's reach is the most important factor for developers when choosing a platform, meaning that developers are still flocking around the Apple and Google platforms.

How does each platform fare in terms of reach? Let us take a look at smartphone shipments and market share per platform in Q2 2012.

¹ www.DeveloperEconomics.com

| Platform | Market Share | Shipments in Q2 | Installed base |
|---------------------------|--------------|-----------------|----------------|
| Android (Google) | 68% | 103 million | 427 million |
| iOS (Apple) | 17% | 26 million | 198 million |
| BlackBerry (RIM) | 5% | 8 million | 108 million |
| Symbian (Nokia) | 4% | 7 million | 259 million |
| Windows Phone (Microsoft) | 3% | 5 million | 14 million |
| bada (Samsung) | 3% | 4 million | 19 million |

(Source: VisionMobile estimates, Tomi Ahonen)

You have to remember that these are global figures - the regional market share of each platform is another matter altogether. In a world where localised content is increasing in importance, it is essential to know the details and characteristics of your home market. For example, China is now the largest smartphone market, having recently outpaced the US in terms of handset activations. Moreover, while Android's global market share was close to 70% in Q2 2012, its share was even greater in China, with a staggering 81%.

To find out about market share in your target region, check out online resources such as comscore², StatCounter³, Vision-Mobile⁴ or Gartner⁵.

2 www.comscore.com/category/mobile

3 gs.statcounter.com

4 www.visionmobile.com

5 www.gartner.com

Another piece of the puzzle is available apps and downloads per platform. Android and iOS are the clear winners in that respect, but Windows Phone is steadily growing. Let us take a look at available apps and downloads per smartphone platform, as of Q2 2012.

| Platform | Available Apps | Cumulative downloads |
|---------------------------|----------------|----------------------|
| iOS (Apple) | 670 thousand | 31 billion |
| Android (Google) | 600 thousand | 20 billion |
| Windows Phone (Microsoft) | 110 thousand | N/A |
| BlackBerry (RIM) | 90 thousand | 3 billion |

It is clearly evident that Android is quickly catching up to iOS in terms of available apps - and is also creeping up to Apple in terms of cumulative downloads. However, Apple users are generally more likely to be willing to actually spend money on apps while Android users are expecting to get their software for free.

Another piece of the puzzle is how many developers are competing over the same piece of the pie. Usage of each platform varies greatly, with Android and iOS again being in the lead. The Developer Economics 2012 research Developer Economics 2012⁶ shows that Android and iOS are at the top in terms of mindshare - i.e. the percentage of developers using each platform, irrespective of which platform they consider to be their 'primary'. Other platforms, like Symbian and BlackBerry are quickly losing ground.

⁶ www.DeveloperEconomics.com

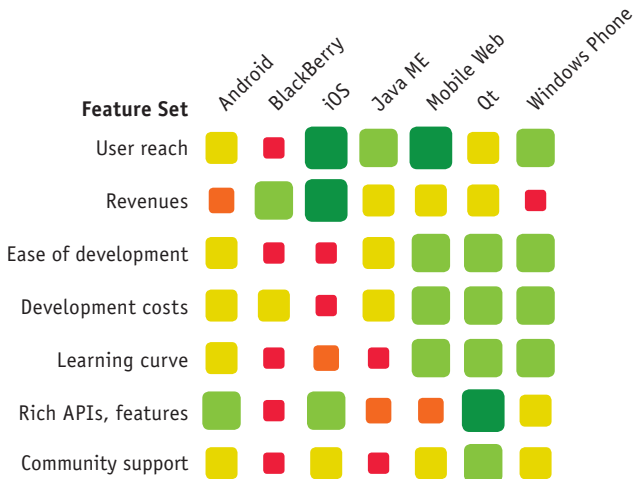
The table below shows the “Mindshare Index” of each platform, as well as the YoY gain or loss.

| Platform | Mindshare Index | YoY change |
|---------------------------|-----------------|------------|
| Android (Google) | 76% | +13% |
| iOS (Apple) | 66% | +12% |
| mobile web | 53% | -5% |
| Windows Phone (Microsoft) | 37% | +3% |
| BlackBerry (RIM) | 34% | -24% |

The Developer’s Perspective: Technology

Market reach is not the only deciding factor when choosing a platform - technology is also a critical factor. Many developers choose a platform based on their existing skills and knowledge - as is the case with Java and Android. Other developers choose their platform depending on revenue potential, while others consider the associated costs of developing an app on a given platform. Each platform comes with its own advantages and disadvantages.

The table on the right presents developers’ opinions on each platform - based on the Developer Economics 2012 research.



Poor Between 5% and 10% below average
Bad 10%+ below average
Average Average across platforms $\pm 5\%$
Good Between 5% and 10% above average
Great 10%+ above average



There are many challenges when it comes to development - fragmentation being the top one for many developers today. But there are also many post-launch challenges, such as tracking errors and bugs or customer support. The Developer Economics research identified the top challenges for developers: "Tracking errors and bugs" is identified as a challenge by 38% of developers, followed by "Getting users to review your apps" (30%) and "Updating the applications in the field" (25%).

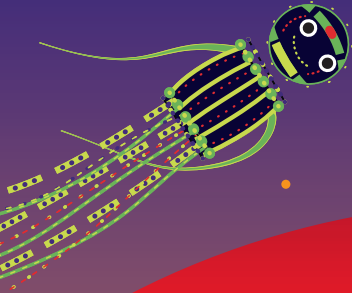
The Developer's Perspective: Marketing

Marketing; a dreaded word for many developers! But marketing is an integral part of every business venture and mobile development is no exception. Even if you build the perfect app, it is not important if nobody is using it. As any textbook says, marketing strategy is a combination of four parameters: product, price, promotion and place. All four are important when creating your marketing mix, but let us focus on price for now.

There are more than ten revenue models to choose from⁷, leaving developers dazed and confused when it comes to picking the right one. In terms of use, pay-per-download and advertising are the most popular models - but that does not mean they suit your strategy. The Developer Economics 2012⁸ research identified not just the most common revenue models, but also the most lucrative ones. The table to the right presents the percentage of developers using each model, as well as the average per-app month revenue for each one.

⁷ See the monetization chapter in this guide for details

⁸ www.DeveloperEconomics.com



| Revenue model | Percentage (%) of developers using model | Average revenue per app month |
|------------------|--|-------------------------------|
| Subscriptions | 12% | \$3,683 |
| In-app purchases | 19% | \$3,033 |
| Pay-per download | 34% | \$2,451 |
| Freemium | 18% | \$1,865 |
| Advertising | 33% | \$1,498 |

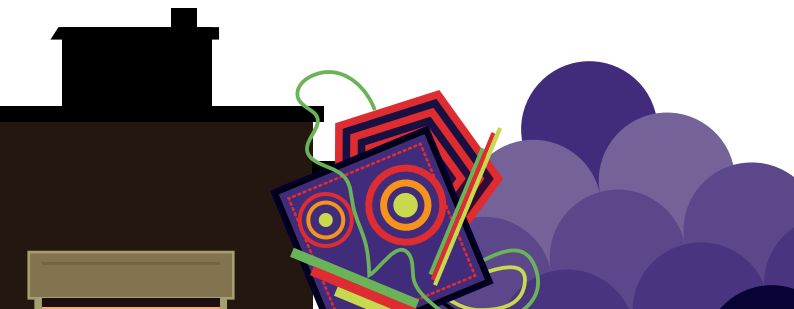


Since you are already using multiple platforms - according to a recent research, the average number of platforms used by each developer is 2.7 - you may then need to adjust your strategy according to your retail channel or the specific platforms you are targeting.

According to the Developer Economics research, it is actually BlackBerry and iOS that top the per-app month revenue chart. Android is third behind them, with Windows Phone bringing up the rear. The table below presents the average per app-month revenue for each platform.

| Platform | Revenue per app-month |
|---------------|-----------------------|
| Android | \$2,735 |
| BlackBerry | \$3,853 |
| iOS | \$3,693 |
| Windows Phone | \$1,234 |

Note: These two tables calculate revenues across the lower 95% of developer by per-app revenue. Which are the most common promotional channels used by developers to promote their apps? According to the research, 47% of developers use Facebook to promote their apps, while 28% use mobile or desktop keyword search and 26% use free demos or freemiums.



The Developer's Perspective: The Final Choice

At the end of the day, the final choice is yours. We hope we have given you some good info to mull over and answered some of your questions. There is no single, all-encompassing answer that will make you the next Rovio - it has all got to do with planning, hard work and planning. Word to the wise, choose your environment not based on what you already know, but on what you are trying to achieve and plan your strategy accordingly.





Epilogue

Thanks for reading this 11th edition of our Mobile Developer's Guide. We hope you have enjoyed reading it and that we helped you to clarify your options.

If you like to contribute to this guide, sponsor upcoming editions or get the book as a hardcopy, please send your feedback to **developers@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 hashtag **#mdgg**

About the Authors

Andrej Balaz / Enough Software

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 dive into digital art and illustration.

www.enough.de www.behance.net/andrejbalaz

Benno Bartels / InsertEFFECT

Benno started developing mobile software as a student. After graduating high school, he and two friends founded InsertEFFECT in Nuremberg, Germany. Today, the company comprises 15 people designing and developing mobile apps. At InsertEFFECT, Benno consults for companies that include Deutsche Bahn, Immowelt and O2. He also loves to share his experience in workshops and at BarCamp and Web Monday events.

www.inserteffect.com

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.

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.

Oliver Graf / Enough Software

Oliver has been coding software for several platforms since 2000. He works as a multi-platform developer for Enough Software and writes about mobile development for several magazines. Oliver was among the first registered developers for bada. As one of the Samsung developer advocates, he connects developers with Samsung (and vice-versa) to improve the bada ecosystem.

www.enough.de **www.dm-graf.de**

Roland Gülle / Sevenval

In 2001, Roland joined Sevenval to experience the mobile industry. As CTO, he is responsible for Sevenval's product, technology and development. Roland specializes in web technologies and internet standards. He has extensive expertise in the areas of usability, system architecture, project implementation and solution-oriented approaches. He talks at conferences about web, mobile, adaptation and performance.

www.sevenval.com **www.fitml.com**

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.

Bob Heubel / Immersion Corp.

Bob is a haptic technology evangelist with Immersion Corporation who specializes in assisting developers implement what is known as force-feedback, tactile-feedback or rumble-feedback effects. He has spent more than ten years working with developers, carriers, OEMs and ODMs to design and implement these sensations aimed at improving both gaming and user interaction experiences. Bob graduated from UC Berkeley in 1989 with a BA in English Literature.

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.

www.enough.de www.oviduiulescu.com

Gary Johnson / Sharkfist Software, LLC

Gary is currently contracting for mobile development across the board Windows Phone, iOS and Android. His past experience includes deep expertise *in .NET*, Silverlight and WPF. He has a strong passion for all things mobile as well as creating great UI and UX.

Alex Jonsson / MoSync

Alex likes anything mobile, both apps and web technologies as well as cleverly connecting physical stuff to digital stuff. He holds a Doctorate in Computer Science and still gives lectures now and then. Alex has an eclectic urge to create new values by finding new combinations of things, transferring knowledge between disciplines and exploiting aspects of communication and media with the aim of bringing people together. Alex is CTO at MoSync Inc.

www.mosync.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, 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.

www.enough.de

Tim Messerschmidt

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. Since 2012 he is working at PayPal as Developer Evangelist for Europe. He is passionate about UI, UX and Android development in general. Furthermore he loves speaking at conferences, writing articles and all kind of social media.

www.timmesserschmidt.de

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.

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.

www.zahlenhelfer-consulting.de

Thibaut Rouffineau / WIP

Community and passion builder with a mobile edge, for the past 5 years Thibaut has been working to move the mobile developer community towards greater openness and exchange. Thibaut is VP for Developer Partnerships at WIP, the organizer of Droidcon London. He is an enthusiast speaker on mobile topics and has been heard around the world.

www.wipconnector.com

André Schmidt / Enough Software

André has been developing mobile applications since 2001. He joined Enough Software in 2007, where he heads the development of Open Source products for mobile developers and mobile applications of all kinds. He mainly develops for Java ME, Android and BlackBerry.

www.enough.de

Michel Shuqair / AppValley

Michel's experience with telecoms started in 1999 and he closely watched the mobile development space evolving from Japan. Starting with black and white WAP applications, iMode and SMS games, he 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 as well taking responsibility for finding sponsors and merging the input provided by the mobile community.

www.enough.de

Ian Thain / SAP


Ian is a Mobile Evangelist at SAP, though he started 12 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.

ianthain.ulitzer.com www.sap.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 and many mobile apps.

www.enough.de www.j2mepolish.org



Please spread the word about this project.
On Twitter use the hashtag *#mdgg*

THANK YOU

an initiative by:



printing sponsors:



powered by



**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!