Don't Panic

MOBILE DEVELOPER'S GUIDE TO THE GALAXY

published by:

ENOUGH SOFTWARE

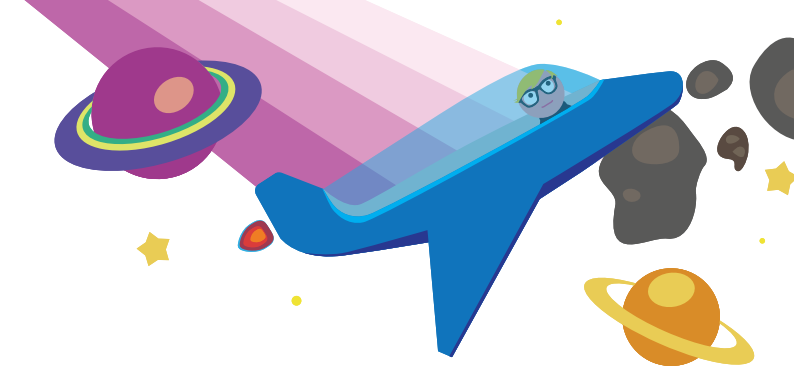
Services and Tools for All Mobile Platforms

Enough Software GmbH + Co. KG
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*

10th Edition February 2012

Design and Artwork by **Andrej Balaz** (Enough Software)
Character Artwork by **Johanna Kromp** (*www.organisiertekunst.de)*

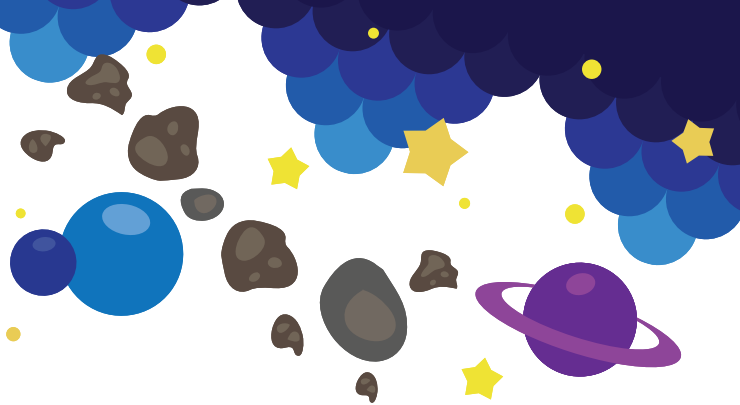


Introduction

Welcome to the 10th anniversary edition of our Mobile Developers Guide To The Galaxy! We introduced the first version of this overview of mobile technology exactly 3 years ago, at Mobile World Congress 2009. And we are flattered to see that it has grown from a thin brochure into a small book, and become one of the best-known publications about app development. All this would not have been possible without the community's support: Our thanks go out to all the writers who are sharing their know-how on the following pages, to the printing sponsors for their financial support and to all the individuals who are helping to distribute the book at events and among universities, schools and other organizations worldwide.

We are also glad to inform you that our friends from WIP have started to put out a companion guide to this book: Get their "Mobile Developer's Guide To The Parallel Universe" on *www.wipconnector.com* or on their WipJams and learn everything about app marketing.

As always the ever-changing kaleidoscope of the mobile industry does just that — it keeps changing. Since our last edition: Windows 8 was revealed, Adobe discontinued Flash for



mobile browsers, Tizen released a preview SDK, HP changed its mind and announced webOS would continue as an Open Source project, Sony separates from Ericsson, Nokia released its first Windows Phone products and acquired Smarterphone, BlackBerry seemingly jumps off the Java bandwagon with BlackBerry 10, Android expanded its reach to TVs and other appliances, Apple made the world speak to its phone and much more. As usual these changes are reflected in this guide. We have also added two completely new chapters: how to implement haptics into your app and how you can protect your code against security attacks right from the start.

We hope you enjoy the book and we are looking forward to seeing the project continue to grow. Please get involved and let us know what content might be missing, at *developers@enough.de*.

Robert + Marco / Enough Software
Bremen, February 2012

An Overview Of Application Platforms

There is a wide selection of platforms with which you can realize your mobile vision. This section describes the most common environments and outlines their differences. More detailed descriptions follow in the platform-specific chapters.

Native Applications

There are many mobile platforms used in the market – some are open source, some are not. The most important native platforms are (alphabetically) Android, bada, BlackBerry, BlackBerry PlayBook OS (QNX), iOS, Symbian, Windows 8 and Windows Phone.

All these platforms enable you to create native applications without establishing a business relationship with the respective vendor.

The main benefits of programming apps natively include better integration with the platform's features and often better performance. Typical drawbacks are the effort and complexity of supporting several native platforms (or limiting your app to one platform).

Most mass market featurephones are, however, equipped with embedded operating systems that do not offer the opportunity to create native applications. Examples include but are not limited to Nokia Series 40, Samsung SGH and LG featurephones.

The following table provides an overview of the main mobile platforms:

Platform	Language(s)	Remarks
Android	Java, C, C++	Open Source OS (based on Linux) developer.android.com
bada	C, C++	Samsung's mobile platform running on Linux or RealTime OS developer.bada.com
BlackBerry	Java, Web Apps	Java ME compatible, extensions enable tighter integration blackberry.com/developers
BlackBerry Tablet/PlayBook OS (QNX)	ActionScript, C++, HTML, CSS, JavaScript	The forthcoming BlackBerry 10 OS will be based on QNX as well blackberry.com/developers
iOS	Objective-C, C	The mobile OS that introduced mobile apps to ordinary people. developer.apple.com/iphone
Symbian	C, C++, Java, Qt, Web Apps, others	Currently the longest running of all smartphone OSs www.forum.nokia.com/symbian
webOS	HTML, CSS, JavaScript, C	Supports native HTML5 programming, (based on Linux), future should be Open Source developer.palm.com
Windows 8	C#/VB.NET, C++, JavaScript	Microsoft's first real tablet OS msdn.microsoft.com/en-us/windows/apps
Windows Phone	C#, VB.NET	Silverlight, XNA frameworks create.msdn.com

Java ME (J2ME)

Around 70% of all featurephones worldwide support the mobile Java standard (Java ME formerly known as J2ME), making it by far the most widely distributed application environment. In contrast to many other environments, Java ME is a standard rather than a product, which can be implemented by anyone (who pays Oracle the corresponding license fees that is). Standardization is the strength of Java ME but at the same time it's the source of many fragmentation problems.

On many feature phones, Java ME is the only way to realize client side applications. With the increasing penetration of smartphones, Java ME has lost importance, at least in the US and Europe. However, as TechCrunch puts it: With 27% smartphone market share, it´s still a feature phone world out there[1]. That is why, particularly in emerging economies, Java ME remains the main option to target the mass market. It has been also hailed as the fastest growing platform for mobile devices, driven by the demand for feature rich yet inexpensive phones in emerging economies[2].

Flash

Historically, Flash Lite was the mobile edition of Flash, but based on an older version of Adobe's web Flash product with ActionScript 2.0 support. After phasing out Flash Lite for mobile and moving to the full version of Flash, Adobe has now announced it will discontinue Flash for mobile browsers altogether. Instead Flash based Adobe AIR apps along with HTML5 apps are the Adobe's focus for this market.

1 techcrunch.com/2011/11/28/its-still-a-feature-phone-world-global-smartphone-penetration-at-27/

2 www.theregister.co.uk/2012/01/03/apple_losing_the_race_for_emerging_economies_cash

Flash is favored by many designers, since they know the tools already and it can be used to create engaging, powerful user interfaces (UIs). It's relatively easy to code thanks to the ActionScript language, which is very similar to JavaScript.

The drawbacks of Flash on mobile devices used to be poor performance, suboptimal integration into host devices and small market share in comparison to Java ME. Nevertheless there are millions of feature phones supporting Flash Lite today and many smartphones and tablets can support some Flash content including Symbian, iOS (through Adobe AIR), Android and BlackBerry devices.

BREW

The Binary Runtime Environment for Wireless (BREW) is a feature phone programming environment promoted by Qualcomm[3].

BREW services are offered by more than 60 operators in 28 countries, but it's most popular within the US with CDMA devices launched by Verizon, US Cellular and Metro PCS, among others. While previous versions supported C development only, the Brew Mobile Platform (Brew MP), supports applications written in Java, Flash, TrigML or native C code[4].

Websites And Web Apps

The browsing of web pages is supported by most phones, so in principle this should be the environment of choice to get the widest possible reach (after SMS text messaging). However, the sheer number of browsers and their varying feature sets can

3 www.brewmp.com
4 developer.brewmp.com

make this approach challenging. Some browsers are very powerful and support CSS as well as JavaScript; others are less sophisticated and support XHTML only. Thankfully the old WAP standard with its WML pages doesn't play any significant role nowadays.

The main drawback of web pages is that they are available when the device is online only and their access to device features is extremely limited. Even with these challenges, the mobile web as a platform offers benefits in ease of development and control over deployment.

With the introduction of HTML5 and new mobile browsers that support its features, this situation is improving: Offline browsing and new device APIs, such as for location information, are becoming available to mobile websites.

Web Widgets

As HTML5 support matures, it will probably become the most popular way to create web apps for mobile phones. In the meanwhile several platforms offer web widgets (essentially a small website packaged for installation on a device). Web widgets generally offer the advantage of APIs to access platform features and can be distributed through app stores.

Proxy based web widgets for, feature phone platforms, are likely to remain popular even after HTML5 web apps largely replace web widgets on smartphone.

Some notable web widget environment are listed on the following page.

Environment	Language(s)	Remarks
Symbian Web Runtime (WRT) Widgets	XML, HTML, CSS, JavaScript	Standard web technology based widgets, with a proprietary packaging standard. JavaScript APIs offer high degree of access to platform features. www.forum.nokia.com/Develop/Web
WAC	XML, HTML, JavaScript, CSS	A joint initiative by Vodafone, China Mobile and other companies are pushing the W3C widget standard wacapps.net/developers
Samsung	XML, HTML, CSS, JavaScript	innovator.samsungmobile.com
Series 40 web apps	XML, HTML, CSS, JavaScript	Web apps for the proxy based Series 40 Browser enabling UI manipulation on a device through JavaScript. W3C packaging standard used. www.forum.nokia.com/webapps
PhoneGap	HTML, CSS, JavaScript	Cross platform web app platform www.phonegap.com
Sony Ericsson WebSDK	HTML, CSS, JavaScript	Based on PhoneGap developer.sonyericsson.com
BlackBerry Webworks	HTML, CSS, JavaScript	blackberry.com/developers

SMS Text Messaging

Almost everybody who has a mobile phone is also texting. Texting limits interactions to less than 160 characters; and it can be quite costly to send out text messages in bulk. On the positive side, SMS enjoys a global audience of all ages. It also plays an important role in emerging markets where, for example, its use for payments is common.

Note that some tablets such as the iPad or the Samsung Galaxy Tab use WiFi instead of a mobile phone networks and these device cannot send SMS messages directly. However, some operators offer APIs for messaging services that work for WiFi-only devices, such as the network APIs of Deutsche Telekom. [5]

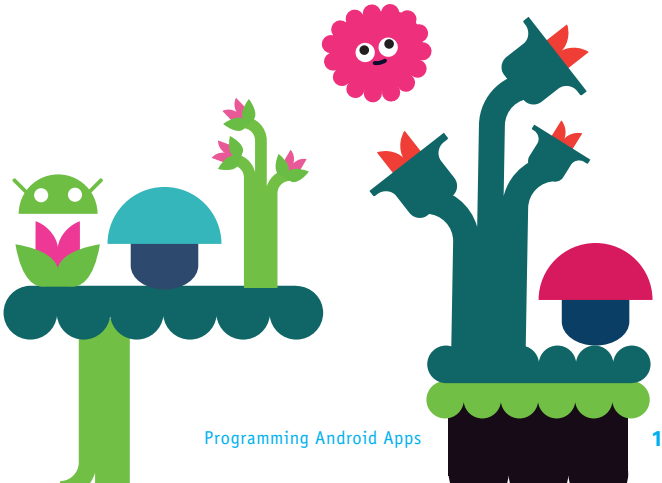
5 www.developergarden.com/apis/

Programming Android Apps

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, 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. Although the acquisition of Motorola's handset business by Google in late 2011 could affect the platform's use by other manufacturers, it is not expected to. According to Gartner, more than 50% of all smartphones sold in Q3 2011 worldwide were based on Android[1], more than 400,000 apps are available in the Android Market[2]. Android is also used in tablets, media players,

1 www.gartner.com/resId=1847315
2 www.distimo.com/blog/2012_01_google-android-market-tops-400000-applications/



set-top boxes, desktop phones and car entertainment systems. Some non-Android devices are also able to run Android applications, such as RIM's PlayBook with its virtual machine called App player[3].

The platform continues to evolve rapidly, with the regular addition of new features every 6 months or so. The current Android OS version 4.0 ("Ice Cream Sandwich") was designed to serve as platform for both phones and tablets. It brings interesting new features such as a Face Detection API, WiFi-Direct, Hardware-accelerated 2D-drawing and an easier to use NFC-API called Beam[4]. The first phone based on Ice Cream Sandwich was the Samsung Galaxy Nexus introduced in October 2011, first tablets with Android 4.0 followed in December.

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 98.3% of the installation base if you decide to develop against Android 2.1 or above[5]. 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 dimensions[6].

With Android 4.0 there is now also a design guide for Android apps available at *developer.android.com/design*.

3 www.theregister.co.uk/2011/03/25/rim_playbook_android/
4 developer.android.com/sdk/android-4.0-highlights.html
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 Guide[7] and to your "How" questions in the reference documentation[8].

To get started, you need the Android SDK[9], 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 Lint[10], which helps in optimizing applications by pointing out common bugs. The optimizations are thankfully easy to implement.

Implementation

An Android application is a mix of activities, services, message receivers and data providers declared in the application manifest.

An activity is a piece of functionality with an attached user interface. A service is used for tasks that should run in the back-

7 developer.android.com/guide
8 developer.android.com/reference
9 developer.android.com/sdk
10 tools.android.com/recent/lint

ground and is 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. 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 others – such as tools to analyze method trace logs, inspect layouts and test apps with random events or backup functionality – 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. IDE plug-ins are available to help manage all these files.

If you are facing issues, such as exceptions being thrown, be sure to check the ddms log. It enables you to check if you have omitted to add necessary permissions, such as `android.permission.INTERNET.`, using the uses-permission flags[11].

If you are going to use Honeycomb related layout features – such as Fragments[12] – for large screens, be sure to add the Android Compatibility package from Google. It's available through the SDK & AVD Manager and helps to develop for Android 3.0+ without causing problems with deployment to Android 1.6[13] through to Android 2.3. Be sure to use the v4 packages in your application to provide maximum backwards support.

11 developer.android.com/reference/android/Manifest.permission.html
12 developer.android.com/guide/topics/fundamentals/fragments.html
13 android-developers.blogspot.com/2011/03/fragments-for-all.html

If you are implementing 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 device. You can the 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 platform[14]. Therefore, testing on a physical device is essential.

Automated Testing

To automate testing, the Android SDK comes with some capable and useful testing instrumentation[15] 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. You can test then for the status of your application after these events have occurred. The automated tests can be run on virtual and physical devices. Open-source testing frameworks, such as Robotium[16] can complement your other automated tests. Robotium can even be used to test binary apk files, if the source is not available. A maven plugin[17] and a helper for the continuous integration of a Hudson server may also assist your testing[18].

14 *For an overview see e.g.* www.androidfragmentation.com
15 developer.android.com/guide/topics/testing/testing_android.html
16 code.google.com/p/robotium
17 code.google.com/p/maven-android-plugin/
18 wiki.hudson-ci.org/display/HUDSON/Android+Emulator+Plugin

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.

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 groups[19].

19 www.wipconnector.com/index.php/appstores/tag/android

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 Market[20].

There are other interesting alternatives to the Android Market which can help you target a certain audience. 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.

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

Programming bada Apps

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 puts a high priority on developer support and training. There is a huge documentation and guide library available from the developer site *developer.bada.com*. This site offers a forum, premium support and direct access to Samsung bada experts as well.

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 website[1], 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. It is possible to generate revenue by placing adverts in your apps too.

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.

bada 2.0 – released in September 2011 – introduced multi-

1 www.samsungapps.com

tasking, 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 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 0 new MyType[10]; // 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)

{
// code to extract the message and set up the

  data to send back goes here
  …

// If this is the event we were expecting, send
// a response
```

```
  if(pReceivedMethod->
    Equals(L"SendDataEvent",true)) {
    pFlash->SendDataEventToActionScript(
    L"TestExtension",dataList);
  }
}
```

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".

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.

These will be explained in more detail in the bada framework documentation under "Basics of bada Web Programming"
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.

Documentation & bada 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. 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.

Once your app has made it to the store, you will get 70% of the revenue. For advertising Samsung own advertising service – AdHub – but also allows the inclusion of third party ad network contents.

Programming BlackBerry 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 Thumb[2]), 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 market share of BlackBerry phones has declined somewhat in the US in 2011[3], but it is still an important smartphone platform. While the general consensus seems to be that BlackBerry tablet – the PlayBook with its new QNX Neutrino-based OS – was been launched too early, the hardware and OS are highly praised.

In this edition we concentrate on Java development for the BlackBerry smartphones.

1 www.rim.com
2 en.wikipedia.org/wiki/Blackberry_thumb
3 gs.statcounter.com

Prerequisites

RIM currently supports two platforms/ operating systems: BlackBerry OS and BlackBerry PlayBook OS. The future platform for BlackBerry devices is called BlackBerry 10 and has not been launched yet.

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.

RIM's next generation operating system is based on the QNX Neutrino Realtime OS (RTOS). Currently RIM calls it PlayBook OS, as it is only supported on their PlayBook. Applications for PlayBook can be written using SDKs for Adobe AIR, WebWorks and Native C/C++. An Android compatibility layer, currently in beta, is also available. There are currently no plans to support the existing BlackBerry Java API and SDK on PlayBook OS, which is a major point of concern for many existing BlackBerry Java developers. However, the Android compatibility layer somewhat compensates for this.

For phones, future OS is called BlackBerry 10 and will be based on PlayBook OS, replacing the current BlackBerry OS eventually. The first regular BlackBerry smartphones running a version of this new operating system are scheduled to arrive in late 2012.

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, for more information on WebWorks (web) and Flash programming please see the respective chapters in this guide.

Java SDK

As for all Java-driven applications and development, you need the Java SDK[4] (not the Java Runtime Edition).

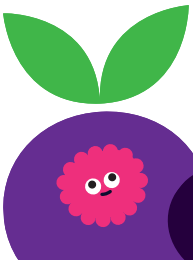
IDE

For native Java development, you first need to decide which IDE to use. The modern option is to use Eclipse and the BlackBerry plugin[5], 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 Manager[7] should be downloaded and installed. It enables you to deploy your app package on a device for testing. For faster deployment, you might also use a tool called javaloader that comes with the JDE.



4 www.oracle.com/technetwork/java
5 us.blackberry.com/developers/javaappdev/javaplugin.jsp
6 us.blackberry.com/developers/javaappdev/javadevenv.jsp
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 others[8]. 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, however overriding the `Field.applyTheme()` method will prevent this.

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 Polish[9] or Glaze[10] which enable you to design and animate your UI using CSS.

Services

BlackBerry offers many services that can be useful in developing your applications including advertising, mapping, payment and push services[11].

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

8 www.blackberry.com/developers/docs/6.0.0api/index.html
9 www.j2mepolish.org
10 www.glaze-ui.org
11 us.blackberry.com/developers/platform

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 service[12] 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.

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.

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

Porting

Porting 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 functionalities 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 don't use this technology anymore.

Porting to other Java platforms such as J2ME and Android is complicated as it's not possible to port the BlackBerry UI. Code written for server communication and storage etc might be reused on J2ME 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 BlackBerry[13]. The signing itself is undertaken using the rapc tool which also packages the application.

Distribution

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

13 www.blackberry.com/SignedKeys/
14 appworld.blackberry.com
15 www.getjar.com

Programming Flash Apps

Adobe Flash is a multimedia platform used to add animation, video, and interactivity to web pages and mobile apps. It combines vector and raster graphics with ActionScript for interactivity.

Flash development experienced a setback in November 2011, when Adobe announced that Flash would no longer develop a version for mobile web browsers. Instead Flash developers should concentrate on Adobe AIR[1] development that can be used to develop native apps based on Flash. At the same time Adobe jumped on the HTML5 bandwagon by demonstrating its HTML5 authoring suite edge and by purchasing PhoneGap. But all is not lost, an active developer community and third-party ecosystem around Flash ensures its continued existence. Also, RIM announced[2] that they will continue their support of the Flash browser plugin for PlayBook itself. Many feature phones have support for the older and discontinued Flash Lite[3] (typically support for Flash Lite 3, 6 or 8 depending on when the device was manufactured).

Development of mobile Flash applications can be undertaken using Adobe products and alternative Flash-compatible SDKs from a number of vendors.

1 www.adobe.com/products/air
2 www.theverge.com/2011/11/9/2550815/rim-to-continue-developing-flash-browser-plug-in-for-playbook
3 en.wikipedia.org/wiki/Adobe_Flash_Lite

Prerequisites

Adobe open sourced the Flash specification, permitting independent developers and companies to develop Flash-compatible SDKs, engines and players. Authoring can be undertaken using the Adobe Flash Professional or Adobe Creative Suite (CS) software.

CS 5 supports ActionScript 3 and Flash 10.X offering the full 3D and 2D feature set on some smartphones and tablets. If you want to use features such as 3D and ActionScript 3 compatibility, using the CS5 package and tools is the way to go.

However, one potential drawback of Flash is poor performance: Large binary files may run slowly on less powerful devices, resulting in a poor user experience. Adobe CS 3, 4 and 5 can be used to author Flash content that runs on alternative Flash-compatible SDKs, engines and players, giving developers more options to optimize an application's performance.

These alternative Flash-compatible SDKs generally support ActionScript 2 and Flash 8 with a full 2D feature set. Note that video and audio playback support was a feature introduced in Flash 6.1, so nearly all current Flash-compatible SDKs have the ability to support video playback.

A Flash application typically consists of two distinct pieces: The animation engine that renders deterministic graphics for the display and the ActionScript engine that uses input events (time, keyboard, mouse and XML) to make runtime decisions about whether animations should be played Flash-internally or externally. ActionScript is a scripting language based on ECMAScript available in three versions.

Developing Flash applications, animations or video for mobile devices does not differ significantly from developing browser based Flash applications for desktop computers. However, you must be aware of the requirements and restrictions of the target device.

Be sure to save your Flash files in a format that is compatible with your target device's software.

Pay special attention to the design of your Flash application.

Adobe CS provides the option to choose between developing a browser-dependent or a standalone Adobe AIR[4] application. An AIR application is essentially a Flash player, browser engine, and the native device's APIs wrapped into one executable file, so that it conforms to the developer terms and security requirements of various mobile platforms. Alternative Flash-compatible SDKs go further and integrate Flash content into existing 2D and 3D software applications.

There are also open source versions including Gnash[5] and GameSWF[6] that are designed for desktop systems. Many of the alternative Flash-compatible platforms run outside the browser environment, working directly with a device's native APIs.

Tips And Tricks

As it is mentioned often in this guide, it is crucial to consider battery life when creating applications for mobile devices, Flash is no exception. You should never create memory-intensive animations purely for the sake of offering a fancy effect. A Flash animation using ActionScript 3 will create a binary that could be more than 3 times larger than that for an ActionScript 2 animation and will likely result in poor performance.

In general, you should think carefully about whether you need ActionScript 3: It's a completely different scripting language to

4 www.adobe.com/products/air.html
5 www.gnashdev.org
6 tulrich.com/geekstuff/gameswf.html

ActionScript 2 and requires a lot more development know-how and experience to implement efficiently.

You might also want to remember the following to avoid your Flash app causing excessive battery drain:

- Avoid sliding a Flash object across the screen, unless you know it performs well. Redrawing every pixel multiple times in a frame without the support of a GPU is a performance killer. Select a SDK toolkit that minimizes CPU utilization to preserve battery life. If the Flash animation is not changing, the SDK toolkit should show 0% CPU utilization.
- If the target smartphone has one display resolution, use correctly sized 2D bitmaps to replace SVG objects that will never change size.
- Minimize network connectivity to that which is required only.
- Use OS APIs with care. The greater number of OS APIs and independent software you use, the more work is being done and the faster the battery runs out of power.
- Design the application to recover gracefully from power failures. Many of the alternative Flash-compatible SDKs have additional APIs to support power failures and include database tools that you can implement in an application (for example to save and restore settings). These tools mean your user doesn't need to re-key data after a power failure.

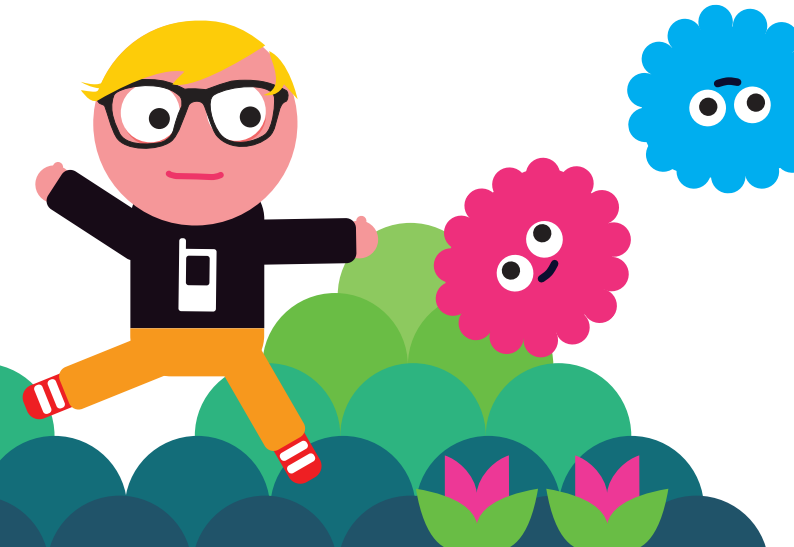
Testing

The best approach for initial testing is determined by your chosen architecture: If you have developed an Adobe Flash browser-based application or Adobe AIR application, then it's best to test the application using the Adobe tools.

However, if you have developed a Flash animation (with or without ActionScript) or Flash animations that will be integrated into another 2D or 3D application, you should consider testing the application with one of the alternative Flash-compatible SDKs or tools.

Adobe CS5 has a built-in smartphone emulator also. This enables developers to virtually test their application on selected handsets and tablets.

In general: Always test on devices to gain information on how much memory and battery is being used by the application.



Packaging And Distribution

When you design Flash content for use in a mobile website, packaging and distribution is straightforward: You simply follow the same rules and procedures you would use in deployment for use in a desktop browser.

Using Flash in a web widget is similar also. Generally you include the Flash content in the widget as you would for a website, package the widget and deploy the resulting application in line with the widget environment's requirements – for more information see the chapter "Programming Mobile Widgets".

When the platform offers a built in Flash player that runs Flash content as an application the packaging requirements can be more complicated. At the simple end of the spectrum is Nokia Series 40, where the packaging requirements are quite simple[7]:

You create some metadata, an icon and pack these with the Flash content into a zip file with the extension *.nfl.

At the complex end of the spectrum is packaging for Symbian devices, where the Flash app has to be given a Symbian C++ launcher and packed in a Symbian SIS file.

While some developers will do this manually, Nokia provides an online packaging service that does the heavy lifting for you[8].

Generally, when the packaging seems complex, it can often be simplified by using the platform's widget option to package and deploy the content.

In general, once Flash content has been packaged into the correct format for the platform, it can then be distributed through any app store that services that platform.

7 bit.ly/aqEmvv
8 esitv008song.itlase.com/sispack

Programming iOS Apps

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 500,000 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 December 2011, Apple announced that users have downloaded more than 18 billion iOS apps. Nearly every quarter device sales are reaching new all-time highs and there is no sign of a slowdown in the billion downloads per month. Over 200 million devices are in the hands of users willing to try apps and pay for content, making the App Store one of the most economically interesting targets for mobile app development.

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. 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.

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/iphone*. This requires a membership, which starts at USD 99/year. If you do not plan on distributing your apps in the App Store and don't 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

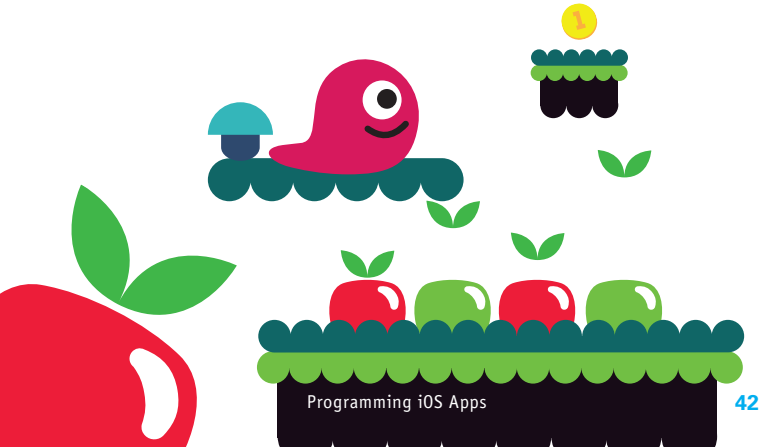
The iOS SDK will work on any Intel-based Mac running Mac OS X 10.6 (Snow Leopard) or 10.7 (Lion, which is the recommended OS X version).

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/iphone/library/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, which usually happens once a year in June or July, with iOS 5.0 being the exception to this rule with its release in October 2011.



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 Kit[1], 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 MonoTouch[2].

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

1 www.udk.com
2 www.monotouch.net

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 AppCode, which is still in beta stage but looks as if it provides some advanced features.

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.

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*[3]. 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 workflow[4]. Also check out some of the sample code that Apple provides online[5] to find out more about various APIs available to you.

3 developer.apple.com/iphone/manage/overview/index.action
4 developer.apple.com/iphone/manage/distribution/distribution.action
5 developer.apple.com/iphone/library/navigation/SampleCode.htm

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.

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 Portal[6]. 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 instructions[7]. 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 projects[8] to facilitate this new feature, as well as commercial services[9].

6 developer.apple.com/iphone/library/referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/index.html#//apple_ref/doc/uid/TP40007594

7 developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhone101/Articles/00_Introduction.html

8 github.com/therealkerni/hockeykit

9 www.testflightapp.com

Google Toolbox for Mac[10] runs test cases using a shell script during the build phase, while GHUnit[11] 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 OCUnit; an example of how to create the unit tests is available online[12].

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 and a full reference is available in the iOS Reference Library[13]. Several other third party testing automation tools for iPhone applications are available, including FoneMonkey[14] and Squish[15].

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 isn't 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 shouldn't 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

10 code.google.com/p/google-toolbox-for-mac
11 github.com/gabriel/gh-unit
12 www.mobileorchard.com/ocunit-integrated-unit-testing-in-xcode
13 developer.apple.com/library/ios/#documentation/DeveloperTools/Reference/UIAutomationRef/_index.html
14 www.gorillalogic.com/fonemonkey
15 www.froglogic.com/products

other aspect of the iPhone ecosystem. A list of common rejection reasons can be found on *www.apprejections.com*. 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* 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 some 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 besides 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'll 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 list[16]. 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*.

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

Programming J2ME / Java ME Apps

J2ME (or Java ME as it is officially called) is the world's most widespread mobile application platform and the oldest one 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 BlackBerry smartphones.

J2ME's major drawback is that, due to its age and primary market segment, it doesn't fare all that well compared to more modern platforms, such as Android, iPhone, BlackBerry and Symbian: 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, in favor of development on smartphone platforms.

So why would you want to develop for J2ME? Mainly for one reason: market reach.

With over 70% of phones worldwide supporting it, J2ME is miles ahead of the competition in this regard. In late 2011, J2ME even overtook Android when it comes to mobile browsing: While most of mobile browsing was done on iOS devices (51%), J2ME was on position two with 21%. Only 16% of the mobile web users had an Android device[1]. 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.

1 www.netmarketshare.com

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 Symbian).

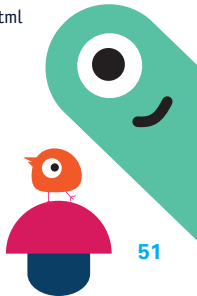
That being said, it should be noted that Java ME's capabilities are constantly improving thanks to the Java Community Process that standardizes new APIs: for example, modern features such as GPS, sensors, 3D graphics and touchscreens are all supported by the platform today. In addition, J2ME-compatible hardware is becoming more powerful and less expensive all the time, and with more and more devices implementing the advanced features mentioned. Overall the platform is evolving and changing for the better, though admittedly at a considerably slower pace compared to the competition.

Prerequisites

To develop a Java ME application, you will need:

- The Java SDK[2] (not the Java Runtime Environment) and an IDE of your choice, such as Eclipse Pulsar for Mobile Developers[3], NetBeans[4] with its Java ME plug-in or IntelliJ[5].
- An emulator, such as the Wireless Toolkit[6], the Micro Emulator[7] or a vendor specific SDK or emulator.

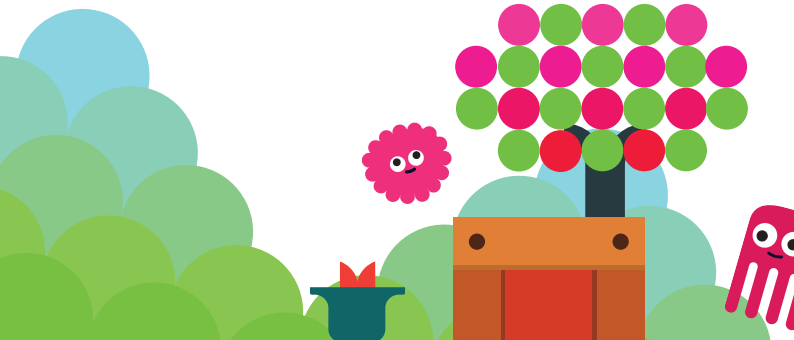
2 www.oracle.com/technetwork/java/javame/downloads/index.html
3 www.eclipse.org
4 www.netbeans.org
5 www.jetbrains.com
6 www.oracle.com/technetwork/java/download-135801.html
7 www.microemu.org



- Depending on your setup you may need an obfuscator like ProGuard[8]. If you build applications professionally you will probably want to use a build tool such as Maven[9] or Ant[10] also.
- You may want to check out J2ME Polish, the open source framework for building your application for various devices[11].

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

8 www.proguard.sourceforge.net
9 maven.apache.org
10 ant.apache.org
11 www.j2mepolish.org



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.

Manufactures 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 minimized; it cannot run in the background. Some devices support concurrent

12 java.sun.com/products/cldc/overview.html
13 java.sun.com/products/midp/overview.html
14 www.jcp.org/en/jsr/tech?listBy=1&listByType=platform

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 287[15].

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 and more powerful UIs than are possible with the standard high-level LCDUI components.

15 www.jcp.org/en/jsr/detail?id=287

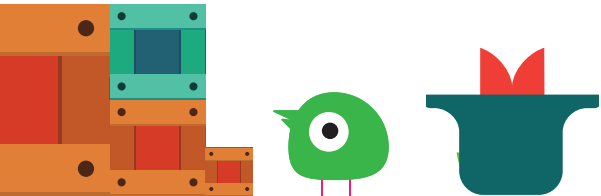
1. J2ME Polish[16]: This tool separates the design in CSS and you can use HTML for the user interface. It is backward-compatible with the highlevel LCDUI framework
2. LWUIT[17]: A Swing inspired UI framework
3. Mewt[18]: Uses XML to define the UI

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 176x208/220 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 applica-

16 www.j2mepolish.org
17 lwuit.dev.java.net
18 www.mewt.sourceforge.net



tion's size as small as possible. This results in cheaper downloads for your users (as less data traffic is needed) and greater market reach (as some devices have a limit on the maximum application size). A great free tool for this is PNGGauntlet[19], 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.net*[20], for example the Bluetooth project Marge[21].

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 Nokia[22] and Samsung[23] provide subsidized or even free remote access to selected devices.

19 www.pnggauntlet.com
20 mobileandembedded.dev.java.net
21 marge.dev.java.net
22 www.forum.nokia.com/rda
23 innovator.samsungmobile.com

Automated Testing

There are various unit testing frameworks available for Java ME, including J2MEUnit[24], MoMEUnit[25] and CLDC Unit[26]; System and UI testing is more complex given the security model of J2ME, however JInjector[27] 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.

Direct Support

The best but hardest solution is to code directly for different devices and platforms. So you create a J2ME app for MIDP devices, a native BlackBerry app, a native Windows Mobile app, a Symbian app, an iPhone app, a Web OS app, and so on. As you can imagine, this approach has the potential to bring the very best user experience, since you can adapt your application to each platform's UI. At the same time your development costs will skyrocket. We advise the use of another strategy first, until your application idea has proved itself to be successful.

24 www.j2meunit.sourceforge.net
25 www.momeunit.sourceforge.net
26 snapshot.pyx4me.com/pyx4me-cldcunit
27 www.code.google.com/p/jinjector

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 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 Distillery[28] that is licensed per month, Bedrock from Metismo[29] that provides a suite of cross compilers on a yearly license fee and J2ME Polish from Enough Software[30] 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 SDK[31].

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

28 www.mobile-distillery.com
29 www.metismo.com
30 www.enough.de
31 www.mosync.com

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

Applications signed by JavaVerified[32], Verisign[33] or Thawte[34] 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. Intertek[35] 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-

32 www.javaverified.com
33 www.verisign.com
34 www.thawte.com
35 www.intertek.com/wireless-mobile

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:

- Handmark[36] and Mobile Rated[37] provide carrier and vendor independent application stores.
- GetJar[38] is one of the oldest distributors for free mobile applications – not only Java applications.
- LG distributes apps on *www.lgapplication.com*
- Nokia Store[39] targets Nokia users worldwide and provides a revenue share to the developer at 70% from credit card billing and 60% from operator billing
- Carriers are in the game also, such as Orange[40] and O2[41].

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 Catalogue[42]. Furthermore there are various vendors who provide solutions for provisioning of Java applications over a Bluetooth connection, including Waymedia[43] and Futurlink[44].

36 store.handmark.com
37 www.mobilerated.com
38 www.getjar.com
39 www.publish.ovi.com
40 http://bit.ly/A3Dde0
41 www.o2litmus.com
42 www.wipconnector.com/appstores/
43 www.waymedia.it
44 www.futurlink.com

Programming Qt Apps

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. Today, Qt can be used to create applications for phones based on Symbian and the Nokia N9 smartphone.

In fact, Qt can now be thought of as a platform in its own right – you can create a Qt application and deploy it to phones utilizing a number of different underlying operating systems. In the future, Qt will underpin Nokia's next billion strategy.

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++ development. 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

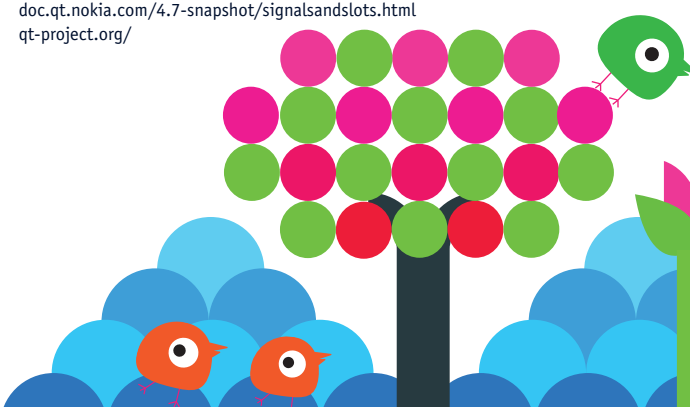
communication between objects[1]; the output from one object is a "signal" that has a receiving "slot" function in the same or another object.

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 QGraphicView 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 Project[2].

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



Prerequisites

Qt SDK[3] 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.
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 Quick[4], a comprehensive solution for declarative UI definition. Qt Quick uses QML, a language similar to JavaScript' 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

3 developer.nokia.com/Develop/Qt/Tools
4 qt.nokia.com/qtquick/

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 APIs[5].

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 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
(constQGeoPositionInfo&gpsPos) {
latitude = gpsPos.coordinate().latitude();
longitude = gpsPos.coordinate().longitude();
}
```

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 –

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

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.

QTestLib[6] provides both unit testing and extensions for testing GUIs. It replaced QtTestLib, however you may find useful tips by searching for this term. A useful overview is available at *qtway.blogspot.com/2009/10/interesting-testing.html*.

6 doc.qt.nokia.com/latest/qtestlib-manual.html

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 website[7], but in summary:

- You sign up as a Nokia Publisher[8]
- 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 to run it on the five phones elected and test it to ensure it complies with the Symbian Signed Test Criteria[9]
- 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.

7 www.developer.nokia.com/Distribute/Packaging_and_signing.xhtml
8 publish.nokia.com
9 www.developer.nokia.com/Community/Wiki/Symbian_Signed_Test_Criteria_V4_Wiki_version

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 your receive 70% of the sale price of revenue after any applicable taxes and costs.

One significant advantage of Nokia Store is that consumers in 46 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.

Programming Symbian Apps

Symbian[1] is a software platform for mobile phones. It consists of an operating system (formerly known as Symbian OS), middleware and user interface layers (formerly known as S60). Its development is stewarded by Nokia. Although Nokia has announced a transition to Microsoft Windows Phone for its high-end smartphones, Symbian still offers a viable development option with over 200 million compatible phones in use.

It is, however, worth noting that Nokia recommends creating apps for Symbian phones using Qt rather than the platforms native C++. It would be our recommendation too. Qt has the advantage that Nokia has committed to it over the long term, indicating that it will be focused on offer in apps for "the next billion". The precise nature of the opportunity it will offer remains to be disclosed however.

So, unless you have specialist development requirements – such as low level video manipulation – we would suggest you go to the "Programming Qt Apps" chapter and skip this section altogether.

Should you still want to create native Symbian apps or middleware, you will be using Symbian C++, the native programming language of the Symbian platform. Symbian C++ is a specialized subset of C++ with Symbian-specific idioms[2].

The native Symbian C++ APIs provide the most comprehensive access to phone features and enable rich application development.

The APIs provide fine-grained control over all aspects of the operating system, including memory, performance and battery life; and deliver a consistent performance advantage over other runtimes.

1 symbian.nokia.com
2 wiki.forum.nokia.com/index.php/Fundamentals_of_Symbian_C++

Prerequisites

The official desktop development platforms for Symbian C++ are Microsoft Windows XP with Service Pack 2, Windows Vista and Windows 7. All of the kits and tools supplied for Symbian development are free. If your computer meets the requirements, setting it up for Symbian C++ development is as simple:

1. Download the Symbian^3 SDK for Nokia phones[3] and install it
2. Install Carbide.c++[4]

Linux and Mac OS X are not officially supported platforms. One way around this is to use a virtual machine that hosts Windows. Other options are more complex, but information can be found online[5].

Carbide.c++

Carbide.c++ is designed for developers who wish to create applications that run on production phones – that is "on top" of the Symbian platform. Typical users include professional application and games developers, professional service companies, hobbyist developers, students and research groups.

Carbide.c++, however, requires the installation of one or more S60 or Symbian SDKs to enable development. Based on Eclipse, Carbide.c++ includes the GCCE compiler, a debugger that enables debugging on both the emulator and production phones, analysis tools, and more.

3 www.developer.nokia.com/Develop/Featured_Technologies/Symbian_C++/Tools/

4 *same as above*

5 www.developer.nokia.com/Community/Wiki/Develop_Symbian_C++_on_Linux_using_Gnupoc_and_Eclipse_CDT

Symbian/S60 Software Development Kits

The Symbian and the earlier S60 SDKs contain the libraries and header files that enable you to develop applications. Each SDK provides access to the APIs that are guaranteed to work on phones based on the corresponding version, that is the APIs in the Symbian Belle SDK will work on all Symbian Belle phones. Once you have installed the SDKs for the Symbian/S60 versions you wish to build for, you can use the built-in application wizard to create your first native application – you can then debug, run and download it to a Symbian phone – without having to write a single line of code.

Testing

For automated unit testing, googletest[6] works on Symbian, and other mobile C++ platforms. Each SDK includes an emulator that enables apps to be run and debugged on the development computer. And, as with all mobile technologies, testing on a phone is highly recommended.

Signing

Symbian uses a trust-based platform security model. This means some APIs are protected by platform security "capabilities". If you use APIs protected by capabilities, your application will need to be signed before it can be distributed. In addition, it is necessary to sign such applications during development in order to install it onto a phone: This is done using a "development certificate". For most applications, signing and the provision of "development certificates" is free-of-cost as part of the services offered by Nokia Store (see the "Programming Qt Apps" chapter for more information on Nokia Store). For a limited number of

6 www.code.google.com/p/googletest

applications, those using more advanced APIs, it will be necessary to obtain Certified Signed. More information on signing can be found in the user guide[7].

Distribution

Nokia Store will probably be your primary distribution channel (see the "Programming Qt Apps" chapter for more information), but you can distribute applications independently or through a number of operator and third-party application stores also.

7 www.developer.nokia.com/Community/Wiki/User_guide:_Symbian_Signed

Programming Windows Phone Apps

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 business users as well as consumers, and offers the simple-to-use Metro UI that focuses on typography and content. A marketing budget of 500 million USD has been spent on promoting the Windows Phone platform and 1.5 million handsets were sold in the first six weeks after launch[1]. However, according to Gartner, only 1.5% of all smartphones sold in Q3 2011 were based on Windows Phone[2].

In early 2011 Nokia announced that Windows Phone would become its smartphone platform. This partnership is expected to raise the market share of the platform in the near future. Estimates of the global sales numbers for Nokia's first Windows Phone handsets (the Nokia Lumia 710 and Lumia 800), released in November 2011, were suggesting these phones might not be fulfilling the high expectations. However, Nokia announced that in the UK the Lumia 800 was the most successful smartphone launch ever for the Finnish company[3].

1 www.microsoft.com/Presspass/Features/2010/dec10/12-21AchimBergQA.mspx
2 www.gartner.com/it/page.jsp?id=1848514
3 conversations.nokia.com/2011/11/16/nokia-hails-lumia-800-a-hit

UI Design

Windows Phone introduced a new UI paradigm called Metro[4] that now has been extended to the Xbox 360 and Windows 8 as well. So as Steve Ballmer stressed in his CES 2012 keynote, the future of Microsoft will be "Metro, Metro, Metro". 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
- **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.

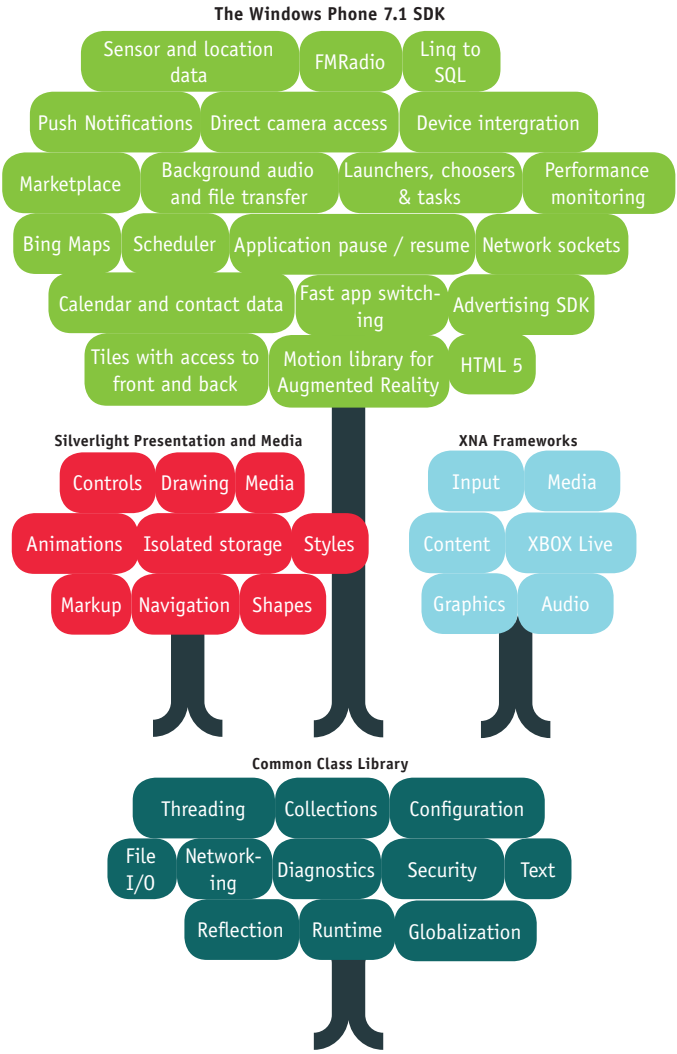
You should embrace the Metro design principles in your application, especially when porting over existing apps. Designers will find many inspirations and information in the Microsoft Design Toolbox[5]. You should also use one of the available grid solutions. A grid helps you to align your content with the Metro style. One solution that you can also use in the emulator is MetroGridHelper[6]. 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.

4 en.wikipedia.org/wiki/Metro_%28design_language%29
5 www.microsoft.com/design/toolbox/
6 www.jeff.wilcox.name/2011/10/metrogridhelper

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 PhoneGap[7], however web development is not covered in this chapter.

7 phonegap.com


The Windows Phone 7.1 SDK
Sensor and location data
FMRadio
Linq to SQL
Push Notifications
Direct camera access
Device intergration
Marketplace
Background audio and file transfer
Launchers, choosers & tasks
Performance monitoring
Bing Maps
Scheduler
Application pause / resume
Network sockets
Calendar and contact data
Fast app switch-ing
Advertising SDK
Tiles with access to front and back
Motion library for Augmented Reality
HTML 5
Silverlight Presentation and Media
Controls
Drawing
Media
Animations
Isolated storage
Styles
Markup
Navigation
Shapes
XNA Frameworks
Input
Media
Content
XBOX Live
Graphics
Audio
Common Class Library
Threading
Collections
Configuration
File I/O
Network-ing
Diagnostics
Security
Text
Reflection
Runtime
Globalization

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

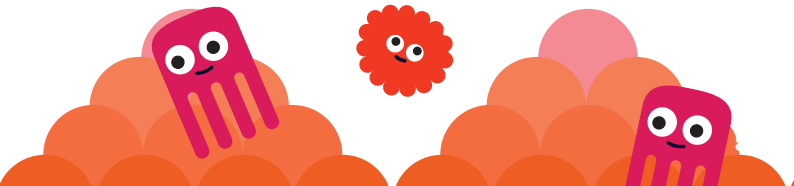
Windows Phone applications have access to input data such as location, multi-touch screen, accelerometer, gyroscope, and microphone.

Available services include FM radio, media playback, raw camera feed and push notifications[8] that can also update the live tiles of your app. You can also consider using the freely available SkyDrive cloud space for you app[9].

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 life-

8 msdn.microsoft.com/en-us/library/ff402558%28v=vs.92%29.aspx
9 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



cycle called Tombstoning[10]. To make Tombstoning possible, the Windows Phone framework provides the hooks needed to perform actions during different stages of the application lifecycle (such as caching and restoring data and UI states).

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.

10 msdn.microsoft.com/en-us/library/ff817008(v=vs.92).aspx

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 Hub[11]. 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

11 create.msdn.com

fee is waived for students in the DreamSpark program[12] 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 Kit[13] to test your application locally before you submit them.

There is also the ChevronWP7 Labs project[14] which, in partnership with Microsoft, enables a Windows Phone Device to be unlocked for development for 9 USD. While this option is less expensive than App Hub, it doesn't offer the free Marketplace submission. As of January 2012, ChevronWP7 has been "sold out" of unlock tokens with no confirmation if more will be made available[15].

Testing And Analytics

You can unit test applications using the Windows Phone Test Framework[16] or the Silverlight Unit Test Framework[17].

For behavior-driven development, the Windows Phone Test Framework by Expensify[18] 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. Localytics[19], PreEmptive Solutions[20] and Flurry[21] all provide analytics tools and services that are

12 www.dreamspark.com
13 msdn.microsoft.com/en-us/library/hh394032%28v=VS.92%29.aspx
14 labs.chevronwp7.com
15 www.windowsphonethoughts.com/news/show/129966
16 wptestlib.codeplex.com
17 www.jeff.wilcox.name/2011/06/updated-ut-mango-bits
18 github.com/Expensify/WindowsPhoneTestFramework
19 www.localytics.com/docs/windows-phone-7-integration
20 www.preemptive.com/windowsphone7.html
21 www.flurry.com

compatible with Windows Phone 7. Developers can also use the Silverlight Analytics Framework[22] 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.

Monetization

There are two primary methods of monetizing your Windows Phone apps, as paid for applications and as ad-servings 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 instead[23]. Notably there is no in app purchase mechanism currently (Windows Phone Mango), although it is expected to be supported soon.

For ad-based monetization, there are several options. Microsoft has their own Microsoft Advertising Ad Control[24] (currently available in 18 countries), while Smaato[25], inneractive[26], AdDuplex[27] and Google[28] all offer alternative advertising solutions. For more general information about monetization, please the dedicated chapter in this guide.

22 msaf.codeplex.com
23 msdn.microsoft.com/en-us/library/ff967558(v=vs.92).aspx#Best_Practices
24 advertising.microsoft.com/mobile-apps
25 www.smaato.com
26 inner-active.com
27 www.adduplex.com
28 code.google.com/mobile/ads

Resources

Visit *create.msdn.com* for news, developer tools and forums. The development team posts on their blog on *windowsteamblog.com/windows_phone* or their Twitter account *twitter.com/wp7dev*. For a large collection of developer and designer resources, visit *windowsphonegeek.com* and *reddit.com/r/wp7dev*.

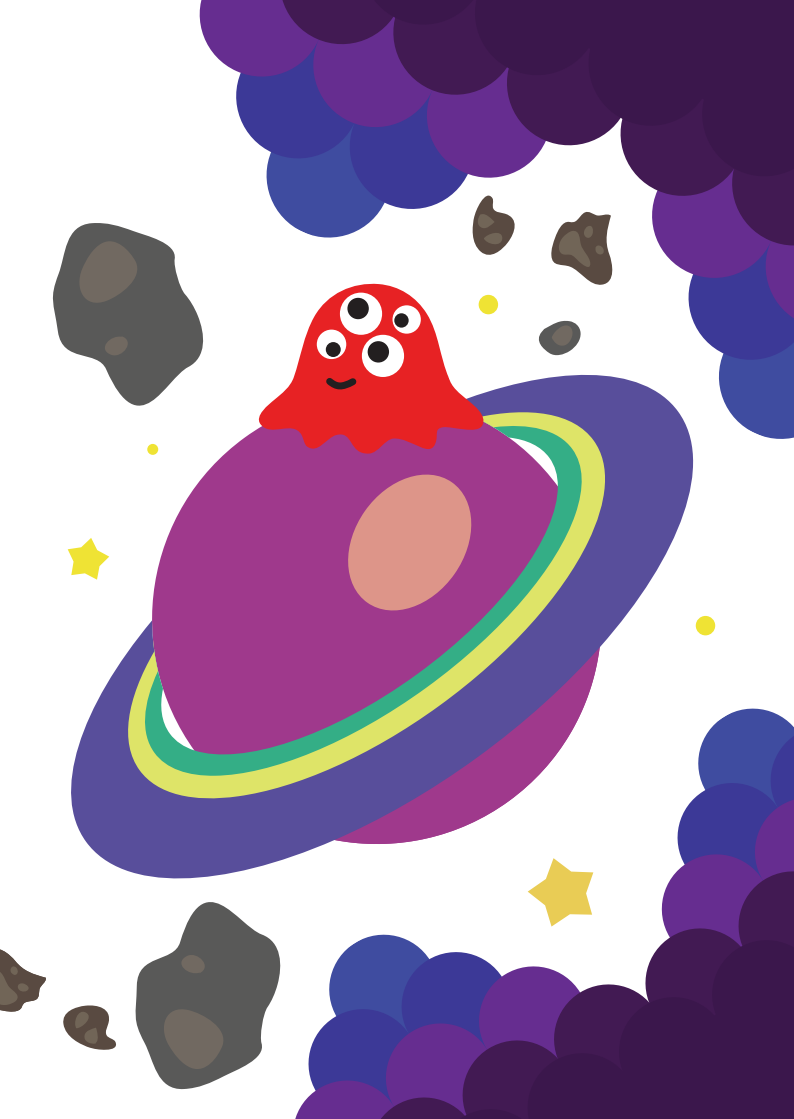
There are currently several built-in OS controls that are not included in the Windows Phone SDK, such as context menu, date picker, and others. Those controls are available as part of the Silverlight Toolkit for Windows Phone, available at *silverlight.codeplex.com*. This project is maintained by Microsoft and sees frequent updates. Other popular Windows Phone projects include *coding4fun.codeplex.com* and *mvvmlight.codeplex.com*.

Use *silverlightspy.com for* checking out the visual tree and properties of XAML based user interfaces.

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 Depth[30].

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

30 windowsphonegeek.com/WPToolkitBook



Programming Windows 8 Apps

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, Metro UI style apps built for Windows 8 will run on PCs and ARM based tablets alike. 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 Metro style apps. The design language of Metro is discussed in the Windows Phone chapter.

Prerequisites

To develop Metro style apps you require Visual Studio 11 or Expression Blend 5, which are both available as part of the Windows 8 Developer Preview release[1]. You can install Windows 8 within a virtual machine or side by side with your existing 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 with the preview release, such a license is automatically and free of charge: it is acquired when you start Visual Studio 11 for the first time.

1 msdn.microsoft.com/en-US/windows

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	no	no
Codesharing	Legacy Windows Apps, professional Xbox, other platforms, ...	Legacy Windows Apps, indie Xbox, Windows Phone apps, ...	Websites, HTML5 apps, ...

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 msdn[2], 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** use Windows Live as an authentication mechanism with zero click single sign-on[3] share data between a user's various devices, using SkyDrive[4]
- **Windows.Security.Authentication.Web** integrate with web services that use OAuth or OpenID Facebook, Twitter, etc
- **Windows.Security.Credentials** enables access to and storage of passwords

2 msdn.microsoft.com/en-us/library/windows/apps/br211377.aspx
3 msdn.microsoft.com/en-us/windowslive/hh561433
4 msdn.microsoft.com/en-us/windowslive/hh528485

— **Windows.ApplicationModel.Contacts** access or provide contacts

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.appxmanifest` file, then implement the required functionality.

There are following contracts[5]:

— **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.ShareTarge`.

5 msdn.microsoft.com/en-us/library/windows/apps/08800074-78ba-410a-962f-876da3463629

— **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`.
— **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.

Views and Form Factors

Metro style apps can run in different layout modes[6]:

— **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 acccessible, 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

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

snapped mode and your user selects a 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)[7] This also enables you to update the live-tiles of your app. Using WNS seems to be free of charge. You can use the Windows Azure Toolkit for Windows 8[8] to simplify the implementation of a push server.

7 msdn.microsoft.com/en-us/library/windows/apps/hh465460%28v=vs.85%29.aspx

8 watwindows8.codeplex.com

Live Connect

Windows 8 provide user credential management services[9] and using Windows Live for its user authentication. You can leverage this to provide single sign to your apps, enabling you to identify the user directly without further authentication[10].

Distribution

Metro style apps can be distributed through Windows Store[11] 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.

9 msdn.microsoft.com/en-us/library/windows/apps/br229572.aspx
10 msdn.microsoft.com/en-us/library/windows/apps/hh465098.aspx
11 msdn.microsoft.com/en-us/library/windows/apps/hh694084.aspx

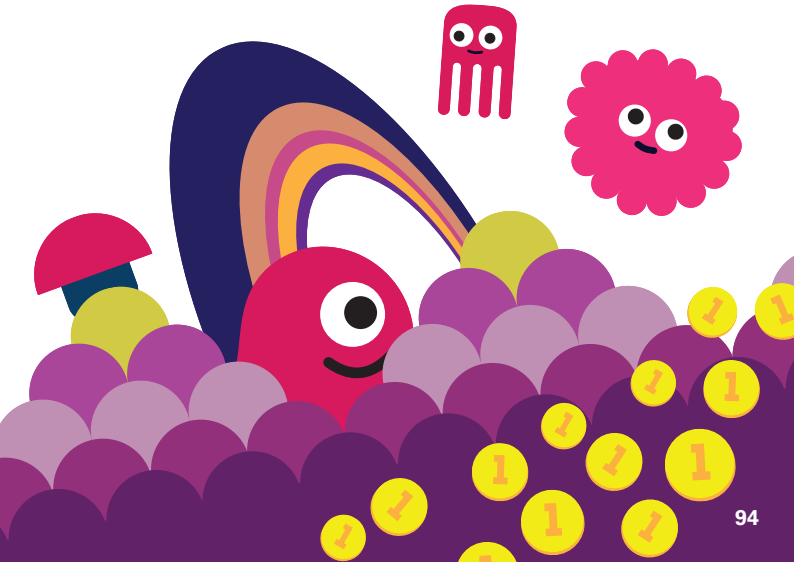
Resources

Your starting point for Windows 8 development is *msdn.microsoft.com/en-us/windows/apps/default*.

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%28v=VS.85%29.aspx*



Programming Mobile Widgets

We have mentioned that some approaches to mobile development require you to learn multiple languages and the unique features of individual platforms. One of the latest approaches to solving this problem, and offering one development technology for many devices, is mobile web widgets. These widgets are created using the scripting and mark-up languages used for websites (HTML, CSS and JavaScript) and bundle this web content into a zip archive that is installed on a device and run just like any other application.

The big advantage of widgets is that they offer probably the easiest route into mobile development. If you are a web developer widgets enable you to create mobile apps using your existing web design skills and code in the languages you already know. Equally, for anyone taking their first steps into mobile development – or first steps into programming – HTML, CSS and the JavaScript language are a lot easier to learn than the relatively complex native languages.

Widget Characteristics

In general, a widget can be characterized as a small website installed on a device. But if that's the case, why not simply use a website? Well, widgets have several advantages when compared with web pages:

- Widgets can be more responsive than websites: In a widget you work with raw data not HTML pages, the reduction in data overhead means widgets make better use of mobile network bandwidth.
- Widgets are already first class apps on some phones: Although widget environments vary, a user can open a widget in just a few clicks, there is no URL to type or bookmark to find. For example, on Symbian or BlackBerry devices widgets are installed and accessed in the same way as native applications.
- Widgets can look like native applications: Some widget environments include features that replicate the device's native menus and UI. Widgets that behave like native applications are much easier to use than websites.
- Widgets can run on a device's home screen: Some widget environments, such as Symbian, are able to provide summary views users can add to their device's homepage while others, such as Samsung's Touchwiz can incorporate arbitrarily sized widgets.
- Widgets can use device data: The ability to use device data, such as location or contact records, enables widgets to offer information that has context, such as identifying social network contacts based on the entries in the device's address book.
- Widgets can generate revenue: They can be packaged and distributed via application stores so you can sell them just as would a native application.

It is worth noting the emergence of a new type of widget: Proxy-based web browser widgets. These widgets fall into two broad categories:

- Server-based, such as those for Opera Mini, which at the time of writing were available through Vodafone only. These widgets run entirely on the proxy server. An obvious consequence of this is that these widgets cannot offer device side features.
- Hybrid, such as Series 40 web apps for Nokia phones. In these widgets some JavaScript can be executed on the device. In the case of Series 40 web apps, they can run code that reads a device's location, creates an SMS message, implement element transitions and enable dynamic alterations to the UI on the device. In addition to offering a richer user experience this hybrid approach reduces round trips to the server, for example by eliminating the need for the server to paint every screen refresh.

It is worth noting the emergence of a new type of widget: Proxy-based web browser widgets. These widgets fall into two broad categories:

- Server-based, such as those for Opera Mini, which at the time of writing were available through Vodafone only. These widgets run entirely on the proxy server. An obvious consequence of this is that these widgets cannot offer device side features.

— Hybrid, such as Series 40 web apps for Nokia phones. In these widgets some JavaScript can be executed on the device. In the case of Series 40 web apps, they can run code that reads a device's location, creates an SMS message, implement element transitions and enable dynamic alterations to the UI on the device. In addition to offering a richer user experience this hybrid approach reduces round trips to the server, for example by eliminating the need for the server to paint every screen refresh.

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 Wiki[1]. Because the standards are not complete, it is important to note that each widget tech-

1 www.w3.org/2008/webapps/wiki/WidgetSpecs

nology has slightly different ways of implementing 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):** *bit.ly/blackberry-widgets*
- **Nokia WRT (on selected S60 3rd Edition, Feature Pack 2 devices and all S60 5th Edition and Symbian^3 devices):** *bit.ly/nokia-wrt*
- **Nokia Browser for Series 40 (selected Nokia Series 40 devices):** *www.forum.nokia.com/webapps*
- **Vodafone360:** *bit.ly/vf-widgets*
- **WAC/ JIL:** *www.jil.org*
- **Windows Mobile (v6.5):** *bit.ly/winmo-widgets*

To port your widget over to platforms that don't natively support widgets, such as iPhone or Android, you can use tools such as PhoneGap[2], Titanium from Appcelerator[3], and Rhomobile[4] among others. Of these options, PhoneGap offers a solution that is closest to the W3C approach.

2 www.phonegap.com
3 www.appcelerator.com
4 www.rhomobile.com

Prerequisites

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 have a tool for web development it can be used for widget development. The primary advantage of using a web editor is the support these tools provide for composing HTML, CSS and JavaScript.

There are a number of tools specifically designed for developing widgets also. These may be delivered as plug-ins or add-ons to web authoring tools, as with the BlackBerry Widget SDK[5] which works in conjunction with Adobe Air, or standalone tools such as Nokia Web Tools[6]. These tools generally provide template projects, a preview environment, validation, packaging, and deployment features.

Writing Your Code

In general, there are no special requirements for writing code for a widget. The principal area where a widget differs from a website is the variety of relatively small screen sizes it has to work on. Devices running widgets may offer WVGA, nHD, QVGA, or other resolution screens. CSS provides an elegant solution to reformatting information to accommodate these varying screen sizes.

By the way: Try to use CSS3 whenever possible and remove any old compatibility code or you may run into issues.

5 us.blackberry.com/developers/browserdev/widgetsdk.jsp
6 www.forum.nokia.com/Develop/Web/Tools#NWT

You can start by simply:

1. Creating `index.html` and `config.xml` files.
2. Zipping them at the command line using zip `myWidget.wgt index.html config.xml`.
3. Opening the `myWidget.wgt` file in Opera.

Of course, your widget can use AJAX also and one of the many JavaScript libraries, such as jQuery, MooTools, YUI, Dojo, or Guarana.

Depending on the widget platform you are targeting you may be able to use more advanced technologies such as Canvas, SVG, Flash Lite, or even HTML5 features such as the `<audio>` and `<video>` tags.

In addition, each environment's APIs for retrieving device information, accessing user data, storing data, or other environment specific tasks will need to be mastered. In most cases these APIs follow JavaScript conventions and are easy to learn. For example, the following code uses Nokia Platform Services 2.0 APIs to asynchronously determine a device's location:

```
serviceObj = nokia.device.load("geolocation");
serviceObj.getCurrentPosition(success_callback,
error_callback,positionOpts);
…
function success_callback(result){
  Lat = result.coords.latitude;
  Long = result.coords.longitude;
}
```

While standards are still to be finalized, overall the APIs are moving in very similar directions. The W3C Geolocation API Specification proposes an almost identical API for the same task:

```
navigator.geolocation.getCurrentPosition(
  successCallback,errorCallback,
  positionOptions);
```

All the widget runtimes are advancing quickly, with new features being added regularly. While keeping up with these developments may be a challenge it is certainly worthwhile if you want to create leading edge widgets.

Testing

It is always good to be able to test on a PC. If your widget is W3C compliant you can use Opera 9 or later as an all-purpose option. However, if your widget includes device integration or platform specific features you will need to look to other tools and fortunately most widget development tools provide a computer based preview environment as well.

For example, when creating Series 40 web apps, Nokia Web Tools include Web Apps Simulator that runs your app on Microsoft Windows, Ubuntu Linux and Apple Mac computers. The features offered by these widget specific preview tools vary, but common features include being able to display widgets in various screen resolutions and orientations, issue device triggers (such as removal of a memory card) to the widget, and testing against simulated device data (such as contacts and location data). Some of the tools support debugging too, as Nokia Web Tools do through an implementation of Web Inspector.

Of course, once you finish desktop testing, final testing on your own phone will be essential. The way a widget looks and behaves can only be fully assessed on a real screen, under realistic lighting conditions, and in a real network.

Signing

Currently most widget environments don't require widgets to be signed, although there are exceptions, such as BlackBerry. This situation may change as the APIs to access device features become more advanced. It is worth noting that the W3C standards include a proposal for widget signing.

Distribution

While the W3C is working on standards that will enable widgets to be discovered from websites, in very much the same way RSS feeds are today, there is no universal mechanism for widget discovery, yet.

However, some widget environments enable you to add a link to a widget on a website so that the widget installs directly into the environment or device when downloaded. For example, by identifying a Nokia WRT widget with the MIME type `AddType x-nokia¬widget .wgz` downloading the widget on a Symbian device will automatically initiate the installation process.

Distribution via a website is not the only option. Many application stores welcome widgets. As we went to press, the only store that supports W3C widgets is the Vodafone Widget store[7], but by packaging your widgets appropriately you can upload them into Nokia Store[8], the Windows Marketplace[9] or RIM BlackBerry AppWorld[10]. You can use tools, such as PhoneGap, to port your widget to a native application environment, thus gaining the option to use other stores, such as Apple AppStore and Android Market among others.

7 widget.vodafone.com
8 store.ovi.com
9 www.windowsmarketplace.com
10 appworld.blackberry.com/webstore

Programming With Cross-Platform Tools

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, BlackBerry PlayBook OS, iOS, Symbian and Windows Phone are among the most important smartphone and tablet platforms while Java ME and Brew MP dominate on feature phones.

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 in 2011, 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.

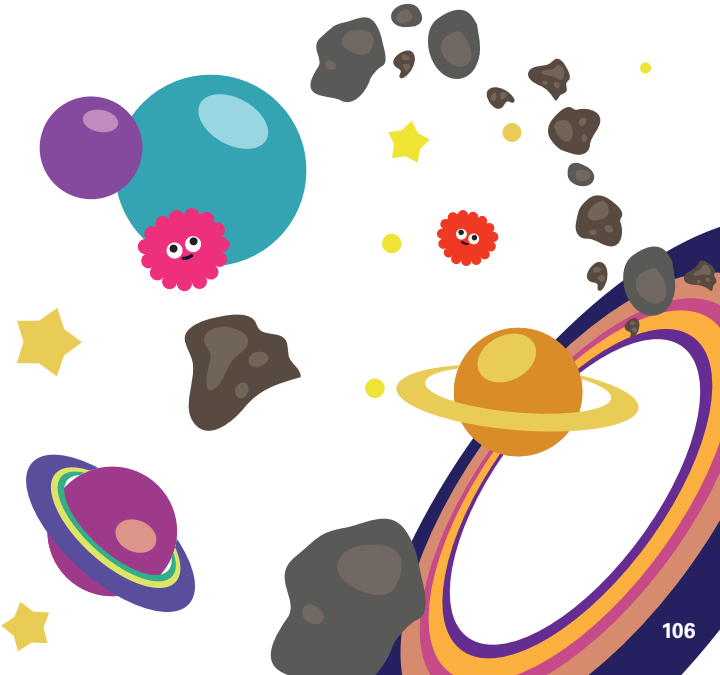
By the way, we are not talking about app stores here; this is a different market fragmentation problem. The more than 120 app stores, from operators, manufacturers and independent companies create challenges of their own, outlined in the "Appstores" chapter.

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	1st Class Citizen[A]	2nd Class Citizen[B]	Target Platforms
ActionScript	1	0	BlackBerry PlayBook OS (QNX)
C, C++	4	3	1st class: bada, BlackBerry PlayBook OS, Brew MP, Symbian, Windows 8, Windows Phone Classic. 2nd class: Android (partially, using the NDK), iOS (partially)
C#	1	0	Windows 8, Windows Phone and Windows Phone Classic (formerly Windows Mobile)
Java	3	2	1st class: Android, BlackBerry, Java ME devices 2nd class: Symbian, Windows Phone Classic
JavaScript	1	2	1st class: BlackBerry PlayBook OS, Tizen, webOS, Windows 8. 2nd class: BlackBerry (WebWorks), Nokia (WRT)
Objective-C	1	0	iOS

A) Supported natively by the platform, for example either the primary or only language for creating applications

B) Supported as an option by the platform, for example can be used as an alternative to the native language but generally won't provide the same level of access to platform features.

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-independent[1].

1 *For example:* www.developergarden.com/apis/

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.

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.

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

2 urbanairship.com/

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.

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 don't 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. Typically you have no access to hardware features and native UI elements, so in our opinion they don't 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.

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

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 func-

tionality 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 Solutions

There are many cross-platform solutions available, so it's hard to provide a complete overview. You may call this fragmentation, I call it competition. A word of warning: we don't 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*.

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 www.applicationcraft.com	Commercial	HTML, CSS, JavaScript	Android, BlackBerry, iOS, Symbian, webOS, Windows Phone, mobile sites
appMobi www.appmobi.com	Commercial	HTML, CSS, JavaScript	Android, iOS
Bedrock www.metismo.com (Metismo)	Commercial	Java ME	Android, bada, BlackBerry, brew, Consoles, iOS, PC, webOS, Windows Phone, Windows Phone Classic
Celsius mobile-distillery.com (Mobile Distillery)	Commercial	iOS, Java ME	Android, BlackBerry, brew, iOS, Symbian, Windows Phone Classic
Corona www.anscamobile.com (Ansca Software)	Commercial	JavaScript	Android, iOS
EDGELIB www.edgelib.com (elements interactive)	Commercial	C++	Android, iOS, PC, Symbian
id Tech 5 www.idsoftware.com (id)	Commercial	C++	Consoles, iOS, PC
Irrlicht irrlicht.sourceforge.net gitorious.org/irrlicht-android	**Open Source**	C++	Android & iOS with OpenGL-ES version, PC
J2ME Polish www.j2mepolish.org (Enough Software)	Open Source + Commercial	Java ME, HTML, CSS, JavaScript	Android, BlackBerry, iOS, J2ME, PC, Windows Phone Classic

Solution	License	Input	Output
Flash adobe.com/devnet/devices.html (Adobe)	Commercial	Flash	Android, iOS, PC
Feedhenry feedhenry.com	Commercial	HTML, CSS, JavaScript	Android, BlackBerry, iOS, Nokia WRT, Windows Phone
Marmalade madewithmarmalade.com (Ideaworks3D)	Commercial	C++	Android, bada, iOS, webOS
MobiForms www.mobiforms.com	Commercial	Drag and Drop + MobiScript	Windows PC, Linux, Unix, Windows CE, Windows Mobile, Android, iOS
Mono for Android http://xamarin.com/monoforandroid (Xamarin)	Commercial	C#	Android
Mono Touch xamarin.com/monotouch (Xamarin)	Commercial	C#	iOS
MoSync www.mosync.com	Open Source + Commercial	C	Android, iOS, Windows Phone 7, Symbian, Java Mobile, Windows Phone Classic, Moblin, BlackBerry (beta)
PhoneGap www.phonegap.com (Nitobi)	Open Source	HTML, CSS , JavaScript	Android, BlackBerry, iOS, Symbian, webOS, Windows Phone (Qt announced)

Solution	License	Input	Output
Qt qt.nokia.com (Nokia)	Open Source + Commercial	C++	PC, Symbian, MeeGo and Windows Phone Classic, desktop Windows, Apple & Linux OS
Rhodes rhomobile.com/products/rhodes (rhomobile)	Open Source + Commercial	Ruby, HTML, CSS, JavaScript	Android, BlackBerry, iOS, Symbian, Windows Phone Classic
SIO2 sio2interactive.com (sio2interactive)	Commercial	C	iOS, other announced
Spot Specific www.spotspecific.com	Commercial	Drag and Drop, JavaScript	Android, iOS. (BlackBerry & Windows announced)
Titanium www.appcelerator.com	Open Source	JavaScript	Android, Consoles, iOS, PC
Verivo verivo.com	Commercial	(Visual)	Android, BlackBerry, iOS
Unity3 unity3d.com (Unity Technologies)	Commercial	C#	Android, iOS, PC
Unreal www.unrealtechnology.com (Epic Games)	Commercial	UnrealScript, C++	Consoles, iOS, PC
XML VM xmlvm.org	Open Source + Commercial	Java, .NET, Ruby	C++, Java, JavaScript, .NET, Python

Creating Mobile Websites

Why create a mobile website instead of an application? Using the web has a number of advantages, websites can be browsed on most devices, the technology is flexible, and it is easy to update sites so all users get the latest version. You only need to modify a single codebase if you want to add or change content or even features, rather than updating each installed application.

Context Is King

When you create a website for desktop browsers, you typically design and develop for users who have a large display, a fast and persistent internet connection, powerful PC processing and plenty of time. None of these are guaranteed when using the internet on mobile devices.

Although new devices are gaining ever larger screens, they are still small compared to the typical PC. Your design should therefore ensure that all elements are easy to read, and for touch-screen devices, easy to select. Very often less is more.

Issues with the speed of mobile internet connections and processor power on the phone go hand in hand. If pages take too long to download and render, you can easily lose a user and they may look to obtain the information elsewhere. Therefore you should try to keep content, particularly external scripts and images, as small as practical.

But perhaps the most significant factor to consider is that mobile users often access a website when they are on the go. This means they normally have limited time to browse your site. So you should focus your mobile website on the content a user is most likely to be looking for: They are unlikely to be interested in your awesome company main page's Flash animation,

but will probably appreciate finding your address or phone number quickly.

Overall you should think about the contexts in which your mobile site will be used before you begin to design a mobile website. It could be used in a train with a weak signal, in a village with poor connection speed, outside in a sunny environment on a high-end smartphone with touchscreen display or on an older feature phone with a limited browser and keypad operation. You cannot influence the context a user is in, but you can design your site to be useable under all these circumstances.

Usability Aspects

After thinking about context and content, it is equally important to consider usability. It is not only a matter of what content is interesting for a user and the contexts in which it will be consumed, but also how your target audience will use your site.

Navigating your site is less fun when using a device's keyboard compared to using a large touch-screen. However, a keyboard can be easier to use if the links within your page are so small that it is almost impossible to tap them without causing unwanted behavior: Tapping another link, rather than the one you wanted.

Here are some basic hints to help make sure your content is adapted well and offers the best usability:

1. **Make it "mobile" not only "small":** Create a concept that utilizes the possibilities of the technology. You won't satisfy many people by simply offering a smaller version of your classic website. Mobilize, don't miniaturize!
2. **Keep all paths open:** Leave it up to the user to access either the mobile or desktop version of your website.

3. **Keep it simple:** Avoid complex navigation structures; users will not dig that deep while they are mobile.
4. **Avoid text input wherever possible:** Text input on mobiles is hard. If you ask the user to enter text, use wide input boxes so that they see what they are typing. Buttons to clear an input field on click/touch can also be helpful.
5. **Adapt the media:** Adapt all pictures, videos and alike to be displayed optimally on the handset (check the "Implementing Rich Media" chapter in this guide for more information). Try to avoid formats such as .doc and pdf if possible.
6. **The user is a creature of habit:** Respect this and adapt usage patterns from classic websites, such as linking logos to the homepage or offering corrections to mistyped search requests.
7. **Think of stubby fingers:** When optimizing your content for touch screen phones do not use clickable areas smaller than 50 x 50 pixels.
8. **Use sharp contrasts:** Use fonts and background colors that guarantee legibility in any surrounding, particularly in bright sunlight.
9. **Reflect continuously:** Ask yourself if you would use the implemented features yourself. Ask your friends and colleagues as well, before realizing your ideas.
10. **Do not require the user to think too much:** Try to implement intuitive navigation, do not force users to make decisions more often than necessary.

Technical Limits of Web Technologies

When it comes to the decision whether to create an app or a mobile website for a specific project, many mobile developers tend to say "app" first because this approach seem to offer more possibilities and better performance. This answer is seldom wrong, but it is only half the truth. Always take a detailed look at the advantages and disadvantages of mobile websites or web apps compared with native mobile apps.

If you are coming from the "desktop world" and you have never created a website for mobile devices before (or if the last time you did was some time ago) you may be surprised by the capabilities of web browsers on modern mobile phones. It is the capabilities of these newer browsers we will focus on.

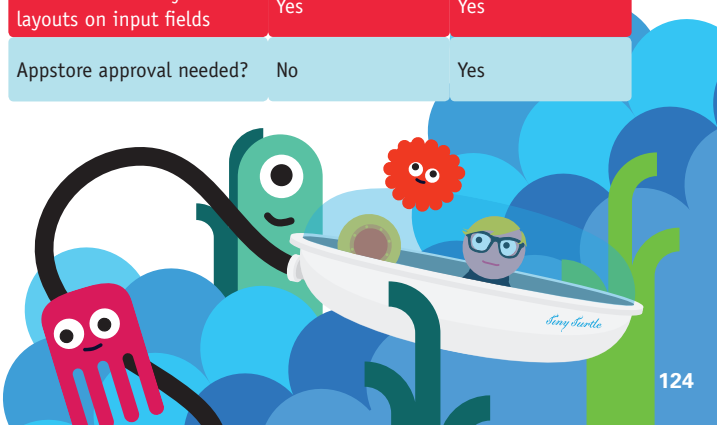
The current generation browsers on the major platforms support a variety of modern HTML5, CSS and JavaScript features, such as Geolocation, WebGL (interesting for mobile game development), hardware acceleration, offline storage and many more. You can easily find out if a user is online or offline and synchronize online data to the device for later offline use, when a user has no active internet connection. You can ask for permission to query the user's current position, just as you can do in a native app. You can also access the gyroscope of an iPhone and the accelerometer on a Nokia N9 using pure JavaScript directly in the browser. Mobile browser vendors are working on making it possible to access the phone's camera, network status or address book data among others.

The capabilities of JavaScript are often underestimated. However, it can be used to rapidly create high class web apps that make extensive use of a device's capabilities, (almost) without the need to create proprietary versions of your web app for each platform.

And that's not all: almost all recent mobile browsers support many of the current CSS3 standard's features and so you can create appealing effects such as transitions, custom web fonts, drop shadows or rounded corners with little effort. And this makes it relatively easy to create web applications that look and feel like native apps.

But these are still issues. Although modern browsers may offer a wide range of device APIs, there are still things you cannot yet do inside a browser: Accessing the camera, as already mentioned, is one. You cannot prevent the device from going to standby mode on a website, which can sometimes be a problem. And sure: you will need to implement your own user interface in HTML, CSS and JavaScript instead of being able to use native GUI objects — although several vendors provide template and CSS libraries you can use. Also, if your code is clean and any UI effects are used wisely, the performance difference to native apps can be so small that a user may not realize they are using a web app instead of a native app.

Feature	Mobile Website	Native App
Detect online status	Yes	Yes
Offline data storage	Yes	Yes
Access GPS/geolocation	Yes	Yes
Access gyroscope	Yes	Yes
Access camera	Not yet (planned)	Yes
Access address book	Not yet (planned)	Yes
Notifications (i.e. vibration, push, messages)	Not yet	Yes
Cross-platform compatible	Yes	No
Check battery status	Not yet (planned)	Yes
Different touch keyboard layouts on input fields	Yes	Yes
Appstore approval needed?	No	Yes



Fragmentation

"In mobile, fragmentation is forever"[1]. Unfortunately it is not always as easy to create a cross-device, cross-platform, cross-browser, cross-markup, "cross-blahblah" mobile website as you might think. Dealing with many different devices also results in an annoying fragmentation jungle. Some devices use their own implementation of device APIs (such as Geolocation on Blackberry OS 4.6) or even have no support for certain features (such as filesystem access on iOS).

This means you have to write workarounds of your code for different platforms and even for the same browser running on different platform versions. But the web wouldn't be the web if there wasn't already a solution for almost all of these problems. And so there are sites, libraries and services such as *caniuse.com*[2], *Modernizr*[3] or *fitml.com*[4] where you can build quick and clean workarounds to handle fragmentation issues with only a minimum of effort. But remember, for most code you will only need to write most parts of your web app once.

1 Richard Wong, Techcrunch, March 2010
2 www.caniuse.com
3 www.modernizr.com
4 www.fitml.com

Website Adaptation

To deliver a great user experience you need to adapt your site to the device it is being viewed on. Typical adaptations include:

- **Images scaling:** In most cases it is not necessary to deliver a 800×600 JPG with a file size of 120K to a device whose display resolution is only 320×480: You should re-size the image to the display size of the mobile device and then serve a much smaller version to the client.
- **Content markup:** If you have a visitor with an iPhone or an Android phone you can serve a rich HTML5 document, while users with an old Nokia feature phone can be sent an old fashioned XHTML 1.1 document.
- **Feature disabling/swapping:** Your website or web may need to disable feature on some browsers, or replace a "rich" feature with a simpler one. For example, if you use location to show the user nearby services, but the browser does not support geolocation you may offer the ability to select a location on a map or enter a postcode as an alternative.

Adapting you content relies on determining the characteristics of the device – such as screen size and input method – as well as the capabilities of the browser. To do this you will have to determine which (type of) device is sending a request to your website. Then there are two methods for obtaining more information: "guessing" (server-side) or testing (client-side) for the supported features.

Server-Side Detection and User Agent Sniffing

Server-based "user agent sniffing" is usually based on large databases containing the user agents of thousands of devices and their capabilities.

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 some commercial providers of device databases (such as WURFL[5], DeviceAtlas[6] and fitml.com). These services are well worth considering, as they have full time employees actively maintaining their databases and do a lot of work that would be impractical for you to do.

However, be aware that user agents can be "wrong", manipulated or unknown. You should therefore provide a fallback in case your user agent detection fails. Such a fallback could be a document in a format (such as HTML4) that almost every mobile browser released in the past five years can understand.

Client-Side Feature Detection And Adaptation

Client-side feature detection general involves use of JavaScript to detect whether a certain capability is supported on a device. For example, you can use `if(navigator.geolocation)` to check whether a device supports the acquisition of the user's current position.

Because feature detection occurs on the device, it means you have to deliver a comprehensive document containing all features of the website and then gracefully degrade the document by removing features that are not supported. This means you are sending a lot of content to all devices, regardless of whether a certain feature is supported on a device.

5 wurfl.sourceforge.net
6 www.deviceatlas.com

The one big advantage of this approach, compared to server-side adaption, is that when a feature test passes you can (under normal conditions) be sure that your desired feature or behavior will work as expected. This guards against cases where a user agent has been modified and therefore provides the wrong information from a device capability database.

The Modernizr JavaScript library is a good option for client-side feature detection. Modernizr covers many browser capabilities and provides a simple checking API. For example, if you need to know if a browser supports Drag and Drop, use the Modernizr library like this:

```
if(Modernizr.draganddrop) {
       // browser supports native drag and drop
     } else {
       // fallback
}
```

Another task that belongs on the client-side is the adaption of layout. On modern mobile browsers you can use media queries that let you apply CSS rules only if a client or browser matches certain conditions. You can easily show a two column layout when a device exceeds a display width of 800px (on tablets for example) and fall back to a one column layout when a device's display has a resolution of less than 800px.

The challenges to be aware of include browser bugs, which may be hard to detect on the client side.

Using The Best Of Both Worlds

Given that agent detection may yield false results and client-side optimization may require overly large files, combining server-side and client-side approaches can be of benefit. In this approach you determine the basics features ("which mark-up should be used?", "what is the correct size for images?", "JavaScript support, yes or no?" and alike) from the user agent on the server and then let the client handle the rest.

Note that you should provide server-side fallbacks for JavaScript detection, to make a site accessible even on old devices.

A commercial platform working according to this principle is *fitml.com*[7] where you describe your content in an abstract XML markup called FITML, the platform then converts your markup to the best suitable output format and optimizes both on server- and client-sides.

Hybrid Apps

If you want (or need) to publish your mobile app in an app store, you can create a "hybrid app". Using this approach, you create your app using common web technologies (HTML, CSS, JavaScript) and then compile it as native app. There are several hybrid app frameworks that make this easy and from a single HTML5 web app code base that reduces your development and maintaince costs. Such frameworks include PhoneGap[8], Appcelerator[9] and Apparat.io[10] among others.

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 a web

7 www.fitml.com
8 www.phonegap.com
9 www.appcelerator.com
10 apparat.io

app – such as vibration, access to camera or address book – but they are rarely as comprehensive as the features for native apps. And you still have to develop your own UI that might be slower than a native one.

Lessons Learned

The gap between native apps and web apps is rapidly decreasing. Browser vendors have done much good work recently to bring new features to web apps — and this is only going to get better.

Creating mobile websites or mobile web apps makes your content accessible on almost every platform with much less effort than native development for several platforms: saving you in development and maintenance costs. Using hybrid app frameworks you can even publish your apps in app stores. However, you still need to intelligently optimize content, because of the variation in the wide variety of browser used on mobile devices. Using a combined approach of server- and client-side optimization can be your best option for doing this.

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.

If you have never developed a mobile website or web app before or were put off by the early, poor browsers you should give it another try: features have improved by at least 1000% since the rise and success of Android and iOS.

But always have one thing in mind: **make your mobile website mobile, not just smaller!**

Developing Accessible Apps

Regardless of the technology you choose to develop your apps, you'll 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 could 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.

General Platform Accessibility

Some of the mobile platforms have built-in accessibility features that can help make it easier for people to use your apps. 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:** Enable people to choose what works best for them
- **Voice Control and Siri:** This enables users to make phone calls and control music playback using voice commands.

Many users rely on third-party applications such as TalkBack on the Android platform, available from the market place or Talks

from Nuance for the Symbian platform, which provides screen reading and screen magnification features.

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.

Making Your App Accessible

As a developer, you should keep in mind these guidelines to make your software accessible for users with disabilities. 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
- 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 won’t be able to identify errors if the are marked by coloring them red only
- Ensure good colour contrast throughout your app.
- Use the Accessibility API for your platform, if there is one. This will enable you to make custom UI elements more

accessible and will mean less work on your part across your whole app

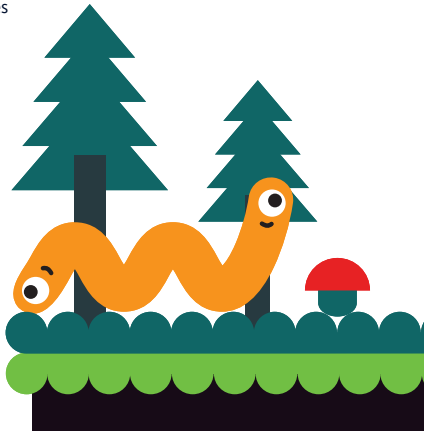
- Support programmatic navigation of your UI. This will not only enable your apps to be used with an external keyboard but will enhance the accessibility of your app on platforms such as Android where navigation may be performed by a trackball or virtual d-pad.
- Test your app on the target device with assistive technology such as VoiceOver on the iPhone

You can find a more comprehensive list of guidelines online[1].

Developing Accessible Android Apps

Android has several accessibility features including an accessibility API. Again, when developing Android apps 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.

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



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 Designing for Accessibility[2].
For more information about Android accessibility including how to use the text to speech API, see the Eyes-Free project[3].

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 developers[4].

Developing Accessible iOS Apps

iOS has good support for accessibility. However, it only works well if the accessibility guidelines[5] have been followed. 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 con-

2 developer.android.com/guide/practices/design/accessibility.html
3 code.google.com/p/eyes-free
4 docs.blackberry.com/en/developers/deliverables/11936
5 developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/iPhoneAccessibility

trols where possible. If you are developing using Qt, then please check the web for details of their accessibility API[6].

Developing Accessible Mobile Web Apps

Though the focus of this chapter has been how to develop accessible native apps, we can never ignore the role the web has to play, especially as the line between native app and web app is blurring.

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

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

If you are a web content developer, then you should take a look at the Web Content Accessibility Guidelines (WCAG) Overview[7].

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 Practices[8] as this is likely to form the foundation of any mobile web application accessibility standard that emerges in the future..

6 doc.qt.nokia.com/qq/qq24-accessibility.html
7 w3.org/WAI/intro/wcag
8 w3.org/TR/mwabp

In Conclusion

As the various mobile platforms are becoming increasingly sophisticated, we are seeing it become a more accessible place for users with disabilities and though we eagerly await what Windows Phone, Tizen, BlackBerry 10 and others have to offer in this regard, a level playing field will only be achieved to the degree that developers are aware of the appropriate APIs and features within their platform of choice.

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. Below are the recommended formats:

Container	mp4, 3gp, avi (BlackBerry only), wmv (Windows Phone only)
Protocol	HTTP (progressive or download) or RTSP (streaming)
Video	H.264, H.263
Audio	AAC-LC, MP3, AAC+
Resolution	320x240, 480x320, 480x800 (tablets only), 1024x768 (iPad only), 176x144

Streaming vs. Local Storage

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

To stream content through relatively unstable mobile networks, a specific protocol called RTSP was developed that solves

latency and buffering issues. Typical frame rates are 15 fps for MP4 and 25 fps for 3gp, with data rate up to 48 kbps for GPRS (audio only), 200 kbps for Edge, 300 kbps for 3G/UMTS/WCMDA and 500 kbps for Wi-Fi and 4G.

Apple's open source Darwin streaming server[1] can serve streaming video and audio with the highest level of compatibility and reliable RTSP combined with FFMPEG[2] and is always a good choice to stream 3gp or mp4 files.

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

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

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

1 dss.macosforge.org
2 www.ffmpeg.org
3 technet.microsoft.com/en-us/windowsserver/

tiple 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 Mp4box[4]. Note that an mp3 file doesn't need and cannot be hinted.

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, AAC and AMR encoder support included. There are good converters available based on FFMPEG, such as "Super" from eRightSoft[5]. For MAC users, QuickTime pro (paid version) is a good alternative to encode and hint 3gp files. If you are looking for a complete server solution with a Java/ open source background, check out Alembik[6].

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

Implementing Location-Based Services

Knowing where a mobile user is located geographically enables mobile services to be more accurate and timely: helping find a nearby parking space, analyzing pollen reports for your local area, finding friends at the trade fair or obtaining directions to the local zoo. The zip code of your current location may be good enough to locate a nearby barber, while higher precision will be required to find your GPS-tagged hunting dog or lost toddler.

How To Obtain Positioning Data

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

— **Network positioning:**
Each GSM or UMTS base station carries a unique ID, containing its country code, network id, five-digit Location Area and two-digit Routing Area, from which geo coordinates for the base station can be obtained by looking up the operator's declaration. More accurate methods include measuring the difference in time-of-arrival of signals from several nearby base stations (multilateration). For phones with WiFi capabilities, known wireless LAN access points can also be used. Several companies monitor WiFi signals by driving around cities, and sell these data sets to third parties or use in-house. In general, accuracy depends on

the cell size (base station density): Higher accuracy is obtained in urban areas than in rural areas.

— **GPS positioning:**
The built-in GPS module in the phone (or an external one) gives you an accuracy ranging from 5 to 50 meters, depending on quality of the hardware as well as the terrain, canopy and wall materials. 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. Modern phones sometimes have an on-board assisted GPS chip, which minimizes the delay until the first GPS fix is obtained by providing orbital data, accurate network time and network-side analysis of snapshot GPS information from devices. Note; the assisted GPS does not provide a more accurate position, only a faster result when the GPS is initially enabled, or when exiting from an area of bad GPS satellite coverage.

— **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 simply contain an id that needs to be looked up to obtain a position, while they can contain more data, e.g. provide the latitude and longitude directly.

— **Manual input:**
The user selects a location on a map, inputs 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 several image bitmaps, represented as vector data or a combination of both. Vector data has the advantage of consuming much less bandwidth than bitmaps do. Vector data also allows for arbitrary zooming, but 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 Map Open Street Map[1] or CloudMate[2], while Nokia Maps – through Nokia Location Platform[3]– are free for apps targeting Nokia phones only. Commercially available maps include NAVTEQ[4], Garmin and Microsoft[5] to name a few. Some solutions, such as Google Maps[6], 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, such as 1000 tiles from a single map key. 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 their own purposes. Android developers also have access to high-level libraries and

1 wiki.openstreetmap.org/wiki/Software
2 www.developers.cloudmade.com/projects
3 www.developer.nokia.com/Develop/Maps/
4 www.nn4d.com
5 www.microsoft.com/maps/developers
6 www.code.google.com/apis/maps

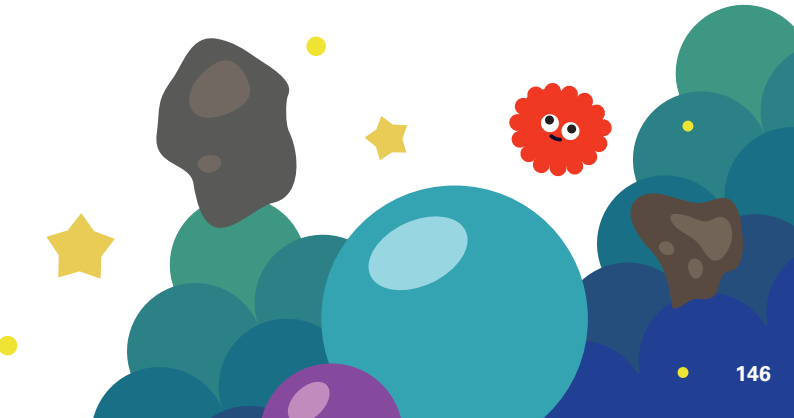
these devices are more liberal with the choice of map sources, though they do default to Google's Maps. On Symbian devices, Nokia Maps can be used for Nokia phones free of charge also for commercial use.

Maps can be overlaid with geodata, in a number of formats.

One of the simplest standards is called geoRSS, and could look like this for a single point-of-interest:

```
<entry>
  <title>Byvikens fortress</title>
  <description>Swedish 1900 century army
install, w. deep mote
  </description>
  <georss:point>18.425 59.401</georss:point>
</entry>
```

There are many more formats for geodata, but the basic idea is similar; increasingly sources are harmonizing their data streams for interoperability. Other important formats include the Geography Markup Language (GML), an XML encoding specifically for the transport and storage of geographic information, and KML that is an elaborate geoformat used in Google Earth.



Tools For LBS Apps

Several players in the industry 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 few mobile platforms.

Location aware does not always mean maps, it could as well be a clever way to sort waypoints in a list, with the closest one first. Companies Admob and NAVTEQ offer developers a standalone location aware advertisement program, where applications can exchange location data for smarter display of location relevant ads.

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

- **Garmin Mobile XT SDK:** *developer.garmin.com*
- **Android offline maps project:** *code.google.com/p/bigplanet-tracks/*
- **TeleAtlas:** *developerlink.teleatlas.com*
- **Nutiteq:** *www.nutiteq.com*
- **RIM:** *us.blackberry.com/developers/* (search for "map api")
- **Spime:** *www.spime.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 standards[1] provide for three modes of operation that can be used in mobile devices:

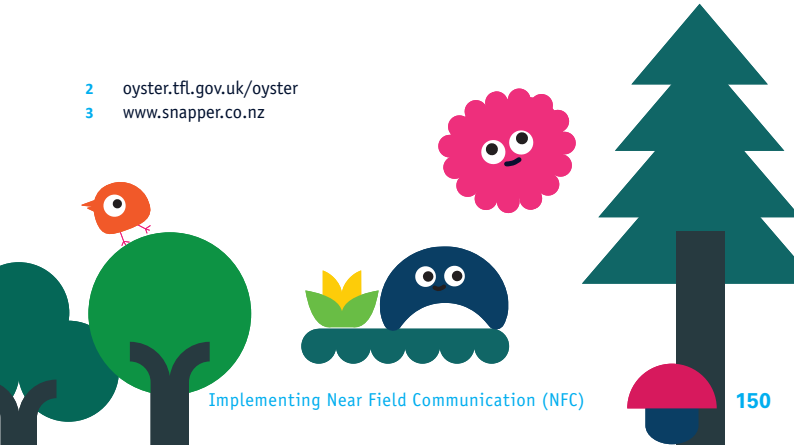
- **Read/Write:** where a phone can read or write data to a tag
- **Peer to Peer:** where two NFC enabled phones can exchange data, from information for creation of a Bluetooth connection through to business cards and digital photos
- **Card Emulation:** where a phone can act as a tag or contactless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

1 www.nfc-forum.org/specs/spec_list/

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 Oyster[2] or payments cards such as Snapper[3].

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 Nokia[4] 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 capabilities[5].

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 APIs[6], 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 Android[7].

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

The Android platform is unique for vibration control. It provides native support and has more vibration control options than any other platform. In comparison, iOS is limited to one method, `AudioServicesPlaySystemSound (kSystemSoundID_Vibrate)`. Furthermore, there are ways to extend Android vibration control to be similar to what is found in console games system, like X-Box or PlayStation.

This is why for now the following chapter is focusing on how to implement haptic vibration feedback on Android. We hope that the other OS will provide similar capabilities soon, we will then make sure upcoming editions of this guide covers those opportunities as well.

When designing a user interface, keep in mind the ultimate experience of the user. Spend some time planning before starting your Haptic implementation. Answer these basic questions:

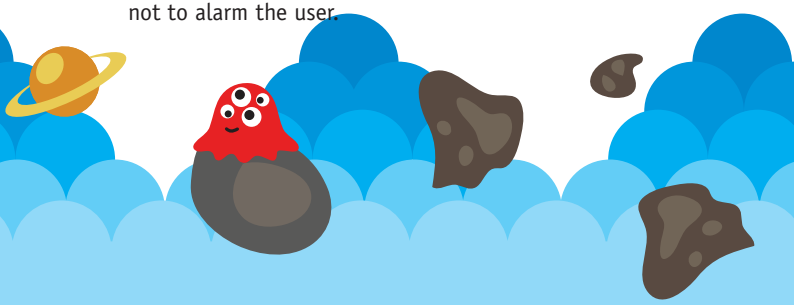
- What is the intention of the user interface? Is it showing something? selling something? explaining something? exposing something? Does it deal with tangible things or abstract concepts?
- How should the user feel when experiencing the final product? Relaxed? Excited? Frightened? Curious?
- Are the Haptic (touch) sensations a central part of the presentation, or are they intended to support and enhance other content?
- Will the user be watching something passively, doing something actively on the handset, or both?

- Should everything about the interface be straightforwardly presented to the user, or will the user have an opportunity to explore the interface to find hidden "gifts" or "surprises"?
- What type of touch-enabled device will most users have for experiencing the finished work, a handset, a tablet, something else?

There are certainly other questions of this type which go into the planning process for any new interface design project. Once the project is defined and taking shape in your mind, consider the following guidelines:

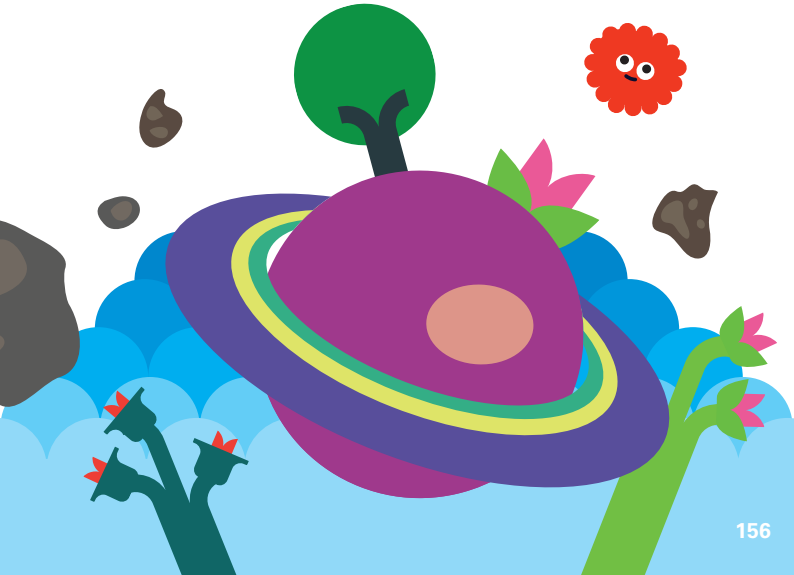
- Simple sensations are often the most effective. It is sometimes surprising to realize that something like a very simple Pop or Click sensation can enhance menu interactions and increase user confidence within the application. Simple is often very satisfying. A designer does not have to make wildly complex effects to make a compelling user interface.
- Sensations that fit with graphical and sound content in time and space make the whole greater than the sum of its parts. Seeing, hearing, and feeling an object or activity, where the visual, audio, and tactile parts are well-synchronized in time and space, and where each part is as realistic as possible, promotes sensory harmony, and draws a user into the presentation in a way just seeing and hearing alone cannot. In contrast, poor synchronization of touch feedback content with other content in time can actually be distracting to a user and detract from the experience. A little care and planning go a very long way in making an exceptional final product.

- It's 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 or other occasional attention-grabbing effect, continuous high-pitched buzzing that occurs frequently will eventually cause a user to leave the site or application, annoyed. Remember the BLINK tag in HTML? It's not used much anymore. Just because you can do something, doesn't mean you don't have to be careful with how or where you do it. Preventing a negative user experience is primarily a matter of thoughtful interface design.
- It's 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. Each individual sensation will lose its impact, and the user will get tired quickly. Haptic sensations should be placed and spaced so each one has the intended impact on the user and doesn't blur into the other sensations (unless, of course, this is the intended effect). At the same time, too few or too sparsely spaced effects will disrupt the haptic continuity of the presentation. A decision to create a haptic interface requires a commitment to incorporate enough touch sensations so as not to alarm the user.



- 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. Don't use the power up effect for ending an application for example. 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.
- The platform may include haptic effects, for instance the KickBack Accessibility Service[1] provides haptic feedback. So, consider how haptic effects generated by your application may interact with, or disturb, such services.

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



Basic Vibration Control on Android

The Android platform allows you to set vibration motor-on and motor-off durations. This method allows you to create short click-type effects for use with UI confirmations or longer alert-type effects. Both types of vibe effects are played at full motor voltage, or full amplitude. The effect may vary because of physical differences across Android devices, such as different vibration motors. For consistency of your vibe playback across multiple devices, like handsets vs. tablets, developers may want to compensate for these differences themselves by setting different vibration durations in code for each device they wish to optimize for. This is not a major issue for long duration alert-type vibe effects but is a big issue for short duration click-type effects. You may find a short vibe duration for one device cannot be felt but feels too strong and annoying on another.

Because you can only control the duration when using the core Android vibration control, and all vibrations are at full amplitude, you can run into the risk of increased battery consumption for the device. Having control over the amplitude and frequency, as well as the duration, will help decrease battery consumption.

Imagine you have a racing game where you want to feel the engine of your vehicle. The basic control is unusable because the motor will simply play at 100% amplitude and would annoy the user if you use the vibration for any length of time. You cannot simulate the kind of acceleration/deceleration vibe effects you might find in a console gaming experience with the basic vibration control.

Extending Basic 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's MOTIV Haptic SDK allows full vibration motor control of duration, amplitude and pulsing frequency with a library of pre-defined vibration/haptic effects. With this type of control, application developers have the capability of designing vibration effects rivaling console gaming vibration experiences. It also allows for smarter use of frequency level, taking up less battery life would be a better solution for extended vibration effect features. Immersion also offers haptic solutions for other mobile platforms including Windows and bada. You can find more information at *http://developer.immersion.com*.

Implementing Security Measures

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.

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 don't 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 ProGuard[1] and Arxan's GuardIT[2], 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.

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 isn't 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 set-

1 proguard.sourceforge.net
2 www.arxan.com

ting for these compilers is to prepare every function to be exported from a shared object, and add it to the dynamic symbol table in the binary. The dynamic symbol table is different to the symbol table used for debugging and is much harder to strip after compilation. Dumping the dynamic symbols can give a hacker a very helpful index of every function in the native code. Using the –f visibility compiler switch[3] correctly is an easy way to make it harder to understand the code.

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-compiler[4]. 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.

3 gcc.gnu.org/wiki/Visibility

4 www.hex-rays.com

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.

Protection Tools

Basic Java code renaming can be done using Proguard[5], 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 Technologies[6] and PreEmptive Solutions[7].

For native code protection tools and white-box cryptography libraries, the main vendors are Arxan and Irdeto[8].

5 www.proguard.sourceforge.net
6 www.arxan.com
7 www.preemptive.com
8 www.irdeto.com

Resources

Here are some useful resources and references which may help you. Apple provide a general guide to software security[9]. It also includes several links to more detailed topics for their platform. Commercial training courses are available for iOS[10] and Android[11] and O'Reilly have published a book on Android security[12].

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.

9 www.securecoding-iphoneapps.com/
10 marakana.com/training/android/android_security.html
11 developer.apple.com/library/mac/navigation/#section=Topics&topic=Security
12 Jeff Six: Application Security For The Android Platform. Processes, Permissions and Other Safeguards (Dec 2011), available at shop.oreilly.com/

Testing Your Application

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.

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 doesn't work as hoped.

Headless Client

The user-interface (UI) of a modern mobile application can constitute over 50% of the entire codebase. If you limit your testing to testing using the GUI designed for users you may needlessly complicate your testing and debugging efforts. One approach is to create a very basic UI that's 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.

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.

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 Mobile Test Automation tools are not capable of allowing TDD for all aspects of a mobile application, we have seen it used successfully on a variety of mobile projects, particularly when used for the generic aspects of the client code.

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're out and about. It's 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.

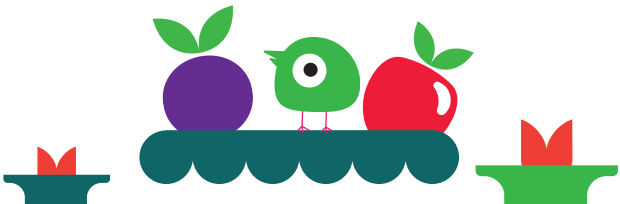
For platforms such as Android and Java ME where there are so many manufacturers and models, it's particularly useful to test on a range of these devices. A good start is to pick a mix of popular, new, and models that include specific characteristics or features such as: touch screen, physical keyboard, screen resolution, networking chipset, etc. 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 don't, 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.

These days many of the manufacturers provide this service free-of-charge for their new and popular phone models to registered software developers. You can also use commercial services of companies such as PerfectoMobile or DeviceAnywhere for similar testing across a range of devices and platforms.

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.



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.

Commercial companies tried to provide automated testing "solutions"; however several have been mothballed. In comparison, there are several open source cross platform tools: Mobile End-to-end Testing (MOET)[1] supports Android, BlackBerry and iPhone devices with a consistent set of commands which can be used interactively or automated. Google have released the Native Webdriver open source project[2] which can test native applications for Android, iOS and Windows Phone. Also, Tampere University in Finland have had some success creating automated tests for various mobile platforms, including Android and Nokia's S60 phones[3].

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.

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

1 github.com/eing/moet
2 code.google.com/p/nativedriver/
3 tema.cs.tut.fi

quickly and relatively inexpensively, compared to maintaining a larger dedicated software testing team.

These services can augment your other testing, we don't 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, etc.

Web-Based Content And Applications

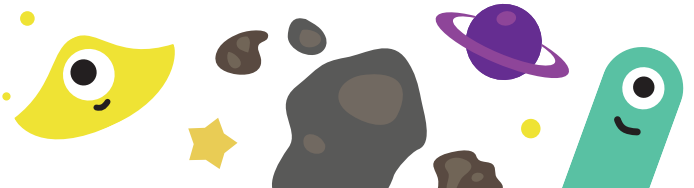
We can benefit from the extensive history of test automation tools for desktop web-based content and applications to automate aspects of our Mobile equivalents.

Tools such as WebDriver wrap web browsers, including, headless WebKit, Android, iPhone, Mobile Opera, and BlackBerry as well as the main desktop web browsers.

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's an article on the Google Testing blog[4] that includes an example of how to emulate the iPhone browser[5].

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 Mobile[6].

4 googletesting.blogspot.com
5 googletesting.blogspot.com/2009/05/survival-techniques-for-web-app.html
6 www.opera.com/developer/tools/



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?

Developers claim a lack of monetization options is one of the biggest hurdles they face in entering the mobile app market. The 2011 Developer Economics report suggests the issue is one of innovation: the high pace of technological innovation is not matched by monetization innovation[1]. 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 ads:** Earn money from advertising.
4. **Indirect sales:** Affiliates, data reporting and physical goods among others.
5. **Revenue sharing:** Earn revenue from operator services originating in your app.

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.

1 www.visionmobile.com/developers

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.

Recently, 57 of the world's largest mobile phone network operators and manufacturers founded the Wholesale Applications Community (WAC)[2], a not for profit organization that helps to standardize the mobile applications ecosystem. It is expected that pretty soon developers will only need one account with WAC instead of one with each operator.

The alternative is to use a mediator that can do this for you. Each operator will take a revenue share typically 45% to 65% of

2 www.wacapps.net

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

Among US operators, T-Mobile USA and AT&T both allow Android[3] and Nokia Store users to purchase apps and bill them directly to their mobile bill. Android made a major gain in its European markets recently when Vodafone and Deutsche Telekom introduced direct operator billing for Android Market customers in European markets[4]. This enables end users to purchase apps and games without the need for a credit card. In addition to customers on monthly billing arrangements, pre-pay customers can use their credit 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.

The developer API model is strategically important for developers, as it enables them to build monetization right into the heart of their applications. For example TextDeck[5] is a Mac OS app, available from the Mac OS App Store. The app allows Mac users to send SMS messages direct from their desktop using the BlueVia SMS API. Every time a user sends an SMS message using TextDeck, Telefónica bills the end user for the cost of the message, and then revenue shares back with the developer of TextDeck.

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

3 *Android developer blog on more payment options:* android-developers.blogspot.com/2011/09/more-carrier-billing-options-on-android.html

4 *Deutsche Telekom press release:* www.telekom.com/media/consumer-products/69850

5 glimmerdesign.com/textdeckpro

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.

The last option is to create your own website and implement a payment mechanism through that, such as PayPal, PayPal mobile, credit card (not supported on all devices), dial-in to premium landlines[6], or similar.

Using PPD can typically be implemented with no special design or coding requirements for your app.

Overall, we would recommend starting with an app store as it involves minimal setup costs and administrative overhead.

6 www.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/O2 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 *www.Smaato.net*, *www.Madgic.com* or *www.inner-active.com* 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 20% to 30% on top of that.

If your app is doing really well and has a large volume in a specific country you might consider selling ads directly to advertising agencies or brands (Premium advertising) or hire a media agency to do it 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.

Similarly, in application advertising will require you to design

and code your application with it in mind. Also, the placing of adverts needs to be considered with care. 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. Currently only offered by BlueVia for services delivered over its connected networks in 7 countries, the revenue sharing model 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

The final 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 *www.mobpartner.com*. This can be considered a variation on mobile advertising.
2. **Data reporting:** Track behavior and sell data to interested parties. Mobile radio applications often use this

business model. 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.

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 fan websites and such like. Nokia provides a particularly useful page of information on marketing your apps[7].

For a comprehensive introduction into app marketing for developers, check out "Developer's Guide To The Parallel Universe" available from *www.wipconnector.com* and at WIPJam events.

7 www.forum.nokia.com/Distribute/Public_relations_guidelines.xhtml

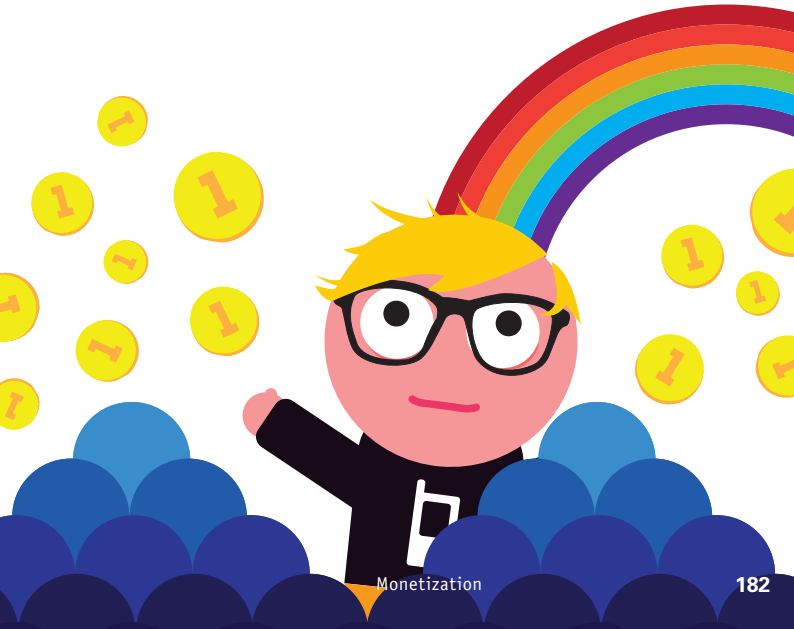
Strategy

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 when you have more than 100 thousand users worldwide or split the app into free and paid versions.
- 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.

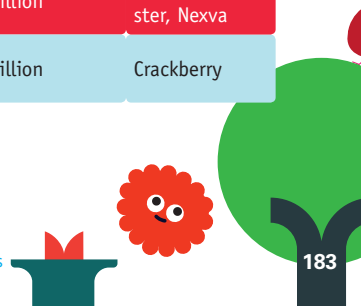


Appstores

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 stand out from the crowd and be successful, leading many to compare the chances of appstore success to the odds of winning the lottery. So, here are a few tips and tricks to help your raise your odds.

Top 5 Appstores

Appstore	Platform	Daily Downloads	Alternatives
Apple iTunes App Store	iOS	>31 million	Cydia (Jailbroken iPhones)
Android Market	Android	>22 million	GetJar, Samsung, Motorola, Amazon and 50+ others
Nokia Store	Symbian, Qt, Widget, Java	>9 million	GetJar
GetJar	Java, Symbian, Android, widget	>3 million	Appia, Handster, Nexva
Blackberry App World	BlackBerry	>3 million	Crackberry



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 achieve[1]. Also, it's worth noting that downloads don't 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 Distimo[2]. 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're not going to market your app for you. You can't rely on the app stores to pump up your downloads, unless you happen to get into a top-ten list. But don't play the lottery with your apps, have a strategy and plan to market your app.

We have asked many developers about the tactics that brought them the most attention and higher rankings in appstores.

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/

Many answers came back and one common theme emerged: there's 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 op-

portunity 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?

Unfortunately there is no definitive answer, we wish there was. So, the short answer is: It depends.

However, you can still find the best solution to your need, but it is a longer answer: think about your target region, the market share achieved by each technology, define your business model, your target users, their needs, their devices and data plans. Then consider your vision and the requirements for your application as well as your existing technology skills.

The Business Perspective: Market Reach

While smartphone platforms take the lion's share in terms of the sheer number of apps, downloads and the other goodies that make up what we call the platform's ecosystem, we are still in a feature-phone dominated world. Smartphone penetration is now at only 27%, with Asia Pacific, the US and some European countries being the largest markets for smartphones.

Many developers choose to focus on feature phones, banking on the large number of devices out there. Fragmentation issues aside, the most widely used platform for feature phones is Java ME – a platform compatible with nearly 80% of feature phones.

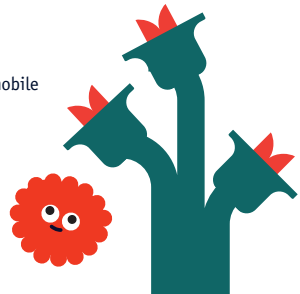
In the smartphone platform race, the trend is strongly in favor of Android and iOS, with traditional players like RIM and its BlackBerry phones losing ground.

As of the second quarter of 2011, the top smartphone platforms in terms of shipments were (note that this table excludes feature phone platforms, which make up the majority of sales):

Platform	Market Share	Shipments
Android (Google)	48%	52 million
iOS (Apple)	19%	20 million
Symbian (Nokia)	18%	17 million
BlackBerry (RIM)	12%	12 million
bada (Samsung)	1%	2 million
Windows Phone (Microsoft)	1%	1,5 million

Keep in mind: The regional differences in market shares are huge. To find out about market share in your target region, check out online resources such as comscore[1], StatCounter[2], VisionMobile[3] or Gartner[4].

1 www.comscoredatamine.com/category/mobile
2 gs.statcounter.com
3 www.visionmobile.com
4 www.gartner.com



However, a big market share of a certain technology does not automatically mean that you can make a lot of money by offering apps for that platform. There is a large discrepancy between the number of devices shipped and the number of available apps on some platforms.

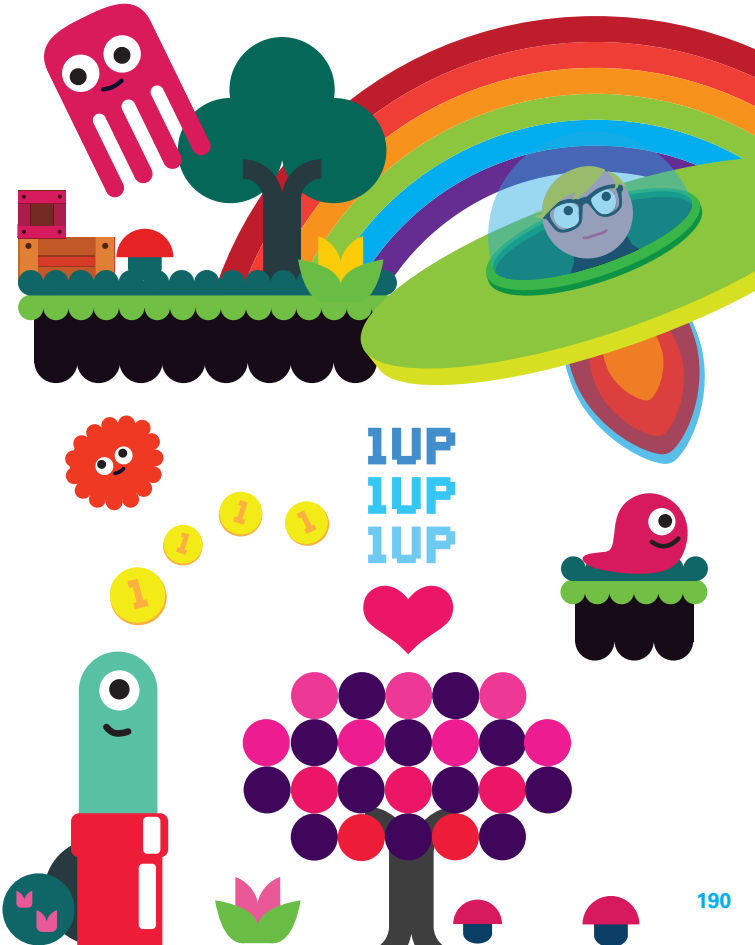
At one end of the spectrum, Java ME has been shipped in over 3 billion devices since its launch, but the number of available apps is estimated at around 22,000. Similarly, Nokia's feature phone OS, Series 40, has been shipped in over 1.6 billion devices, but the number of apps available for the platform is around 27,000 (which includes Adobe Flash applications) . Feature phones are often used principally as telephones and SMS devices, it is less common for people to install apps on their feature phone or even pay for mobile software, although there is evidence that this is changing.

At the other end of the spectrum, Android has 400,000 available apps, while it has been shipped in approximately 250 million devices[5]. Similarly, iOS has over 500,000 apps available although Apple has shipped "only" around 200 million mobile devices and applies a strict app approval policy.

Always remember that you are not necessarily restricted to a single application environment. It often makes sense to combine different environments, for example by providing a mobile website for your casual users, a native smartphone application for power users and a Java ME app for regions where smartphones take a smaller market share. If you want to concentrate on smartphone platforms only, you may need to adjust your application concept: On iOS you might make money by direct app sales or in-app purchase (if you somehow manage to gain visibility among the vast number of apps). For Android this approach may not work, Android users are less likely to pay for mobile

5 www.asymco.com/2011/12/21/how-many-android-phones-have-been-activated/

software, therefore in-app advertising could be the better choice for generating revenue. Please see the “Monetization” chapter to learn more about your options.



The Developer's Perspective: Technology

Aside from the marketing aspect of each platform, such as the relative strength of the ecosystem, there are also technical factors to consider. One of the most important is the ease with which a developer can master a platform. Many developers will not want to invest time in a platform that takes many long months to properly master. According to the Developer Economics research, Symbian and Java ME are the hardest platforms to master, while Android and BlackBerry are the easiest. The table below presents the average time in months required to master each platform, as identified by the respondents of this research:

Platform	Approximate time to master
Android	5.7 months
BlackBerry	6.1 months
iOS	6.8 months
mobile web	8.7 months
Symbian	9.8 months
Java ME	10.6 months

A recent VisionMobile research on mobile development[6] introduced a new metric for a platform's success. The Developer Mindshare is the percent of developers who have recently worked on a given platform, irrespective of the main platform they spend most of their time on.
Here are the top-5 platforms in terms of Developer Mindshare:

Platform	% of developers developing for it
Android (Google)	67%
iOS (Apple)	59%
mobile web	56%
Java ME	46%
BlackBerry	45%

Android and iOS are the clear winners in terms of mindshare, mirroring the traction these two platforms have. However, the presence of mobile web demonstrates the increasing importance of web apps in the mobile industry, as well as an influx of non-mobile developers.

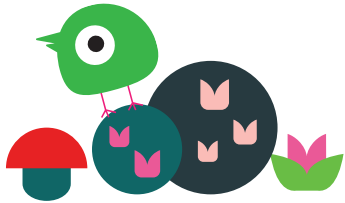
6 www.DeveloperEconomics.com

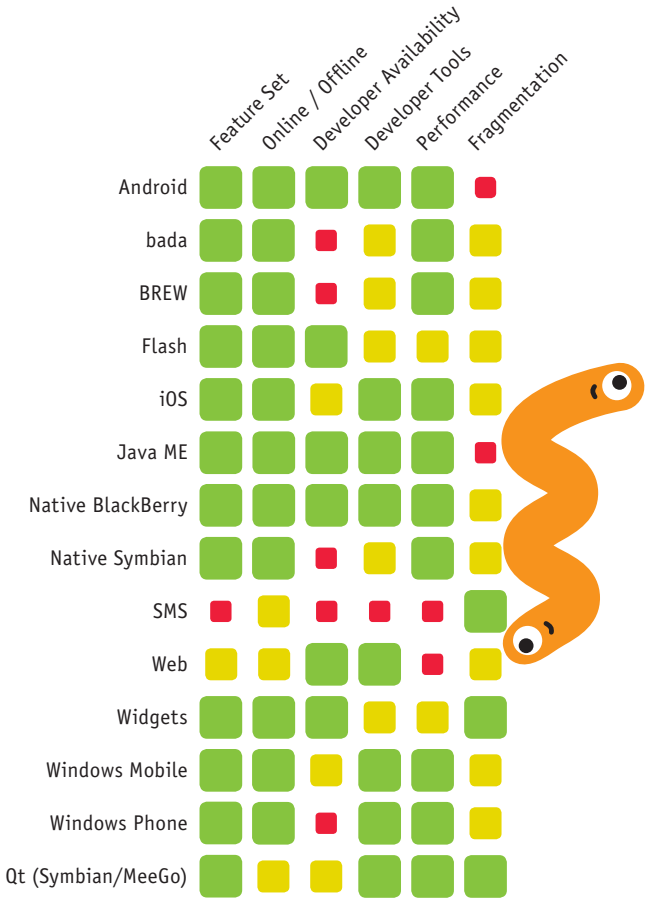
The research also investigated which are the top platforms being abandoned developers:

Platform	% of participants abandoning it
Symbian	39%
Java ME	35%
Palm OS	28%
BREW	28%
webOS	19%

These two metrics demonstrate how developers are abandoning older, more cumbersome platforms (Java and BlackBerry) and flocking towards newer platforms (Android, iOS, Qt and Windows Phone).

Other important aspects of each platform, from a developer's point of view, are the size of the developer community, the quality of developer tools, the range of APIs, possible dependencies on internet connectivity and performance or fragmentation issues:





Green indicates good coverage or support, **yellow** for limited and **red** for bad coverage of the respective topic

Epilogue

Thanks for reading this 10th edition of our Mobile Developer's Guide. We hope you have enjoyed reading it and that we helped you to clarify your options.

You can also order free hardcopies of this publication on our website: **www.enough.de/products/mobile-developers-guide.**

If you like to contribute to this guide or sponsor upcoming editions, 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

As a graduate of the University of the Arts Bremen, Andrej focuses on UI, UX and visual design for mobile applications and other interactive technologies. He is also in charge of the layout and design of this guide. When not involved with something mobile, he loves to experiment with digital art and illustration.
www.enough.de

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.

Winston Bond / Arxan Technologies

Winston has 20 years experience developing software on a wide variety of embedded, desktop and mobile platforms. Since 2008, he has been focused on helping businesses, from large banks to emerging digital media companies, protect apps against tampering and reverse engineering for unauthorized use, piracy and theft of intellectual property. Winston is European Technical Manager for Arxan Technologies Inc., a leader in application security solutions and the fight against cybercrime.

www.arxan.com

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. Since then his mission has been to expand the WWW world so it is usable on mobile devices. He is responsible for the development of the FITML platform which enables developers to create mobile internet portals and answer the challenge of device fragmentation. Roland is an active member of the Mobile Web Initiative (MWI) and participates in various open source projects.

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

Julian Harty / eBay

Hired by Google in 2006 as the first Test Engineer outside the USA and told he was responsible for testing Google's mobile phone 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. The material is also freely available at *tr.im/mobtest*. He continues to work on test automation for mobile phones and applications. He now works for eBay where his mission is to revamp testing globally.

www.ebay.com

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's been undertaking 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.ovidiuiliescu.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

Matos is Marketing Manager at VisionMobile, a leading industry analyst firm. He is responsible for VisionMobile's marketing, communications and PR, and he has also taken on the mantle of the company's blog editor and is a regular contributor. As the lead in the company's data gathering efforts and the Developer Economics projects, Matos is well-acquainted with the facts and figures of the mobile industry.

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. He is passionate about UI, UX and development in general for Android. Furthermore he loves speaking at conferences, writing articles and all kind of Social Media.

www.mobile.neofonie.de **www.sera-apps.de**

Suzanne Nguyen / Immersion Corp.

Suzanne has been a part of the mobile ecosystem, driving innovation in the mobile space for over 15 years. She was one of the first evangelists for Java ME and the originator of the Java ME developer program. Suzanne was a key driver of growth in the Android community, through T-Mobile's developer community support. As director of developer marketing at Immersion, she heads the effort to educate and increase awareness of MOTIV technologies on the Android Platform.

www.immersion.com

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

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 any kind. He mainly develops for J2ME, 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

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:

ENOUGH SOFTWARE
www.enough.de

wireless industry partnership
www.wipconnector.com

Printing sponsors:

NOKIA Connecting People

BlueVia
www.bluevia.com

immersion

developer garden
powered by T

A non-commercial, community-driven overview on mobile technologies for developers and decision-makers.

> 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."
> *Daniel Hudson, www.webtechman.com*

> Extremely helpful content, also for non-developers. And the design is nothing but fantastic!
> *Monika Lischke, Community Manager, Intel AppUp developer program*

> Impressive. The most comprehensive and concise guide to developing for mobile.
> *Andreas Constantinou, Founder & MD, VisionMobile*