

Don't Panic

MOBILE DEVELOPER'S

GUIDE TO THE GALAXY



published by:



Services and Tools for All Mobile Platforms

Enough Software GmbH + Co. KG

Sögestrasse 70

28195 Bremen

Germany

www.enough.de

Please send your feedback, questions or sponsorship requests to:

developers@enough.de

Follow us on Twitter: *[@enoughsoftware](https://twitter.com/enoughsoftware)*



9th Edition October 2011

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

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

Introduction

The mobile galaxy is still evolving. Since our last edition Google purchased Motorola Mobility, HP discontinued webOS, MeeGo will be replaced by the new Linux-based Tizen OS and of course new SDKs came out for iOS, Qt, Windows Phone, bada and other platforms. So we have updated the whole guide for you once again. We have added new content on topics such as accessibility, operator billing and market shares. And: we kicked out the chapter about webOS, since the future of the platform is unsure. You will neither find a MeeGo chapter anymore – maybe we'll have a Tizen chapter in the 10th edition?

However, those of you who used to develop webOS or MeeGo apps will probably appreciate this book even more now: It gives you an objective overview on all the remaining mobile platforms. So read through the pages to find out what your next playground will be! If you are still afraid of making the wrong decision and taking a dead-end road, you are probably thinking about cross-platform programming. If this is the case, then refer to the dedicated chapter right away and learn more about multi-platform solutions and the limitations of this approach.

If you are not a developer but a decision maker who is wondering how to enter the mobile arena, this book is also a good entry point. You may be asking yourself if you should spend your budget on an iOS app or a mobile website? You want to know what your programmers are talking about when they are complaining about fragmentation and too many platforms? If so, you will find the answers in the pages that follow.

Whatever is your role in the mobile galaxy, please continue spreading the word about this project or even better, get involved as a writer or a sponsor!



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 Tablet OS (QNX), iOS, Symbian, 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 non-smartphones 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 Sony Ericsson Java Platform phones.

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 OS (QNX)	ActionScript, C++, HTML, CSS, JavaScript	Java announced blackberry.com/developers
iOS	Objective-C, C	Requires Apple Developer Account developer.apple.com/iphone
MeeGo	Qt, C++, others	Intel and Nokia guided open source OS (will be replaced by Tizen) meego.com/developers
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 widget style programming, (based on Linux), probably dead since it has been abandoned by HP developer.palm.com
Windows Mobile	C#, C	.NET CF or Windows Mobile API, most devices ship with Java ME compatible JVM developer.windowsmobile.com
Windows Phone	C#, VB.NET	Silverlight, XNA frameworks create.msdn.com



Java ME (J2ME)

Around 80% of all mobile handsets 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, for many emerging economies it remains the main option to target the mass market.

Flash

Historically, Flash Lite was the mobile edition of Flash, an older version of Adobe's web Flash product with ActionScript 2.0 support. Adobe is phasing out Flash Lite for mobile and simply using the full version of Flash.

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. However, all these things are improving: There are millions of feature phones supporting Flash today and many smartphones and tablets can support some Flash content including MeeGo, 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 Qualcomm¹.

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 code².

Widgets and Web Apps

The main advantage of widget environments is they offer simple, straightforward programming based on web markup and scripting languages.

There are, however, several widget environments and some require a player to be installed. This situation is changing, with a trend towards standardization, based on W3C standards. The move to standard web technology based widgets is alleviating the main drawback of widgets: lack of integration with the underlying platform. The standards-based environments are increasingly offering APIs that enable widgets to access device data, such as location or contacts, among others. All these environments use XML, a script language (usually Java Script) and a page description language (usually HTML) to realize a widget.

1) www.brewmp.com

2) developer.brewmp.com



This table provides an overview of popular widget frameworks:

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

Websites

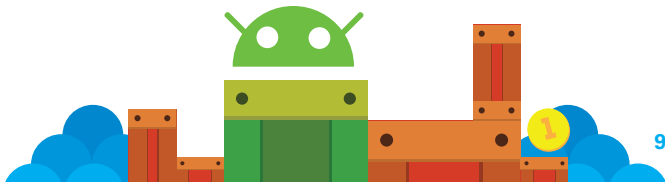
The browsing of web pages is supported by most phones, so in principle this should be the environment of choice to get the widest possible reach (after SMS text messaging). However, the sheer number of browsers and their varying feature sets can make this approach challenging. Some browsers are very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

With the introduction of HTML5 and new mobile browsers that support its features, this situation is improving: Offline browsing and new device APIs are now becoming available for mobile websites, such as location information in the Opera Mobile browser or Nokia Browser for N9. The main benefits of the mobile web as a platform are the ease of development and that, generally, you control the deployment.

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, for example its use for payments is common in these markets.



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. Since the platform is supported by many hardware manufacturers, it is the fastest growing smart-phone operating system. During the second quarter of 2011, more than 50% of all smartphones shipped in the US were based on Android¹. Although the acquisition of Motorola's handset business by Google in late 2011 could affect the platforms use by other manufacturers, it is not expected to. Android is also used in tablets, media players, set-top boxes, desktop phones and car entertainment systems. Some non-Android devices are also able to run Android Applications, such as RIM's Playbook with its virtual machine called App player².

The platform continues to evolve rapidly with the regular addition of new features, every 6 months or so. For example, Android 2.3 (code named "Gingerbread") introduced NFC and VOIP communication, better game development and a much more.

Android 3.0 ("Honeycomb") has been designed with deployment on tablets and other devices with larger screens in mind.

With the next Android OS version ("Ice Cream Sandwich"), Google is bringing together the separated tablet and telephone systems and introducing many new features.

1) www.npd.com/press/releases/press_110822a.html

2) www.theregister.co.uk/2011/03/25/rim_playbook_android/



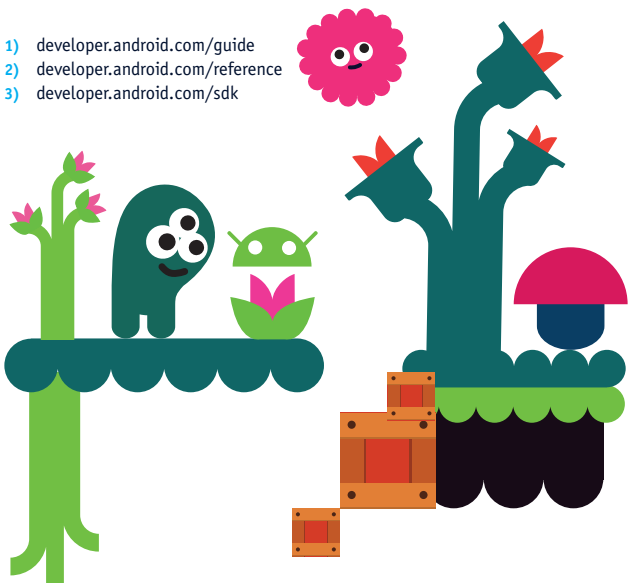
Prerequisites

The main programming language for Android is Java. But beware, only a subset of the Java libraries is supported and there are lots of platform specific APIs. You can find answers to your What and Why questions in the Dev Guide¹ and to your How questions in the reference documentation².

To get started, you need the Android SDK³, 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 a good choice. 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.

- 1) developer.android.com/guide
- 2) developer.android.com/reference
- 3) developer.android.com/sdk



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 background 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.

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` flags¹.

If you are going to use Honeycomb related layout features for large screens like Fragments², 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 Android 1.6³ to Android 2.3 deployment.

1) developer.android.com/reference/android/Manifest.permission.html

2) developer.android.com/guide/topics/fundamentals/fragments.html

3) 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 platform¹. Therefore, testing on a physical device is paramount.

Automated Testing

To automate testing, the Android SDK comes with some capable and useful testing instrumentation² 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 Robotium³ can complement your other automated tests. Robotium can even be used to test binary apk files, if the source is not available. A maven plugin⁴ and a helper for the continuous integration of a Hudson server may also assist your testing⁵.

1) For an overview see e.g. www.androidfragmentation.com

2) developer.android.com/guide/topics/testing/testing_android.html

3) code.google.com/p/robotium

4) code.google.com/p/maven-android-plugin/

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

Signing

Your application will always be signed by the build process, either with a debug or release signature. You can use a self-signing mechanism, so you can avoid signing fees (and security). The same signature is required for updates to your application.

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 new exciting 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 groups¹.

Android 1.6 upwards also supports in-app purchase. This allows you to sell extra content, feature sets and such like from within your app, using the existing infrastructure of the Android Market².

1) www.wipconnector.com/index.php/appstores/tag/android

2) developer.android.com/guide/market/billing/index.html

Programming bada Apps

bada is Samsung's proprietary Smartphone platform, it 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 intention in introducing bada, was to add a new platform to the existing market to accommodate the anticipated need for smartphone features in low-end market. Because bada can run either on top of a Linux kernel, for high-end devices, or on real-time OS kernels, for low-end devices, all market segments can be served.

To reach their target, Samsung puts a high priority on developer support and training. There is a huge documentation and guide library available from the developer site . 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, advertisement and a direct feedback channel for customers. The store is accessible to customs by three methods, the website *www.samsungapps.com*, a client application on bada smartphones and a PC client called Samsung's Kies.

Samsung's app store is available in over 120 countries worldwide and had over 100 million downloads in its first 10 month; the store was launched in June 2010. Applications can be offered as paid apps or free. It is possible to generate revenue by placing advertisements in your apps too. 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. Currently there are six bada-based devices available with the Wave 3 being the current flagship device and Wave 578, the first device with NFC available running bada 2.0 OS.

Getting Started

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

In September 2011, Samsung released bada 2.0. One of the key features is multitasking, which makes real background services possible. In addition, bada 2.0 supports Near Field Communication as well as the possibility for easy ad-hoc WiFi-P2P network setup from within the SDK.

Other interesting features include enhancements to the UIX with speech-to-text (STT) and text-to-speech (TTS), as well as support for 3D sound with OpenAL. Support for web-based applications is extended, with support for more JavaScript frameworks, HTML5 and a lot of API's 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.

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

Implementation

After creating an application manifest you can start with app development using bada SDK/IDE. The IDE has a plentiful library of example code, which 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.

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 are 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 variables 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. A UI layout for the landscape and/or the portrait mode is also the 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.

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 allows for testing and debugging in the simulator or on a device. As mentioned earlier in this guide, we strongly recommend testing on real devices. Without device testing you cannot be sure how the app will perform on a device 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 Tu-

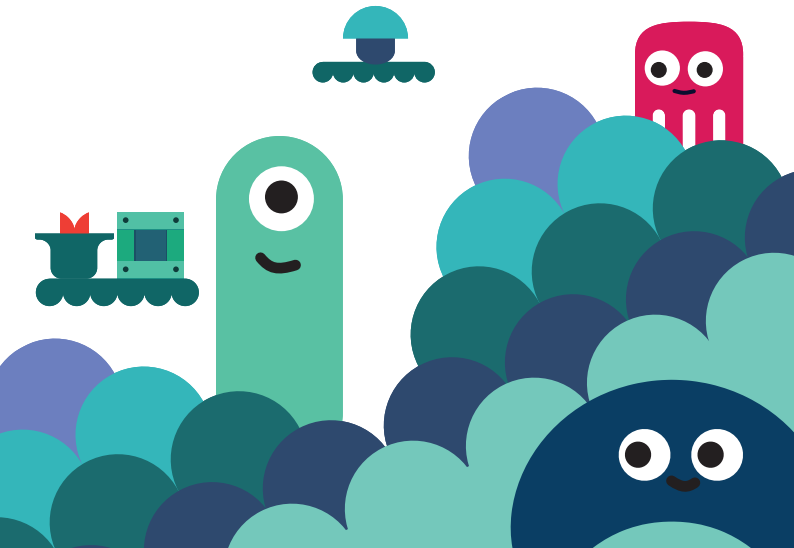
torial.Development Environment.pdf” included in the documents folder in the SDK base directory.

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

Distribution

App distribution is through Samsung’s apps store and it is the only distribution channel. As with Apple’s App Store, there are quite strict acceptance rules for apps submitted and you can find out more in the “Samsung Apps Publisher Guide”. The guide 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 allows the inclusion of third party ad network contents in bada application.



Programming Native BlackBerry Apps

The BlackBerry platform was developed by the Canadian company Research In Motion (RIM)¹ and launched in 1999. BlackBerry devices became extremely popular because they were equipped with a full keyboard for comfortable text input (which spawned a condition named BlackBerry Thumb²), their long battery life and more and more for 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 2011³, but it is still an important smartphone platform. While the general consensus seems to be that BlackBerry tablet, the PlayBook with its QNX OS has been launched too early, the hardware and OS are highly praised.

Prerequisites

RIM currently supports two platforms/operating systems. The first is BlackBerry OS. This is the operating system found on all current BlackBerry smartphones. Its latest iteration (BlackBerry OS 7) offers a multitude of new features over its previous versions. Besides improved browsing support, new APIs are now available for:

1) www.rim.com

2) en.wikipedia.org/wiki/Blackberry_thumb

3) gs.statcounter.com

- NFC
- Compass and positioning
- Video capture
- Window management
- Searching
- Integration with the operating system
- and more

In addition the platform supports OpenGL ES 2.0. These additions make the platform very competitive from a technical standpoint.

For the 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 first choice. Small apps can also be developed with the BlackBerry WebWorks SDK.

RIM's next generation operating system is based on QNX Neutrino Realtime OS (RTOS). RIM calls it Tablet OS and it is currently only supported on their PlayBook (QNX based smartphones have been announced for Q1 2012). For now, applications should be written using Adobe AIR Flash and WebWorks. Native C and Java SDKs have also been announced, as well as an Android compatibility layer. The first regular BlackBerry smartphones running QNX are scheduled to arrive in Q1 2012, and QNX itself will replace the current BlackBerry OS in the long run.

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 SDK¹ (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 plugin², for previous BlackBerry OS versions you can also use the BlackBerry Java Development Environments (JDEs)³.

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

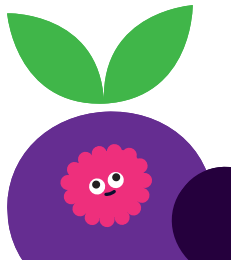
Desktop Manager

The BlackBerry Desktop Manager⁴ 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.

Coding Your Application

The BlackBerry JDE is partly based on J2ME 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 others⁵. This means that BlackBerry apps can be created using J2ME only technologies.

- 1) www.oracle.com/technetwork/java
- 2) us.blackberry.com/developers/javaappdev/javaplugin.jsp
- 3) us.blackberry.com/developers/javaappdev/javadevenv.jsp
- 4) us.blackberry.com/apps-software/desktop/
- 5) www.blackberry.com/developers/docs/6.0.0api/index.html



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 Polish¹ or Glaze² 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 services³.

The push service is useful mainly in mail, messaging or news applications. Its main benefit is that the device waits for the server to push updates to it, instead of the device continuously polling the server to find out if updates are available and then pulling the updates from the server. This reduces network traffic, battery usage and, for users on metered data plans or roaming, lowers costs.

The push service⁴ 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

1) www.j2mepolish.org

2) www.glaze-ui.org

3) us.blackberry.com/developers/platform

4) us.blackberry.com/developers/platform/pushapi.jsp

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 which allows 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.

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 BlackBerry¹. The signing itself is undertaken using the `rapc` tool which also packages the application.

Distribution

BlackBerry's own distribution channel is called App World² where you can publish your apps. For paid applications, you'll get a 70% revenue share. In addition, GetJar³ is a well-known independent website that also publishes BlackBerry apps.

1) us.blackberry.com/developers/javaappdev/codekeys.jsp

2) appworld.blackberry.com

3) www.getjar.com

Programming Flash Apps

Adobe Flash has become the ubiquitous platform for developing web-based applications, animations, and video. The tools are fairly easy to use and enable beautiful graphics, animation and audio to be packaged in a single, compact file that displays on any screen size. Flash is simply a file format that bundles bitmap images, video, audio, animations, and ActionScript into a single SWF file. It is one of the best ways to manage multimedia content in an application or for a web browser.

The commercial potential in using Flash for mobile app development is substantial, as it's a very well-known platform with over 3 million developers worldwide and it is already supported in a large number of devices. Many feature phones have support for Flash Lite (typically support for Flash 3, 6 or 8 depending on when the device was manufactured). Flash Lite is perfect for simple games such as puzzles and card games. Some smartphones and tablets have a Flash player pre-installed; Full Flash 10.x support has been announced for Android-based devices and RIM's BlackBerry PlayBook. For the iPhone, Adobe has released a packager that enables Adobe AIR applications to run on iOS devices.

Development of mobile Flash applications can be undertaken using Adobe products and alternative Flash-compatible SDK from a number of vendors. Flash brings the flexibility of a web browser user interface (UI) to mobile applications, allowing the developer to break free of a platform's UI constraints. Many developers are not aware of how easy it is to implement Flash in an application. Using Adobe AIR requires the entire application to be developed in Flash: It can be a daunting task to learn ActionScript and how to create animations. However, several Flash-compatible SDKs are available that enable the implementation of Flash content

directly as part of a native 2D or 3D mobile application, a consequence of this approach can be better application performance.

Prerequisites

Adobe open sourced the Flash specification, permitting independent developers and companies to develop Flash-compatible SDKs, engines and players. Authoring can be done 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 utilize 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 ECMA-Script 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. We anticipate that Flash support will become standardized on most mobile devices, as the hardware platforms include faster CPUs and GPUs with OpenGL ES graphics acceleration. But until then, you have to find a way to deal with this fragmentation. 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 application. An Adobe 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 Gnash¹ and GameSWF² 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

1) www.gnashdev.org

2) tulrich.com/geekstuff/gameswf.html

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 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.



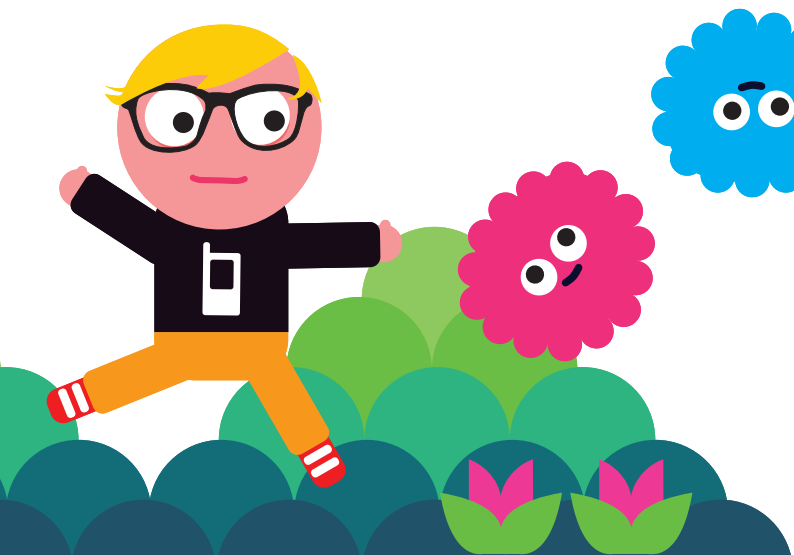
Testing

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 simple¹:

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 you².

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.

1) bit.ly/aqEmvv

2) esitv008song.itlase.com/sispack

Programming iOS Apps

The iPhone, along with the iPod touch and iPad, is an extremely interesting and very popular development platform, one commonly stated reason 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.

Users have downloaded more than 15 billion iOS apps, as of July 2011. 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. As of June 2011, Apple has paid developers over USD 2.5 billion since the launch of the App Store.

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 iPhone OS, 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 allows for easy cloud storage of application-specific data, documents and easy-to-implement Twitter functionality.

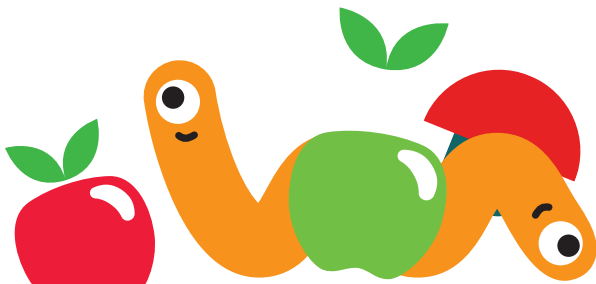
Prerequisites

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.



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 Kit¹, 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 compared to the official way of producing iOS apps. 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 outstanding quality. In most cases the cross platform tool concentrates on the lowest common denominator and the resulting product doesn't 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 which focus solely on iOS development, such as MonoTouch².

The platform allows developers to build iOS apps using C# and .NET while taking advantage of iOS APIs, making 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 is 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*¹. 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 workflow². Also check out some of the sample code that Apple provides online³ to find out more about various APIs available to you.

1) developer.apple.com/iphone/manage/overview/index.action

2) developer.apple.com/iphone/manage/distribution/distribution.action

3) 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 device available for 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 a 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 Portal¹. 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 instructions². Every single step is vital to success, so make sure that you execute them all correctly.

With iOS 4.0, Apple has introduced the possibility for developers to deploy Over-The-Air (OTA) Ad-Hoc builds of their apps

- 1) developer.apple.com/iphone/library/referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/index.html#//apple_ref/doc/uid/TP40007594
- 2) developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhone101/Articles/00_Introduction.html

to beta testers. There are open source projects¹ to facilitate this new feature, as well as commercial services².

Google Toolbox for Mac³ runs the test cases using a shell script during the build phase, while GHUnit⁴ runs the tests on the device (or in the simulator), allowing the developer to attach a debugger to investigate possible bugs. In version 2.2 of the SDK Apple included OCUnit; an example of how to create the unit tests is available online⁵.

In iOS 4.0 Apple introduced a new tool, UIAutomation which 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 Library⁶. Several other third party testing automation tools for iPhone applications are available as well, including FoneMonkey⁷ and Squish⁸.

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,

1) github.com/therealkerni/hockeykit

2) www.testflightapp.com

3) code.google.com/p/google-toolbox-for-mac

4) github.com/gabriel/gh-unit

5) www.mobileorchard.com/ocunit-integrated-unit-testing-in-xcode

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

7) www.gorillalogic.com/fonemonkey

8) www.froglogic.com/products

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 several rejections in the past, the approval process receives more complaints than any 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 estimate their app's success of being approved. Also, the restrictions were relaxed and apps which were previously rejected were approved after being re-submitted.

Approximate review times as experienced recently by other developers are gathered at reviewtimes.shinydevelopment.com for your convenience. 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

Over the past years, 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. A lot of iOS developers are very outspoken and open about what they do, and how they did certain things.

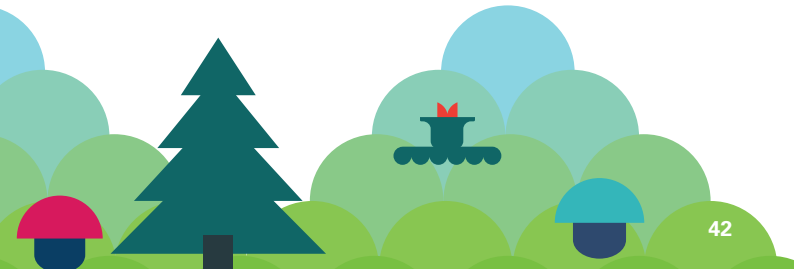
This is even more visible ever since Twitter and Github gained momentum and became widely-known.

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 which even allow commercial usage.

Practically all of the most important and most experienced iOS developers use Twitter to share their thoughts about the platform with others. There are many comprehensible lists of iOS developers out there, a notable and well-curated one being Robert Scoble's list¹. 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*

1) 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 80% of phones worldwide supporting it, J2ME is miles ahead of the competition in this regard. 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 is a great choice.

However, if your business model relies on direct application sales, or if your application needs to make use of state-of-the-art features and hardware, you might want to consider targeting a different platform (such as Android, BlackBerry, iPhone or 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 SDK¹ (not the Java Runtime Environment) and an IDE of your choice, such as Eclipse Pulsar for Mobile Developers², NetBeans³ with its Java ME plug-in or IntelliJ⁴.
- An emulator, such as the Wireless Toolkit⁵, the Micro Emulator⁶ or a vendor specific SDK or emulator.
- Depending on your setup you may need an obfuscator like ProGuard⁷. If you build applications professionally you will probably want to use a build tool such as Maven⁸ or Ant⁹ also.
- You may want to check out J2ME Polish, the open source

1) www.oracle.com/technetwork/java/javame/downloads/index.html

2) www.eclipse.org

3) www.netbeans.org

4) www.jetbrains.com

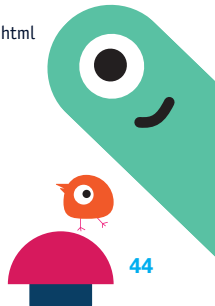
5) www.oracle.com/technetwork/java/download-135801.html

6) www.microemu.org

7) www.proguard.sourceforge.net

8) maven.apache.org

9) ant.apache.org



framework for building your application for various devices¹.

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).

Implementation

The Java ME platform is fairly straight-forward: it comprises the Connected Limited Device Configuration (CLDC)² and the Mobile Internet Device Profile (MIDP)³, 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.

1) www.j2mepolish.org

2) java.sun.com/products/cldc/overview.html

3) java.sun.com/products/midp/overview.html



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"¹.

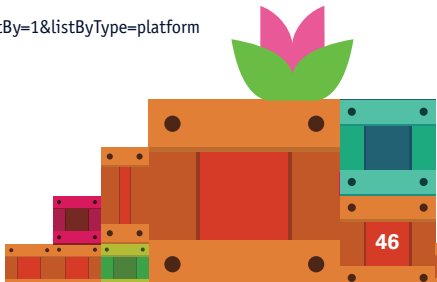
It is very important to remember that not all JSRs are available on all devices, so capabilities available on one device might not be available on another device, even if the two devices have similar hardware.

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 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.

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



Creating UIs

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

1. Highlevel LCDUI components: you use standard UI components, such as Form and List
2. Lowlevel LCDUI: you manually control every pixel of your UI using low-level graphics functions
3. SVG: you draw the UI in scalable vector graphics then use the APIs of JSR 226 or JSR 287¹.

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.

1. J2ME Polish²: 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. LWUIT³: A Swing inspired UI framework
3. Mewt⁴: Uses XML to define the UI
4. TWUIK⁵: A powerful “Rich Media Engine”.

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

2) www.j2mepolish.org

3) lwuit.dev.java.net

4) www.mewt.sourceforge.net

5) www.tricastmedia.com/v1/twuik.php

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: 240 x 320, 176 x 208, 176 x 220, 128 x 160, 128 x 128 or 360 x 640 pixels. By far the most popular is 240x320, while 360x640 is a common resolution for high-end Java ME devices (typically those running Symbian or Blackberry) and 176 x 208/220 is a common resolution for low-end devices. You will also encounter devices that have these resolutions in landscape, for example 320 x 240 instead of 240 x 320 pixels.

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

Another aspect worth considering is the size of your application's assets, especially its graphical assets. Whenever possible, your assets should be optimized, in order to keep your application's size as small as possible. This results in cheaper downloads for your users (as less data traffic is needed) and greater market reach (as some devices have a limit on the maximum application size). A great free tool for this is PNGGauntlet¹, 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.

1) www.pnggauntlet.com



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*¹, for example the Bluetooth project Marge².

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 and Samsung provide subsidized or even free remote access to selected devices³.

Automated Testing

There are various unit testing frameworks available for Java ME, including J2MEUnit⁴, MoMEUnit⁵ and CLDC Unit⁶; System and UI testing is more complex given the security model of J2ME, however JInjector⁷ is a flexible byte-code injection framework that supports system and UI testing. Code coverage can also be gathered with JInjector.

1) mobileandembedded.dev.java.net

2) marge.dev.java.net

3) www.forum.nokia.com/rda and innovator.samsungmobile.com

4) www.j2meunit.sourceforge.net

5) www.momeunit.sourceforge.net

6) snapshot.pyx4me.com/pyx4me-cldcunit

7) www.code.google.com/p/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.

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 Distillery¹ that is licensed per month, Bedrock from Metismo² that provides a suite of cross compilers on a yearly license fee and J2ME Polish from Enough Software³ 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 SDK⁴.

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

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

Signing

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

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

1) www.mobile-distillery.com

2) www.metismo.com

3) www.enough.de

4) www.mosync.com

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 JavaVerified¹, Verisign² or Thawte³ 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. Intertek⁴ is probably the largest such supplier.

Distribution

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

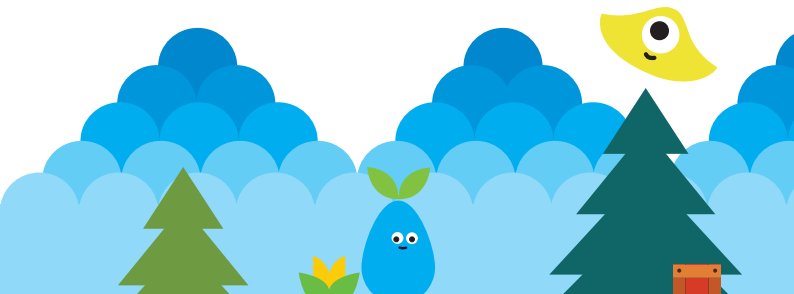
- 1) www.javaverified.com
- 2) www.verisign.com
- 3) www.thawte.com
- 4) www.intertek.com/wireless-mobile

- Handmark¹ and Mobile Rated² provide carrier and vendor independent application stores.
- GetJar³ is one of the oldest distributors for free mobile applications – not only Java applications.
- LG distributes apps on *www.lgapplication.com*
- Nokia Store⁴ 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 Orange⁵ and O2⁶.

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 Catalogue⁷.

Furthermore there are various vendors who provide solutions for provisioning of Java applications over a Bluetooth connection, including Waymedia⁸ and Futurlink⁹.

- 1) store.handmark.com
- 2) www.mobilerated.com
- 3) www.getjar.com
- 4) www.publish.ovi.com
- 5) www.orangepartner.com/site/enuk/mobile/application_shop/p_application_shop.jsp
- 6) mobileapps.o2online.de
- 7) www.wipconnector.com/apppstores/
www.waymedia.it
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 MeeGo (which will be replaced by the new Linux-based Tizen OS). 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.

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 pre-processor (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 objects¹; the output from one object

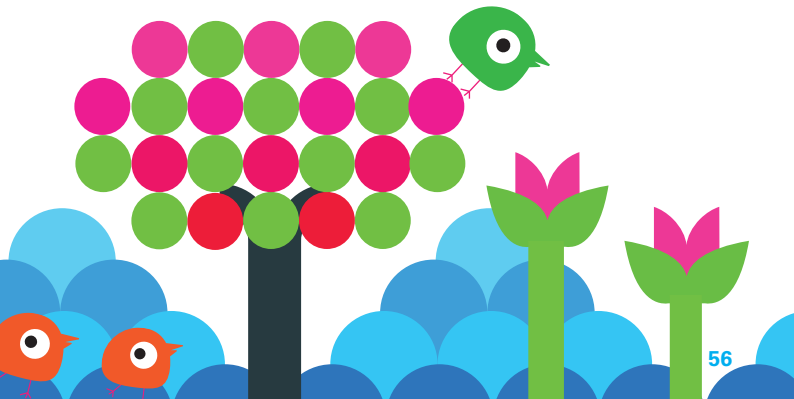
¹ doc.qt.nokia.com/4.7-snapshot/signalsandslots.html

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 the source code is made available over Gitorious¹.

1) qt.gitorious.org



Prerequisites

Qt SDK installs everything you need to create, test, and debug applications for Symbian and MeeGo from a single package. All versions offer tools for compiling Symbian and MeeGo 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 that offers full code completion support and integrated help. One of the neat features of Qt is QML¹, a language for declarative UI definition. While QML generally simplifies UI development, its biggest advantage is that the tools within Qt Creator enable the UI to be defined by graphic designers who do not have to be aware of the technical programming aspects.

In the past, one of the challenges with cross platform applications for mobile has been accessing platform features: Anytime you want to find the phone's location or read a contact record it has been necessary to revert back to the platform's native APIs². 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 and MeeGo phone. (The Qt SDK enables you to work with the native APIs if you want to, as it includes the Symbian APIs too.) As with Qt in general, working with the mobility APIs

1) qt.nokia.com/qtquick/

2) qt.nokia.com/products/qt-addons/mobility/

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 and MeeGo phones. 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 or MeeGo 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 cre-



ate your UI. Qt has many interesting features, such as WebKit integration – enabling you to include web content into your app – and scripting that can be used to add functionality quickly during development or change runtime functionality. It is also worth pointing out that, because Qt applications are compiled to the platform they will run on, they deliver very good performance, too. For most applications the levels of performance will be comparable to that previously achieved by hardcore native applications only.

Testing

Qt SDK¹ 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 and MeeGo phones. 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.

QTestLib² 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

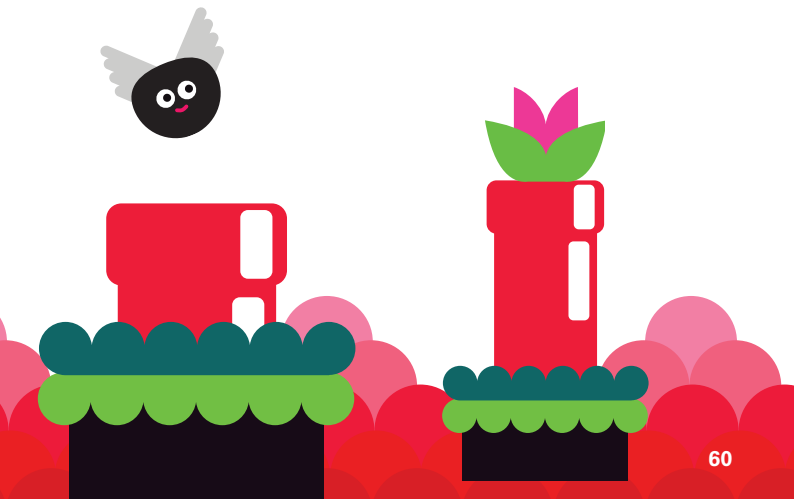
1) developer.nokia.com/Develop/Qt/Tools

2) 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. 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 and MeeGo phones they need to comply with each platform's signing requirements.

MeeGo 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. 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 MeeGo phones, 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 the process is straightforward and described in full in the Distribute section of the Forum Nokia website¹, but in summary:

- You sign up as a Nokia Publisher²
- 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 Criteria³
- Once tested, you submit an unsigned copy of the app to the Nokia publishing portal

1) www.developer.nokia.com/Distribute/Packaging_and_signing.xhtml

2) publish.ovi.com

3) www.developer.nokia.com/Community/Wiki/Symbian_Signed_Test_Criteria_V4_Wiki_version

Distribution

Nokia Store is the latest iteration of the Nokia app store solution, with a history stretching dating back to 2003. Achieving , and has grown to deliver 5 6 million downloads a day 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 is no restriction on the types of applications that can be distributed.

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

To use Ovi Nokia Store you need to register and pay a one-time €1 fee – registration is open to both companies and individuals.

When your application starts selling the revenue depends on the payment method chosen by the user:

- For credit card payments you get 70% of revenue after any applicable taxes and costs
- For operator billing purchases you get 60% of revenue after applicable taxes and costs.

While the reduced revenue from purchases made through operator billing may seem a disadvantage it usually is not. This is because operator billing is universal and trusted. 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. (If you really don't like the idea of losing the 10% margin, you can opt to sell apps through credit card purchases only.)



Programming Symbian Apps

The Symbian platform¹ 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 and Nokia forecasting additional sales in the region of 150 million.

It is, however, worth noting that Nokia recommend creating apps for Symbian phones using Qt rather than the platform's 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 idioms².

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%2B%2B

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 phones¹ and install it
2. Install Carbide.c++²

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 online³.

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.

- 1) www.forum.nokia.com/Develop/Other_Technologies/Symbian_C++/Tools
- 2) www.forum.nokia.com/Develop/Other_Technologies/Symbian_C++/Tools/
- 3) www.forum.nokia.com

Symbian/S60 Software Development Kits

The Symbian and older 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^3 SDK will work on all Symbian^3 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, googletest¹ works on Symbian, and other Mobile C++ platforms. Each SDK includes an emulator which 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 an application 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

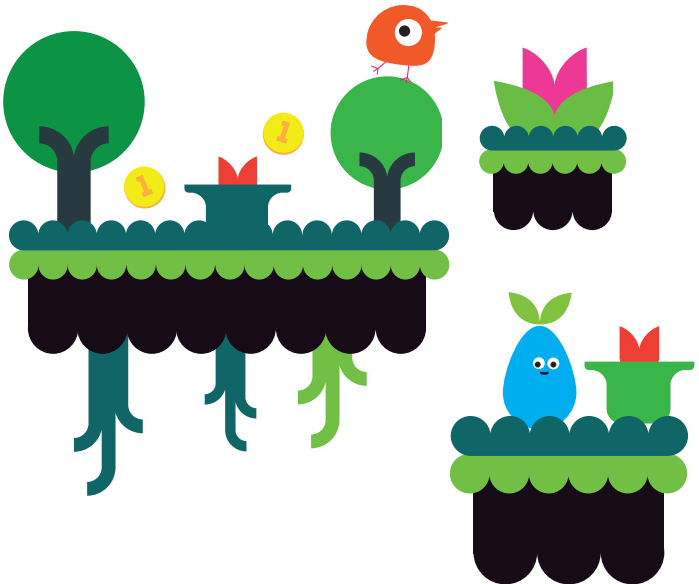
1) www.code.google.com/p/googletest

for more information on Nokia Store). For a limited number of applications, those using more advanced APIs, it will be necessary to obtain Certified Signed through the Symbian Signed website¹.

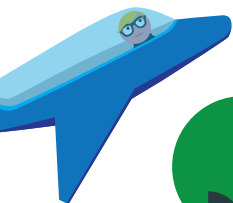
Distribution

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.

1) www.symbiansigned.com



1UP
1UP
1UP



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 in the consumer market. Windows Phone is geared towards business users as well as consumers, and offers a simple-to-use interface that focuses on typography and content. Called Metro, this UI design language is also already implemented on the Xbox 360 and is being introduced into Windows 8. 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 launch¹.

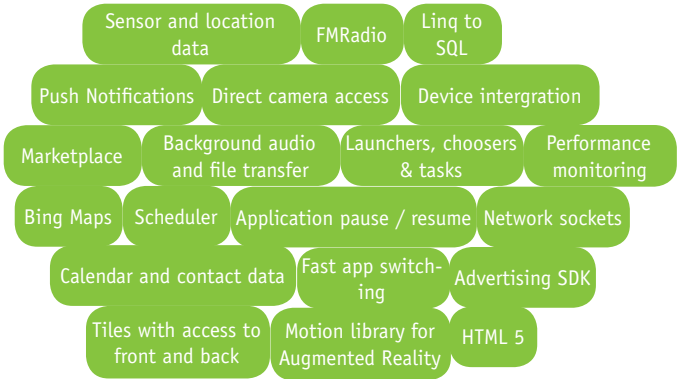
In February 2011, Nokia announced a partnership with Microsoft that underlines the future relevance of the platform: Windows Phone is now the first choice smartphone platform for Nokia phones.

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 use in a single application. The user interface for Silverlight applications can be created either in Microsoft Visual Studio or Microsoft Expression Blend.

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

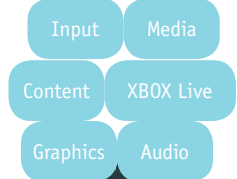
The Windows Phone 7.1 SDK



Silverlight Presentation and Media



XNA Frameworks



Common Class Library



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

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

Use Silverlight if...	Use XNA if...
...you want to create an event-driven application.	...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 both Windows Phone, Windows and the web, re-using some code.	...you want to target Windows Phone 7, 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 equally 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 that can update “live tiles” (animated application widgets that reside on the start page of the phone).

Multitasking and Application Lifecycle

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

Applications suspended in the background may be closed automatically if the OS determines resources are needed elsewhere. To create the appearance of an application that was never closed, Windows Phone has a well-documented application lifecycle called Tombstoning. To support Tombstoning the platform provides the hooks to cache and restore data and UI states.

Native Code

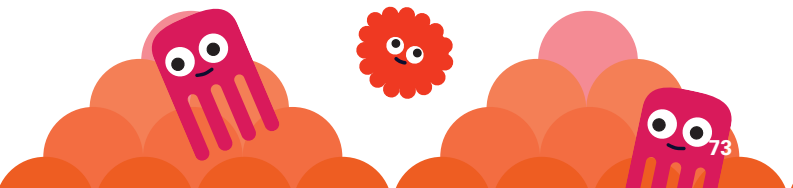
In contrast to some other platforms, developers cannot execute native code or access the device hardware directly using Windows Phone. While this restricts the extensibility of the platform and arguably limits the type of applications that can be devel-

oped, 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 on-screen keyboard cannot be replaced or extended, and low-level access to Wi-Fi or Bluetooth radios is not possible. However, for most applications you are unlikely to encounter any restrictions that will affect your ability to deliver user features.

Distribution

Applications for Windows Phone are distributed through a single endpoint: 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 (create.msdn.com). Although developer tools are provided free of charge, a paid App Hub account is necessary to deploy software to devices and the Marketplace. Currently, a developer account costs 99 USD for an annual subscription and includes 100 free app submissions. The fee is waived for students in the DreamSpark program, available at www.dreamspark.com. The Marketplace also provides for time-limited beta distribution and offers a private distribution channel for enterprises.

There is also the ChevronWP7 Labs project which, in partnership with Microsoft, will enable a Windows Phone Device to be unlocked for development for a small fee. It has been suggested this option will be cheaper than App Hub, but won't offer the free Marketplace submission. This service will be available soon at labs.chevronwp7.com.



Testing And Analytics

You can unit test applications using the Windows Phone Test Framework¹ or the Silverlight Unit Test Framework².

For developers wishing to collect runtime data and analytics, there are several options. Localytics³ and PreEmptive Solutions⁴ both provide analytics tools and services that are compatible with Windows Phone 7. Developers can also use the Silverlight Analytics Framework⁵ to connect to a variety of third-party tracking services such as Google Analytics. Starting in the Windows Phone SDK 7.1 update, there are now robust performance-monitoring tools available in Visual Studio.

- 1) www.wptestlib.codeplex.com
- 2) www.silverlight.codeplex.com
- 3) www.localytics.com/app-analytics/
- 4) www.preemptive.com/windowsphone7.html
- 5) msaf.codeplex.com/



Resources

Visit create.msdn.com for news, developer tools and forums. The development team posts on their blog on windowsteamblog.com/windows_phone. 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.



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, create 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 Wiki¹.

Because the standards are not complete, it is important to note that each widget technology has slightly different ways of

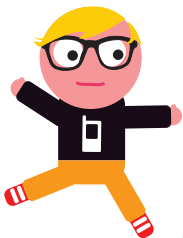
1) www.w3.org/2008/webapps/wiki/WidgetSpecs

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 PhoneGap¹, Titanium from Appcelerator², and Rhomobile³ among others. Of these options, PhoneGap offers a solution that is closest to the W3C approach.

- 1) www.phonegap.com
- 2) www.appcelerator.com
- 3) 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 SDK¹ which works in conjunction with Adobe Air, or standalone tools such as Nokia Web Tools². These tools generally provide template projects, a preview environment, validation, packaging, and deployment features.

1) us.blackberry.com/developers/browserdev/widgetsdk.jsp

2) www.forum.nokia.com/Develop/Web/Tools#NWT



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.

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 Guaranara.

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.

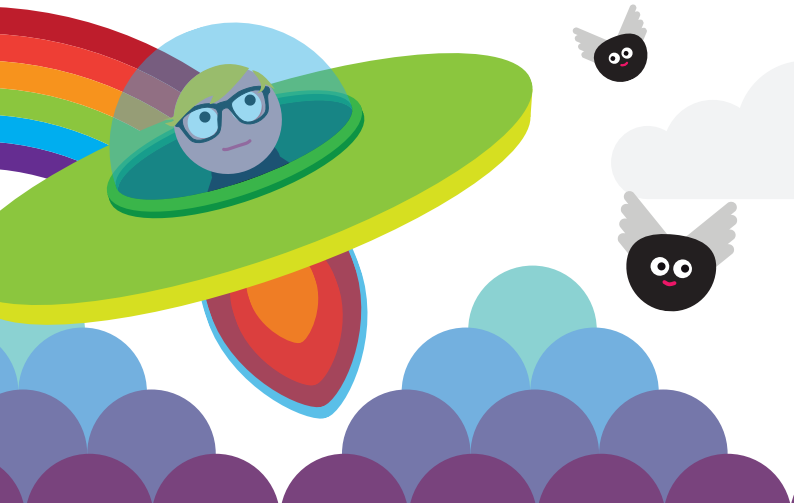
For example, when creating Series 40 web apps, Nokia Web Tools include Web Apps Simulator that runs your app on Microsoft Windows, Ubuntu Linux and Apple Mac computers. The

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 store¹, but by packaging your widgets appropriately you can upload them into Nokia Store², the Windows Marketplace³ or RIM BlackBerry AppWorld⁴. 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.

1) widget.vodafone.com

2) store.ovi.com

3) www.windowsmarketplace.com

4) 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 Tablet OS (QNX), 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.

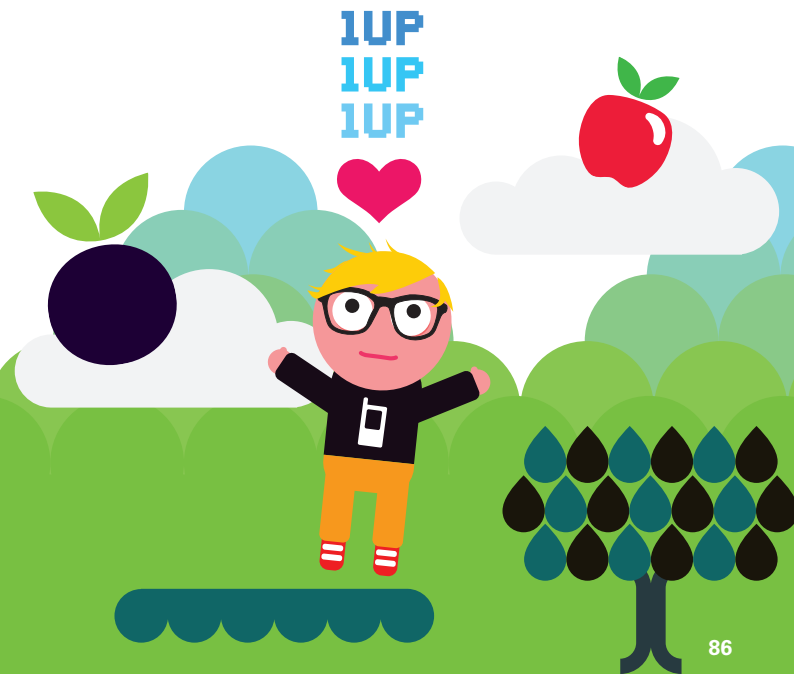
By the way, we are not talking about app stores here; this is a different market fragmentation problem. The more than 120 app stores, from operators, manufacturers and independent companies create challenges of their own, outlined in the "Appstores" chapter.

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 ¹	2nd Class Citizen ²	Target Platforms
ActionScript	1	0	BlackBerry Tablet OS (QNX)
C, C++	4	3	First class: bada, Brew MP, Symbian, Windows Phone Classic Second class: Android (partly, using the NDK), iOS (partly), BB Tablet OS (QNX)
C#	1	0	Windows Phone and Windows Phone Classic (formerly Windows Mobile)
Java	3	2	First class: Android, BlackBerry, Java ME devices Second class: Symbian, Windows Phone Classic
JavaScript	1	2	First class: <i>webOS (R.I.P.)</i> Second class: BlackBerry (WebWorks), Nokia (WRT)
Objective-C	1	0	iOS

- 1) Supported natively by the platform, e.g. either the primary or only language for creating applications
- 2) Supported as an option by the platform, e.g. 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.

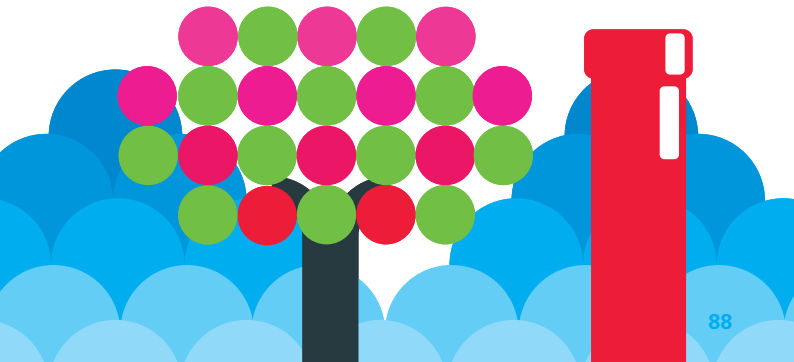
- **Interpretation**

Here the framework delivers an engine for each platform that interprets a common or framework specific language. In the gaming world Lua scripting is quite popular, for example.

- **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 platforms' 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.



UI + 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. 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 any simple 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 cross-platform 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.

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.
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 functionality that you require while keeping most of the functionality on the server side.

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

Cross-Platform 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 purchase and in app advertisement?



Solution	License	Input	Output
Airplay www.airplaysdk.com (Ideaworks Labs)	Commercial	C++	Android, bada, brew, iOS, Symbian, webOS, Windows Phone Classic
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, Black-Berry, brew, iOS, Symbian, Windows Phone Classic
Corona www.anscamobile.com (Anasca 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	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, Black-Berry, iOS, J2ME, PC, Windows Phone Classic

Solution	License	Input	Output
Flash adobe.com/devnet/devices.html (Adobe)	Commercial	Flash	Android, iOS, PC
Mono Touch monotouch.net (Novell)	Commercial	C#	iOS
MoSync www.mosync.com	Open Source + Commercial	C	Android, BlackBerry (Beta), iOS, J2ME, Symbian, Windows Phone Classic
PhoneGap www.phonegap.com (Nitobi)	Open Source	HTML, CSS , JavaScript	Android, BlackBerry, iOS, Symbian, webOS
Qt qt.nokia.com (Nokia)	Open Source + Commercial	C++	PC, Symbian, MeeGo and Windows Phone Classic as well as 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
SI02 sio2interactive.com (sio2interactive)	Commercial	C	iOS, other announced
Spot Specific www.spotspecific.com	Commercial	Drag and Drop, JavaScript	Android, iOS. (BlackBerry & Windows announced)

Solution	License	Input	Output
Titanium www.appcelerator.com	Open Source	JavaScript	Android, Consoles, iOS, PC
Unity3 unity3d.com (Unity Technologies)	Commercial	C#	Android, iOS, PC
Unreal www.unrealtechnology.com (Epic Games)	Commercial	UnrealScript, C++	Consoles, iOS, PC
Whoop www.whoop.com	Commercial	Drag and Drop	Android, BlackBerry, iOS, J2ME, Windows Phone Classic
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 application.

Context Is King

When you create a website for desktop browsers you typically design and develop for a context in which a user has a large display, enough time, power and also a fast persistent internet connection. None of these are guaranteed when using the internet on mobile devices.

Although most new devices have much larger screens than devices a few years before, they are mostly still small compared to desktop devices. Beyond that users often access a mobile website when they are on the go and they may be focusing on other things in addition to the web site they are using because they have to look at the traffic for example.

Other problems to consider are the effects of a weak signal and a slow mobile internet connection. Since the average internet user is impatient, you can easily lose a user who has to wait too long for pages to load – they may obtain the information elsewhere. Therefore you should try to keep content such as external scripts and images as small as practical.

You should consider these factors when you are about to design a website for mobile devices. Focus your mobile web strategy on what content a user is probably looking for when they visit your website. They are unlikely to be interested in your

awesome company intro page animation made in Flash. Beyond you should also think about the contexts in which your mobile site will be used before you begin to design a mobile website.

It could be in a train with a weak signal, in a village with poor connection speed, outside in a sunny environment on a top-notch smartphone with touchscreen display as well as on an older feature phone with an even older 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 content and context 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 it will be consumed but also a matter of *how* your target audience will use your site. Navigating your site is less fun when doing it using the device's keyboard instead of just using your finger on a large screen but it can be easier if the links within your page are so damn small that it is almost impossible to hit them without causing unwanted behavior (like tapping other links as you actually wanted to).

There are some basic hints to make sure your content is adapted in the best possible and usable way for mobile users:

- 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 anyway while they are on the go.

4. **Avoid text input wherever possible:** Text input on mobiles sucks. If you really need the users to enter text, use wide input boxes so that they see what they are typing. Buttons to clear an input field on click/touch are also helpful very often.
5. **Adapt the media:** Adapt all pictures, videos and alike to be displayed properly on the handset (check the corresponding chapter in this guide: “Implementing Rich Media” for more information). Try to avoid formats such as .doc and pdf if possible.
6. **The user is a creature of habit:** respect that: 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 ~50x50px.
8. **Use sharp contrasts:** Fonts and background colors that guarantee legibility in any surrounding, including 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.** 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 you seem to have more possibilities and better performance. This answer is seldom wrong but it is only half the truth. Let’s have a detailed view on 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 is already a long time ago) you will be surprised about the capabilities of modern mobile phone’s web browsers. Assuming that you intend to optimize your mobile website mainly for modern platforms you would also create apps for (iOS, Android, BlackBerry OS, WebOS, bada, Windows Phone) we will primarily focus on modern browsers running on the mentioned platforms in this chapter.

The current generation browsers on major platforms support a variety of modern HTML5, CSS and JavaScript features like 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, you can synchronize online data on a device for later offline use (e.g. if the signal is lost) and you can make whole web applications available even when a user has no active internet connection.

You can ask for permission to query the current position of a user just like you can do in a native app and you can also access the gyroscope of an iPhone using pure JavaScript directly in the browser. In addition mobile browser vendors are also working on making it possible to access the phone’s camera, network status or address book data.

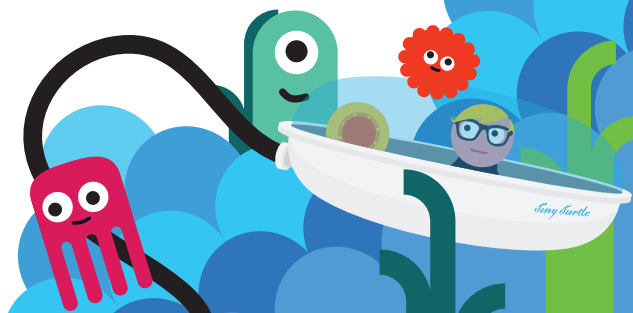
Sounds pretty app like, doesn't it? JavaScript is still often underestimated but if you know how you can rapidly create high class web apps which make extensive use of a device's capabilities (almost) without the need to create proprietary versions for each platform.

And that's not all: almost all recent mobile browsers support a lot of the current CSS3 standard and so you can create nice and shiny things like transitions, custom web fonts, drop shadows or rounded corners with only little effort which makes it very easy to let your web applications look and feel like native apps.

Pitfalls? Issues? Of course there are some. Although modern browsers already have a wide range of device API support there are still things you cannot yet do inside a browser. Accessing the camera as already mentioned is one thing. 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 just using the native GUI functionality which is a bit faster in most cases. But if your code is clean and effects are used wisely the performance difference to native apps is so small that a user probably will not even realise he is just 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 (i.e. Geolocation on Blackberry OS 4.6) or even have absolutely no support for certain features (i.e. 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 like caniuse.com², Modernizr³ or fitml.com⁴ where you can build quick and clean workarounds to handle fragmentation issues with only a minimum of effort. You still 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



Server-Side vs. Client-Side Adaption

Basically there are two different approaches to professional mobile web development if you aim to deliver a great user experience. Either way you will have to determine which (type of) device is sending a request to your website and then you need to “guess” (server-side) or test (client-side) what features are supported by the according device.

Server-Side Detection and User Agent Sniffing

The first possible way is to do so using the so called “user agent sniffing” on the server-side and then let your server create and deliver an optimized version of your site to the client. Server-side detection is usually based on large databases containing the user agents of thousands of devices and their capabilities. The most common use of server side adaption is image scaling on the server to save some bytes when delivering a big image to a mobile device. 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. So you typically resize the image to the display size of the mobile device and then serve a much smaller version to the client.

Service-side detection can be a good choice for other reasons. You can also decide which markup to deliver based on the user agent of the accessing device. If you have a visitor with an iPhone or an Android phone you can serve a nice HTML5 document while users with old Nokia devices will receive an old fashioned XHTML 1.1 document.

The advantage of server-side adaption is that you optimize all the content to serve only what a client probably really needs. And “probably” is also the problem here. New devices are released so often it is hard to keep a device database entirely up-to-date. There are some commercial providers for device databas-

es (WURFL¹, DeviceAtlas², fitml.com for example) if you want to realize mobile web projects with server-side user agent detection I strongly recommend you consider one of these as they have full time employees actively maintaining their databases and do a lot of work that would be impractical for you to do yourself.

Pitfall here: user agents can be “wrong”, manipulated or unknown. You should therefore always provide a fallback in case your user agent detection fails. Such a fallback could be a document in a format (i.e. HTML4) that almost every mobile browser released in the past 5 years can understand.

Client-Side Adaption and Feature Detection

Client-side feature detection is the second approach to create a great user experience on mobile websites. Feature detection in general means that you use JavaScript to look if a certain capability is supported on the accessing device. To give you a first impression: you can use `if(navigator.geolocation)` to check if a device supports acquiring of the user’s current position.

Feature detection also means that you will have to deliver the complete document with all possible features of the website in it and then gracefully degrade it by removing features which are not supported on the device. That means you are sending a lot of content to all devices regardless of whether a certain feature is supported on a device or not.

One big advantage compared to server-side adaption: when a feature test passes you can (under normal conditions) be sure that your desired feature or behavior will work as expected even

1) wurfl.sourceforge.net

2) www.deviceatlas.com

if a user agent was modified and therefore was not recognized properly by a device capability database.

The Modernizr JavaScript library is probably the first place to go when it comes to client-side feature detection. Modernizr covers a lot of possible browser capabilities and provides a simple API to check for support of a certain feature. If you need to know whether a browser supports Drag and Drop or not, just use the Modernizr library this way:

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

Another part that belongs to the client-side is the adaption of layout. On modern mobile browsers you can use media queries which 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 linear one column layout when a device's display has a resolution of less than 800px.

Issues concerning client-side approach: first there are some things, especially browser bugs, which may be hard to detect on client side. Another big problem is that you always have to serve the whole document with large scaled images and a bunch of JavaScript to the client since you do not know if your visitor is using a bleeding-edge hipster browser from Cupertino or an oldschool "I don't know what JavaScript is" browser from 2003.

Why not just use the best of both worlds?

Excellent idea! There is of course no rule that forbids to combine both server-side and client-side technology to create an even better user experience. So you can try to determine the basics (“which markup should be used?”, “what is the correct size for images?”, “JavaScript support, yes or no?”) on the server and let the client handle the rest. You can provide server-side fallbacks for JavaScript enhancement to make a site accessible and so you can create mobile websites which even work fine on old devices. A commercial platform working according to this principle is *fitml.com*¹ 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-side and on client-side.

Hybrid Apps

If you necessarily want (or need) to publish your mobile app on Android Market, Apple Appstore, etc. you can also create a “hybrid app”. Create your app completely by using common web technologies (HTML, CSS, JavaScript) and then compile it as app. Sounds easy? It is. There are several hybrid app frameworks which let you create native apps with only a single HTML5 web app as shared base. There is PhoneGap², Appcelerator³ or Apparatus⁴ and probably many more you can use to achieve this.

How does it work exactly? Write your complete application in HTML5 just as you were about to publish it to the web. Then you use one of the frameworks to compile your web app as native app. The framework creates some sort of “wrapper app” which

1) www.fitml.com

2) www.phonegap.com

3) www.appcelerator.com

4) apparatus.io

embeds your web app in a “web view” and it can then be installed as regular application on several different platforms. The big plus of doing it this way is that you only have one web app as the base and thus you can reduce your costs for development and maintaining but the result is still a “real” native app.

Small downer: although frameworks let you use features in your web app which you usually cannot use in a browser hybrid apps are not a complete substitute for native apps. They offer some nice advanced functions like vibration, access to camera or address book but in its core it still is a web app. That means you will have to create your own user interface which might be slower and you also cannot use really everything as you can when creating an app using the native SDK.



Lessons learned

The gap between native apps and web apps is rapidly decreasing. Browser vendors did a lot of good work in the past few years and implemented many features which were only available in native apps. There are different approaches to create great mobile websites which can look and even feel a lot like a native app and new devices will be released that have even better support for HTML5 and its device specific enhancements.

Creating mobile websites or mobile web apps makes your content accessible over the web on almost every platform with only little work compared to native development for several platforms. It can thus save you a lot of costs for development and maintenance. Due to the existence of hybrid app frameworks you can even publish your apps in app stores.

If you have never developed a mobile website or web app before or if you were not convinced because of the poor browsers in the early days of the mobile web you should give it another try. Consistent support of the many different features is still far from perfect but it has been improved by at least 1000% since the rise and success of Android and iOS.

When you became curious and want to create a mobile website 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, that 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.

The challenge then, is to design your apps accessibly to enable its use by as many people as possible.

Built-In Accessibility Features

Some of the mobile platforms have accessibility features that can help make it easier for people to use your apps. For example, iOS devices include:

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

Users often 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 apps such as screen readers.

So, in order to develop an accessible app, your goal as a developer is to:

- Follow the general guidelines below in order to give your app the best chance of interoperating with any third-party access software the user may be running in conjunction with your app
- Find out what accessibility features and APIs your platform has and follow best practice in leveraging those APIs if they exist.

General Guidelines For Accessible App Development

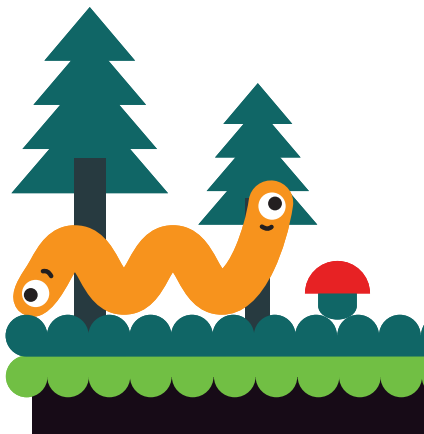
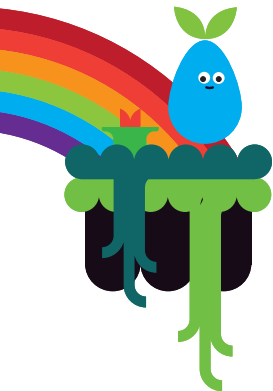
The following guidelines will make your apps more useable by those of your customers with disabilities:

- 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 accessible navigation is 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 online¹.

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



Developing Accessible iOS Apps

iOS has good support for accessibility. However, it only works well if the accessibility guidelines¹ have been followed. This guidelines detail the API and provide an excellent source of hints and tips for maximizing the user experience with your apps.

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 developers².

Developing Accessible Symbian / Qt Apps

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

When developing native Symbian apps your best chance of developing an accessible app is to use the standard UI controls where possible. If you are developing using Qt, then please check the web for details of their accessibility API³.

- 1) developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/iPhoneAccessibility
- 2) docs.blackberry.com/en/developers/deliverables/11936
- 3) doc.qt.nokia.com/qq/qq24-accessibility.html

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.

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*¹.

For more information about Android accessibility including how to use the text to speech API, see the *Eyes-Free* project².

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*³.

1) developer.android.com/guide/practices/design/accessibility.html

2) code.google.com/p/eyes-free

3) w3.org/WAI/intro/wcag

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 Practices¹ as this is likely to form the foundation of any mobile web application accessibility standard that emerges in the future.

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 7 and QNX 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.

¹ w3.org/TR/mwabp



Implementing Rich Media

“As many standards as handsets.” – Again this is true for the list of supported media formats on mobile phones. Contrary to PCs where most audio and video formats are supported or a codec can easily be installed to support it, 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 fail playback, so always test on real devices.

Most smartphones today do 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: Either 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 up to 48 kbps for GPRS (audio only), 200 kbps for Edge, 300 kbps for 3G/UMTS/WCDMA and 500 kbps for Wi-Fi.

Apple's open source Darwin streaming server¹ can serve streaming video and audio with highest compatibility and reliable RTSP combined with FFMPEG² and is always a good choice to stream 3gp or mp4 files.

When targeting Windows Mobile/Phone, Windows Media Server³ 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 the 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 like *download.php* a well-known trick is to add a parameter like *download.php?dummy=.3gp* to allow 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

1) dss.macosforge.org

2) www.ffmpeg.org

3) technet.microsoft.com/en-us/windowsserver/

might handle the request. On some phones, like Android, multiple media players can be available and a popup is displayed to allow the user to select one.

Finally you might of course also simply include the local media file in your mobile app as a resource. On Android devices pay attention to support media located at SD-Cards (Android 3.1 and up) which requires the `android.permission.READ_EXTERNAL_STORAGE`

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). So far the most reliable open source hinting software found is Mp4box. Note that a 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 and 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, e.g. "Super" from eRightSoft¹. 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 Alembik².

1) www.erightsoft.com/super

2) 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 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 in 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, this 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. Yet an A-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 others may provide the latitude and longitude.

- **Manual input**

The user selects a location on a map, inputs an area code or address. This option is used typically for applications on feature phones, which 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 Map¹ or CloudMate², while Ovi Maps³ are at the time of writing free for Nokia phones only. Commercially available maps include NAVTEQ⁴, Garmin and Microsoft⁵ to name a few. Some solutions, such as Google Maps⁶, 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 accuracy, response time, altitude derived from the on-board GPS, and speed based on performing consecutive readings.

With iOS and the iPhone SDK, 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 devices are more liberal with map sources, even though they default to Google Maps. On Symbian devices, Ovi Maps can be used for Nokia phones free of charge and for commercial use.

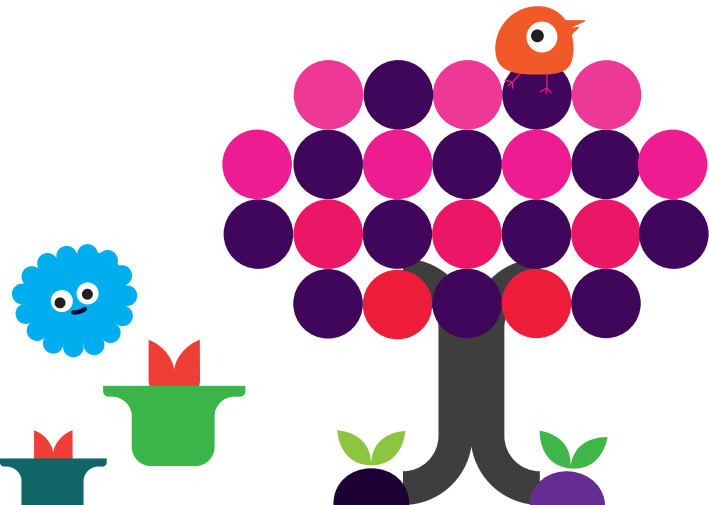
- 1) wiki.openstreetmap.org/wiki/Software
- 2) www.developers.cloudmade.com/projects
- 3) www.forum.nokia.com/Develop/Web/Maps
- 4) www.nn4d.com
- 5) www.microsoft.com/maps/developers
- 6) www.code.google.com/apis/maps



Maps can be overlaid with geodata, in a number of formats. One of the simplest 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, and more and more 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, for example Ad-mob and NAVTEQ both offer developers a stand-alone location aware advertisement program, where applications can exchange location data for smarter display of location relevant ads.

- **Garmin Mobile XT SDK:** *developer.garmin.com*
- **iPhone SDK:** *developer.apple.com*
- **Offline tools:** *code.google.com/p/big-planet-tracks/*
- **Android:** *developer.android.com/guide/topics/location/*
- **NAVTEQ:** *www.nn4d.com*
- **TeleAtlas:** *developerlink.teleatlas.com*
- **Nutiteq:** *www.nutiteq.com*
- **Qt Maps/Navigation API:** *qt.nokia.com/products/qt-addons/mobility*
- **Bing Maps:** *www.microsoft.com/maps/developers/*
- **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 standards¹ provide for three modes of operation that can be used in mobile devices:

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

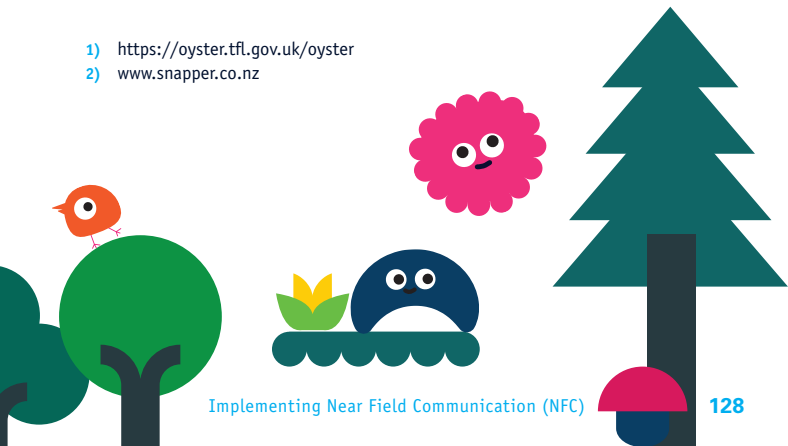
1) www.nfc-forum.org/specs/spec_list/

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 Oyster¹ or payments cards such as Snapper².

1) <https://oyster.tfl.gov.uk/oyster>

2) 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 Nokia¹ 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 capabilities².

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 APIs³, making it likely that a single set of APIs can be used for Symbian and MeeGo devices.

Otherwise it will essentially be one set of APIs per platform, such as the Google APIs for Android⁴.

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

1) www.forum.nokia.com/nfc

2) www.nearfieldcommunicationsworld.com/nfc-phones-list/

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

4) developer.android.com/reference/android/nfc/package-summary.html

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. 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.

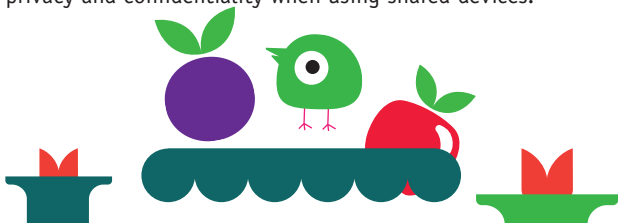
For platforms such as Android and Java ME where there are so many manufacturers and models, it's particularly useful to test on a range of these devices. A good start is to pick a mix of popular, new, and models that include specific characteristics or features such as: touch screen, physical keyboard, 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 project² 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 phones³.

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/nativewebdriver/

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 blog¹ that includes an example of how to emulate the iPhone browser².

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 Mobile³.

1) googletesting.blogspot.com

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

3) 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 innovation¹. 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) *Developer Economics 2011, July 2011:* 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 sending a Premium SMS. This option 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 is very difficult to handle particularly if you want to sell in several countries, as you need to sign contracts with each operator in each country. 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 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 Android¹ and Nokia Ovi app users to purchase apps and bill them directly to their mobile bill. Android made a major gain in its

¹ *Android developer blog on operator billing in the US:* android-developers.blogspot.com/2010/12/more-payment-options-in-android-market_22.html

European markets recently when Vodafone announced¹ it is rolling out direct operator billing for Android Market customers in European markets. This enable 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 TextDeck² 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, Nokia Store has the best coverage (121 operators in 42 markets at the time of writing) while 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 60% 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 (www.daopay.com), or similar.

1) *Vodafone blog post:* developer.vodafone.com/blog/2011/08/18/vodafone-developer-celebrates-operator-billing-android/

2) glimmerdesign.com/textdeckpro

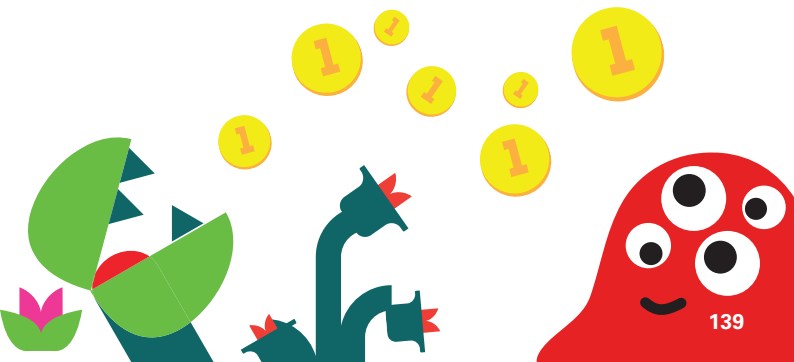
Using PPD can typically be implemented with no special design or coding requirements for your app.

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

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 too.

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 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 O2 and Movistar networks, 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 also *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 vendors 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 apps¹.

1) 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.

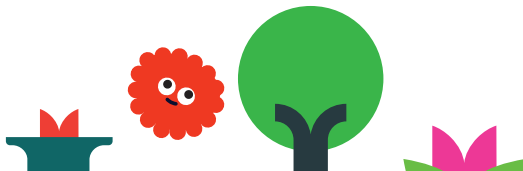


Appstores

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 you 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 AppStore and Android's App Market were behind Nokia's Ovi Store, Windows Marketplace and BlackBerry's App World when it comes to the number of downloads any particular app might achieve¹.

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. 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.

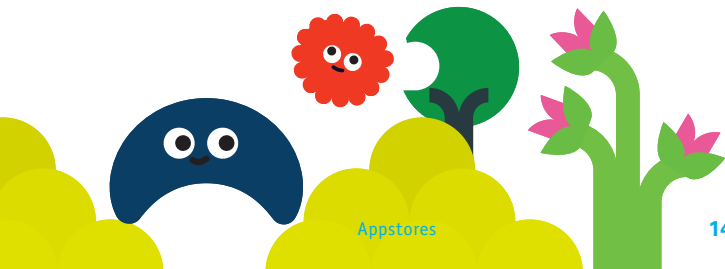
1) www.research2guidance.com/apps-on-nokia's-ovi-store-had-2.5-times-higher-download-numbers-in-q2-2011-compared-to-apps-on-apple-app-store/

- Polish your icons and images in the appstore, work on your app description, and carefully choose your keywords and category. If unsure of or unsatisfied with the results, experiment.
- Getting reviewed by bloggers and magazines is one of the best ways to get attention. In return some will be asking for money, some for exclusivity, and some for early access.
- Get (positive) reviews as quickly as possible. Call your friends and ask your users regularly for a review.
- If you are going to do any advertising, use a burst of advertising over a couple of days. This is much more effective than spending the same amount of money over 2 weeks, as it will help create a big spike, rather than a slow, gradual push.
- Do not rely on the traffic generated by people browsing the appstore, make sure you drive traffic to your app through your website, SEO and social media.

Multi-Store vs Single Store

With 120+ appstores available to developers, there are clearly many application distribution options. But the 20 minutes needed on average to submit an app to an appstore means you could be spending a lot of time posting apps in obscure stores that achieve few downloads. This is why a majority of developers stick to only 1 or 2 stores, missing out on a potentially huge opportunity but getting a lot more time for the important things, like coding! So should you go multi-store or not?

Multi-store	Single store
The main platform appstores can have serious limitations, such as payment mechanisms, penetration in certain countries, content guidelines.	90%+ of smartphone users only use a single appstore, which tends to be the platform appstore shipping with the phone
Smaller stores give you more visibility options (featured app)	Your own website can bring you more traffic than appstores (especially if you have a well-known brand)
Smaller stores are more social media friendly than large ones.	Many smaller appstores scrape data from large stores, so your app may already be there.
Operators' stores have notoriously strict content guidelines and can be difficult to get in, particularly for some types of apps.	For non-niche content, operator or platform stores may offer enough exposure to not justify the extra effort of a multi-store strategy.
Smaller stores may offer a wider range of payment or business model options, or be available in many countries.	Some operators' stores have easier billing processes – such as direct billing to a user's mobile account -- leading to higher conversion rates.
Some developers report that 50% of their Android revenues come from outside of Android Market	iPhone developers only need 1 appstore



Now What – Which Environment Should I Use?

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 just reaching 25%, 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 mobile phones currently in the market.

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.

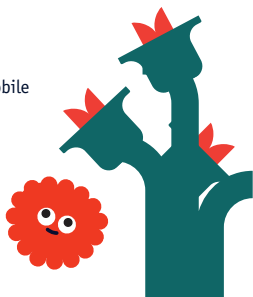
As of the second quarter of 2011, the top smartphone platforms in terms of shipments were (note that this table excludes feature phone platforms, which make up the majority of sales):



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](http://comscore.com)¹, [StatCounter](http://statcounter.com)², [VisionMobile](http://visionmobile.com)³ or [Gartner](http://gartner.com)⁴.

- 1) www.comscore.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.

At the other end of the spectrum, Android has around 300,000 available apps, while it has been shipped in approximately 160 million devices. Similarly, iOS has over 500,000 apps available, but Apple has shipped barely 130 million iPhones.

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 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 development¹ 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.

1) www.DeveloperEconomics.com

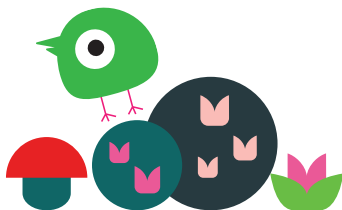


The research also investigated which are the top platforms being abandoned developers:

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).

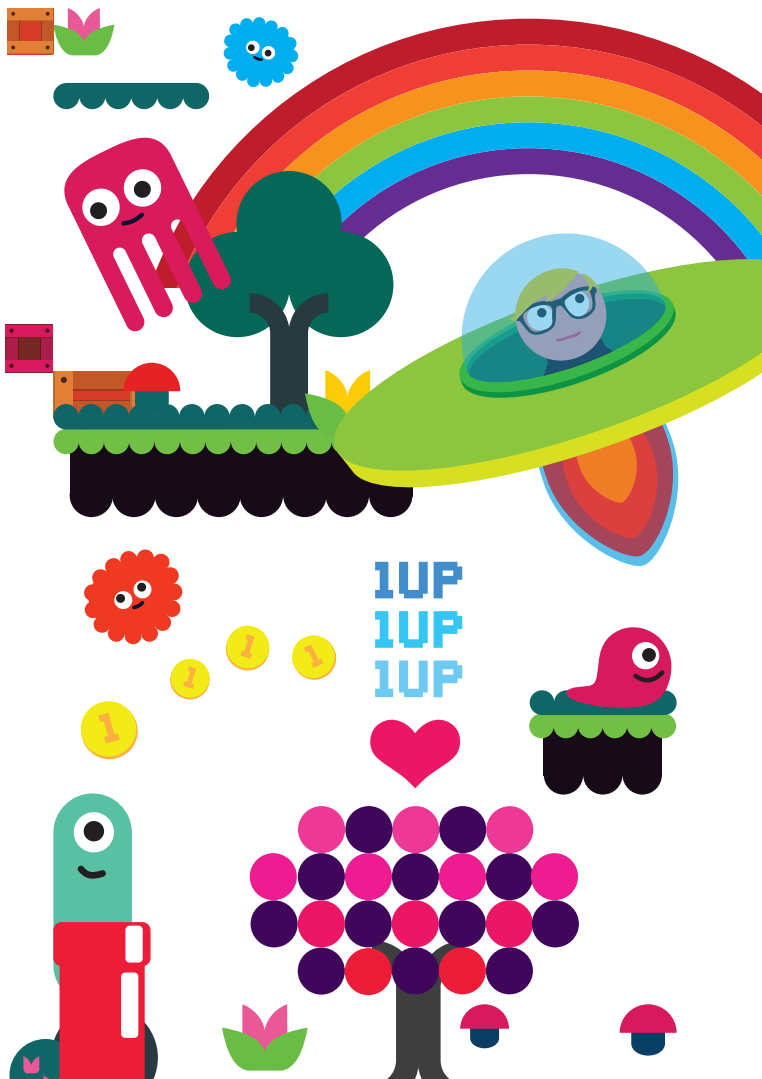
Other important aspects of each platform, from a developer's point of view, are the size of the developer community, the quality of developer tools, the range of APIs, possible dependencies on internet connectivity and performance or fragmentation issues:



	Feature Set	Online / Offline	Developer Availability	Developer Tools	Performance	Fragmentation
Android	Green	Green	Green	Green	Green	Red
bada	Green	Green	Red	Yellow	Green	Yellow
BREW	Green	Green	Red	Yellow	Green	Yellow
Flash	Green	Green	Green	Yellow	Yellow	Yellow
iOS	Green	Green	Yellow	Green	Green	Yellow
Java ME	Green	Green	Green	Green	Green	Red
Native BlackBerry	Green	Green	Green	Green	Green	Yellow
Native Symbian	Green	Green	Red	Yellow	Green	Yellow
SMS	Red	Yellow	Red	Red	Red	Green
Web	Yellow	Yellow	Green	Green	Red	Yellow
Widgets	Green	Green	Green	Yellow	Yellow	Green
Windows Mobile	Green	Green	Yellow	Green	Green	Yellow
Windows Phone	Green	Green	Red	Green	Green	Yellow
Qt (Symbian/MeeGo)	Green	Yellow	Yellow	Green	Green	Green



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



Epilogue

Thanks for reading this ninth edition of our Mobile Developer's Guide. We hope you've enjoyed reading it and that we helped you to clarify your options. Don't be put off by the difficulties in entering the mobile arena – once you're in the water, you can and will swim.

Would 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](https://twitter.com/hashtag/mdgg)



About the Authors

Robert Virkus / Enough Software

Robert has been working in the mobile space since 1998. He experienced Java fragmentation first hand by 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 that helps developers to 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

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

Thibaut Rouffineau / WIP

Community and passion builder with a mobile edge, Thibaut has been conversing with the mobile developer community for the past 5 years as the head of developer engagement at Symbian, where he spearheaded the migration to open source. Today he is the VP for Developer Partnerships at WIP (Wireless Industry Partnership).

www.wipconnector.com

Chris Brady / Animated Media Inc. (AMI)

Chris is an expert on graphics and GPUs and has been developing software since the 1980's. He founded ALT Software Inc. growing it to the leading provider of safety critical, real-time, OpenGL 3D device drivers and software in the aerospace market. As AMI's CEO, he is now leading the charge to bring Flash technology to devices and markets outside of Adobe's focus – including Flash on the iPhone.

www.animatedmedia.ca

Tim Messerschmidt

Tim has been developing Android applications since 2008 for his own business Messerschmidt-IT. He is currently writing his bachelor thesis about Android and Google App Engine.

www.messerschmidt-it.de

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

Benno Bartels / InsertEFFECT

Benno's entry to the mobile space was his diploma thesis about porting J2ME applications. Afterwards he founded InsertEffect, a company focusing on mobile web development. Today, the team consists of 10 people focused mainly on usability optimization of mobile websites, social network applications and widgets.

www.inserteffect.com

Ovidiu Iliescu / Enough Software

After developing desktop and web-based applications for several years, Ovidiu decided mobile software is more to his liking. He's been doing J2ME 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.oviduiiliescu.com

Michel Shuqair / AppValley

Michel built his experience with Telecoms since 1999 where he closely watched the mobile development space evolving from Japan. Starting with black and white WAP applications, iMode and SMS games, he led the mobile social network *m.wauwee.com* with almost 1,000,000 members, supported by a team of Symbian, iPhone, BlackBerry and Android specialists with headquarter in Amsterdam (acquired by MobiLuck).

www.appvalley.nl

Peter Nowak

Peter is a .NET expert at global operating ICT - Services Company since 2002 and has been a Microsoft MVP since 2008. He writes for several magazines, speaks at developer conferences, and published his own book about Windows.

www.winphonedev.de

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. 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

Alex Jonsson / MoSync

Alex likes anything mobile, both apps and web technology and connecting physical stuff to digital stuff. He holds a doctors degree in on-line publishing and distributed education. Behind this tech surface lies an eclectic urge to create new value by exploiting aspects of communication and media to bring people together. Alex holds a position as VP Creative Products at MoSync Inc.

www.mosync.com

Richard Bloor / Sherpa Consulting Ltd

Richard has been writing about mobile applications development since 2000. He contributes to popular websites, such as *AllAboutSymbian.com*, and assists 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.

Jens Weller / Code Node

In 2002 Jens started his career in the mobile business at Vodafone. 5 years later, he founded his own company Code Node Ltd. Since then Jens has been working in the industry as a specialist in C++ and Qt. In 2009 he also started to offer mobile development for bada. Jens has a blog about mobile development with C++, and has spoken at various events on this topic. He likes to dance Salsa in his free time.

www.codenode.de

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

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 joined the development team at Enough Software in 2005. He has not only headed the development of numerous mobile projects (mainly for Windows Mobile and BlackBerry), but is also an expert on server technology. Of course he is an open source enthusiast, just like everybody at Enough Software.

www.enough.de

Patrick Getzmann

Patrick is working for a global operating ICT - Services Company since 2000. He is responsible for mobile app development, conceptioning and architecture with a focus on Windows Phone. As a Microsoft MVP ("Most Valuable Professional") he is always happy to share his know-how at conferences and in several publications.

Gary Johnson / Hyland Software, Inc.

Gary has been working as a software developer for Hyland Software, Inc. since 2005. He works primarily in Silverlight and WPF, and has a strong passion for UX and mobile development. As a hobbyist, he is heavily involved in Windows Phone 7 development.

www.hyland.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

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 has worked on a number of apps, one of which was featured in an Apple TV commercial, and 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

Manuel Bieh / Sevenval

Manuel started to create websites soon after the internet gained mass popularity. He started focusing on the mobile web in 2007 and in 2008 published his first German technology book on mobile web design and development. He loves to play around with JavaScript, HTML5 and all the latest web and mobile tech. In 2010 he joined Sevenval, as Developer and Community Evangelist, and since then has been working to make the mobile web a better experience for users and developers.

www.sevenval.com www.fitml.com

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



An initiative by:



www.enough.de



www.wipconnector.com

Printing sponsors:



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

// This excellent guide is a huge hit in Appsterdam, thank you so much for making this resource available.

Mike Lee, Mayor of Appsterdam

// Extremely helpful content, also for non-developers. And the design is nothing but fantastic!

Monika Lischke, Community Manager, Intel AppUp developer program

// Packed with valuable information if you need a crash course in mobile development. Highly recommended!

Tom Deryckere, www.mobiledrupal.com

// Impressive. The most comprehensive and concise guide to developing for mobile.

Carlos Bernardi, Team Leader Handset Embedded Programs, Gameloft