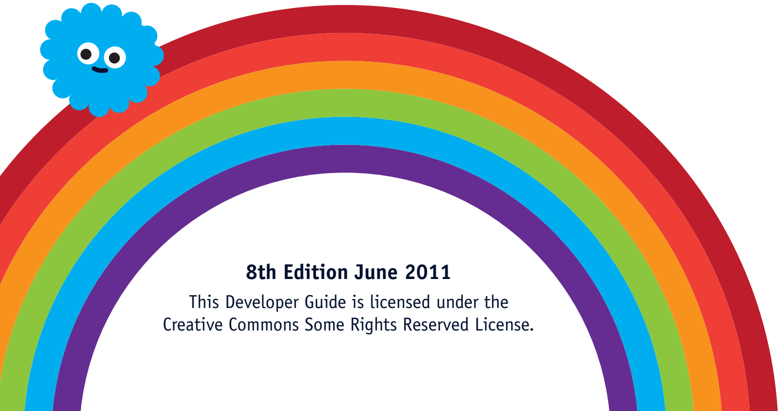
Don't Panic
Mobile Developer's Guide to the Galaxy

published by:

ENOUGH SOFTWARE
Services and Tools for All Mobile Platforms

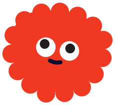
Enough Software GmbH + Co. KG
Sögestrasse 70
28195 Bremen
Germany
www.enough.de

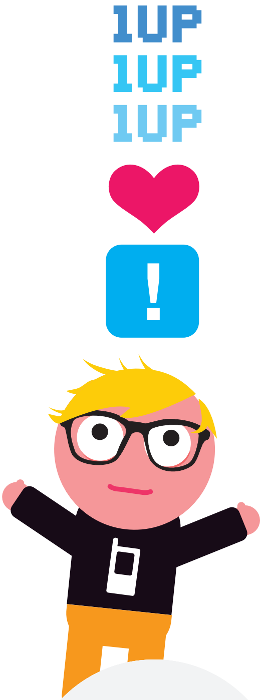
Please send your feedback, questions or sponsorship requests to:
developers@enough.de

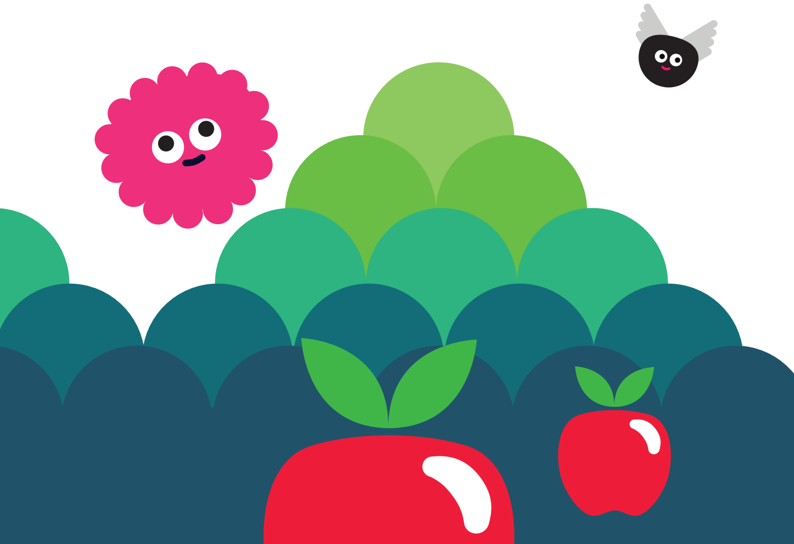


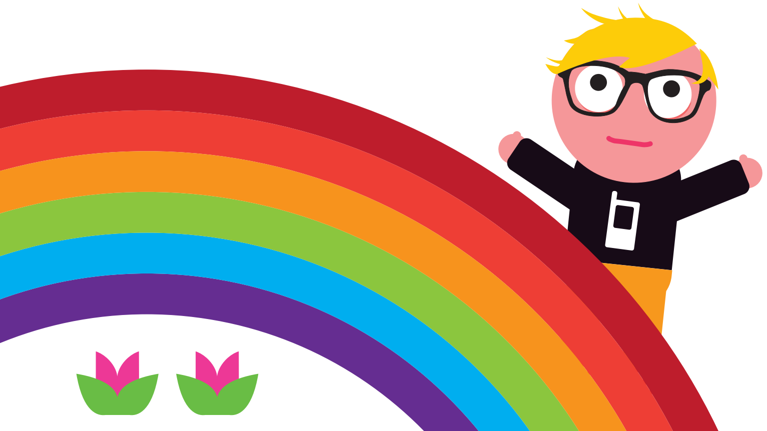
8th Edition June 2011

Mobile Developer's Guide
Table of Contents




1UP
1UP
1UP
!





Introduction

This is already the eighth edition of the Mobile Developer's Guide To The Galaxy. As you can see, it has become something of a book – taking us a long way from where we started . The first version was published for Mobile World Congress 2009 and consisted of just 40 pages. At that time we did not expect the guide would become a regular publication, with new editions every three months. But the overwhelming volume of feedback coupled with the ever-changing mobile market has driven us.

For this release, all the writers have checked their chapters and updated them where necessary. This means that you will learn everything about the latest developments in all the major mobile platforms and technologies.

New for this edition is a chapter on Near Field Communication (NFC). While it is still early days for NFC, it might just change the way we use our mobile phones – all over again.

Another new chapter covers monetization from a platform-independent perspective. We feel this is an important addition because many of the considerations are the same no matter what technology you are using to create your mobile software.

While the guide has changed in size, we hope you will agree that we have stuck with our original philosophy: Providing an objective overview of mobile technology that is useful to decision makers and developers alike. Every business case is different and there is no one platform that can cover them all. As a result, decisions about how to deliver a mobile application can be difficult; we hope that this booklet will help you make the best decision. Despite the challenges it creates we hope you will agree with us: Diversity is a good thing, equally so in the mobile world.

Robert + Marco / Enough Software
Bremen, June 2011

1UP
1UP
1UP

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

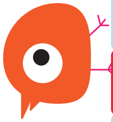
There are many mobile platforms used in the market – some are open source, some are not. The most important native platforms are (alphabetically) Android, bada, BlackBerry, iOS, MeeGo, Symbian, webOS and Windows Mobile/Windows Phone. All these platforms enable you to create native applications without establishing a business relationship with the respective vendor.

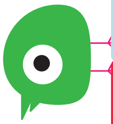
The main benefits of programming apps natively include better integration with the platform's features and often better performance. Typical drawbacks are the effort and complexity of supporting several native platforms (or limiting your app to one platform).

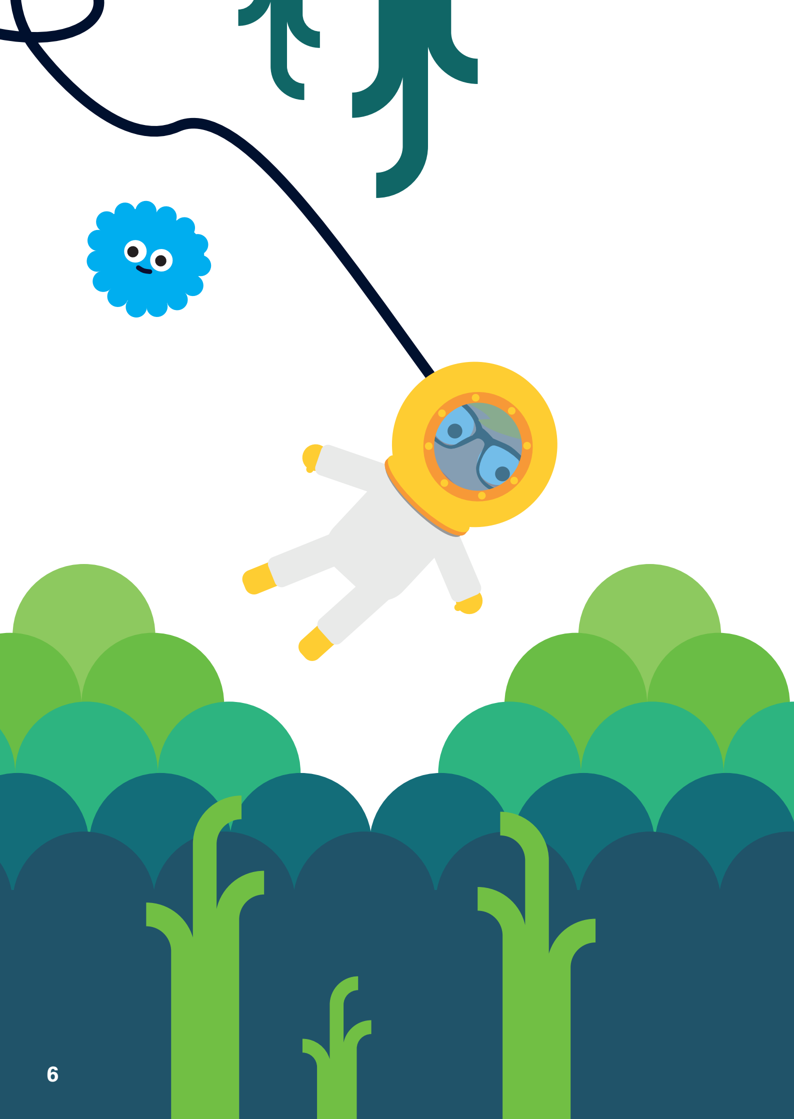
Most mass market phones are, however, equipped with embedded operating systems that do not offer the opportunity to create native applications. Examples include but are not limited to Nokia Series 40, Samsung SGH and Sony Ericsson Java Platform phones.

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, Web Apps, Java ME compatible, extensions enable tighter integration na.blackberry.com/eng/developers
iOS	Objective-C, C	Requires Apple Developer Account developer.apple.com/iphone
MeeGo	Qt, C++, others	Intel and Nokia guided open source OS (based on Linux) meego.com/developers
Symbian	C, C++, Java, Qt, Web Apps, others	OS built from the ground up for mobile devices www.forum.nokia.com/symbian
webOS	HTML, CSS, JavaScript, C	Supports widget style programming, (based on Linux) 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 Qualcomm[1].

BREW services are offered by more than 60 operators in 28 countries, but it's most popular within the US with CDMA devices launched by Verizon, US Cellular and Metro PCS, among others. While previous versions supported C development only, the Brew Mobile Platform (Brew MP), supports applications written in Java, Flash, TrigML or native C code[2].

Widgets

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 / JIL	XML, HTML, JavaScript, CSS	A joint initiative by Vodafone, China Mobile and other companies are pushing the W3C widget standard www.jil.org
Samsung	XML, HTML, CSS, JavaScript	innovator.samsungmobile.com
Series 40 web apps	XML, HTML, CSS, JavaScript	Web apps for the proxy based Ovi Browser enabling UI manipulation on a device through JavaScript. W3C packaging standard used. www.forum.nokia.com/webapps
PhoneGap	HTML, CSS, JavaScript	Cross platform widget platform www.phonegap.com
Sony Ericsson WebSDK	HTML, CSS, JavaScript	Based on PhoneGap developer.sonyericsson.com
BlackBerry	HTML, CSS, JavaScript	na.blackberry.com/eng/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, 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. 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, where its use for payments is common.

Programming Android Apps

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

Android is an operating system and an application framework (Dalvik) with complete tooling support and a variety of preinstalled applications. In late 2010, Google announced that every day 300,000 Android devices are shipped to end users. Since the platform is supported by many hardware manufacturers, it is the fastest growing smartphone operating system: In March 2011 it has been announced that Android surpassed both iOS and BlackBerry in terms of subscriber share in the U.S.[1]

Additionally, Android is used (or planned to be 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 like RIM's Playbook with its virtual machine called App player[2].

The platform also evolves rapidly with the regular additions of new features every 6 months or so. For example, Android 2.3 (so-called "Gingerbread") introduced NFC and VOIP communication, better game development and a lot more[3], Android 3.0 ("Honeycomb") has been especially designed for deployment on tablets and other devices with larger screens.

1) comscore.com/Press_Events/Press_Releases/2011/3/
2) http://www.theregister.co.uk/2011/03/25/rim_playbook_android/
3) developer.android.com/sdk/android-2.3-highlights.html

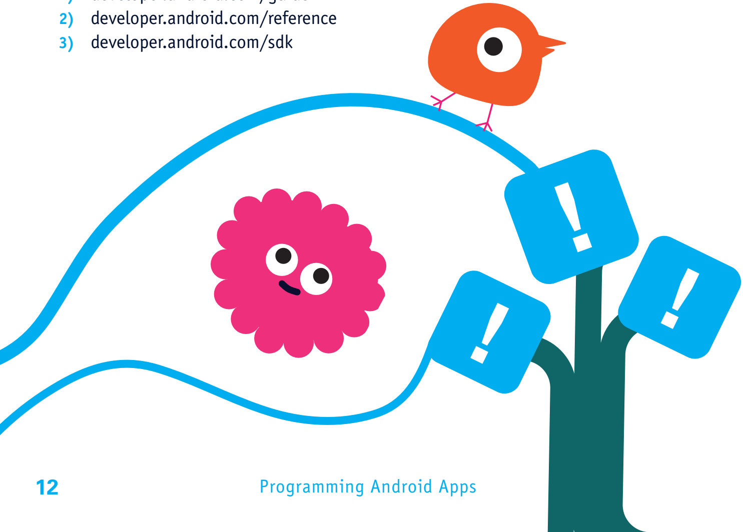
Prerequisites

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 Guide[1] and to your How questions in the reference documentation[2].

To get started, you need the Android SDK[3], which is available for Windows, Mac OS X and Linux. It contains tools to build, test, debug and analyse applications. You will probably also want a good Java IDE. Eclipse or IntelliJ seem a good choice as there is good support for development, deployment and importantly, so-called library projects that allow to share code and resources between several projects.

Command line tools and Ant build scripts are also provided so you can concoct 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 which should run in the background and is therefore not tied directly to a visual representation. A message receiver can handle messages broadcast by the system or other applications. A data provider is an interface to the content of an application and thereby abstracts from 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 like the user's location or an URL with an action. These intents trigger behaviours in the platform. For instance, the intent of showing a web page will open the browser activity. The nice thing about this building-block philosophy is that functionality can be replaced by other applications and the Android system will use 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 a lot of tools from the SDK at your disposal, the most important ones are:

- **android**: Create an initial project or manage virtual devices and versions of the SDK.
- **adb:** Query devices, connect and interact with them (and virtual devices) by moving files, installing apps etc.
- **emulator:** Start it with a virtual device and it will emulate

the defined features. It takes a while to start so do it once and not on every build.

- **ddms:** Look inside your device or emulator, watch log messages, and control emulator features like network latency and GPS position. View memory consumption or simply kill processes. If this tool is running, you can also connect the Eclipse debugger to a process running in the emulator.

These tools and more – e.g. to analyze method trace logs, inspect layouts, or to 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 Java code. To this end, localized strings and images are organized in separate resource folders. IDE plugins are available to help manage all these files.

If you are facing issues, e.g. exceptions are thrown, be sure to check the ddms log. It allows you to check if you have omitted to add necessary permissions like *android.permission.INTERNET* via the uses-permission flags[1].

If you are going to use Honeycomb related layout features for large screens like Fragments[2], be sure to add the Android Compatibility package from Google. It's available via the SDK & AVD Manager and helps to develop for Android 3.0 without causing troubles with Android 1.6[3].

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

Testing

The first step to test an app is to run it on the emulator or device and debug it if necessary through the ddms tool. Android is built to run on different devices and OS versions without modification but hardware manufacturers might have changed pieces of the platform[4]. Therefore, testing on a physical device is paramount.

Automated Testing

To automate testing, the Android SDK comes with some capable and useful testing and instrumentation[5] tools. Tests can be written using the standard JUnit format with some Android mock objects that are contained in the SDK.

The Instrumentation classes can monitor the UI, send system events like key presses, et cetera. You can test for the status of your application after these events occur. The automated tests can be run on physical devices or in a virtual device. Open-source testing frameworks such as Robotium[6] can complement your other automated tests; it can even be used to test binary apk files, if the source is not available. A maven plugin[7] and a helper for the continuous integration server Hudson may also assist your testing[8].

Signing

Your application will always be signed by the build process, either with a debug signature or a real one. Your signature may be self-signed, so forget about signing fees (and security). The same signature is required for updates of your application.

4) *For an overview see e.g.* www.androidfragmentation.com
5) developer.android.com/guide/topics/testing/testing_android.html
6) code.google.com/p/robotium
7) code.google.com/p/maven-android-plugin/
8) wiki.hudson-ci.org/display/HUDSON/Android+Emulator+Plugin *and* hudson-

Distribution

After you have created the next killer application and tested it, you should put it in the Android Market. It 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 used by other app portals as a source for app meta data. Furthermore, it is part of the Google Backup Service[1], the Cloud to Device Messaging (C2DM)[2] and the License Verification Library (LVL)[3]. To upload your application to the Android Market, start at *market.android.com/publish*.

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

Make sure that you have defined a versionName, versionCode, an icon and a label in your *AndroidManifest.xml*. Furthermore, the declared features in the manifest (uses-feature nodes) are used to filter apps for different devices. As there are lots of competing applications in Android Market, you might want to use alternative application stores. They provide different payment methods and may target specific consumer groups[4].

Android 1.6 upwards also supports in-app purchase. This allows you to sell extra content, feature sets etc. from within your app by using the existing infrastructure of the Android Market[5].

1) code.google.com/android/backup/index.html
2) code.google.com/android/c2dm/index.html
3) developer.android.com/guide/publishing/licensing.html
4) www.wipconnector.com/index.php/appstores/tag/android
5) developer.android.com/guide/market/billing/index.html

Programming bada Apps

Samsung announced bada their new mobile platform in late 2009 and released it to the public in June 2010. This new platform has its foundation in Samsung`s proprietary platform. Building on more than 10 years experience with low end devices bada can run either on top of a Linux kernel for high-end devices or on real-time OS kernels for low-end devices.

Developer support is available from the developer site[1] offering forum, premium support and giving direct access to Samsung bada experts.

Samsung's Application store gives developer a route to market, but also specific features such as sales and download statistics, advertisement and a direct feedback channel for customers. The store is available in 3 ways, the website *www.samsungapps.com*, a client application on the handset and through a PC client, Samsung's Kies. Samsung's app store is available in over 120 Countries worldwide and has had over 100.000.000 downloads within the first 10 month since June 2010. Applications can be offered as paid apps or for free. It is possible to get revenues by placing advertisement in your app. In case of a paid application, you will receive 70% of sales.

Over 5 million bada phones were sold between June and December 2010. In the future, Samsung plans to sell about 20 million bada phones per year. Currently there are 6 bada-based

1) http://developer.bada.com

devices available with the Wave S8500 & Wave II being the current flagship devices and Wave 578 with NFC coming mid 2011.

Getting Started

To start developing for bada you need to register (for free) at *developer.bada.com* and download the latest bada SDK, which is currently only available for Windows only. The SDK includes the bada IDE (based on eclipse CDT), a simulator and a GNU toolchain to provide developers with a familiar development environment Samsung also offer a GUI tool available for UI design.

Before starting programming you should familiarize yourself with the application manifest which is a unique and needed for deploying application onto devices and distribute them into the store. A manifest can be generated and managed on *developer.bada.com* under the menu item “My Application”.

Implementation

After creating an application manifest one can start with App development with the bada SDK/IDE. Inside the IDE you will find a lot of example code, which you can copy with one click into your own workspace to familiarize yourself with Bada features and programming paradigm.

Native bada apps are developed in C++. Some restrictions apply however, for example, the language does not use exceptions. Instead, it return 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 to the developer. You will have to delete any pointer returned by a method ending in 'N'. You should also make sure that each of your own new variables has its delete:

```
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 only uses some parts of STL. Even though Samsung claims that you could use STL in your own code the current available STL implementation shipping with bada is missing some parts. You might need to use STLPort for full STL support. Similarly you could port modern C++ Libraries like Boost to work on bada, but the lack of RTTI and exceptions can make it hard.

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 all your UI must be positioned by hand or within the code. You need to provide a UI layout for the landscape and/or the portrait mode. The API also provides most standard classes for XML, SQL or Network and resembles a pretty complete framework. Also make use of the callbacks for important phone events in your application class, like low battery level or incoming calls.

If you want to write games for bada, the SDK supports OpenGL ES 1.1 and 2.0. Also the SDK wraps parts of OpenGL for use in its own classes, so you can also easily 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* where you will find great tutorials about coding for bada. There is an IRC channel #bada at *irc.freenode.net*, and of course there are groups for bada on most social networks.

Testing

The bada API offers its own logging class and an AppLog method, so you should make extensive use of logging in debug builds. The AppLog will show up in the IDE directly. With the IDE you can test and debug in the simulator or on a device. As mentioned in other chapters of this guide, we strongly recommend testing on real devices in general.

Otherwise you can never be sure how the app performs and it also might turn out that the 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 Eclipse-plugin.
Tools and frameworks for unit testing are available within the IDE/SDK. For details about these tools, check out the „bada *Tutorial.Development Environment.pdf*" included in the documents folder in the SDK base directory.

Distribution

The distribution will be done through Samsung's apps store and is the only distribution channel. Similar to Apples AppStore, there are quite strict acceptance rules applied which are described in the "Samsung Apps Publisher Guide", downloadable after you registered 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 you the inclusion of third party ad network contents in your bada application.

What Comes Next?

During the Developer Day at the Mobile World Congress 2011, Samsung revealed features of the upcoming bada 2.0 version. One of the key features will be Multitasking, which should make real background services possible. In addition, bada 2.0 will add support for Near Field Communication as well as the possibility for easy ad-hoc WiFi-P2P network setup from within the SDK.

Other interesting features will enhance the UIX like speech-to-text (STT) and text-to-speech (TTS) and support for 3D sound with OpenAL. Support for web-based application will also be extended by supporting more Javascript frameworks, HTML5 and a lot of API's from the WAC 2.0 standard within the Webcontrol.

Another great new feature will be the MIME-type registration for you application, so that you can register your application to the system for handling specific types like MP3. Beside the SDK updates the IDE will extend its tools with code coverage and performance monitoring functionality to support you code optimization.

Programming Native BlackBerry Apps

The BlackBerry platform developed by the Canadian company Research In Motion (RIM)[1] was launched in 1999. BlackBerry devices became extremely popular because they were traditionally equipped with a full keyboard for comfortable text input (which spawned a condition named BlackBerry Thumb[2]), their long battery life and more and more for Blackberry Messenger, their mobile social network offering. Add PDA applications such as address book, secure mail, calendar, tasks and memopad to these features and you will understand why the platform is very popular among business and mainstream users alike.

The marketshare of BlackBerry phones has declined somewhat in the US in 2011[3], but it is still a succesful 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

For the Blackberry OS, different 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.

For the next generation BlackBerry OS based on QNX Neutrino Realtime OS (RTOS), which RIM calls Tablet OS, and currently

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

only supported on ther the PlayBook Adobe AIR Flash and Web-Works programming are supported., Native C and Java SDKs have also been announced as well as an Android compatibility layer. This chapter focuses on Java development, for more information on WebWorks (web) and Flash programming please see the respective chapters in this guide.

Java SDK

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

IDE

For native Java development, you first need to decide which IDE to use. The modern option is to use Eclipse and the BlackBerry plugin[5], for previous BlackBerry OS versions you can also use the BlackBerry Java Development Environments (JDEs)[6].

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

Desktop Manager

You should download and install the BlackBerry Desktop Manager[7] also. It enables you to deploy your app package on a device for testing. For faster deployment, you might also use a tool called javaloader that comes with the JDE.

4) www.oracle.com/technetwork/java
5) us.blackberry.com/developers/javaappdev/javaplugin.jsp
6) us.blackberry.com/developers/javaappdev/javadevenv.jsp
7) us.blackberry.com/apps-software/desktop/

Coding Your Application

The BlackBerry JDE is partly based on J2ME and some of its JSR extensions: Integrated into the SDK is the MIDP 2.0 standard with popular JSR extensions that provides APIs for UI, audio, video, and location services among others[1]. This means that one option is to create BlackBerry apps with J2ME technologies alone.

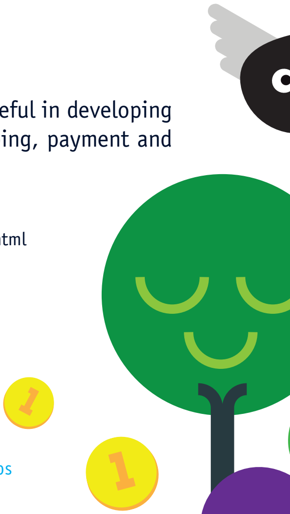
Another option is to use BlackBerry's proprietary extensions and UI framework that enable you to make full use of the platform. Native UI components can be styled to an extent, however they inherit their look from the current theme (unless you override the `Field.applyTheme()` method).

From OpenGL-ES over homescreen interaction to cryptography the BlackBerry APIs provide you with everything you need to create compelling apps. In addition to the official BlackBerry tools, there are third party extensions that enable you to enhance your apps, for example J2ME Polish[2] or WildBerry[3] enables you to design and animate your UI using CSS.

Services

BlackBerry offers many services that can be useful in developing your applications including advertising, mapping, payment and push services[4].

1) www.blackberry.com/developers/docs/6.0.0api/index.html
2) www.j2mepolish.org
3) www.wildberry-project.org
4) http://us.blackberry.com/developers/platform



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 service[1] works as follows: Your server sends a data package of up to 8KB to the BlackBerry push infrastructure.

The infrastructure then broadcasts the message to all or a group of clients (for content such as a news report) or to one specific client (for content such as a chat message). The device client then receives the message through BlackBerry's Push API and may confirm the reception 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.

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

Porting

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

- Some classes and functionalities are available on specific OS versions only. For example the FilePicker that is used to choose a file is available on OS 5.0 onwards only.
- 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. Self-written classes for server communication and storage among others might be reused on J2ME and Android in certain cases. In general, cross-platform portability strongly depends on how frequently your app uses native BlackBerry components. For example it is not possible to reuse BlackBerry push services classes on other platforms.

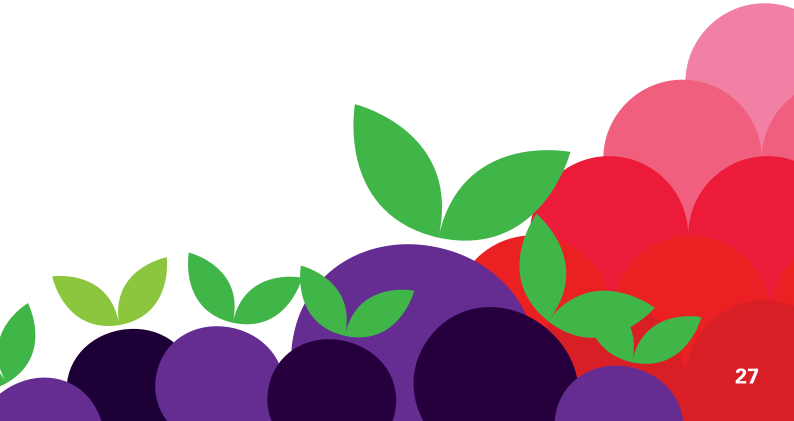
Signing

Many security-critical classes and features of the platform (such as networking or file APIs) require an application to be signed such that the publisher can be identified. To achieve this, you need to obtain a signing key directly from BlackBerry[1]. The signing itself is undertaken using the rapc tool which also packages the application.

Distribution

BlackBerry's own distribution channel is called App World[2] where you can publish your apps. For paid applications, you'll get a 70% revenue share. In addition, GetJar[3] 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

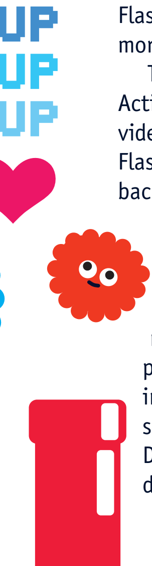
of a native 2D or 3D mobile application, a consequence of this approach can be better application performance.

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 all players 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.

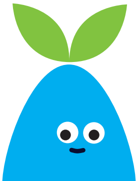
Pay special attention to the design of your Flash application. Adobe CS provides the option to choose between developing a browser-dependent or a standalone Adobe AIR application. An Adobe AIR application is essentially a Flash player, browser engine, and the native device's APIs wrapped into one executable 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 Gnash[1] and GameSWF[2] 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.

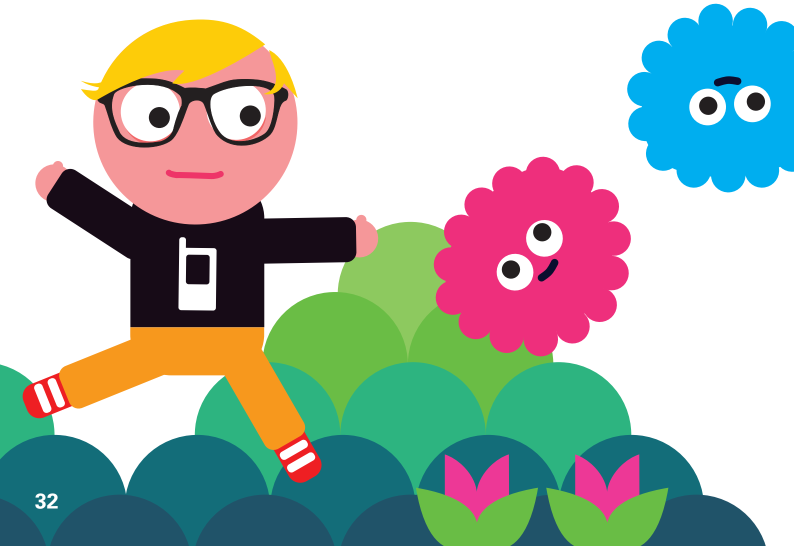
Testing

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 Widgets.

When the platform offers a built in Flash player that runs Flash content as an application the packaging requirements can be more complicated. At the simple end of the spectrum is Nokia Series 40, where the packaging requirements are quite simple[1]:

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

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

While some developers will do this manually, Nokia provides an online packaging service that does the heavy lifting for you[2].

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 a highly interesting and very popular development platform for many reasons, a commonly named one being the App Store. When it was introduced in July 2008, the App Store took off like no other marketplace did before. Now there are far more than 350,000 applications in the App Store, and the number is growing daily. This reflects the success of the concept but it also means that it is getting more and more difficult to stand out in this mass of applications.

Users have downloaded more than 10 billion iOS apps as of January 2011, and with device sales reaching new all-time highs just about each quarter, there is no sign of the current volume of about a billion downloads per month slowing down. Over 190 million sold devices in the hands of users willing to try apps and pay for content makes the App Store the most economically interesting target for mobile app development.

The iOS SDK offers high-level APIs for a wide range of tasks which helps to cut down on development time. New APIs are added in every major update of iPhone OS, such as MapKit in iPhone OS 3.0, (limited) multitasking in iPhone OS 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 skills acquired in iPhone development can be used in iPad development. A single binary can even contain different versions for both platforms with large parts of the code being shared. Since iOS version 4.2 was released in November 2010, all iOS devices currently sold use 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.

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 on the Mac App Store for USD 4.95.

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 allows the developer to test apps quicker than by deploying to a device.

The iOS SDK will work on any Intel-based Mac running Mac OS X 10.6 (Snow Leopard) or the yet to be released 10.7 (Lion).

A guide to get you started and introduce you to the tools is included, 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, among others:

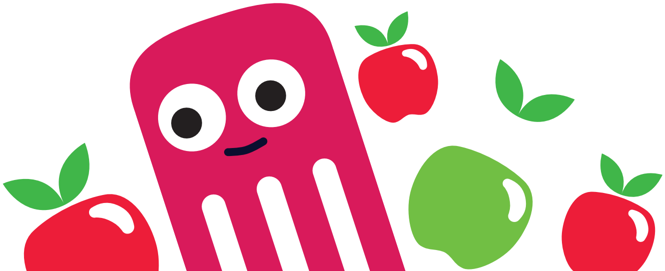
- Cocoa Touch, which consists of the UI libraries, various input methods such as multi-touch and accelerometer.
- Media frameworks, such as OpenAL, OpenGL ES, Quartz, Core Animation and various audio and video libraries
- Core Services, such as networking, SQLite, threading and various other lower level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

The list of available frameworks constantly 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 Kit[1], which Epic released for iOS to much fanfare in December 2010.

1) www.udk.com



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 MonoTouch[2]. 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 what the original SDK has to offer and still allowing code re-use, for example when creating similar Windows Phone 7 apps.

Some of those alternative IDEs might, however, carry additional fees in addition to Apple's yearly $99 for access to the develop program and Apple's 30% cut of all sales. As the perceived quality of mobile apps comes in large part from a usable and beautiful user interface, cross-platform development using third party IDEs makes the most sense for games, which can share almost all their code between different platforms. Java IDE makers JetBrains recently released an Objective-C IDE of their own, called AppCode, which is still in beta stage but looks to provide advanced features.

2) www.monotouch.net

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 workflow[2]. Also check out some of the sample code that Apple provides online[3] to find out more about various APIs available to you.

Testing

As performance in the iPhone Simulator may be superior to actual devices by several orders of magnitude, it is absolutely vital to test on devices. It is highly recommended that you have at least one device available to test on for each class of devices you want to deploy your apps on.

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

For example, an iPhone-only app shouldn't need to be tested separately on an iPad, when an universal app would. However, it cannot hurt to have several classes of devices, 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. For end-user testing you can distribute builds of your application to up to 100 testers through Ad-Hoc Provisioning, which you can set up in the Program Portal[4]. Each iPhone (and iPad/ iPod touch) has a unique identifier (UDID – universal device identifier), which is a string of 40 hex characters based on various hardware parts of the device.

If you choose to test using Ad-Hoc-Provisioning, simply follow Apple's detailed set-up instructions[5]. 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 to beta testers. There are open source projects[6] to facilitate this new feature, as well as commercial services[7].

Google Toolbox for Mac[8] runs the test cases using a shell script during the build phase, while GHUnit[9] 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

4) developer.apple.com/iphone/library/referencelibrary/GettingStarted/Learning_Objective-C_A_Primer/index.html#//apple_ref/doc/uid/TP40007594
5) developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhone101/Articles/00_Introduction.html
6) github.com/therealkerni/hockey
7) www.testflightapp.com
8) code.google.com/p/google-toolbox-for-mac
9) github.com/gabriel/gh-unit

the SDK Apple included OCUnit; an example of how to create the unit tests is available online[1].
In version 4 of iOS 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 Library[2]. Several other third party testing automation tools for iPhone applications are available as well, including FoneMonkey[3] and Squish[4].

Distribution

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

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

After Apple has approved your application, which usually shouldn't take more than 2 weeks, your app will be available to customers in the App Store. Due to 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

1) www.mobileorchard.com/ocunit-integrated-unit-testing-in-xcode
2) developer.apple.com/library/ios/#documentation/DeveloperTools/Reference/UIAutomationRef/_index.html
3) www.gorillalogic.com/fonemonkey
4) www.froglogic.com/products

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. Use this only as a guideline.

Books

Over the past years, a number of great books have been written on iOS development. Here is a short list of good tutorials and references which makes no attempt to be complete:

Beginner books

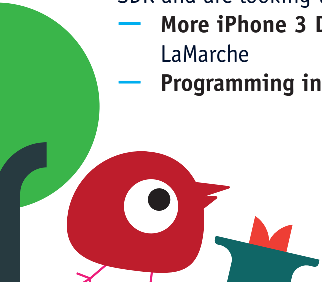
These books are best for someone looking into getting started with iOS development.

— **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 skills.

- **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.

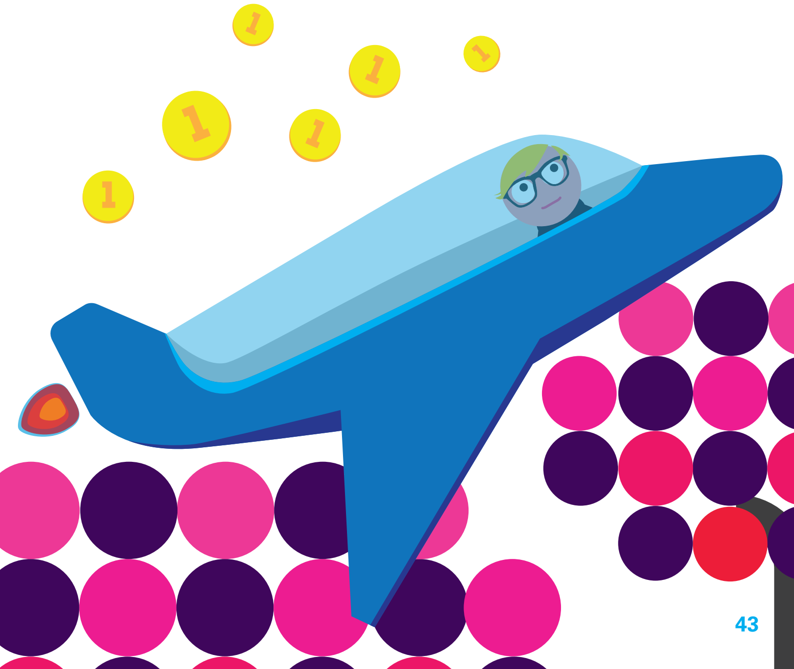
This is even more visible ever since Twitter and Github gained momentum and became widely-known.

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 list[1]. Following a list like that means staying up to date on current issues and generally interesting things about iOS development.

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 applications 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 the 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 SDK[1] (not the Java Runtime Environment) and an IDE of your choice, such as Eclipse Pulsar for Mobile Developers[2], NetBeans[3] with its Java ME plug-in or IntelliJ[4].
- An emulator, such as the Wireless Toolkit[5], the Micro Emulator[6] or a vendor specific SDK or emulator.
- Depending on your setup you may need an obfuscator like ProGuard[7]. If you build applications professionally you will probably want to use a build tool such as Maven[8] or Ant[9] also.

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

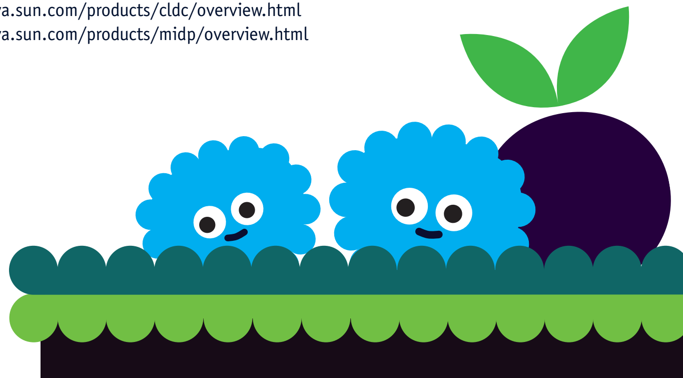
- You may want to check out J2ME Polish, the open source framework for building your application for various devices[1].

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], which are both quite easy to understand. These form the basis of any J2ME environment and provide a standardized set of capabilities to all J2ME devices. As

1) www.j2mepolish.org
2) java.sun.com/products/cldc/overview.html
3) java.sun.com/products/midp/overview.html



both CLDC and MIDP were designed a decade ago, the default set of capabilities they provide is rudimentary by today's standards.

Manufactures can supplement these rudimentary capabilities by implementing various optional Java Specification Requests (JSRs). JSRs exist for everything from accessing the device's built in calendar, address book and file system (JSR 75) to using the GPS (JSR 179) and Near Field Communication (JSR 257). For a comprehensive list of JSRs related to Java ME development, visit the Java Community Process' "List by JCP Technology"[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 applications, in which case 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 287[1].

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 Polish[2]: This tool separates the design in CSS and you can use HTML for the user interface. It is backward-compatible with the highlevel LCDUI framework
2. LWUIT[3]: A Swing inspired UI framework
3. Mewt[4]: Uses XML to define the UI
4. TWUIK[5]: 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/index.php?s=twuik

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

Open Source

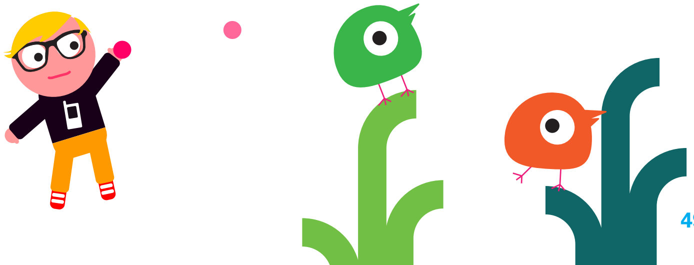
There is a rich open source scene in the J2ME sector. Interesting projects can be found via the blog on *opensource.ngphone.com*. You will also find fascinating projects on the Mobile and Embedded page of *java.net*[6], for example the Bluetooth project Marge[7].

Testing

Because of the fragmentation in the various implementations of Java ME, testing your 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 you have to test on devices.

Thankfully, vendors like Nokia and Samsung provide subsidized or even free remote access to selected devices[8].

6) mobileandembedded.dev.java.net
7) marge.dev.java.net
8) www.forum.nokia.com/rda *and* innovator.samsungmobile.com/bbs/lab/view.do?platformId=2



Automated Testing

There are various unit testing frameworks available for Java ME, including J2MEUnit[1], MoMEUnit[2] and CLDC Unit[3]; System and UI testing is more complex given the security model of J2ME, however JInjector[4] is a flexible byte-code injection framework that supports system and UI testing. Code coverage can also be gathered with JInjector.

Porting

One of the strengths of the Java environment for mobile devices is that it is backed by a standard, so it can be implemented by competing vendors. The downside is that the standard has to be interpreted, and this interpretation process can cause differences in individual implementations. This results in all kinds of bugs and non-standard behavior. In the following sections we outline different strategies for porting your applications to all Java ME handsets and platforms.

Direct Support

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

1) www.j2meunit.sourceforge.net
2) www.momeunit.sourwceforge.net
3) snapshot.pyx4me.com/pyx4me-cldcunit
4) www.code.google.com/p/jinjector

Lowest Common Denominator

You can prevent many porting issues by limiting the functionality of your application to the lowest common denominator. In the J2ME world this usually means CLDC 1.0 and MIDP 1.0. If you only plan to release your application in more developed countries/regions, you may consider targeting CLDC 1.1 and MIDP 2.0 as the lowest common denominator (without any additional APIs or JSR support).

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

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

Porting Frameworks

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

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

For Java ME some of the options you can choose from are:

Celsius from Mobile Distillery[1] that is licensed per month, Bedrock from Metismo[2] that provides a suite of cross compilers on a yearly license fee and J2ME Polish from Enough Software[3] that is available under both the GPL Open Source license and a commercial license. Going in the other direction (from C++ to Java ME) is also possible with the open source MoSync SDK[4].

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.

1) www.mobile-distillery.com
2) www.metismo.com
3) www.enough.de
4) www.mosync.com

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 largely depends on the implementation.

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

Applications signed by the manufacturer of a device enjoy the highest security level and can access every Java API available on the handset. Applications signed with a carrier certificate are similarly trusted.

Applications signed by JavaVerified[5], Verisign[6] or Thawte[7] 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 Ovi Store by Nokia does.

Another option is to consider using a testing and certification service provider and leaving the complexity to them. Intertek[8] is probably the largest such supplier.

5) www.javaverified.com
6) www.verisign.com
7) www.thawte.com
8) www.intertek.com/wireless-mobile

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:

— Handmark[1] and Mobile Rated[2] provide carrier and vendor independent application stores.
— GetJar[3] is one of the oldest distributors for free mobile applications – not only Java applications.
— LG distributes apps on *www.lgapplication.com*
— Ovi Store[4] targets Nokia users worldwide and provides a revenue share to the developer at 70% from credit card billing and 60% from operator billing
— Carriers are in the game also, such as Orange[5] and O2[6].

Basically almost everyone in the mobile arena has announced an app store.

An overview of the available app stores (not those selling J2ME apps alone) can be found in the WIP App Store Catalogue[7].

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) www.o2litmus.coma
7) www.wipconnector.com/appstores/

Furthermore there are various vendors who provide solutions for provisioning of Java applications over a Bluetooth connection, including Waymedia[1] and Futurlink[2].

1) www.waymedia.it
2) www.futurlink.com

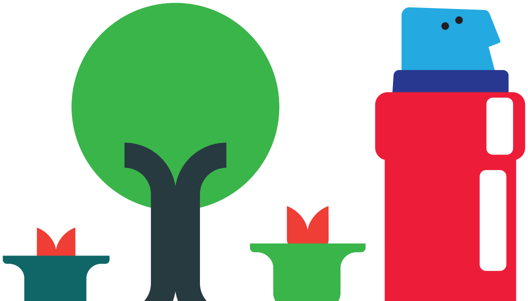
Programming MeeGo Apps

MeeGo is a mobile platform, launched by Intel and Nokia at Mobile World Congress in 2010. The concept behind MeeGo is one of a modular mobile operating system; consisting of a core OS module, which is the same for all MeeGo platforms, and specialized system modules for the supported platforms. Currently there are six platform targets: MeeGo Handset, MeeGo Tablet, MeeGo Netbook, MeeGo Smart TV, MeeGo In-Vehicle and MeeGo Media Phone.

MeeGo is a Linux based OS, which is the successor to Nokia's Maemo and Intel's Moblin. As a result it shares traits of both; Based on a SuSE Kernel (as was Moblin), it uses a UI platform based on Nokia's Qt Framework.

Currently, Intel is the main driving force behind MeeGo, as Nokia has announced that Windows Phone will be its primary high-end smartphone platform. But Nokia is still supporting MeeGo, having stated it has a role in exploring disruptive mobile technologies, and is bringing its first MeeGo device to market in 2011.

The lack of devices is the biggest problem for the platform right now, but as it is highly customizable and offers modular design, many companies are already interested in MeeGo.



Prerequisites

You can start developing for MeeGo using the MeeGo SDK[1] or Qt[2]. The SDK includes Qt Creator, which is the main IDE for developing for MeeGo and Qt. One nice feature of Qt is that apps able to run on MeeGo will run on other Qt platforms, such as Symbian and desktop computers. The SDK is available for Linux, Windows and Mac.

Implementation

MeeGo apps are based on Qt. Qt offers C++ based app development combined with Qt Quick that provides a new declarative UI technique based on the QML language. Qt offers a full selection of UI controls and backend classes for file IO, network communication and other tasks. Qt Mobility adds a set of 12 APIs to access features and data on a device, such as location information and contacts records. For more information about Qt, please see the respective chapter in this guide.

For the UI QML offers the most modern approach. It even allows building MeeGo apps purely in JavaScript since it is a JavaScript-based UI modeling language. It is possible to mix QML and Qt, so you can have a C++ backend coded in Qt connected with a QML UI.

Games can be implemented in OpenGL ES 2.0, and Qt offers OpenGL support of its own. QML is rendered through OpenGL, and will be rendered in Qt 4.8 with the help of an Open Scene Graph — currently it is implemented with Qt GraphicsView.

You can use the MeeGoTouch Library, which is build on top of Qt, to develop touch friendly Qt applications for MeeGo. But this is not recommended, as MeeGoTouch is seen by the com-

1) www.meego.com/downloads
2) http://qt.nokia.com

munity as (almost) deprecated. Techniques like QML and Qt are more powerful and MeeGoTouch is not supported on other Qt platforms.

Testing

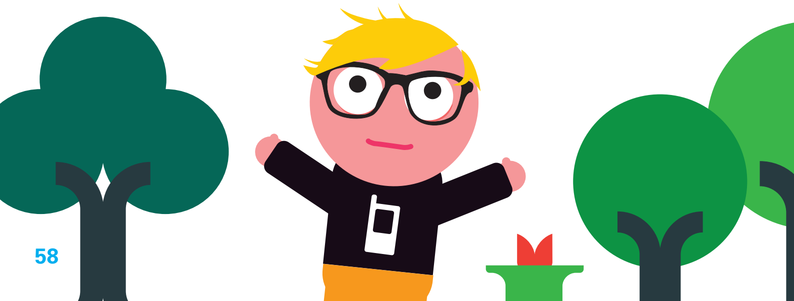
Qt offers support for testing through the QTestLib. There are also a number of testing libraries for C++, such as CPPUnit or boost::test, but most of them will need to be built on MeeGo first before they are usable. Besides, you might want to check out the respective QA pages in the MeeGo wiki[1].

Intel offers a number of free tools to analyze and optimize your MeeGo app from a Windows or Linux host system[2]:

- **Intel C++ Compiler:** A cross compiler which will build your binary for MeeGo.
- **Vtune:** Intel's profiling tool, which will analyze your C++/Qt Code during runtime on MeeGo.

With the Qt Creator IDE you can test and debug in a computer-based simulator or using a device.

1) http://wiki.meego.com/Quality
2) *Available at* http://appdeveloper.intel.com/en-us/meego-sdk-suite



Distribution

MeeGo as an OS is held by the Linux Foundation and has Nokia and Intel as partners. Both companies will offer their own app stores. AppUp[1] is Intel's cross platform app store, which is already offering some MeeGo apps. In order to use AppUp as a distribution platform, you need to install Intel's AppUp SDK[2] on top of the MeeGo SDK. Your app will be packaged as an rpm file for AppUp on MeeGo.

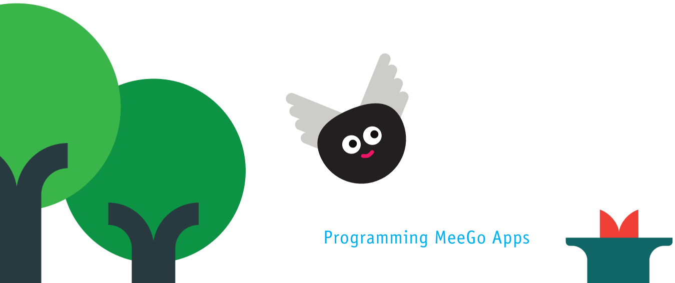
Nokia will offer MeeGo apps through its Ovi Store[3], as it already does for Java ME, Symbian and Maemo apps.

Both stores have their own validation processes.

Learn More

If you want to find out more about MeeGo, the best places to start are *meego.com/community* and *appdeveloper.intel.com/meego*. More information can be found at *www.forum.nokia.com/MeeGo*. For those of you who speak German, *www.meetmeego.org* is a good source. There is also an IRC channel #meego at *irc.freenode.net*. The official Twitter account is @meegocom.

1) www.appup.com
2) http://appdeveloper.intel.com
3) www.ovi.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 on the desktop 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 devices. Today, Qt can be used to create applications for devices based on Symbian, Maemo and MeeGo – the open source platform initiated by Nokia and Intel. In fact, Qt can now be thought of as a platform in its own right – you will create a Qt application and deploy it to devices 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, a 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 burden by the usual C++ housekeeping. For example, instead of callbacks, a paradigm of signals and slots is used to simplify communication between objects[1]; the output from one object

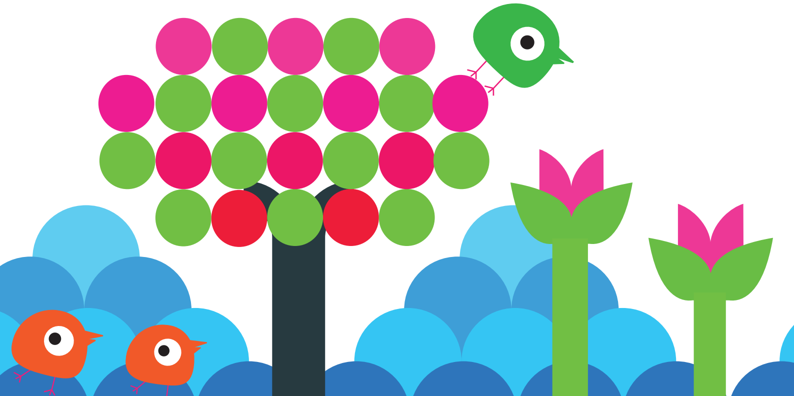
1) doc.qt.nokia.com/4.7-snapshot/signalsandslots.html

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 Qt Quick content or creating GUIs (using the QWidget object), 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 Gitorious[1].

1) qt.gitorious.org



Prerequisites

Qt SDK installs everything you need to create, test, and debug applications for Symbian and Maemo from a single package. It's also future proofed for MeeGo apps development. All versions offer tools for compiling Symbian and Maemo 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 QML, 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 device's location or read a contact record it has been necessary to revert back to the platform's native APIs. This is where the Qt Mobility APIs come in. The APIs provided by Qt Mobility offer a common interface to device data such as contacts, location, messages, NFC, and several others.

This means that if you, for example, need the device's location the same API will obtain the location information on both a Symbian and Maemo device. (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 is quite straightforward. The following code, for example, shows that only a few lines are needed to access a device'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 device accelerometers are transposed between Symbian and Maemo devices. 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 on 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 create your UI. Qt has many interesting features, such as WebKit

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

Testing

Qt SDK includes a lightweight simulator enabling applications to be tested and debugged on the development computer (Qt SDK runs under Microsoft Windows, Ubuntu Linux and Apple Mac OS X). The simulator includes tools that enable device 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 device testing.

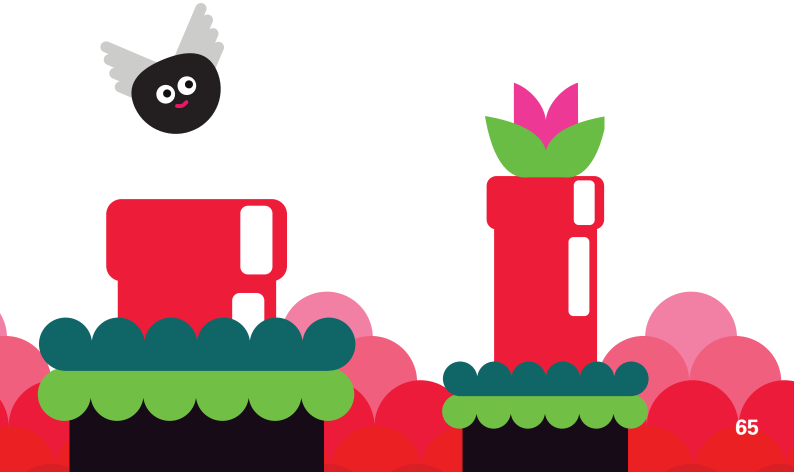
In addition, the Qt SDK includes tools to perform on-device debugging on Symbian and Maemo devices. This feature can be handy to track down bugs that come to light only when the application is running on a device. Such bugs are rare and tend to surface in areas such as comms, where the Qt simulator uses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

QTestLib 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*

Packaging

For a Qt application to run on a mobile device the Qt API framework has to be present. The Nokia N900 has the Qt APIs built in. In addition, Maemo and MeeGo devices provide 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 devices the situation is a little different. Symbian^3 devices have the APIs built in. However, Symbian does not include a built-in mechanism to add the APIs to earlier devices 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 device, 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 devices.



Signing

As Qt applications install as native applications on Symbian and Maemo devices they need to comply with each platform's signing requirements. In the case of Maemo this means that signing is not required. For applications to be installed on Symbian devices, signing is necessary. If you choose to use Ovi Store to distribute your apps, Nokia will organize for your app to be Symbian Signed, at no cost.

The process is straightforward and described in full in the Distribute section of the Forum Nokia website[1], but in summary:

- You sign up as an Ovi publisher
- You provide up to five device IMEIs and request a UID for your application
- The Ovi 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 devices elected and test it to ensure it complies with the Symbian Signed Test Criteria[2]
- Once tested, you submit an unsigned copy of the app to the Ovi publishing portal

1) www.forum.nokia.com/Distribute/Packaging_and_signing.xhtml
2) http://wiki.forum.nokia.com/index.php/Symbian_Signed_Test_Criteria_V4_Wiki_version

Distribution

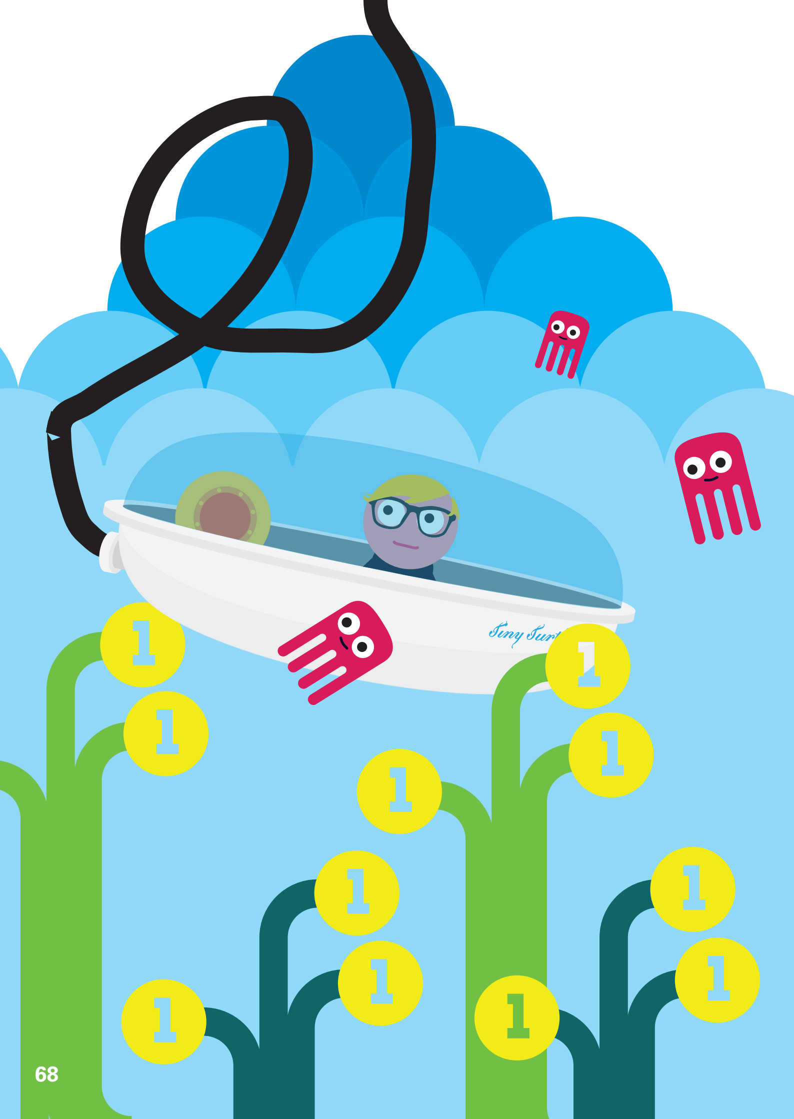
Ovi Store is the latest iteration of the Nokia app store solution, with a history stretching back to 2003, and has grown to deliver 5 million downloads a day. Importantly, once an application has met the store's quality requirements – beyond removing indecent or illegal applications – there is no restriction on the types of applications that can be distributed.

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

To use Ovi 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.)


Tiny Turt
1
1
1
1
1
1
1
1
1
1
1

Programming Symbian Apps

The Symbian platform[1] is a software platform for mobile devices. 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 C++ still offers a viable development option with over 200 million compatible devices in use and Nokia forecasting additional sales of at least 150 million.

As a third-party developer you can create applications and middleware in Symbian C++, the native programming language of the Symbian platform. Symbian C++ is a specialized subset of C++ with Symbian-specific idioms[2].

However, it has a steep learning curve and your first steps can be more frustrating than in other environments. It is for this reason that Nokia is promoting Qt as the primary application development for Symbian (and MeeGo).

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.

That said, native Symbian C++ APIs provide the most comprehensive access to device 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. For these reasons Symbian C++ is still of interest to many developers.

1) symbian.nokia.com
2) http://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 devices[1] and install it
2. Install ActivePerl version 5.6.1.638[2] from the SDK
3. Install Carbide.c++[3]

Linux and Mac OS X are not officially supported platforms. One way around this is to use a virtual machine that hosts Windows. Other options are more complex, but information can be found online[4].

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.

1) www.forum.nokia.com/Develop/Other_Technologies/Symbian_C++/Tools
2) found in the Symbian^3 SDK at epoc32\tools\distrib\ActivePerl-5.6.1.635-MSWin32-x86.msi
3) www.forum.nokia.com/Develop/Other_Technologies/Symbian_C++/Tools/
4) www.forum.nokia.com

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 devices, analysis tools, and more.

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 devices based on the corresponding version, that is the APIs in the Symbian^3 SDK will work on all Symbian^3 devices. 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 build, debug and run your first native application and download it to a Symbian phone, without having to write a single line of code.

Testing

For automated unit testing, googletest[5] 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 device 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,

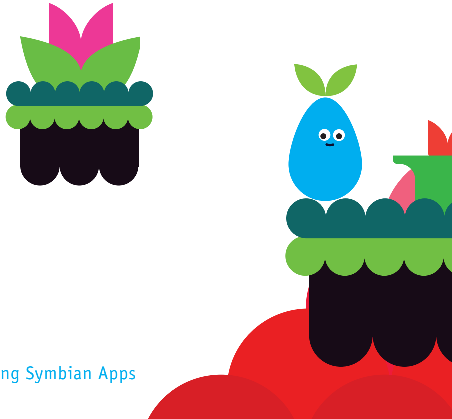
5) www.code.google.com/p/googletest

it is necessary to sign an application during development in order to install it to a device: 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 Ovi Store (see the chapter on Programming Qt applications for more information on Ovi Store). For a limited number of applications, those using more advanced APIs, it will be necessary to obtain Certified Signed through the Symbian Signed website[1].

Distribution

Nokia Ovi Store will probably be your primary distribution channel (see the chapter on Programming Qt applications 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



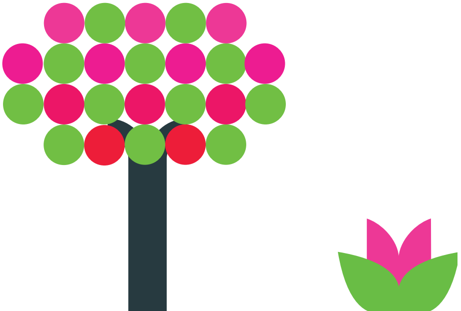
1UP
1UP
1UP

Programming webOS Apps

WebOS is a multitasking mobile operating system based on a Linux kernel introduced January 2009 by Palm. Palm released two webOS handsets: the Palm Pre and Palm Pixi. In Spring 2010, Palm was bought by Hewlett Packard who are now introducing webOS based tablets to increase the OS market share (in Q1 2011, only 4% of US smartphones supported webOS[1]).

Since early 2011 the prototype-based framework "Mojo" is replaced by "Enyo". Enyo improves the performance of your apps and enhances the layout handling. Probably the biggest advantage of the platform is that there are very few coding prerequisites needed. It is very easy to get started, especially when you have a little experience with web developing: WebOS applications can be built using common web standards like JavaScript, HTML and CSS. To implement advanced technologies like 3D graphics, you will need to code in C language.

1) http://blog.nielsen.com/nielsenwire/online_mobile/who-is-winning-the-u-s-smartphone-battle/



Prerequisites

As a first step download the HP webOS SDK (which also includes a PDK installer) and VirtualBox (needed for the Palm Emulator) from *developer.palm.com*. The SDK/PDK is available for Windows, Mac and Linux.

There is no need to sign up for an account at this point, but you will need one for publishing your application later on. It is also helpful to have a look into the user interface guidelines before starting your project.

Implementation

Before you start developing you need to decide which Development Kit can fulfill your needs.

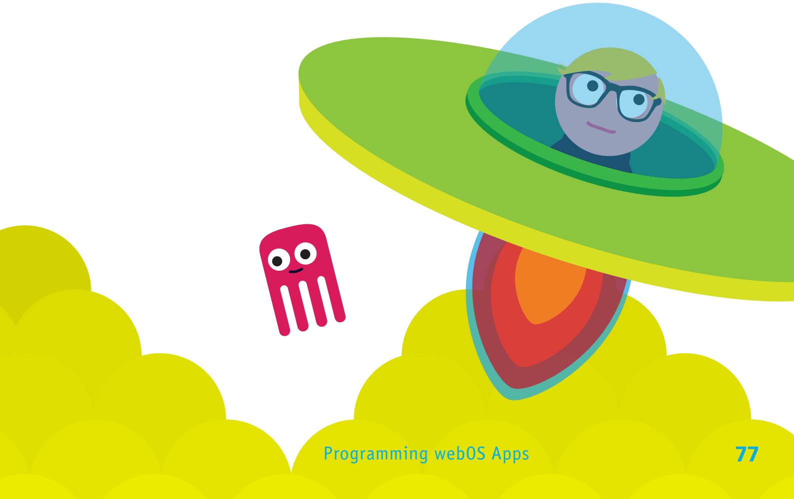
SDK (Software Development Kit)	PDK (Plug-in Development Kit)
Javascript, CSS, Prototype based framework (Enyo)	C, C++
No compiling needed (no compiling errors)	Porting applications from other platforms
Easy developing for web developers	Implementing 3D graphics with OpenGL and SDL
Ares, an online interface builder via drag&drop and a built-in editor	Integrate C/C++ code into apps built with web technologies
Swap between Ares and hand-built workspace (since Enyo)	Eclipse plugin
Source is viewable by users	Pre-configured Xcode template with headers and libraries for MAC machines, source is not viewable for users

To use Ares[1], Palm's browser based IDE, you need to sign up for a free account at *developer.palm.com*. The source code is saved online so you can access it and continue your work from anywhere you want.

Some major Ares features:

- Drag-and-drop interface builder
- Code editor
- Visual debugger
- Log viewer
- Source control integration
- Drag-and-drop calls to phone services and sensors
- Preview apps in the browser
- Run apps directly on the webOS emulator or device (requires SDK installation)

1) ares.palm.com



Enyo vs Mojo

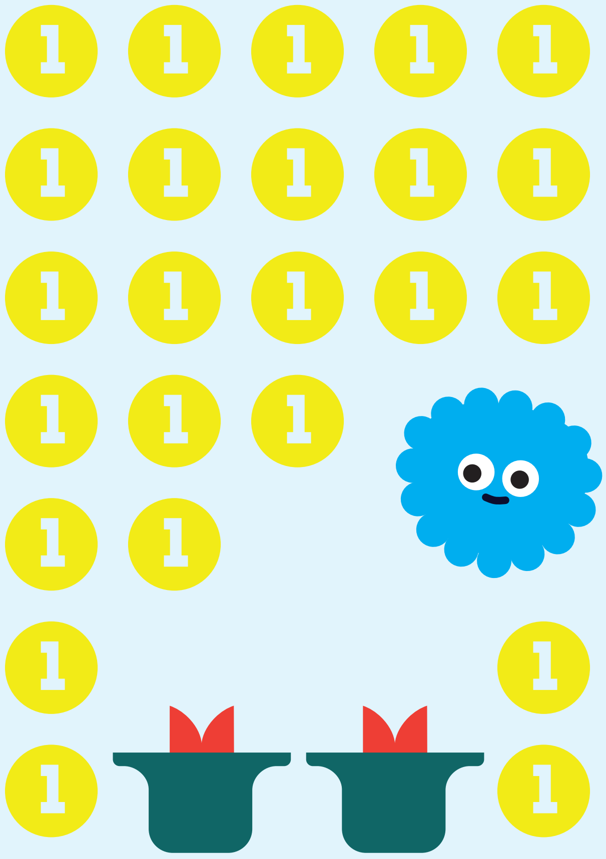
With webOS 3.0 a new framework called Enyo was introduced to replace Mojo. Compared to Mojo, Enyo has a shortened structure and gets rid of the assistants and scenes. The sources are centralized into one sources folder.

Enyo

- Project
  - Source
    - test.js
  - CSS
    - style.css
  - Images
  - appinfo.json
  - depends.js
  - icon.png
  - index.html

Mojo

- Project
  - App
    - Assistants
      - stage-assistant.js
      - test-assistant.js
    - Views
      - Test
        - test-scene.html
  - Stylesheets
    - style.css
  - Images
  - appinfo.json
  - sources.json
  - icon.png
  - index.html


1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1
1

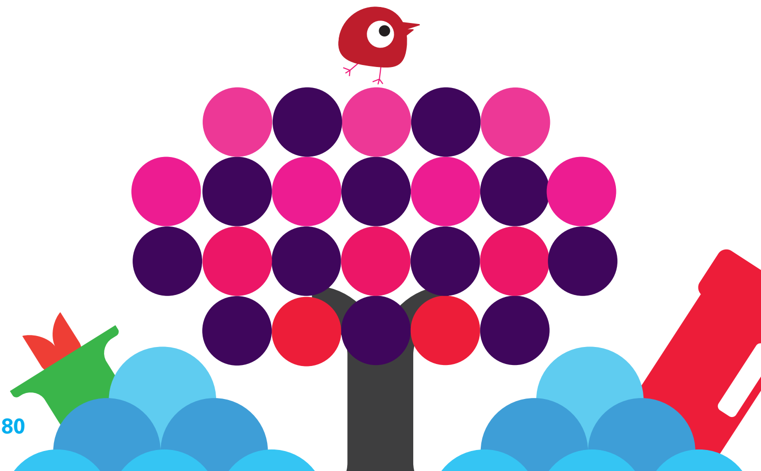
You will notice that the source folder contains only your JavaScript files, the sources.json file is replaced with the depends.js and there are no HTML files needed but index.html. All controllers are structured in the components properties of the kinds.

Here is the comparison of depends.js and sources.json:

Enyo	Mojo

`enyo.depends(`
`  “source/test.js”,`
`  “css/style.css”`
`);`
	`[`
	` {“source”: “app\/assistants\/stage-assistant.js”},`
	` {“source”: “app\/assistants\/test-assistant.js”,`
	` “scenes”: “Test”`
	` }`
	`]`

So every time you add new files to the application, you also need to add them to depends.js.



Let us have a look at the index.html:

```
<!doctype html>
<html>
  <head>
    <title>App Title</title>
    <script src=”../../enyo/enyo.js”
type=”text/javascript”>
  </script>
  </head>
  <body>
    <script type=”text/javascript”>
    new test().renderInto(document.body);
    </script>
  </body>
</html>
```

The `<head>` element contains a `<title>` tag with the application name. There is also a `<script>` tag referencing the location of the Enyo framework file (make sure to set the path to enyo.js to match the location of your Enyo installation).

Within the `<body>` element is a call to instantiate a new FeedReader object, which will then be rendered onscreen.

```
enyo.kind({
  name: “test”,
  kind: enyo.VFlexBox, components: [
    {kind: “PageHeader”, content: “Enyo Test
Application”},
    {kind: “RowGroup”, caption: “Test Group”,
components: [
      {kind: “FancyInput”, components: [
        {kind: “Button”, caption: “Press me”,
onclick: “btnClick”},
        ]}
      ]}
    ],
    btnClick: function() {
      // handle the button click
    }
});
```

Taken as a whole, FeedReader.js defines a kind called “test”. That is to say, it contains instructions for creating new test objects.

The kind property tells you that a test object is a view that inherits from enyo.VFlexBox. (A VFlexBox is a box that stacks content vertically). The components property will eventually contain UI elements that you want to show when you instantiate a new test.

Then you add a header (of kind PageHeader) and a text input box (of kind FancyInput) which includes an button.

Testing

There are various ways of testing and debugging. For testing you can always run your application on the device or on the emulator using these command-line tools.

- **palm-generate:** Generates an empty project with the necessary files and directory structure.
- **palm-package:** Creates an installable application package.
- **palm-install:** Installs an application package on the device.

The debugging tools depend based on which development kit you decide to use:

1. If you are building with web technologies the most common and easiest way of debugging is on-browser-development: You can run your application on any (webkit standards) browser and use its native tools for debugging (e.g. Chrome inspector). This is especially useful for developing for different devices and different screen resolutions. Just resize the browser to the required resolution and the application automatically behaves like on a device with that resolution.
2. The Palm Inspector is another SDK debugging tool. It allows to examine the DOM of an application running on the emulator. First you need to launch the app using the -i option with the palm-launch command (palm-launch -i com.myapplication.app), then start the Palm Inspector
3. WebOS also provides on-device debugger C,C++ apps. In order to use them, you need to shell into the device or the emulator. Use novaterm on Mac OS X and putty (putty -P 10022 root@localhost) on Windows. Type debug to launch the debugger, or gdb for a standard linux debugger. You can use breakpoints, display variables, and trace the stack.

Distribution

Be sure to have a look into the app submission checklist on *developer.palm.com* to ensure you are following the HP guidelines. Based on your type of application you will have to choose between the SDK and the PDK app submission checklists. Before you submit an application you will need to create a developer account. There are three different types, all of them are free. Since you cannot change the membership type after the fact, choose carefully.

After uploading your application it takes approximately 7 days for HP to review and publish your app. Your app can now be purchased in the webOS app catalogue.

Account Type	Fees	Publish Software
Full account	Free	Yes
Community account	Free	No
Open source account	Free	Yes

	Access Forum	Revenue Share	Notes
	Yes	70%	
	Yes	N/A	This type is obsolete. It has no advantages but you cannot publish any software.
	Yes	70%	All software must be open source

Programming Windows Phone Apps

Microsoft has made a fresh start with the Windows Phone 7 platform. The Windows Mobile operating system was declining in both user acceptance and market share, so the need for innovation was clearly felt. Windows Phone is geared towards consumers as much as business users, and has a simple user interface that is focused on typography and content. A marketing budget of 500 million USD has been spent to promote the new platform and 1.5 million handsets were sold in the first six weeks after launch[1].

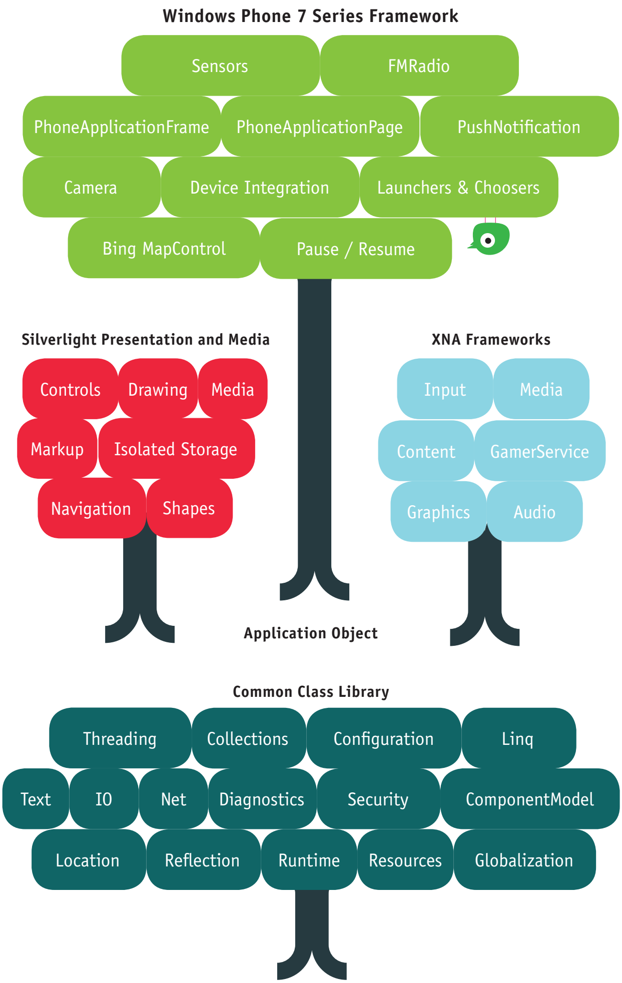
In February 2011, Nokia announced a partnership with Microsoft that underlines the future relevance of the platform: Windows Phone will be the first choice smartphone platform for Nokia devices from now on. However, the first Nokia Windows Phone devices are not expected before 2012.

Development

Windows Phone development is undertaken in C# or VB.NET, using the Microsoft Visual Studio IDE. Depending on the type of application you are developing, you will use one of two platforms – Silverlight for event-driven applications or XNA for games driven by a "game loop". The UI for Silverlight applications can be created either in Microsoft Visual Studio or Microsoft Expression Blend.

The Windows Phone Developer Tools are free of charge and include "Express" (feature-limited) editions of both Visual Stu-

1) http://www.microsoft.com/Presspass/Features/2010/dec10/12-21AchimBergQA.mspx


Windows Phone 7 Series Framework
Sensors
FMRadio
PhoneApplicationFrame
PhoneApplicationPage
PushNotification
Camera
Device Integration
Launchers & Choosers
Bing MapControl
Pause / Resume
Silverlight Presentation and Media
Controls
Drawing
Media
Markup
Isolated Storage
Navigation
Shapes
XNA Frameworks
Input
Media
Content
GamerService
Graphics
Audio
Application Object
Common Class Library
Threading
Collections
Configuration
Linq
Text
IO
Net
Diagnostics
Security
ComponentModel
Location
Reflection
Runtime
Resources
Globalization

dio 2010 and Microsoft Expression Blend. A notable limitation is that the free tools support C# development only. The tools also include a device emulator to run code against. The device emulator uses hardware acceleration and as such performs reasonably well when running 3D XNA games.

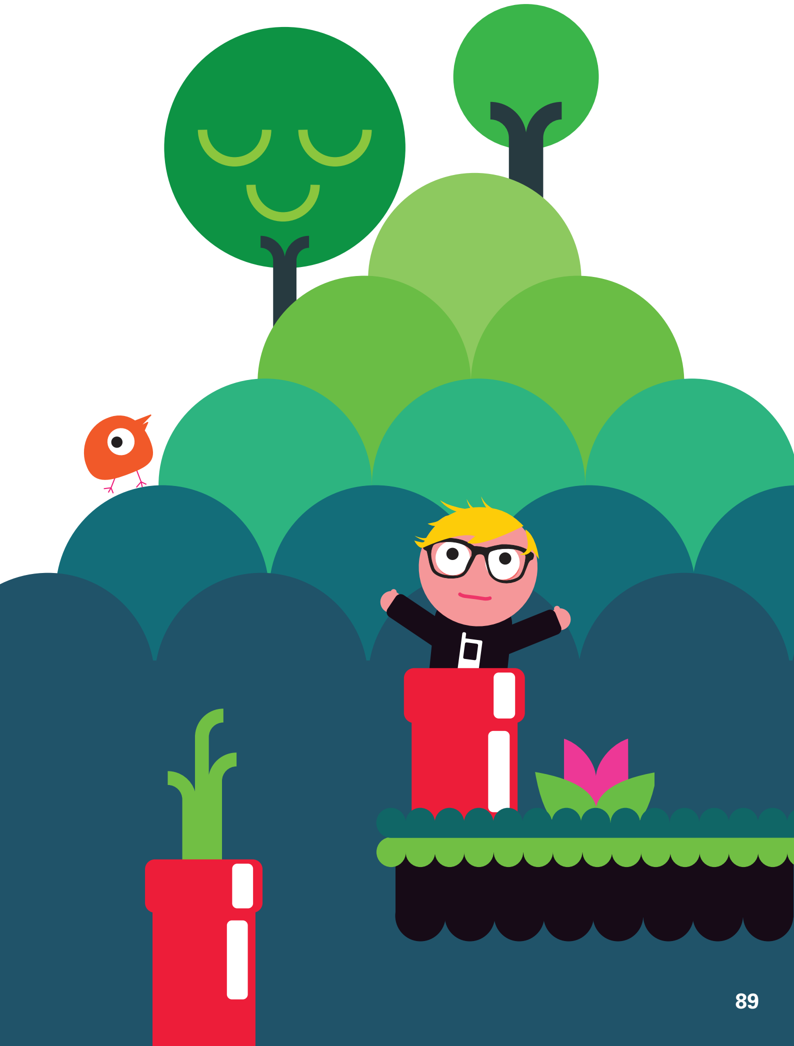
It is important to consider which platform you should leverage when building your application.

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 7 controls.	...you want to manage art assets such as models, meshes, sprites, textures, or animations.
...you want to target both Windows Phone 7 and the web, re-using some code.	...you want to target Windows Phone 7, Windows, and Xbox 360, re-using lots of code.

However, in the next major update of the platform – Windows Phone 7.5 due for release during fall 2011 – it will be possible to use Sliverlight and XNA in the same application.

Functions and Services

Windows Phone applications have access to input such as location, multi-touch screen, accelerometer, and microphone. Available services include media playback and push notifications that can update "live tiles" (animated application widgets that reside on the start page of the phone). The Windows Phone 7.5 update will add features such as a limited form of multitasking similar to the iOS platform, pinning custom content to the start



page, access to raw camera feed, TCP/IP sockets, and a database engine, among other improvements[1].

Currently Windows Phone 7 does not support multitasking, and instead encourages developers to implement best practices regarding saving and restoring an application state ("tombstoning"). This means the onus will be on the developer to cache and restore things like data and UI state when resuming the application, to create the appearance of an application that never stopped running.

In contrast to Windows Mobile, developers cannot execute native code or access the Windows API directly. While this ensures applications are sandboxed and cannot do anything that will permanently affect the usability of the phone, it restricts the extensibility of the platform and arguably limits the type of applications that can be developed. For example, core platform features such as the dialer and on-screen keyboard can generally not be replaced or extended.

There are currently several native controls that are not included in Silverlight for Windows Phone, such as context menu, date picker, and others. At *silverlight.codeplex.com* you can download an intermediate solution if you want to use such controls.

Distribution

Applications for Windows Phone 7 are distributed through a single endpoint, Microsoft's 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.

1) http://channel9.msdn.com/Events/MIX/MIX11/KEY02

Testing And Analytics

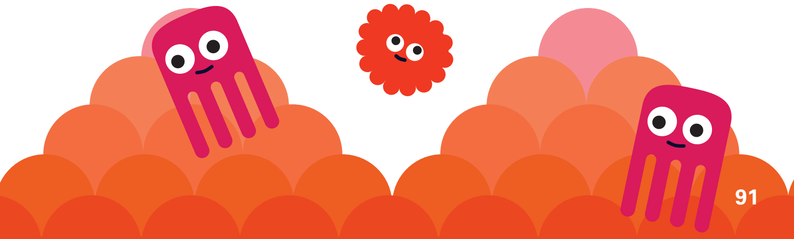
You can unit test applications using the Windows Phone Test Framework[1] or the Silverlight Unit Test Framework[2]. Check the web for several public articles on unit testing Silverlight and Windows Phone applications[3].

For developers wishing to collect runtime data and analytics, there are several options. Localytics[4] and PreEmptive Solutions[5] both provide analytics tools and services that are compatible with Windows Phone 7. Developers can also use the Silverlight Analytics Framework[6] to connect to a variety of third-party tracking services such as Google Analytics.

Resources

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

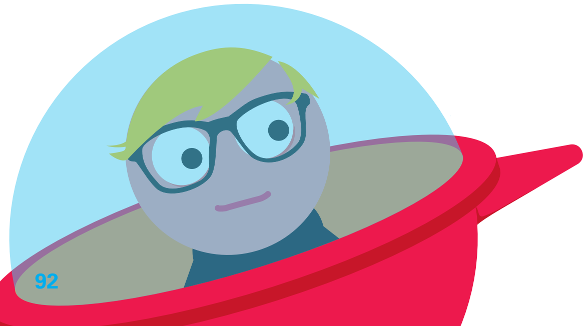
1) www.wptestlib.codeplex.com
2) www.silverlight.codeplex.com
3) live.visitmix.com/mix10/sessions/cl5 *and* dotnet.dzone.com/news/test-driven-development *and* www.smartypantscoding.com/a-cheat-sheet-for-unit-testing-silverlight-apps-on-windows-phone-7
4) http://www.localytics.com/app-analytics/
5) http://www.preemptive.com/windowsphone7.html
6) http://msaf.codeplex.com/



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 which 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 of a widget 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 like native applications.

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 Ovi Browser. In these widgets some JavaScript execution can be performed on the device. In the case of Series 40 web apps the Ovi Browser client can run code that implements element transitions and enables dynamic alterations to the UI on the device. In addition to offering a richer user experience this hybrid approach reduces round trips to the server, for example by eliminating the need for the server to paint every screen refresh.

If there is a challenge in creating widgets, it is the lack of universal support for a common standard. W3C, together with Wholesale Application Community (WAC) and Joint Innovation Lab (JIL), is pushing forward with the definition of standards. This standardization is still underway and information on its progress can be found in the W3C Wiki[1].

Because the standards are not complete, it is important to note that each widget 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*
- **Ovi Browser (selected Nokia Series 40 devices):** *www.forum.nokia.com/webapps*
- **Vodafone360:** *bit.ly/vf-widgets*
- **WAC/ JIL:** *www.jil.org*
- **Windows Mobile (v6.5):** *bit.ly/winmo-widgets*

To port your widget over to platforms that don't natively support widgets, such as iPhone or Android, you can use tools such

as PhoneGap[1], Titanium from Appcelerator[2], and Rhomobile[3] among others. Of these options, PhoneGap offers a solution that is closest to the W3C approach.

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-

1) www.phonegap.com
2) www.appcelerator.com
3) www.rhomobile.com

ons to web authoring tools, as with the BlackBerry Widget SDK[1] which works in conjunction with Adobe Air, or standalone tools such as Nokia Web Tools[2]. These tools generally provide template projects, a preview environment, validation, packaging, and deployment features.

Writing Your Code

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

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

You can start by simply:

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

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

Depending on the widget platform you are targeting you may be able to use more advanced technologies such as Canvas, SVG,

1) us.blackberry.com/developers/browserdev/widgetsdk.jsp
2) www.forum.nokia.com/Develop/Web/Tools#NWT

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

Signing

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

Distribution

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

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

Distribution via a website is not the only option. Many application stores welcome widgets. As we went to press, the only store that supports W3C widgets is the Vodafone Widget store[1], but by packaging your widgets appropriately you can upload them into Nokia Ovi store[2], the Windows Marketplace[3] or RIM BlackBerry AppWorld[4]. 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, iOS, Symbian, webOS and Windows Phone are among the most important smartphone 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 don't 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 taking a cross platform approach, even one that concentrates 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 or adopting a cross platform strategy.

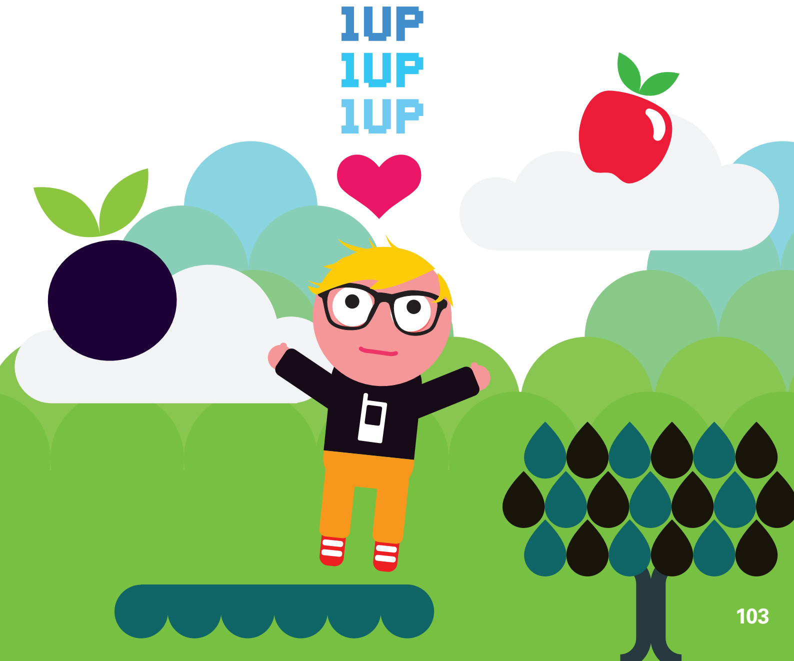
By the way, we are not talking about app stores here; this is a different market fragmentation problem. The more than 100 app stores, from operators, manufacturers and independent companies create challenges of their own, outlined in the Appstores chapter.

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
C, C++	4	3	First class: bada, Brew MP, Symbian, Windows Phone Classic Second class: Android (partly, using the NDK), iOS (partly), webOS (partly)
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: webOS Second class: BlackBerry (Widget), Nokia (WRT)
Objective-C	1	0	iOS

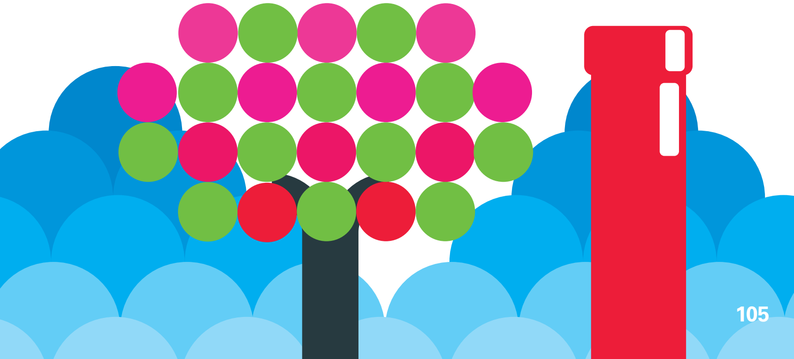
1) Supported natively by the platform, e.g. either the primary or only language for creating applications

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, Black-Berry 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 we have a limited selection of background tasks that may continue to run after the app's

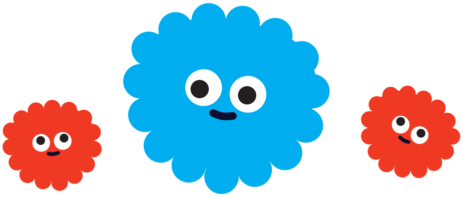
exit. And then there is a freezing and unfreezing mechanism on Windows Phone (although an announced update to the platform will bring background services during 2011). So if background services can improve your app's offering, you should evaluate cross platform strategies carefully to ensure it enables 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 some 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.



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 in one platform only.

Often player concepts tend to use a '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 automatically. 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:

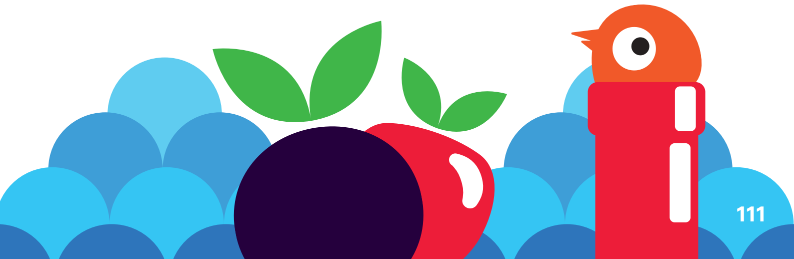
1. You can put a web app in an app store. Even when not directly supported by the vendor, you can use web based tools such as PhoneGap in combination with a web app solution to make web apps available in app stores.
2. Web apps can work offline.
3. Web apps can look and behave in a similar fashion to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

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.



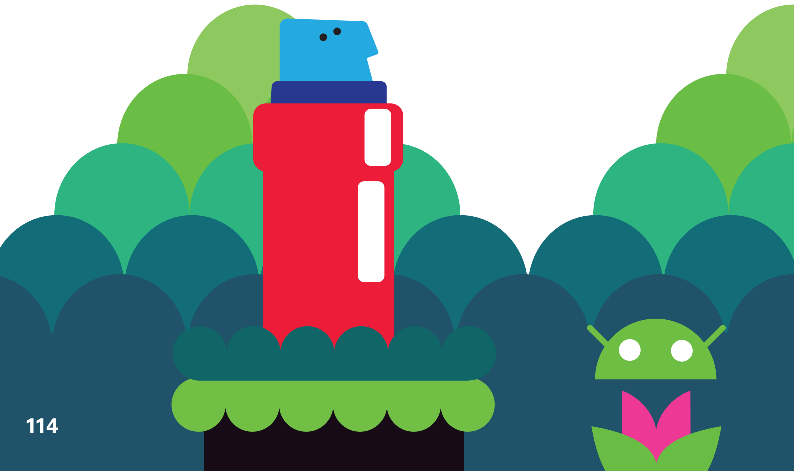
Cross-Platform Solutions

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*

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 (Ansca Software)	Commercial	JavaScript	Android, iOS
EDGELIB www.edgelib.com (elements interactive)	Commercial	C++	Android, iOS, PC, Symbian

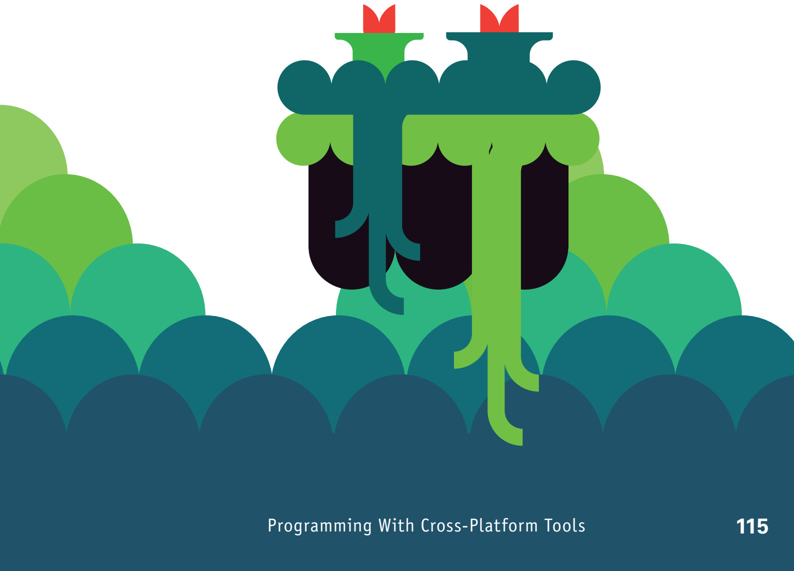
Solution	License	Input	Output
id Tech 5 www.idsoftware.com (id)	Commercial	C++	Consoles, iOS, PC
Irrlicht irrlicht.sourceforge.net gitorious.org/irrlicht-android	Open Source	C++	Android & iOS with OpenGL-ES version, PC
J2ME Polish www.j2mepolish.org (Enough Software)	Open Source + Commercial	Java ME, HTML, CSS, JavaScript	Android, Black-Berry, iOS, J2ME, PC, Windows Phone Classic
Flash adobe.com/devnet/devices.html (Adobe)	Commercial	Flash	Android, iOS, PC
Mono Touch monotouch.net (Novell)	Commercial	C#	iOS
MoSync www.mosync.com (MoSync)	Open Source + Commercial	C	Android, J2ME, Moblin, 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++	MeeGo, PC, Symbian, 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, Black-Berry, iOS, Sym-bian, Windows Phone Classic

Solution	License	Input	Output
SIO2 sio2interactive.com (sio2interactive)	Commercial	C	iOS, other announced
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 (Whoop)	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



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.

- 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?



Creating Mobile Websites

Why create a mobile website instead of an application? The discussion on web vs. apps is still ongoing and it depends on your particular project which approach is the better choice. But aside that discussion one thing is undoubtedly clear: the impact of mobile internet is constantly increasing.

Using the web has a number of advantages, websites can be browsed on almost all devices, the technology is flexible, and it is easy to update sites so all users get the latest version. Because of this experts such as like Sundar Pichai, vice president for product management for Google Chrome, forecast that the web will prevail. Other see web delivering 70% of applications while 30% will remain with apps running on a device. However the distribution eventually looks: If you want to participate effectively in the growing mobile web market you have to follow some guidelines. This chapter will explain what those guidelines are.

Usability In A Limited Environment

The usage pattern at a desktop computer is often described as “hunt and gather“. Mobile usage is totally different. When using internet on a mobile handset, users are usually on the move. They usually want to know more about their surrounding or occupy some spare time. Therefore, mobile internet usage is often described as “quick fulfillment“.

Applications have to take these differences into account. For example, a search interface for users at a desktop browser has to offer comprehensive options, on the mobile handset it has to be more straight-forward, focused on the primary action.
Other things to consider during design are that the mobile user has no mouse, often they have no real keyboard and the size of

the screen is very limited. This means the content of a mobile website has to be arranged accordingly: Images should not be too large, all relevant elements should be easily accessible as the user is not able to move a cursor freely around the site.
In addition to factors like markup, image formats and navigation, you should never forget about the most valuable feature to a mobile user: battery life. Complex websites with many JavaScript, CSS and Flash elements need a lot of processing power, which means battery power.

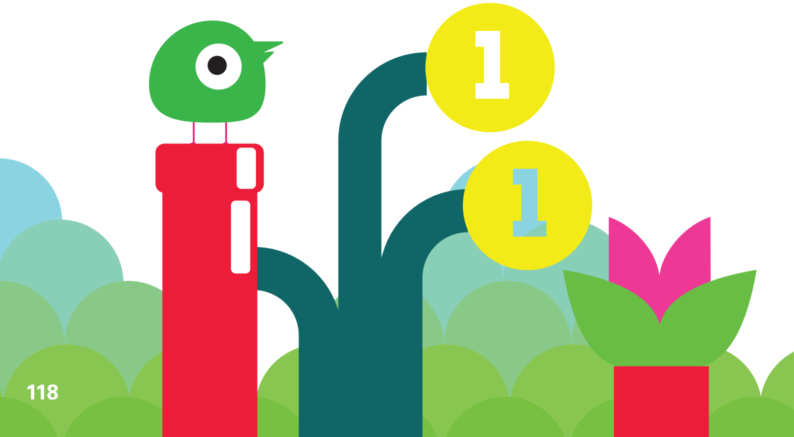
These adjustments cannot be realized by a machine. This is one reason why it is indispensable to create special mobile versions of websites rather than relying on proxy browsers to reformat content.

These 10 basic hints should help you to adapt your content properly for the mobile user:

1. **Mobilize, don't miniaturize:** 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.
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.
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). Avoid formats such *as .doc and* pdf.

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 1cm x 1cm.
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. Users will often click the first link in a list anyway.

Analyze Your Target Markets

“In mobile, fragmentation is forever”[1]. When programming a mobile website, the simple truth is that you have to deal with a fragmented device market. Do you know what kinds of devices your visitors use most? If not, you should start at the low-level and implement a tool that detects the devices that are accessing your site.

Several providers are offering this kind of analysis software:

- **AdMob:** *www.admob.com*
- **Bango:** *www.bango.com*
- **Sevenval:** *www.device-trends.com*
- **TigTags:** *www.tigtags.com/mobile_analytics/mobile_site_tracking*

Some providers also offer access to the data they collect, which provides you an insight into which devices are accessing the mobile web:

- **AdMob:** *http://metrics.admob.com*
- **mobiForge:** *http://mobiforge.com/designing/story/effective-design-multiple-screen-sizes*
- **Opera:** *www.opera.com/smw*
- **Sevenval:** *www.slideshare.net/sevenval/tag/devicetrends*
- **StatCounter Global Stats:** *http://gs.statcounter.com*

When using these data, bear in mind where they come from and how they have been generated: In which region have they

1) Richard Wong, www.techcrunch.com, Mar. 4, 2010

been collected, are they really reflecting the users' mobile web usage or just the general market share of various devices? Is it possible that the data reflect the user base of certain operators or technology platforms only? Are these data for page impressions, visits or unique users?

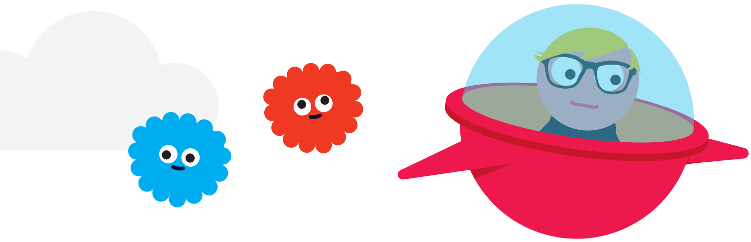
Never trust any report blindly. Use several sources and do your own analysis.

Content adaptation

Static Version

Of course you can simply ignore the possibilities of automatic optimization and leave it up to the users: Create different versions of your content, let the user start with a low-level version and let them decide manually which version they prefer for their devices and usage patterns.

However, since you are dealing with a user who is on the move and does not want to spend a lot of time discovering which version suits them best, this is probably not the best way to go.



Automatic adaption technologies

To adapt the content to different devices, you basically need two components: The first contains logic that detects the device and has information on its browser and that of the browser's characteristics – a device database. The second component uses these characteristics to adjust the content for optimal usability.

The most popular open source device database is the WURFL project[1] that offers comprehensive information via XML. A number of tools deliver APIs for detecting the browser, gathering its properties and adjusting the content: the markup and the images. One example of an open source project that provides adjusting content is MyMobileWeb[2], financed by public funds and Telefonica.

There are also commercial providers. Some concentrate on device data and detection while others focus on offering software platforms that adjust the content accordingly.

Examples of commercial device data and device detection providers are:

— **dotMobi**[3] offers several APIs to access their device database DeviceAtlas[4] and intelligent detection
— **DetectRight**[5]

1) wurfl.sf.net
2) mymobileweb.morfeo-project.org
3) www.mtld.mobi
4) www.deviceatlas.com
5) www.detectright.com

While, some commercial content adaptation software providers are:

- **Sevenval**[1], a platform independent technology that works via HTTP and markup. This solution is available as a service or can be installed on Linux systems
- **Volantis Mobility Server**[2] runs on a variety of Java-based web application servers and SQL-compliant databases. There are two versions available: one open source community version and one professional version
- **Mobile Interaction Server**[3] runs on BEA, IBM WebSphere, JBoss, Tomcat and Caucho Resin

HTML Standards For Mobile

Today the vast majority of all mobile browsers support XHTML/MP 1.0, for many low-end devices this language still delivers the best results. On the other hand, many devices offer full web browsers that support HTML/4.0, but these browsers still have their differences when it comes to content-type or doctype and choosing views that distinguish between mobile and desktop websites.

The next big thing is HTML5 which is still under development. However, some of the latest browsers already support parts of this new standard: Scripting, GPS access, CSS3 elements, CSS animations and the possibility of offline (on-device) data storage.

1) www.sevenval.com
2) www.volantis.com
3) www.mobileaware.com

At the same time, the W3C Device APIs and Policy Working Group (DAP) is defining client-side APIs. These APIs will make it possible to develop web applications and web widgets that interact with device services such as calendar, contacts and camera among others. Such websites might be expected to progressively evolve to offer the features of web widgets (for more information on widgets, see the Programming Mobile Widgets chapter).

Websites For Feature Phones

It is likely that the majority of your users will access your site with a basic handset such as described by the W3C in the Default Delivery Context[4] or by Luca Passani in his baseline device description[5].

To fit the needs of feature phones, these are our recommendations:

- XHTML MP/1.0
- screen size: 176x144 pixels
- GIF and JPEG image
- maximum of 20kb page size (including images and CSS)
- basic Table Support, without “rowspan”, “colspan“ — no nested tables
- minimal CSS (WCSS 1.0)

Although even these basic characteristics exceed the capabilities of many older handsets, mobile internet users usually own handsets that properly display pages that follow the guidelines above. Surfing the internet with much older devices is no fun anyway.

4) www.w3.org/TR/mobile-bp/#ddc
5) www.passani.it/gap/#baseline

Websites For Full Web Browsers

Full web browsers are characterized by their capability to display any website. These include browsers based on WebKit, Opera Mobile, Microsoft Internet Explorer, Netfront (version 3.4 and later), UCWEB, Nokia Web Browser and Fennec. Devices that use a full web browser don't necessarily require any technology concessions in website design, but we suggest the following guidelines:

- HTML/4.0 strict
- screen size: 240x320 pixels
- GIF, JPEG Images and PNG (without alpha transparency) images
- maximum of 50kb page size (including images and CSS)
- basic table support, without "rowspan","colspan" — no nested tables
- keep CSS simple, tables are not bad
- script can be included, but avoid unnecessary or excessive script as well as animation effects — think of the battery
- if available, use AJAX to get content.

Websites For Touch Devices

Many modern devices with a powerful browser such as the iPhone, Android or Symbian handsets also offer a touch screen. Touch UIs offer users the possibility of moving more freely within a website. At the same time, touch interaction makes choosing one line from a list and other smaller elements within a page difficult.

If your mobile site complies with the following requirements, any touch device user will be able to navigate properly:

- HTML/4.0 Strict
- screen size: 320x480 pixels
- GIF, JPEG and PNG (with alpha transparency) images
- maximum of 100kb page size (including images and CSS)
- full tables
- script is allowed, but not needed
- use CSS, especially Webkit styles for rounded corners
- if available, use AJAX to get content
- keep the limited battery life in mind
- use a large font size (for example 18px) for links and clickable lists
- set a default viewport (`<meta name = "viewport" content = "width = device-width">`).

Satisfy The Browser

Markup

Of course it would be great if there were one universal markup standard – unfortunately this is not the case. There are many standards, so be sure of validating your markup by using one of these tools:

- **W3C Markup Validator:** *http://validator.w3.org*
- **W3C mobileOK checker:** *http://validator.w3.org/mobile*
- **dotMobi testing tool:** *http://mobiready.com*
- **Korean MobileOK Test and Validation Service:** *http://v.mobileok.kr*

As general advice, it is recommended that you stick to UTF-8 encoding. You can also consider going low-level creating a XHTML MP/1.0 site with WCSS/1.0 and without JavaScript.

Page Width

Always use dynamic layout. Avoid static width settings in pixels, it is better to use percentage values. Even when using device databases, the browsers still have different display methods (such as full screen, landscape and portrait) and not all provide for displaying a scrollbar. The web is dynamic, so is the hardware landscape – keep your layout dynamic as well.

Images

Not everybody has a mobile data flat rate, so do not use images excessively, avoid any unnecessary images. To reduce data and processor workload, the images should always be scaled on the server and not by the browser. ImageMagick offers functionality to easily scale your images[1].

When choosing image formats, GIF and JPEG are still the best options. PNG offers more flexibility, but you cannot always be sure to what degree transparency is supported.

Tables

For desktop web pages, tables are no longer used for web design. In the mobile environment they are still an effective way to create simple layouts – just avoid unnecessary nested tables and colspan/rowspan. Even though it breaks the rules for valid XHTML MP/1.0, some browsers need the attributes `cellpadding="0"` and `cellspacing="0"` to prevent unwanted spaces from being displayed. CSS simply cannot assure this with all browsers.

1) www.imagemagick.org

CSS

Do not use CSS excessively. CSS interpretation is sometimes not properly implemented and shortens battery life. To be on the safe side, stick to the WCSS/1.0 set.

When determining sizes, avoid defining them in pixels, use percentage values.

Fonts

Don't be too adventurous with fonts: Mobile browsers usually support a limited set of font-types. Better to limit the number of fonts and focus on the font size to create differentiation. Use relative values (small/medium/large) rather than fixed values (pixels).

Cookies

You can use cookies and should do so, but only when really needed. However, don't put too much trust in cookies: Although it might work fine during one session, the cookie might no longer be available at the start of the next.

This is why you should always offer alternatives such as an URL based parameter or a personalized bookmark for permanent settings.

Script / AJAX

According to *www.device-trends.com*, 85% of the mobile web users who visited the participating websites, were using a browser that supports JavaScript and most of them were able to handle AJAX. So it would be prudent to use the possibilities of these technologies, but you should make sure that your site works fine without them as well.

Meta Settings

You can influence the rendering mode by a number of meta settings[1]:

Viewport	Defines view and zoom	`<meta name="viewport" content="width=device-width; initial-scale=1.0;"/>` www.quirksmode.org/mobile/tableViewport.html
MobileOptimized (Windows Mobile)	Defines width of mobile page	`<meta name="MobileOptimized" content="width" />` msdn.microsoft.com/en-us/library/ms890014.aspx
HandheldFriendly (RIM Browser)	Deactivates zoom	`<meta name="HandheldFriendly" content="true"/>` docs.blackberry.com/en/developers/deliverables/6176/HTML_ref_meta_564143_11.jsp
Format Detection	Numbers will not automatically be displayed as phone numbers	`<meta name="format-detection" content="telephone=no"/>` msdn.microsoft.com/en-us/library/ms890014.aspx
Auto Match (RIM Browser)	Numbers will not automatically be displayed as phone numbers	`<meta name="x-rim-auto-match" http-equiv="x-rim-auto-match" forua="true" content="none"/>`

1) Please compare learnthemobileweb.com/tag/handheldfriendly

Using GPS

Devices like the iPhone (firmware version 3 and later), the Blackberry (firmware 4.2. and later) Opera Mobile and browsers with Google Gears enable the use of GPS information within the browser. There is a W3C specification published[1], but unfortunately this is not supported by all devices. Nevertheless you can still use a simple JavaScript API that overlays the different implementations[2].

For further information about how to use GPS location information for web-based apps and services, check out mobiforge[3].

Hybrid Apps

The term "hybrid" is used in many technology fields. The auto industry uses it where a internal combustion engine is combine with a generator in an electric vehicle. The concept is the same for hybrid web apps.

In the mobile space you find that hybrid web apps combine a classic web browser with the features of a native application. These may include:

- a completely native app that is controlled by the downstream content (arguably this is just a browser)
- a client that in addition to its existing function, displays content in a web view format
- an application framework that enables the execution of web content that can interface with native application code, such as by utilizing the WebKit integration provided in Qt.

1) dev.w3.org/geo/api/spec-source.html
2) code.google.com/p/geo-location-javascript
3) www.mobiforge.com/developing/blog/location-location-location

Testing your Mobile Website

There are several ways to test your mobile website:

User-Agent / Browser

Several desktop browsers offer the ability to change the user-agent and thereby enable the emulation of device detection for the testing of automatic adaption. For Firefox users, the User-Agent Switcher is available[4]. MobiForge offers a configuration file for this Add-On which contains some properties of mobile handsets[5].

Emulators / SDKs

Emulators or SDKs offered by manufacturers are the better option for testing. We have had good results using the following tools:

- **Apple iPhone:** *developer.apple.com/iPhone/program*
- **Palm Pre:** *developer.palm.com*
- **Android:** *developer.android.com*
- **BlackBerry:** *www.blackberry.com/developers/downloads/simulators*
- **Windows Mobile:** *msdn.microsoft.com/en-us/windowsmobile*
- **Opera Mini:** *www.opera.com/mini/demo*
- **Symbian SDK:** *www.forum.nokia.com/S60SDK*

Additional emulators can be found on
mobiforge.com/emulators/page/mobile-emulators

4) addons.mozilla.org/en-US/firefox/addon/59
5) www.mobiforge.com/developing/blog/user-agent-switcher-config-file

Remote And Real Device Testing

It is best to test the usability of a site under real-life conditions: Take your mobile handset to a busy public place and try to find all the relevant information in your application by using one hand. You will see very quickly where your mobile site may need to be trimmed or optimized. Test on as many different devices as possible.

Remote testing is an alternative, particularly where your access to handsets is limited. Consider services like *www.deviceanywhere.com* or *www.perfectomobile.com*.

Crowd sourcing testers, with the help of *www.mob4hire.com*, is also worth considering.

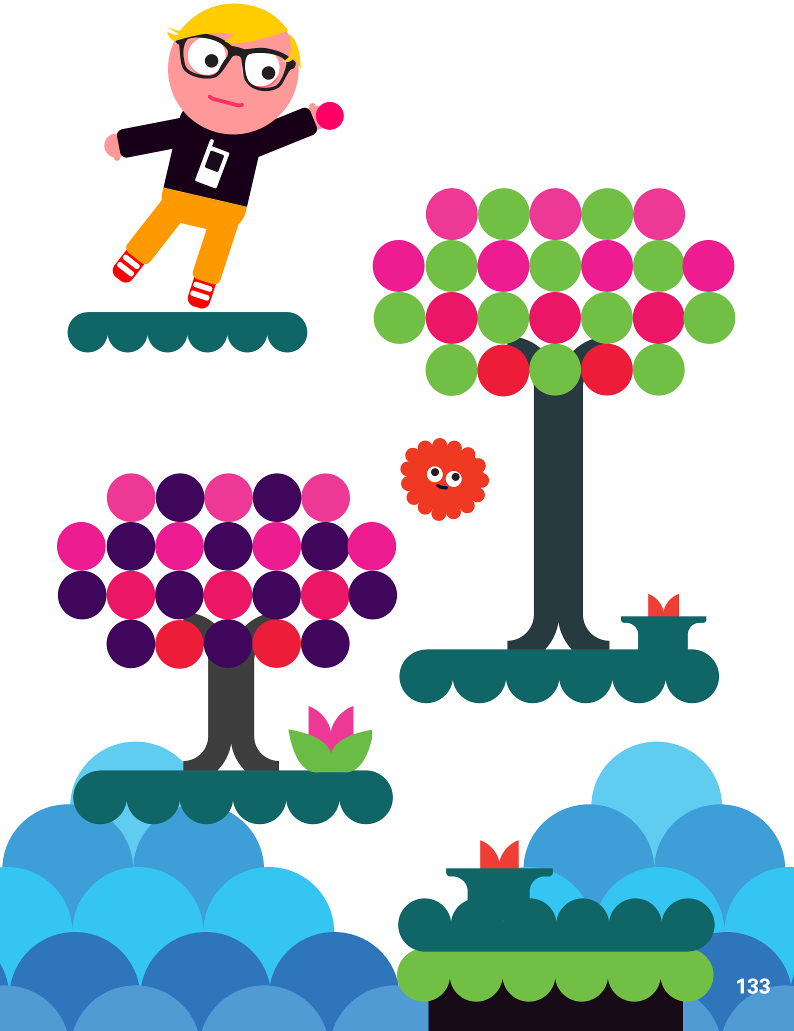
Learn More – On The Web

If you want to dig deeper and learn more about how to satisfy the mobile user with your web-based service, check out these websites:

- **Mobile Best Practices / W3C:** *www.w3.org/TR/mobile-bp*
- **dotMobi Mobile Web Developer's Guide / dotMobi:** *mobiforge.com/starting/story/dotmobi-mobile-web-developers-guide*
- **The Wireless FAQ / Andrea Trasatti and others:** *www.thewirelessfaq.com*
- **Global Authoring Practices for the Mobile Web / Luca Passani:** *www.passani.it/gap*

And always remember:

A mobile website is not simply a small website; it is a website for mobile phone users. Keep it simple.

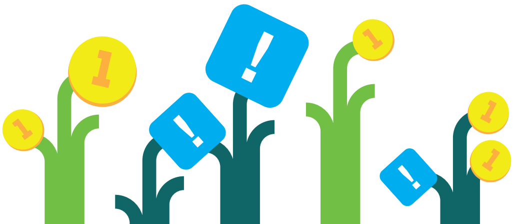


Implementing Rich Media

“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 Mobile/ 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/WCMDA and 500 kbps for Wi-Fi.

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

When targeting Windows Mobile/Phone, Windows Media Server[3] is preferred to support HTTP streaming. The tablet specific Android version 3.0 is also supposed to support HTTP streaming. 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.

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

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

Finally you might of course also simply include the local media file in your mobile app as a resource.

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 eRightSoft[1] . For MAC users, QuickTime pro (paid version) is a good alternative to encode and hint 3gp files. If you are looking for a complete server solution with a Java/ open source background, check out Alembik[2].

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.

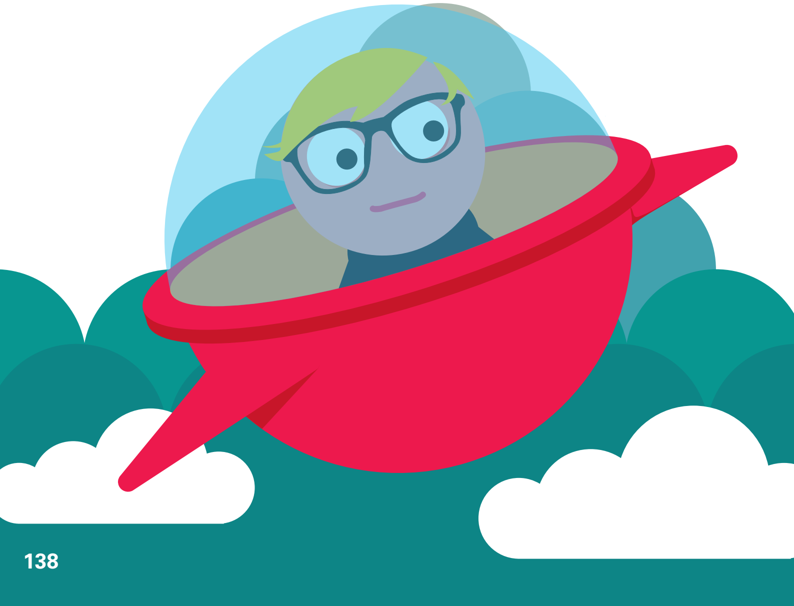
How To Obtain Positioning Data

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 Map or CloudMate, while Ovi Maps are at the time of writing free for Nokia phones only. Commercially available maps include NAVTEQ, Garmin and Microsoft to name a few. Some solutions, such as Google Maps, 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.

- **Cloudmade:** *developers.cloudmade.com/projects*
- **Google Maps:** *code.google.com/apis/maps/*
- **Microsoft Bing API:** *www.microsoft.com/maps/developers*
- **NAVTEQ:** *www.nn4d.com*
- **Nokia:** *www.forum.nokia.com/Develop/Web/Maps/*
- **Open Street Map:** *wiki.openstreetmap.org/wiki/Slippy_map_tilenames*

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.

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 Admob 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 simply needs 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 not have a direct analogy in current behavior. For example, for how many users will the action of touching a poster be an obvious way of opening a related website? However, there is an entire industry poised to educate users on the technology, so there are many opportunities for early adopters.

The NFC standards[1] provide for three modes of operation that can be used in mobile devices:

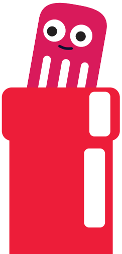
- Read/Write – where a phone can read or write data to a tag
- Peer to Peer – where two NFC enabled phones can exchange data, from simple setup details for a Bluetooth connection through to business cards and digital photos
- Card Emulation – where a phone can act as a tag or contactless card

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

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

- Service Initiation – here a phone can read a tag embedded or attached to everyday objects, the tag would provide a URL, phone number or application specific string that can be used to open a website, dial a number or initiate application specific functionality. A practical application might involve embedding a tag in a product's packaging to provide a way of opening the product's website.
- Sharing – here two NFC enabled phones could share a piece of information, a business card for example.
- Connecting devices – an NFC enabled phone could read connection settings from another phone or peripheral. For example, a Bluetooth headset could include a tag that provides the information for pairing the headset with a phone.
- Ticketing – the NFC phone could be delivered a ticket which is then "redeemed" by being read from the phone.
- Rechargeable or cashless payment cards – here the phone can act as a replacement for a credit card or bank card, travel cards such as Oyster[2] or payments cards such as Snapper[3].

2) https://oyster.tfl.gov.uk/oyster
3) www.snapper.co.nz



Support For NFC

Support for NFC in mobile devices is still relatively new. However, the technology is arriving in the mainstream with Apple, BlackBerry, Google, Microsoft and Nokia[1] all having announced NFC support in their platforms and manufacturers such as Google, BlackBerry, Nokia and Samsung having announced or already started shipping smartphones with NFC capabilities[2].

Creating NFC Apps

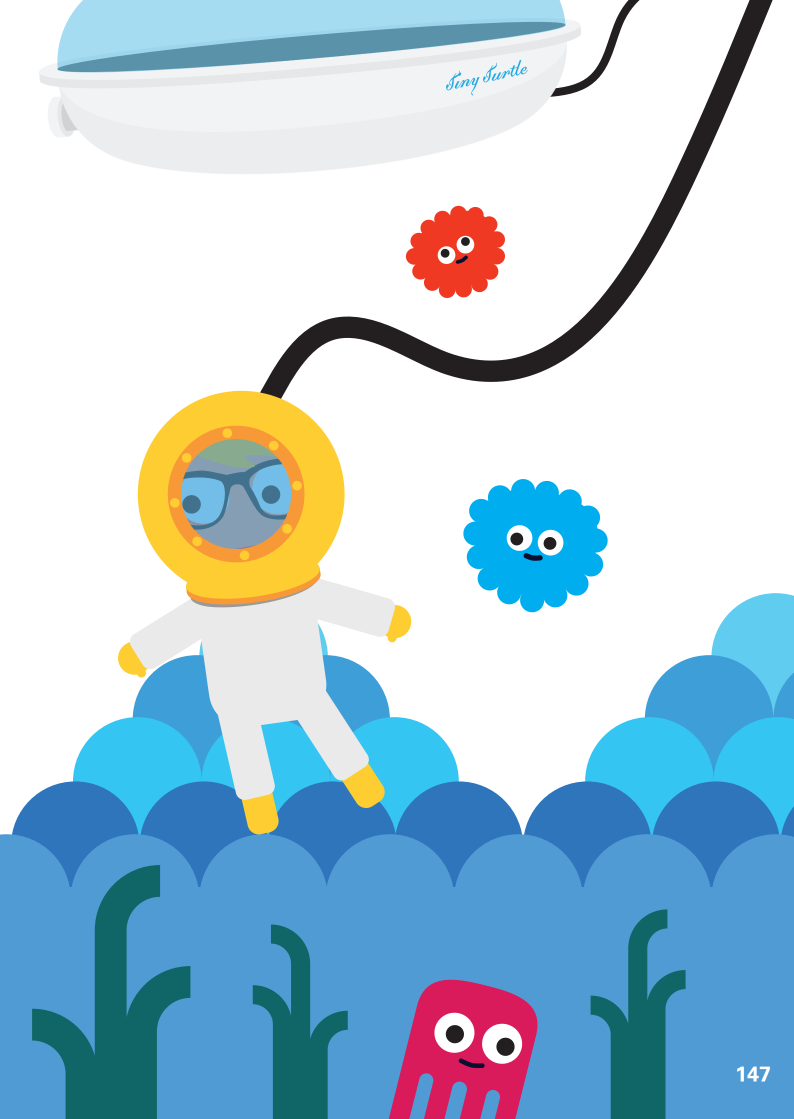
One challenge in creating NFC applications is that there is no single standardized API. While Contactless Communication API (JSR-257) provides a standard, it is not universally available (Apple and Google for example certainly will not provide support for it). Where it is offered, it can be supplemented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

Nokia provides support for NFC in the Qt Mobility APIs[3], 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 Android[4].

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) http://labs.qt.nokia.com/2011/04/12/qt-mobility-1-2-beta-package-released/
4) http://developer.android.com/reference/android/nfc/package-summary.html


Tiny Turtle

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. 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).

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:

- **UI:** 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 fast 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.

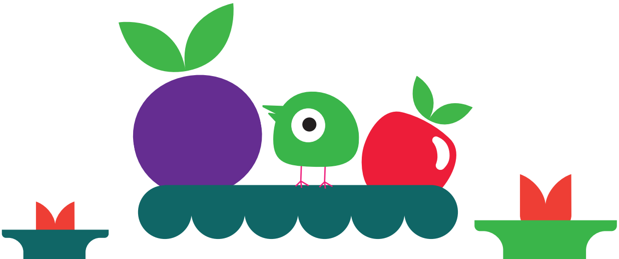
For platforms such as Java ME and Android where there are so many manufacturers and models, it's particularly useful to test on a range of these devices. A good start is to pick a mix of popular, new, and models that include specific characteristics or features such as: touch screen, physical keyboard,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

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) *github.com/eing/moet* supports Android, BlackBerry and iPhone devices with a consistent set of commands which can be used interactively or automated. Google announced they have recently started work on a project that will support the WebDriver protocols to test native applications. Also, Tampere University in Finland have had some success creating automated tests for various mobile platforms, including Android and Nokia’s S60 phones. See *tema.cs.tut.fi* for more information on their work.

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

These services can augment your other testing, we don’t recommend using them as your only formal testing. To get good results you will need to devote some of your time and effort to defining the tests you want them to run, and to working with the company to review the results, etc.

Web-Based Content And Applications

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

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

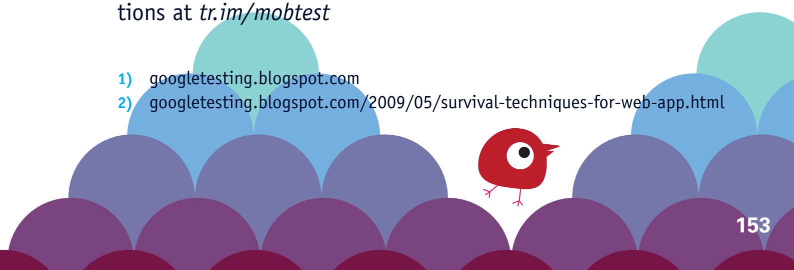
On the desktop the ability to wrap Firefox means it can crudely emulate most mobile browsers by programmatically changing browser parameters such as the user-agent string. There's an article on the Google Testing blog[1] that includes an example of how to emulate the iPhone browser[2].

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

Next Steps

There's more material available on testing your mobile applications at *tr.im/mobtest*

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



Monetization

Finally you have finished your app or mobile website and polished it as a result of beta testing feedback. Assuming you are not developing as a hobby, for branding exposure or for a charity, now it is time to make some money. But how do you do that, what are your options? In general, you have the following monetization options:

1. **Pay per download:** Sell your app per download
2. **In-app payment:** Add payment options into your app
3. **Mobile ads:** Earn money from advertising
4. **Indirect sales:** Affiliates, data reporting and physical goods among others.

When you come to planning your own development, determining the monetization business model should be one of the key elements of your early design as it might affect the functional and technical behavior of the app.

Pay Per Download

Using pay per download (PPD) your app is sold once to each user as they download and install it on their phone. Payment can be handled by an app store, mobile operator, or you can setup a mechanism yourself.

When your app is distributed in an app store — in most cases it will be one offered by the target platform's owner, such as Apple, Google, RIM, Microsoft or Nokia — the store will handle the payment mechanism for you. In return the store takes a revenue share (typically 30%) on all sales. In most cases stores offer a matrix of fixed price points by country and currency ($0.99, EUR 0.79, $3 etc) to choose from when pricing your app.

Operator billing enables your customers to pay for your app by sending a Premium SMS. This option is still very popular for mobile web applications, Java games, wallpaper and ringtones. However, this mechanism 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.

It is worth noting that most of the vendor app stores are pursuing operator billing agreements, Ovi Store by Nokia has the best coverage (112 operators in 36 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 Pay Pal, PayPal mobile, credit card (not supported on all devices), dial-in to premium landlines *(www.daopay.com)*, or similar.

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

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

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.

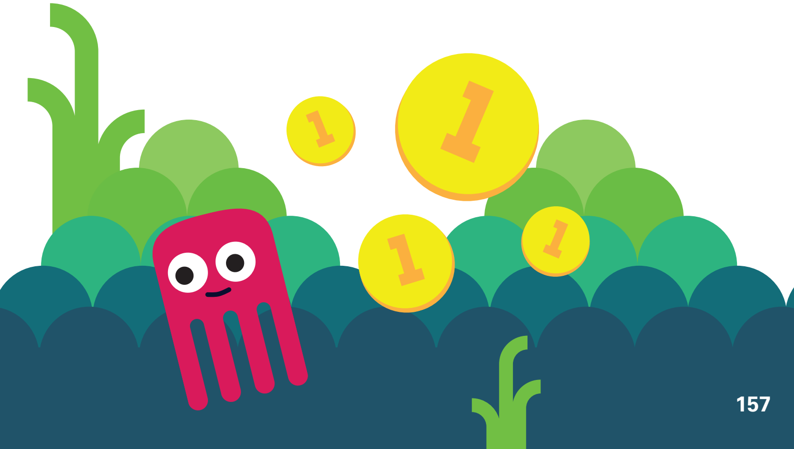
This type of payment is particularly popular in games (for features such as buying extra power, extra levels, virtual credits and alike) and can help achieve a larger install base as you can offer the basic application for free. Note, however, that some app stores do not allow third party payment options 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.Inneractive.de* 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 advertizing 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.

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 apps[1].

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.

1) http://www.forum.nokia.com/Distribute/Public_relations_guidelines.xhtml

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 have 100,000+ 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.

Here are a few tips and tricks to help your raise your odds.

Top 5 Appstores

Appstore	Platform	Daily Downloads	Alternatives
Apple Appstore	iOS	~34 Million a day	Cydia (Jailbroken iPhones)
Android Market	Android	~11 Million a day	GetJar, Samsung, Motorola, Amazon and 50+ others
Nokia Ovi	Symbian, Qt, Widget, Java	~5 Million a day	GetJar
GetJar	Java, Symbian, Android, widget	~4 Million a day	Appia, Handster, Nexva
Blackberry App World	BlackBerry	~3 Million a day	Crackberry

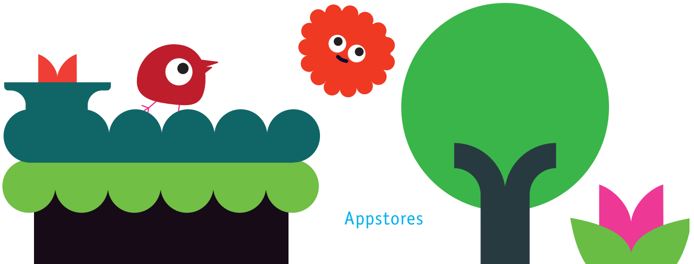
The volume of downloads makes some of these stores look promising. However, the stores with the highest volume attract the largest number of applications fighting for attention, so you might want to pick your niche carefully or spend more time marketing your app.

Basic Strategies To Get High

We have asked many developers about the tactics that brought them the most attention and higher ranking in appstores, many answers came back and one common theme emerged: "there's no silver bullet" you have to fire on all front!

However it will help if you try to keep the following in mind:

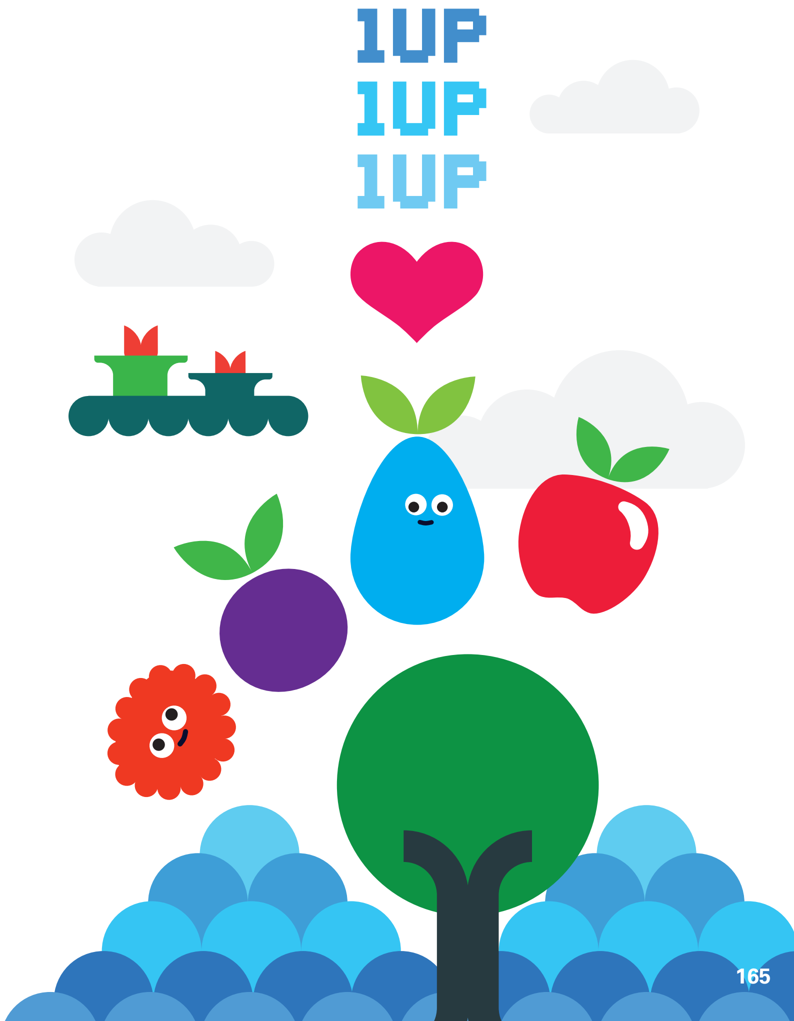
- You need a kick ass app: Entertaining, easy to use, not buggy… make sure you put it in the hands of users before you put it in a store.
- Polish your icons and images in the appstore, work on your app description, and choose carefully your keywords and category. If unsure or unsatisfied of the results, experiment.
- Getting reviewed by bloggers and magazines are some 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 for a review regularly.
- 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.
- 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 115+ appstores there are many possibilities for you to get your application distributed. However with 20 minutes needed on average to submit an app to an appstore, you could be spending a lot of time posting apps in obscure stores that provide less downloads. This is why a majority of developers stick to only 1 or 2 stores missing out on a huge opportunity but getting a lot more time for the important things... coding! So should you go multi-store or not?

Multi-store	Single store
The main platform appstores can have serious limitation, payment mechanism, 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.
Some operators' stores have easier billing processes leading to higher conversion rates	Operators' stores have notoriously strict content guidelines and can be difficult to get in.
Some developers report that 50% of their Android revenues comes from outside of Android Market	iPhone developers only need 1 appstore


1UP
1UP
1UP

Now What – Which Environment Should I Use?

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

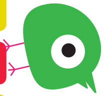
However, you can still find the best solution to your need, but it is a longer answer: think about your target region, the market share achieved by each technology, define your target users, their needs, their devices and data plans. Then consider your vision and the requirements for your application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

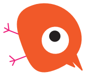
Remember that you are not necessarily restricted to a single application environment. It often makes sense to combine different environments, for example by providing a mobile website for your casual users, a native smartphone application for power users and a Java ME app for regions where smartphones take a smaller market share. If you want to concentrate on smartphone platforms only, you may need to adjust your application concept: On the iPhone you might make money by direct app sales (if you somehow manage to gain visibility among the 350,000 iOS 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.

The following table provides a very rough overview of the strengths and limitations of each application environment. However, one crucial aspect is not reflected here: market share. The differences among regions are too big to simply talk about global market share. To find out about market share in your target region, check out online resources such as comscore[1], StatCounter[2] or Gartner[3].

1) www.comscoredatamine.com/category/mobile/
2) http://gs.statcounter.com
3) www.gartner.com

	Interactivity	Online / Offline	Developer Availability	Developer Tools	Performance	Fragmentation
Android	🟩	🟩	🟩	🟩	🟩	🟨
bada	🟩	🟩	🟥	🟨	🟩	🟨
BREW	🟩	🟩	🟥	🟨	🟩	🟨
Flash	🟩	🟩	🟩	🟨	🟨	🟨
iOS	🟩	🟩	🟨	🟩	🟩	🟨
Java ME	🟩	🟩	🟩	🟩	🟩	🟥
Native BlackBerry	🟩	🟩	🟩	🟩	🟩	🟨
Native Symbian	🟩	🟩	🟥	🟨	🟩	🟨
SMS	🟥	🟨	🟥	🟥	🟥	🟩
Web	🟨	🟨	🟩	🟩	🟥	🟨
Widgets	🟩	🟩	🟩	🟨	🟨	🟩
Windows Mobile	🟩	🟩	🟨	🟩	🟩	🟨
Windows Phone	🟩	🟩	🟥	🟩	🟩	🟨
webOS	🟩	🟩	🟨	🟩	🟩	🟩
Qt (Symbian/MeeGo)	🟩	🟨	🟨	🟩	🟩	🟩





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

Epilogue

Thanks for reading this eighth 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**

About the Authors

Robert Virkus / Enough Software

Robert is 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. With this experience he launched the Open Source J2ME Polish project in 2004 that helps developers to overcome device fragmentation barriers. 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 by creating a comfortable mobile world. Therefore he is responsible for the development of the adaptation technology and 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 attends several 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

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 was leading the mobile social network *m.wauwee.com with* almost 1,000,000 members and supported by a team of Symbian, iPhone, BlackBerry and Android specialists with headquarter in Amsterdam (acquired by MobiLuck).

www.appvalley.nl

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. Since then, he has worked on a number of apps, one of which was featured in an Apple TV commercial, and written a series of articles on iPhone development. As of April 2011, he is an independent software developer and contractor.

www.alexrepty.com

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

Malte Adomeit / Sevenval

Malte got in touch with the mobile world in 2004. Since then he experienced the mobile development in the telecommunication sector from a marketing perspective. A market oriented approach and the aim to satisfy customer needs guided him dealing with device innovation, content enrichment and mobile strategy over the last few years. In 2011 he joined Sevenval as a marketing manager.
www.sevenval.com **www.fitml.com**

Tim Messerschmidt

Tim is developing Android applications since 2008 in his own business Messerschmidt-IT. Besides he is currently writing his bachelor thesis about Android and Google App Engine.
www.messerschmidt-it.de

Ovidiu Iliescu / Enough Software

After developing desktop and web-based applications for several years, Ovidiu decided mobile sofware 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.ovidiuiliescu.com**

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

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 started to offer mobile development for bada as well. 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

Julian Harty / eBay

Hired by Google in 2006 as the first Test Engineer outside the USA and told he was responsible for testing Google's mobile phone applications. He helped others inside and outside Google to learn how to do likewise; and he ended up writing the first book on the topic. The material is also freely available at *tr.im/mobtest* He continues to work on Test Automation for mobile phones and applications. He now works for eBay where his mission is to revamp testing globally.

www.ebay.com **www.tr.im/mobtest**

André Schmidt / Enough Software

André is 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 is mainly developing for J2ME, Android and BlackBerry.

www.enough.de

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. And of course he is an open source enthusiast, just like everybody at Enough Software.
www.enough.de

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 is coding software for several platforms since 2000. He is working 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**

An initiative by:

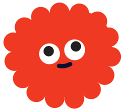
ENOUGH SOFTWARE

wireless industry partnership

www.enough.de www.wipconnector.com

Printing sponsor:

NOKIA Connecting People



»A knowledgeable read for anyone trying to understand the difference between programming for different mobile platforms. Kudos to the authors!«
— Mob4Hire Blog

»This guide is the best short document I have read ever about mobile development.«
— David Contreras Magaña, Director I+D+i, Esidea

»Wow, what an awesome guide. It gave me an excellent overview of the alternatives available with their pros and cons.«
— John Klippenstein, CTO, Cascading Glass

»Short and sweet! Worth to read for beginners as well as decision makers when entering the mobile business.«
— Ralph Buchfelder, CEO, i-locate

»Really cool.«
— Carlos Bernardi, Team Leader Handset Embedded Programs, Gameloft