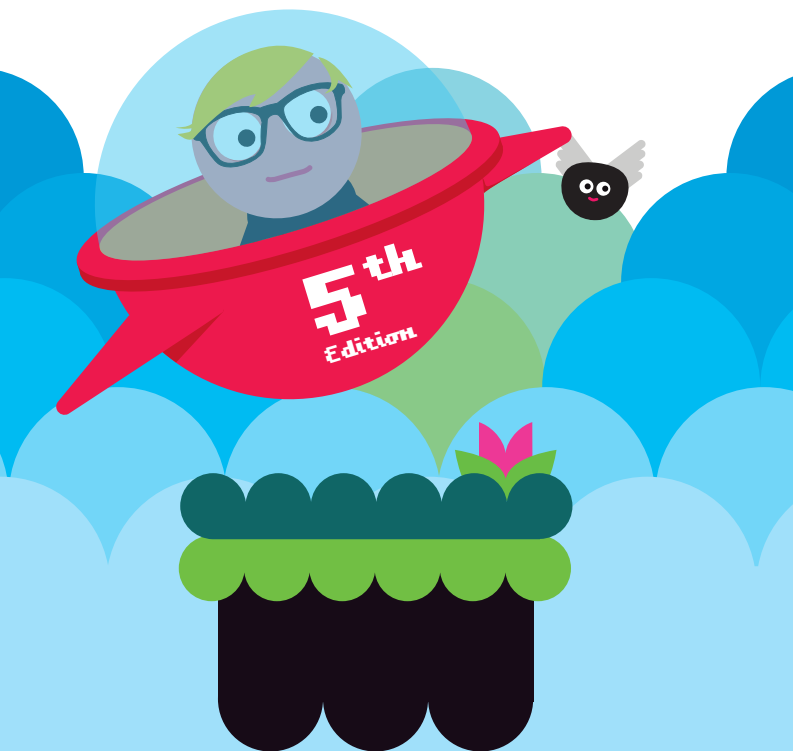


Don't Panic Mobile Developer's Guide to the Galaxy



Mobile Developer's Guide

Table of Contents



5 Introduction

6 Application Environments Overview

6 Native Applications

9 J2ME / Java ME

9 Flash Lite

10 BREW

10 Widgets

12 Websites

12 SMS Text Messaging

13 Programming Android Apps

13 Prerequisites

14 Implementation

16 Testing

16 Signing

16 Distribution

18 Programming iPhone Apps

19 Prerequisites

19 Implementation

21 Testing

22 Distribution

IUP
IUP
IUP



- 23 Programming Symbian Apps**
- 24 Prerequisites
- 26 Porting to Symbian
- 27 Signing
- 28 Distribution

- 29 Programming J2ME / Java ME Apps**
- 29 Prerequisites
- 30 Implement your App
- 31 Testing
- 33 Porting
- 36 Signing
- 37 Distribution

- 38 Programming Qt Apps**
- 39 Prerequisites
- 40 Creating Your Application
- 41 Getting Qt
- 41 Testing
- 42 Signing
- 42 Distribution

- 44 Programming Mobile Web-Widgets**
- 45 Widget Characteristics
- 48 Prerequisites
- 48 Writing Your Code
- 50 Testing
- 50 Signing
- 50 Distribution

- 52 Programming Flash Applications**
- 53 Prerequisites
- 56 Developing Flash Applications for Smartphones
- 58 Testing
- 58 Distribution





- 60 Bringing Your Content to the Mobile Web**
- 63 Mobile Usage Patterns and its consequences
- 64 Some History on the Mobile Web
- 66 How to adapt content for the mobile user
- 68 Satisfy the Interpreter of Your Content: The Browser
- 70 Let's do some pigeonholing: Device categories
- 74 Use GPS in the Browser
- 74 Testing your Mobile Website
- 76 Learn More – On the Web

- 77 Implementing Rich Media**
- 77 Streaming
- 77 Progressive download
- 78 Ringtones
- 78 Flash
- 80 Mobile Video
- 82 Media Converters

- 83 Implementing Location-Based Services**
- 83 How to obtain Location Data
- 84 How to obtain Mapping Material
- 85 Implementing Location Support on Different Platforms
- 86 Tools for making Map Apps

- 87 Now what — Which Environment Should I Use?**

- 90 Epilogue**

- 91 About the Authors**
- 95 Imprint/Contact



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



Mobile Developer's Guide

Introduction

Welcome to the fifth edition of our Mobile Developer's Guide To The Galaxy! It will help you to realize your ideas on mobile handsets by outlining the different options you have and by explaining how to overcome typical problems.

Developing for mobile handsets is a schizophrenic experience – on one side there is the never-ending stream of innovations and new technologies, on the other side it can take a painfully long time until new technologies are actually used by the end customers. Point in case is for example the mobile internet which took more than 10 years until it is now at last used frequently!

The most important issues for mobile application developers still are fragmentation and distribution. Thankfully solutions for those problems do exist.

Development is a passion – have fun in the mobile world!



Mobile Developer's Guide

Application Environments

Overview

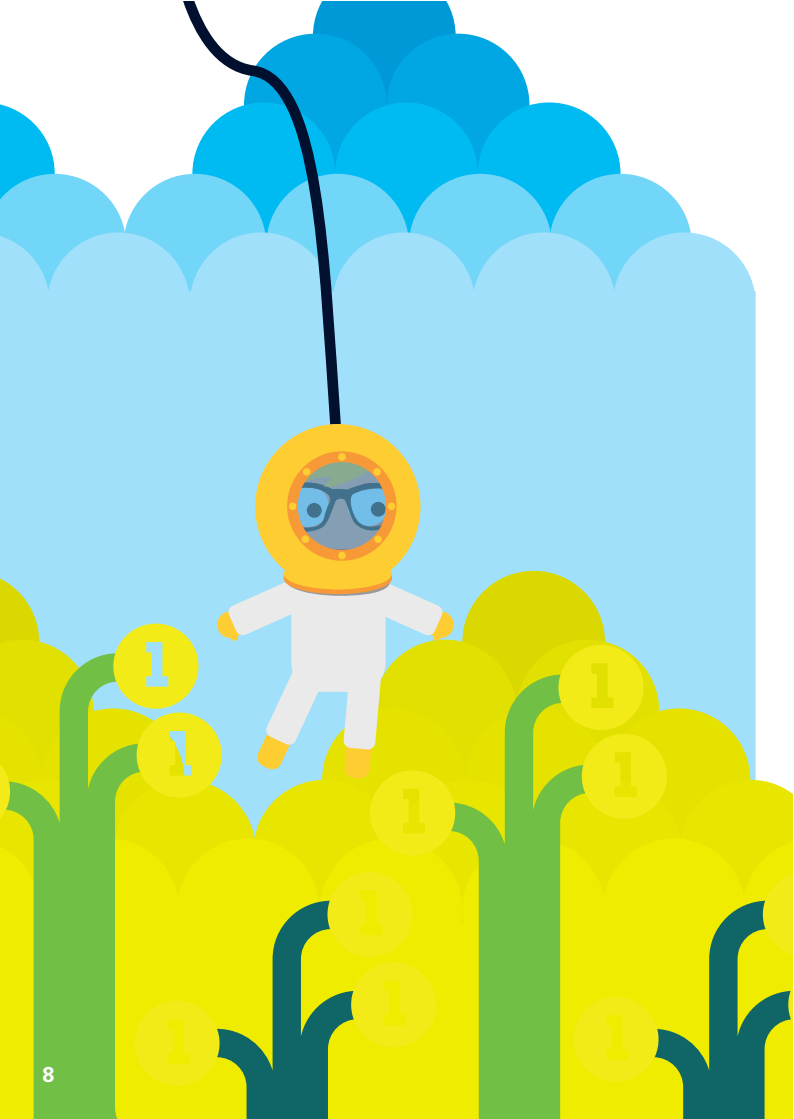
You can choose between different environments for realizing your mobile vision. This section describes the most common environments and outlines differences between them. A more detailed description follows in the environment specific chapters later onwards.

Native Applications

There are a lot of mobile operating systems used in the market today – some are Open Source, some are not. Most important OS are (alphabetically) Android, Bada, BlackBerry, MeeGo, OSX, Symbian, webOS and Windows Mobile/Windows Phone. All these OS allow you to create native applications for them without establishing a business relationship with the respective vendor. Main benefits for programming native applications include the best possible integration of your application and often a very good performance. Typical drawbacks are the required effort and the complexity of supporting several native platforms (or the limitation of your app to only one platform). Most mass market phones are, however, equipped with embedded operating systems that cannot be programmed natively for by outsiders at all. Examples include but are not limited to Nokia Series 40, Samsung SGH and Sony Ericsson Java Platform phones.

The following table provides an overview about the main mobile operating systems:

OS	Language(s)	Remarks
Android	Java, C	Open Source OS (based on Linux) developer.android.com
Bada	C, C++	Samsung C based APIs that allow a tight integration developer.bada.com
BlackBerry	Java	J2ME compatible but extensions allow a tighter integration na.blackberry.com/eng/developers
MeeGo	C++	Intel and Nokia powered Open Source OS (based on Linux) meego.com/developers
OS X / iPhone	Objective-C, C	Requires Apple Developer Account developer.apple.com/iphone
Symbian	C, C++, others	Open Source OS www.symbian.org
webOS	HTML, CSS, JavaScript, C	Widget style programming, OS (based on Linux) developer.palm.com
Windows Mobile	C#, C	.NET CF or Windows Mobile API, most devices ship with J2ME compatible JVM developer.windowsmobile.com
Windows Phone	Silverlight, XNA	New Windows based OS targeted at consumers developer.windowsphone.com



J2ME / Java ME

Around 80% of all mobile handsets shipped in 2009 support the mobile Java standard (J2ME/Java ME), so Java is by far the most widely distributed application environment. In contrast to many other environments, J2ME 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 J2ME but at the same time it is the source of many fragmentation problems.

On many handsets with embedded operating systems, J2ME is the only way to realize client side applications. The main problem is the fragmentation, meaning the little differences between the implementations and handsets.

Flash Lite and Alternative Flash-compatible Platforms

Flash Lite is the mobile edition of Flash, which pretty much is the same as an older version of Adobe's web Flash product and only supports ActionScript 2.0. Flash Lite gains traction and is especially favored by many designers, since they know the tools already. Flash Lite features a powerful UI but it lacks a good integration into the host device, although this is improving. Programming Flash Lite is relatively easy thanks to the ActionScript language that is very similar to Java Script. The drawbacks include poor performance and small market share in comparison to J2ME. However, there are millions of feature phones supporting Flash Lite today and some smartphones support Flash Lite including Android-based devices. Full Flash support has been announced for Android and BlackBerry devices.

BREW

The Binary Runtime Environment for Wireless (BREW) is a programming environment pushed forward by Qualcomm¹. BREW services are offered by more than 60 operators in 28 countries, but it's most popular within the US with CDMA devices launched by Verizon, US Cellular, Metro PCS among others. While previous versions only supported C development, the Brew Mobile Platform (Brew MP), supports applications written in Java, Flash, TrigML or native C code².

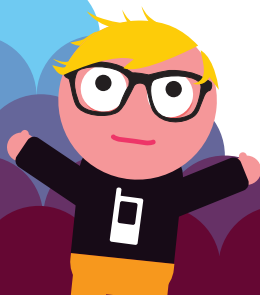
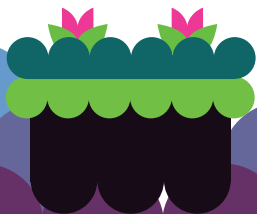
Widgets

There are several widget environments that you can choose from. Main benefit of these environments is the simple programming, main drawback is that you need to ensure that your customers will install both – the widget player as well as your widget. More and more devices feature, however, a preinstalled widget player – not necessary for the environment that you have chosen, though.

The other significant drawback is that widgets do only provide a limited access to the handsets native functionalities.

1) brew.qualcomm.com

2) brew.qualcomm.com/brew/en/developer/overview.html



All these environments use XML, a script language (e.g. JavaScript) and a page description language (like HTML) for realizing a widget. The following table gives you an overview about popular widget frameworks:



Environment	Language(s)	Remarks
Web Runtime Widgets on Symbian	XML, HTML, CSS, JavaScript	tiny.symbian.org/runtimespace
Skylight	HTML, CSS, JavaScript	Skylight is an Open Source widget platform for featurephones as well as smartphones. www.skylightmobile.org
W3C / Vodafone Widgets	XML, HTML, CSS, JavaScript	Vodafone, China Mobile and other companies try to push the W3C widget standard for mobile adoption. widget.vodafone.com/dev
Samsung	XML, HTML, CSS, JavaScript	innovator.samsungmobile.com
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

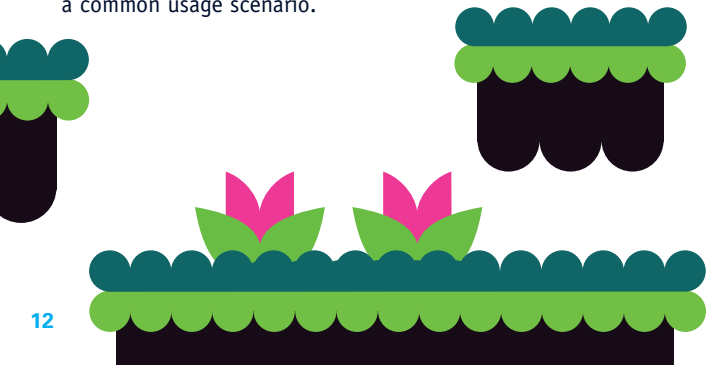
Websites

Webpages are supported by all phones, so in principle this should be the environment of choice to get the widest possible reach (after SMS texting). The only problem is the sheer number of browsers and their varying feature sets. Some browsers are very powerful and support CSS as well as JavaScript, others are less sophisticated XHTML only. Thankfully the old WAP standard with its WML pages does not play any significant role nowadays.

Main problems for web pages are that they are only available online and that they cannot access device features today. Main benefits are the easy development and a deployment that lies solely in your hands.

SMS Text Messaging

Almost everybody who has a mobile phone is also texting. Texting provides the limitation that any interaction needs to be pressed into 160 characters and that it can be quite costly to send out text messages in bulk. On the positive side it enjoys a global audience of any age and plays an important role especially in the emerging markets where SMS banking is a common usage scenario.



Mobile Developer's Guide

Programming Android Apps

The Android platform is one of the latest systems for the mobile market. Created by Google and the Open Handset Alliance in late 2007, Android is an operating system and an application framework with complete tooling support and a variety of preinstalled applications. For 2010, fifty new Android based smartphones are expected to be introduced. Additionally, Android is planned to be used for tablets, media players, setup boxes, desktop phones and car entertainment systems.

Prerequisites

The main programming language for Android is Java5. But beware that not the complete Java library is supported and that a lot of platform specific API is present. You find answers to your What and Why questions in the Dev Guide¹ and to your How questions in the reference documentation².

To get started you need the Android SDK³ and a decent Java IDE. The SDK is available for Windows, Mac OS X and Linux and contains tools to build, debug and analyse applications. So you have no constraints on your development platform. As for IDE support, you can install the ADT plugin for Eclipse which coordinates building and deploying. But as the complete build cycle of Android applications is Ant based you can basically use any IDE you like.

1) developer.android.com/guide/index.html

2) developer.android.com/reference/packages.html

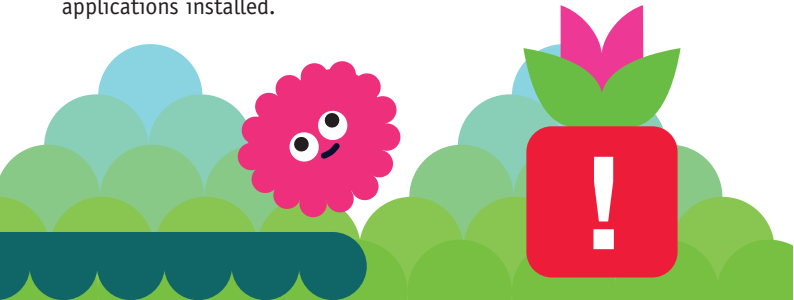
3) developer.android.com/sdk/1.6_r1/index.html



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. The message receiver can handle messages broadcasted by the system or other applications. The data provider is an interface to the content of an application and thereby abstracts from underlying storage mechanisms. These application components are composed such that the user can fulfil a certain task. The 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 are a way of triggering all kinds of behavior 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 on the applications installed.



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:** Scan devices at your USB bus, 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 some time to startup 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. Have a look at the memory consumption or simply kill some 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 - 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 and the ADT plugin may help to manage all these files.

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. Therefore testing on a physical device is paramount. If you do not have a lot of devices at your disposal, try remote device services like Perfecto Mobile or Device Anywhere. They provide you a remote access to actual device hardware in different configurations. To automate testing, the Android SDK comes with instrumentation classes to send system events like key presses or an incoming phone call. You can test for the status of your application after these events occur. A maven plugin and a helper for the continuous integration server Hudson may also assist your testing.

Signing

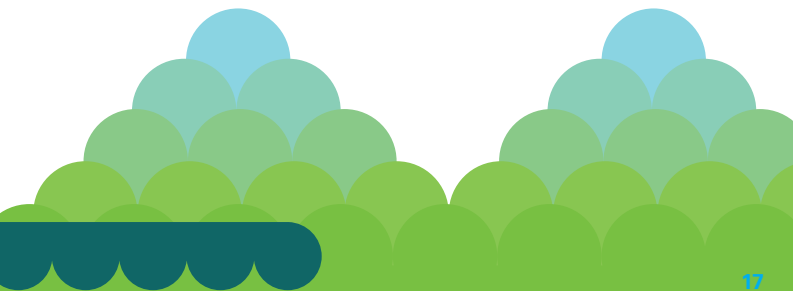
Your application will always be signed by the build process, either with a debug signature or a real one. Your signature may be self signed, so forget about signing fees (and security). The same signature is required for updates of your application.

Distribution

After you have created the next killer application and tested it, you should put it in the Android Market. It is the first place to reach all customers and developers of the Android platform alike, browse for new exciting apps and sell your own. To upload your application, 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`. As Android Market is not available world wide and the number of competing applications in the Android Market is large, you might want to use alternative application stores. They provide different payment methods and may target specific consumer groups.



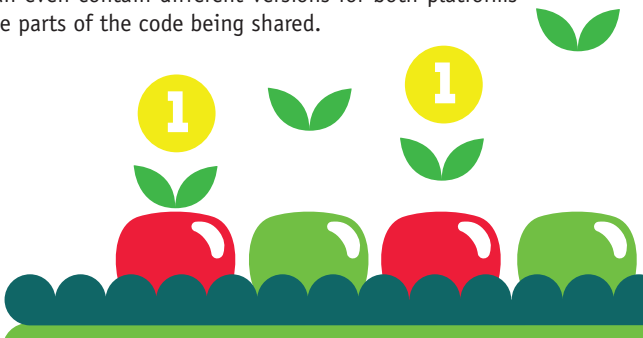
Mobile Developer's Guide

Programming iPhone Apps

The iPhone 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. Since its inception, App Store users have downloaded over four billion apps, which makes the App Store very interesting for mobile developers. Now there are over 185,000 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.

The iPhone SDK offers high-level APIs for a wide range of tasks which helps to cut down on development time on your part. New APIs are added in every major update of iPhone OS, such as MapKit in iPhone OS 3.0 or (limited) Multitasking in iPhone OS 4.0.

Since the iPad, which went on sale in April 2010 uses the same operating system and APIs as the iPhone, acquired skills 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.



Prerequisites

In order to develop iPhone (and iPod Touch and iPad) apps, you will need the iPhone SDK, which can be downloaded at developer.apple.com/iphone. This requires a membership, which is available for free. If you plan to test your apps on your device or distribute your apps on the App Store, you need to sign up for an account starting at 99USD a year.

The iPhone 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 iPhone SDK
- Interface Builder, to build user interfaces for iPhone app
- Instruments, which offers various tools to monitor app execution
- iPhone Simulator, which allows the developer to test his or her apps quicker than by deploying to a device

The iPhone SDK will work on any Intel-based Mac running Mac OS X 10.5 (Leopard) or 10.6 (Snow Leopard). 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.

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're 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.com¹. What separates Objective-C most from these languages is its dynamic nature, lack of namespace support and the concept of message passing vs. method calls.

A great way to get you started is Apple's guide "Your First iPhone Application", which will explain various concepts and common tasks in an iPhone developer's workflow².

Check out some of the sample code that Apple provides online³ to find out more about various APIs that are made available to you.

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

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



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. If possible, test your application on various different models, since the addition of the iPhone 3GS to the lineup brought with it a faster CPU, GPU and more RAM and may thus behave differently than previous iPhone generations.

You can distribute builds of your application to up to 100 testers through Ad-Hoc Provisioning, which you can set up in the Program Portal¹. Each iPhone (and iPod touch) has a unique identifier (UDID – universal device identifier), which is a string of 40 hex characters based on various hardware parts of the device.

If you choose to test using Ad-Hoc-Provisioning, simply follow Apple's detailed set-up instructions². Every single step is vital to success, so make sure that you execute them all correctly.

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



The iPhone SDK does not include support for unit testing out of the box, but fortunately Google added support in Google Toolbox for Mac (GTM)¹ with detailed instructions available in their wiki². Also, iPhone developer Gabriel Handford has improved on this work and released his project, GHUnit as open source³. GTM runs the test cases using a shell script during the build phase, while GHUnit runs the tests on the device (or in the simulator), allowing the developer to attach a debugger to investigate possible bugs.

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 iPhones, but the potential reach isn't nearly as large as the App Store's.

In order 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 as well as a build of your app that has been properly signed. Log in to iTunes Connect³ 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. The approval process is the most complained component of the iPhone ecosystem. A list of common rejection reasons can be found on www.apprejections.com.

1) code.google.com/p/google-toolbox-for-mac
code.google.com/p/google-toolbox-for-mac/wiki/iPhoneUnitTesting

2) github.com/gabriel/gh-unit

3) itunesconnect.apple.com

Mobile Developer's Guide

Programming Symbian Apps

The Symbian platform is an open-source 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) and, since 2009, has been stewarded by the Symbian Foundation¹.

Close to 300 million Symbian devices (operating on 250 networks across the world) have shipped since the first Symbian OS-based device was launched in 2000. The first official platform release from Symbian Foundation was Symbian^2 in May 2009, devices based on Symbian^3 are shipped in 2010.

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. It has a steep learning curve and the first steps here can be more frustrating than in other environments, but there are lots of books and articles to help you².

Compared with many of the runtimes available for Symbian devices, such as Java ME and Python, native Symbian C++ APIs provide the most comprehensive access to device features and enable rich application development. They provide fine-grain control over all aspects of the operating system, including memory, performance and battery life; and deliver a consistent performance advantage over other runtimes.

¹⁾ www.symbian.org

²⁾ developer.symbian.org/wiki/index.php/Symbian_C%2B%2B_Quick_Start

Since late 2009 you can also create native applications for Symbian phones using cross platform Qt (pronounced “cute”) APIs and tools. Qt’s developer-friendly APIs allow applications to be written once and compiled for Symbian, desktop and windows mobile devices.

Please check www.forum.nokia.com/Technology_Topics/Development_Platforms/Qt/QuickStart.xhtml for details.

Prerequisites

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

1. ActivePerl (use ONLY version 5.6.1.638)¹
2. The Symbian Application Developer Toolkit (ADT)
3. The Symbian Application Development Software Developer Kit (SDK) (full installation)²

Linux and Mac OS X are not officially supported platforms. One way around this is to use a virtual machine that hosts Windows. Other options are more complex, but information can be found online³.

¹) downloads.activestate.com/ActivePerl/Windows/5.6/ActivePerl-5.6.1.638-MSWin32-x86.msi

²) Items 2 and 3 are downloadable on developer.symbian.org/main/tools_and_kits/index.php

³) developer.symbian.org and www.forum.nokia.com

Application Developer Toolkit (ADT)

If you have worked with S60 SDKs and tools, you'll have installed each of the tools you need for development (such as Active Perl, the JRE and Carbide.c++) separately. The new Symbian platform installs all the tools you need in one sweep.

The ADT is for developers who wish to create applications that run on production phones - i.e. "on top" of the Symbian platform. Typical users include professional application and games developers, professional service companies, hobbyist developers, students and research groups. The ADT is intended for use with one or more SDKs.

The Carbide.c++ IDE, based on Eclipse, is installed as part of the Application Development Toolkit (step 2 in the section above). It is the only officially supported IDE for Symbian C++ development. The GCC compiler is also supplied, as are a debugger that allows debugging on both emulator and production phones, analysis tools, and more.

Application Development Software Developer Kit (SDK)

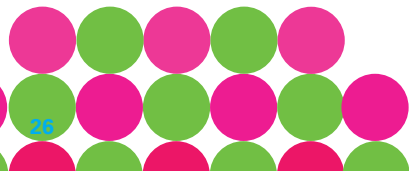
The Application Development SDK contains libraries and header files that enable you to develop applications. It should be used in conjunction with the ADT. The SDK provides access to APIs that are guaranteed to work across a broad range of Symbian-based devices today and in the future. Once you have installed the kits 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 writing a single line of code.

Porting to Symbian

The Symbian platform includes standard libraries that enable developers to build POSIX-compliant C code and use near-ISO standard C++, including the Standard Template Library. These libraries can substantially reduce the time and effort required to port POSIX-based software to the Symbian platform. They are especially useful for porting existing C++ or C-based software from the desktop environment and increase the scope for cross-platform software development using C/C++ code. Existing standards-compliant code will work on the Symbian platform with some minor changes and a rebuild. Developers can benefit from an open-source code-base instead of having to create everything from scratch.

The C language support comprises four 'base' libraries (libc, libm, libpthread and libdl), five additional libraries (libssl, libz, libcrypto, libcrypt and libglib) and related tools. For standard C++ support, the platform provides the popular STLport version of the Standard Template Library and a small but useful subset of the Boost libraries.

There are no standard C/C++ APIs for the UI or application engines, such as Calendar, Contacts and Messaging, and no access to most handset-specific features such as multimedia or telephony. However, if you develop using Qt on Symbian, APIs will be made available for access to these features, through the QtMobility project.





Signing

Symbian uses a trust-based platform security model. Native C++ APIs, which could be deliberately or accidentally used to compromise the end user's private data, the mobile phone network, other applications or the device itself, are protected by platform security "capabilities". If you use APIs protected by capabilities, you will need to submit your application to Symbian Signed¹.

Many developers opt for the Express Signed option when using Symbian Signed. This needs the developer to register for a free Symbian Signed account and use it to purchase a Content ID, which is needed to request the Express signing from Symbian. A separate Content ID is needed for each application to be signed, and each one costs \$20. The developer then validates the application against a set of published test criteria and, when it passes, submits a digitally signed copy of the application to the Symbian Signed portal, using a Publisher ID as the signing key. This Publisher ID may be purchased from a certificate authority for a one-off cost.

When submitting the application, the developer must confirm that the application passes all relevant tests, and list the platform security capabilities it requires. The Symbian Signed portal scans the developer's submission and, if it is acceptable, re-signs it for installation to customers' devices. Symbian Signed conducts regular audits on random applications submitted for Express Signed, to ensure that the test criteria are met by the developer.

You can find out more on [developer.symbian.org/wiki/index.php/Express_Signed_\(Symbian_Signed\)](http://developer.symbian.org/wiki/index.php/Express_Signed_(Symbian_Signed))

¹) developer.symbian.org/wiki/index.php/Symbian_Signed_For_Distributing_Your_Application

If you are writing a simple application or game, you might be able to avoid using protected APIs to avoid Symbian signing. You can simply distribute the application signed by a certificate you have generated for yourself, known as self-signing. However, a self-signed application will present a warning to the user when they install it, saying that the code is untrusted. To reassure your users, you may therefore still prefer to Symbian sign your application even when you don't use protected capabilities.

Distribution

Symbian Horizon¹ is a service that helps developers to write Symbian applications and publish them in app stores like Nokia's Ovi Store, the Samsung Applications Store, Sony Ericsson's Playnow, AT&T's MEdia, and stores from China Mobile, Handango and Orange.

Horizon has a single directory of Symbian Signed tools available which enables developers to display and advertise their applications, and allows users to discover them and find out which stores they are available from.

¹⁾ horizon.symbian.org



Mobile Developer's Guide

Programming J2ME / Java ME Apps

Programming J2ME apps is a fun and rewarding experience. The capabilities of the Java platform are constantly evolving thanks to the Java Community Process¹ that standardizes new APIs (like the Advanced Multimedia API) and even whole platforms (like the Mobile Service Architecture).

Right now J2ME development seems to be a bit unfashionable compared to iPhone or Android development, but J2ME development is still the one and only way to reach out to around 80% of the world wide mobile phone users.

Prerequisites

A mobile Java application (MIDlet) is compiled, obfuscated, preverified and packaged. To implement your application you need a couple of things:

- the Java SDK² (not the Java Runtime Environment) and an IDE of your choice, e.g. Eclipse³ possibly with Mobile Tools for Java (MTJ), NetBeans⁴ with its mobility pack or IntelliJ⁵.
- an Emulator, e.g. the Wireless Toolkit⁶, the Micro Emulator⁷ or a vendor specific emulator.

1) www.jcp.org

2) java.sun.com

3) www.eclipse.org

4) www.netbeans.org

5) www.jetbrains.com

6) java.sun.com/products/sjwtoolkit

7) www.microemu.org

- Depending on your setup you may also need an obfuscator like ProGuard¹. If you build applications professionally you will probably also want to use a build tool like Maven² or Ant³.
- You may want to check out J2ME Polish, the Open Source framework, for building your application for various devices⁴.

A complete installation and setup guide is out of scope of this guide, please refer to the respective documentation of these tools. Beginners often like NetBeans with an installed mobility pack. Also download and read the JavaDocs for the most important technologies and APIs. You can download most JavaDocs from www.jcp.org but there are a couple of useful vendor specific APIs that needed to be tracked down manually from the vendor's pages (Nokia UI-API or Samsung APIs, for example).

Implement Your App

The J2ME environment is pretty much straight forward, the basis is formed by the Connected Limited Device Configuration (CLDC) and the Mobile Internet Device Profile (MIDP), which are both quite easy to understand. Similar to the good ol' fashioned Applets you extend `javax.microedition.midlet.MIDlet` in your main class and then you are ready to go.

1) proguard.sf.net

2) maven.apache.org

3) ant.apache.org

4) www.j2mepolish.org

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

- **Highlevel LCDUI components:** use the components found in the `javax.microedition.lcdui` package, e.g. `Form` or `List`.
- **Lowlevel LCDUI:** use `javax.microedition.lcdui.Canvas` for controlling every pixel of your UI.
- **SVG:** Scalable vector graphics defined in JSR 287 jcp.org/en/jsr/detail?id=287.

There are also different products out there to help you with the UI development:

- **J2ME Polish:** compatible with the highlevel LCDUI framework, separates the design in CSS, you can also use HTML for the user interface (www.j2mepolish.org).
- **LWUIT:** a Swing inspired UI framework (lwuit.dev.java.net).
- **Mewt:** Use XML for defining the UI (mewt.sourceforge.net).
- **TWUIK:** A powerful “Rich Media Engine” (www.tricastmedia.com/twuik)

There is a rich open source scene in the J2ME sector. Interesting projects can be found via blog on opensource.ngphone.com. You will also find many interesting projects on Sun’s page¹, for example the Bluetooth project Marge².

Testing

Thanks to the fragmentation, testing your applications is absolutely vital. Test as early and as often as you can on real devices.

¹⁾ mobileandembedded.dev.java.net ²⁾ marge.dev.java.net

Testing on emulators cannot substitute testing on real devices – you cannot compare your machine with 3GHz Dual CPU, 4 GB of RAM and broadband internet connection with typical mobile phones. Some emulators are quite good (my personal favorites are BlackBerry and Symbian), but there are some things you have to test in real:

- **UI:** 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.
- **Multimedia:** support for audio, video playback and recording facilities can differ dramatically between devices and their respective emulators.
- **Internet connectivity:** establishing internet connections can take an incredible amount of time. Connection delay and bandwidth depend on the network, its current strength and the number of simultaneous users.

Thankfully some vendors provide subsidized or even free remote access to selected devices:

- **Nokia:** apu.ndhub.net
- **Sony Ericsson:** developer.sonyericsson.com/site/global/techsupport/virtuallab/p_virtual_lab.jsp
- **Samsung:** innovator.samsungmobile.com/bbs/lab/view.do?platformId=2

If you require to test network connectivity on a number of carriers or if you need to access certain devices of a specific carrier, you can use Device Anywhere¹ for your testing needs.

¹) www.deviceanywhere.com

Mob4hire¹ provides crowdsourcing services. Of course there are various testing companies as well, that provide testing and QA services, e.g. Absolute Quality² or Mobiquest³. Unit testing can be done with MoMEUnit⁴ or CLDC Unit⁵, coverage testing with Cobertura for J2ME⁶.

Porting

One of the strengths of the mobile Java environment is that it is backed by a standard, so it can be implemented by various vendors that compete with each other. The downside is that a standard has to be interpreted, and interpretation causes differences. Different implementations also suffer from different bugs, which makes things not really easier. In the next sections we outline different strategies for porting your J2ME applications to all J2ME handsets and to different 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 really adapt your application to each platform. At the same time your development costs will skyrocket. We advise to use another strategy first until your idea has been proven to be a success.

1) www.mob4hire.com

2) www.absolutequality.com

3) www.mobiquest.net

4) momeunit.sourceforge.net

5) snapshot.pyx4me.com/pyx4me-cldcunit

6) www.cobertura4j2me.org

Least Common Denominator

You can prevent many porting issues by limiting the functionalities of your application to the least common denominator. In the J2ME world this means CLDC 1.0 and MIDP 1.0. Depending on the target region for the application you might also consider to use either the extensions Java Technology for the Wireless Industry (JTWI, JSR 185) or the Mobile Service Architecture (MSA, JSR 205). Both extensions are supported by many modern devices and provide a lot more power, though in some regions like Africa, South America or India you should be aware that you are limiting the number of your potential users by taking this step.

Using the least common denominator typically is easier, as there are less functionalities to care about. However, the user experience may suffer when limiting your application in this way. Also this approach won't help you to port your app to different platforms like Android or the iPhone. In short: use the least common denominator for simple applications, but be careful to use this approach for complex applications.

Player-Based Solutions

There are various player solutions on the market which mostly provide a consistent set of APIs across various platforms. Probably PhoneGap¹ is the most popular solution among them. The player itself has been ported to different platforms and it interprets or executes the application code.

Player technologies provide an easy and compelling way to realize solutions across a variety of platforms. One drawback can be that player concepts often need to use the least common denominator for realizing a consistent API across platforms, it can be difficult or not possible at all to use platform or device specific functionalities in your application.

1) www.phonegap.com



Porting Frameworks

Porting frameworks automatically adapt your application to different devices and platforms. Such frameworks typically feature the following ingredients:

- Client library that simplifies development
- Build tools chain that converts code and resources to application bundles
- Device database that provides information about devices
- Cross compilers to port your application to different platforms

In the J2ME world there are various frameworks to choose from: Xpress Suite from JavaGround¹ provides porting against revenue sharing. Celsius from Mobile Distillery² is licensed per month. Bedrock from Metismo³ provides a suite of cross compilers on a yearly license fee. J2ME Polish from Enough Software⁴ is available under both the GPL Open Source license and commercial ones. Going the other direction (from C++ to Java ME) is also possible with the MoSync Open Source solution⁵. Good porting frameworks allow you to use platform and device specific code in your project, so that you can provide the best user experience. In other words: A good framework does not hide device fragmentation, but makes the fragmentation manageable.

1) www.javaground.com

2) www.mobile-distillery.com

3) www.metismo.com

4) www.j2mepolish.org

5) www.mosync.com

Signing

The mobile Java standard differentiates between signed and unsigned applications. Some handset functionalities are only available to trusted applications. 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 allow this functionality, on another he'll be asked every time the feature is being used and on the third one you won't 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 handset they desire. Applications signed with a carrier certificate are on a similar level.

Applications signed by JavaVerified¹, Verisign² or Thawte³ are on the lowest security level. The mad thing is that not every phone carries all necessary root certificates. And some well known developers have even stripped away all root certificates in the past. The result is quite a mess, so consider signing your application only when required, e.g. when deploying to an app store or when you absolutely need access to security constrained features. You can of course consider working together with a testing and certification service provider and let them do the job for you. The largest one here would be Intertek NSTL⁴, others are Absolute Quality⁵ or Mobiquest⁶.

1) www.javaverified.com

2) www.verisign.com/code-signing

3) www.thawte.com/code-signing

4) www.nstl.com

5) www.absolutequality.com

6) www.mobiquest.net

Distribution

Unlike the iPhone, you can install J2ME applications directly on your phone – either over bluetooth or over the air (OTA) using a webserver. Thanks to the universal app stores, nowadays distribution is easier than ever. They manage the payment, hosting and advertisements and receive a revenue share for that.

- Ovi¹ targets Nokia users worldwide and provides a 70% revenue share for the developer (of gross sales, net of refunds and returns, less applicable taxes and, where applicable, fixed operator billing costs).
- Java Store² plans to sell Java based content.
- The Samsung App Store³ distributes application for Samsung handsets.
- LG also distributes apps on www.lgapplication.com
- Handmark provides a carrier and vendor independent mobile store.⁴
- Carriers also get into the game, e.g. Sprint⁵ or O2⁶. Basically almost everyone has announced an app store.
- GetJar is one of the oldest distributors for free mobile applications⁷.

An overview on the existing appstores (not only J2ME) can be found in the appstore Wiki on www.wipwiki.com/appstores. Furthermore there are various vendors who provide solutions for provisioning of Java applications over Bluetooth, including Waymedia⁸, Futurlink⁹ and Broadburst¹⁰.

1) publish.ovi.com

2) store.java.com

3) applications.samsungmobile.com

4) www.handmark.com

5) softwarestore.sprint.com

6) www.o2litmus.com

7) www.getjar.com

8) www.waymedia.it

9) www.futurlink.com

10) www.broadburst.com

Mobile Developer's Guide

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 because 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 bought Trolltech — the company behind Qt — it was with the goal of bringing this same ease of development for multiple platforms to Nokia's mobile devices. Today, Qt can be used to create applications for Symbian devices and MeeGo, an open source platform initiated by Nokia and Intel.

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

Qt uses standard C++ but makes extensive use of a special pre-processor (called the Meta Object Compiler, or moc) to deal with many of the challenges faced in standard C++ development. As 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 the communication between objects¹, the output from one object is a "signal" that has a receiving "slot" function in the same or another object.

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

Adding Qt features to an object is simply a case of including QObject (which is achieved by add 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 creating GUIs (the QWidget object), complex graphical views (the QGraphicsView object), managing network connections and communications, using SVG, XML parsing, and using scripts among other.

Many developers who have used Qt report that application can be written with fewer lines of code and with greater in-built reliability when compared to coding from scratch in C++. The result is that 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 also from being open source, with a large community of developers contributes to the content and quality of the Qt APIs, Should you wish to get involved the source code is made available over Gitorious¹.

Prerequisites

Setting up a development environment to create applications for Nokia's Symbian devices was time consuming, although things are improving. The Maemo environment is a little easier, but if you had ambitions to create a cross platform version of your application, then you would have required a Microsoft Windows PC and Linux computer.

The Nokia Qt SDK installs everything you need to create, test, and debug applications for Symbian and Maemo from a single package. At the time of writing, only the Windows version offers full functionalities while the versions for Apple and Linux were still in progress.

¹) qt.gitorious.org/

Creating your application

The 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 with full 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 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's been necessary to revert back to the platform's native APIs. This is where the Qt APIs for Mobility come in. These APIs provide a common interface to device data such as contacts, location, messages, 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 Nokia Qt SDK also enables you to integrate the full platform SDKs so you can use native APIs if you want to.) 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();
}
```

Getting Qt

If you are already familiar with C++ development on the desktop, creating Qt applications for Symbian or Maemo will be straightforward. Once you have mastered the Qt APIs you should find you can code much faster and with fewer of the usual C++ frustrations you are used to. Qt has many interesting features, such as WebKit support enabling you to integrate 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 also deliver very good performance and usability. For most application the levels of performance will be comparable to that previously achieved by hardcore native applications only.

Testing

The Nokia Qt SDK includes a lightweight emulator that enables application to be tested and debugged on the development PC. The emulator includes tools that enable device data, such as location or contacts records to be defined so that the application's functionality can be fully tested. The emulator does not, however, eliminate the need for device testing. Aspects of the application, such as performance and usability, can be fully assessed on a device only. In addition, the Nokia Qt SDK includes the tools required 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 emulator uses the desktop computer's hardware, hardware that differs from the equivalent technology on a mobile device.

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 on signing is required. For application to be installed on Symbian devices, signing is necessary. More information on signing applications for Symbian is given in the Application Signing section of the chapter Programming Symbian Apps.

Distribution

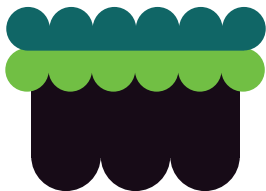
Ovi Store is Nokia's answer to the iPhone App Store and Android Marketplace. Importantly, once an application has met Ovi Store quality requirements — beyond removing indecent or illegal applications — there is no restriction on the application that can be hosted. So you will find many applications in 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 €50 fee. When your application starts selling you get 70% of revenue after any applicable taxes and — where applications are purchased through operator billing rather than a credit card — the operator's fees. Many developers view the operator's fee negatively, as it can account for up to 40% of the sale price. However, the extra sales generated when this billing method is available generally means that you will receive more revenue than from credit card sales.



Mobile Developer's Guide

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 web widgets. Web widgets are based on the scripting and mark-up languages used for websites (HTML, CSS and JavaScript). To run a web widget a device needs to provide browser runtime environment. Most devices have a browser that supports web languages, so it is a small step in the technology to make almost any device capable of running widgets. The biggest advantage of widgets is that they offer probably the easiest route into mobile development. Web developers are able to create widgets using their existing web design skills and code in the languages they 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.



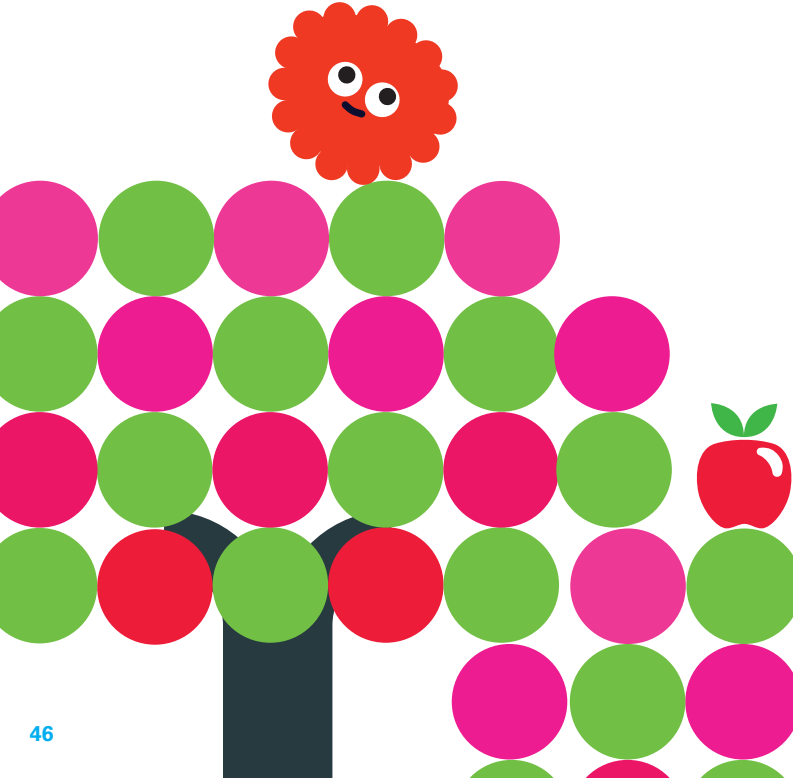
Widget characteristics

In general, a widget can be characterised as a small website installed on a device. But if that's the case, why not simply use a website? Well, there are several advantages of a widget compared with a web page:

- 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, Symbian devices from Nokia place a widget's icon in the device's menu making it as easy to find and open as a native application.
- 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 Nokia's Web Runtime (WRT), are able to provide summary views users can add to their device's homepage.
- 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 net work 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.



If there is a challenge in creating widgets, it is the lack of a universal support of standards. W3C (together with many major players from the mobile ecosystem) is pushing forward with the definition of standards. This standardisation is still underway and information on its progress can be found in the W3C Wiki¹. Because the standards are not complete, it's important to be aware that each widget technology has slightly different ways of implementing the draft specifications and not all environments implement all of the draft standards. In general, a widget that follows the specifications given by the W3C will enable you to target these widget environments:

- **Nokia WRT (Selected S60 3rd Edition, Feature Pack 2 and all S60 5th Edition devices):** bit.ly/nokia-wrt
- **Blackberry (min. v5.0):** bit.ly/blackberry-widgets
- **Windows Mobile (min. v6.5):** bit.ly/winmo-widgets
- **Vodafone360:** bit.ly/vf-widgets
- **J2ME, Blackberry, Android, etc.:** bit.ly/es_skylight

To port your widget over to e.g. iPhone and Android, you can use tools like PhoneGap², Titanium from Appcelerator³, and Rhomobile⁴ among others. Of these options, PhoneGap offers a solution that is closest to the W3C approach.

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

2) www.phonegap.com

3) www.appcelerator.com

4) www.rhomobile.com



Prerequisites

Widgets, just like websites, are created entirely in plain text. These text files are then packaged as a zip archive. This makes it possible to create widgets using a text editor, zip application, and a graphics application (to create an icon and graphics for the widget). If you have a tool for web development it can be used for widget development. The primary advantage of using a web editor is the support these tools provide for composing HTML, CSS, or JavaScript. Nokia WRT for example offers plug-ins for Aptana Studio, Dreamweaver, and Visual Studio. These tools provide template projects, 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 usual 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 will 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 also use AJAX and one of the various JavaScript libraries, such as jQuery, MooTools, YUI, or Dojo. Depending on the widget platform you are targeting you may be able to use more advanced technologies such as Canvas, SVG, Flash Lite, or even HTML5 features such as the <audio> and <video> tags.

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

```
serviceObj = nokia.device.load("geolocation");

serviceObj.getCurrentPosition(success_callback,
error_callback,positionOpts);

...
function success_callback(result){
    Lat = result.coords.latitude;
    Long = result.coords.longitude;
}
```

While standards are still to be finalised, 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 worth while if you want to create leading edge widgets.

Testing

It is always good to be able to test on a PC and you can do this for your W3C compliant widgets in Opera 9 or later. 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. These preview tools enable you 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 finished 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. 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.

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

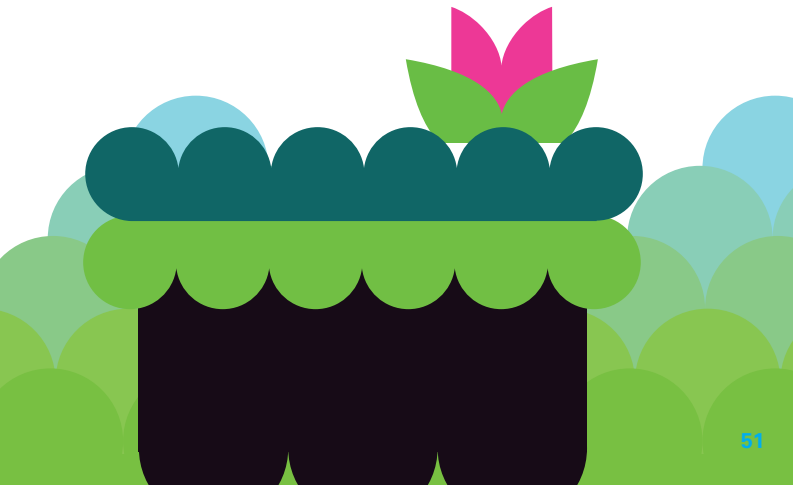
Distribution via a website is not the only option. Many application stores welcome widgets. As we went to press, the only store that supports W3C widgets is the Vodafone Widget store¹, but by packaging your widgets appropriately you can upload them into Nokia Ovi store², the Windows Marketplace⁷ or RIM Blackberry AppWorld⁴. You can use tools, such as PhoneGap, to port your widget to a native application environment, thus gaining the option to use other stores, such as Apple AppStore and Android Market among others.

1) widget.vodafone.com

2) store.ovi.com

3) www.windowsmarketplace.com

4) appworld.blackberry.com/webstore



Mobile Developer's Guide

Programming Flash Applications

Developing Flash applications (and Flash animations within applications) for mobile devices is the same methodology as developing browser-based Flash applications for the desktop, but you must be knowledgeable of the requirements of the target device. There are two components to consider when developing Flash applications and animations: The Flash animation itself, that theoretically can be run on any device, and the Flash engine or player installed in a device that has varied operating functionality.

Adobe open sourced the Flash specification, permitting independent developers and companies to develop Flash-compatible engines and players. Authoring can be done using the Adobe Flash Professional or Adobe Creative Suite (any edition with Flash) software. There are also open source versions including Gnash¹ and GameSWF² which are designed for the desktop systems. Other commercial alternative Flash platforms exist that provide good integration with the host devices. Many of these Alternative Flash platforms run outside the browser environment working directly with a device's native APIs.

Pay special attention to how you design your Flash application. Adobe Creative Suite provides you the option of developing to run inside a web browser or as a standalone Adobe AIR application. Alternative Flash-compatible SDKs go further and can integrate Flash content into existing 2D and 3D software applications.

1) www.gnashdev.org

2) tulrich.com/geekstuff/gameswf.html

The Adobe Flash players installed in mobile devices are not upgradable in the field. This means that you must author the Flash application to match the Flash player version in the device. Flash-compatible engines and players from Alternative Flash-compatible SDK and tools vendors have a different range of functionality, typically based on a vendor's vertical market focus.

Flash support on mobile devices varies widely from full support to no support. We anticipate that Flash support will become standardized on most mobile devices as the hardware platforms include faster CPUs and as they add-in GPUs with OpenGL ES graphics acceleration.

Prerequisites

If you are authoring or modifying Flash animations you require:

- An authoring tool such as Adobe Creative Suite with Flash;
- A Macintosh or Windows computer system that runs the authoring tool;
- Animators, graphics artists and/or ActionScript developers who understand how to develop for mobile phones.

A Flash application (or Flash movie) can be comprised of two independent software engines. The first is the animation engine that renders deterministic graphics to a display; the second is the ActionScript engine that uses input events (time, keyboard, mouse, XML) to make runtime decisions to play internal portions of a Flash animation or to load and play an external animation or video. ActionScript is a scripting language based on ECMA-Script available in three versions.

Then you must decide exactly how the application is going to operate and select the SDK that best supports your target device(s).

It has been reported that Adobe Creative Suite 5 will support ActionScript 3.0 and Flash 10.1 with the full 3D and 2D feature set, and smartphones. If you want utilize the maximum features like 3D and ActionScript 3.0 compatibility, this is the way to go. As stated earlier in this guide, one drawback is poor performance: Large binary files may run too slow on less powerful devices.

Adobe Creative Suite 3, 4 or 5 can be used to author Flash content that runs on Alternative Flash-compatible SDKs, engines and players. These Alternative Flash-compatible SDKs generally support ActionScript 2.0, and Flash 8 or Flash Lite 3.1 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.

Flash is supported or installed on these devices:

— Feature phones

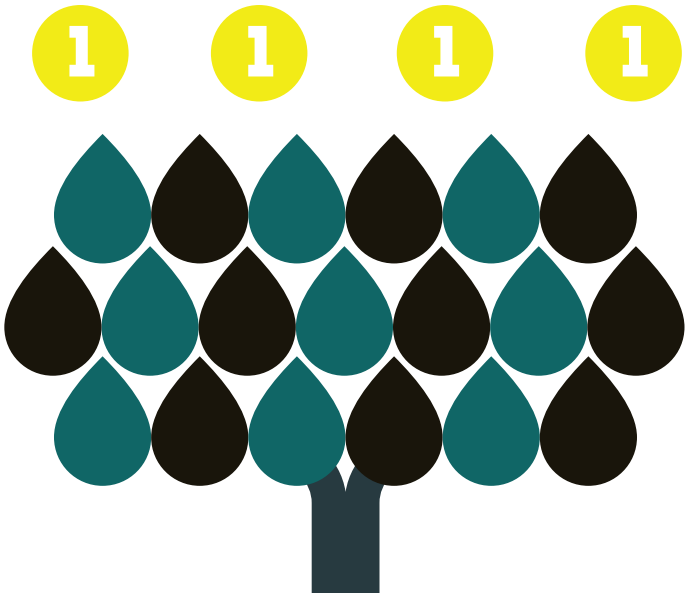
Many feature phones from Nokia, Samsung, Sony and a host of others have included support for Flash Lite. The Flash Lite player is compatible with Flash 3, 6 or 8 depending when the device was manufactured. These are perfect for simple Flash games (e.g., puzzle and card games).

— Smartphones

Some smartphones have a Flash engine or player pre-installed. This is a critical factor in developing Flash applications for these devices, and all applications must comply with the device manufacturer's license terms and API specifications.

Full Flash support has been announced for Android-based devices, and RIM's Blackberry. Developers should clearly understand the restrictions of each platform and also understand that there are a number of Alternative Flash-compatible products that can enable Flash to operate on a number of these devices.

Now that you have selected tools and the target device(s), you need to license the development kit and follow the manufacturer's process for publishing to that platform. You will also need to purchase one of the target devices to do testing or you can use a web-based device testing service. See the Chapter Programming J2ME / Java ME Apps under Testing for more on testing resources.



Developing Flash Applications for Smartphones

The new Adobe Creative Suite 5 has a new, built-in smartphone emulator. This enables developers to test their application on a supported target device. For devices not supported by Adobe Creative Suite 5, a developer may want to seek out Alternative Flash-compatible SDK and tools vendors for testing.

An important consideration for developing smartphone applications is saving your Flash animation in a format that is compatible with your target device. A developer needs to be aware of the implications of saving their Flash animation in Flash 10.1, Flash 10, or Flash 8 (Flash Lite 3) and of using ActionScript 2.0 or 3.0.

Tricks and Traps:

- Avoid Flash animations that have very fancy visual effects, simply for the sake of fancy effects. Test on the target smartphone to know what works well, and what does not. A Flash animation using ActionScript 3.0 will be 3-4 times larger in binary size and will likely result in poor performance. Decide carefully if you really need to use a feature that is only in ActionScript 3.0, as it is a completely different scripting language than ActionScript 2.0. ActionScript 3.0 is more complicated to learn and to program, and all object oriented languages require a lot more software developer experience to use effectively.

- Avoid sliding a Flash object across the screen unless you know it performs well. Redrawing every pixel multiple times a frame without 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 2D bitmaps to replace SVG objects that will never change size.
- Test on your target phones to know how much memory is being use during the application runtime. The more memory being used directly results in more CPU power required to execute the code, which affects the device's battery life.
- Minimize network connectivity to when it is absolutely required. The more OS APIs and independent software you use, the more work is being done, again 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 for the developer to implement in the application. So your user doesn't need to re-key data after a power failure (save and restore settings).

LUP
LUP
LUP



Testing

How your Flash application is architected and on what target device your application will run has a direct impact on what testing is required. If you have developed an Adobe Flash browser-based application or Adobe AIR application, then it is 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, then you should consider testing the application with one of the Alternative Flash-compatible SDKs or tools.

Distribution

Every manufacturer has a different method of publishing your application, and distributing it. Since you have selected your primary platforms, it should be fairly easy to determine the rules and revenue sharing amounts.

Things to keep in mind:

— It may take a few weeks from the time you submit your application to an app store until you receive a response and/or your application becomes available for purchase. Use that time to talk to other developers and media and get your marketing plans together. Once your application is available for purchase from an app store, the hard work begins to market it so that your application rises “above the noise” and gains attention and you start to build a user base. Of course your masterful application submission may also fail. Manufacturers’ can reject it for multiple reasons. Know what they expect before you spend development time.

- During the submission process, you should plan you bug fix release schedule. The first release must be well tested, and free from all obvious or easy to find bugs. Your second release cycle starts the minute you publish the first release.
- Above all, listen to customer feedback, and adjust your application to fulfill those requirements. Customer loyalty is the ultimate goal for an application developer.



Mobile Developer's Guide

Bringing Your Content to the Mobile Web

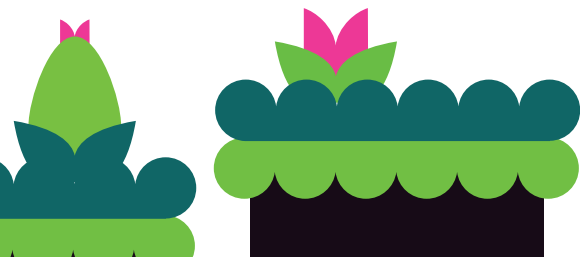
„Don't make me think!“- this title of one of the most popular books about usability¹ still summarizes very well what usability is all about. The developer should always keep in mind that it does not matter how fancy his mobile website might be: The user should be able to easily and intuitively use it. 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 nobody by simply offering a smaller version of your classical website.
2. Keep all paths open: Leave it up to the user to access either the mobile or the desktop version of your website.
3. Keep it simple: Avoid complex navigation structures, the user won't dig that deep anyway while he is on the go.
4. Avoid text input wherever possible: Text input on mobiles sucks. If you really need to implement it, make sure to use wide input boxes so that the user sees what he is typing.

¹) Steve Krug: "Don't make me think - A Common Sense Approach to Web Usability", New Riders, 2nd edition, 2005



5. Adapt the media: Adapt all pictures, videos etc. to be displayed properly on the handset (check the corresponding chapter in this guide: „Implementing Rich Media“). Avoid formats like .doc or pdf.
6. The user is a creature of habit- respect that: Adapt usage patterns from classic websites like linking logos to the starting page or offering corrections of mistyped search requests.
7. Think of stubby fingers: When optimizing your content for touchscreen phones, never use clickable areas smaller than 1cm x 1cm.
8. Use sharp contrasts: Use contrasts, font and background colors which 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. Again: Don't make the user think!: Try to implement intuitive navigation, don't force the user to make decisions more often than necessary. He will often click the first link in a list anyway.





Mobile usage patterns and its consequences

The usage pattern at a desktop computer is often described as „Hunt & Gather“. The mobile usage is totally different. When using internet on the mobile handset, the user is usually on the move. He wants to know more about his surrounding or just fill some minutes. Therefore, mobile internet usage is often described as „Quick Enjoyment“.

Applications have to take these differences into account. For example, a search mask for stationary users has to offer comprehensive options, on the mobile handset it has to be more straight-forward, focused on the certain action.

These adjustment cannot be realized by a machine. This is one reason why it is indispensable to create special mobile versions of websites.

The mobile user has no mouse, often he has 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 although the user is not able to move a cursor freely along the site.

It is best to test the usability of a site under real-life circumstances: Take your mobile handset to a busy public place and try to find all relevant info of your application by just using one hand. You will see very quickly where your mobile site shall still be trimmed.

But apart from things like markup, image formats and navigation, you should never forget about the most valuable good of the mobile user: Battery power. Complex websites with a lot of JavaScript, CSS and Flash elements need a lot of processing power which means battery power.

Some History on the Mobile Web

WAP: The stone age of mobile internet

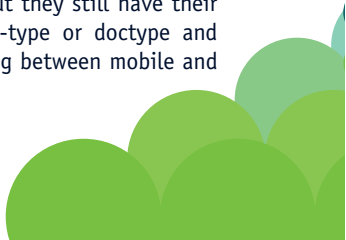
In the beginning, everybody in the business expected the mobile internet to be a cashcow. The WAP technology made it possible to send small texts on monochrome displays and b/w images in the WBMP format to the user's cellphones, the industry charged him for that by inventing complicated data transfer rates or strange subscription models. Many players expected to get rich by selling information which was already available for free on the internet.

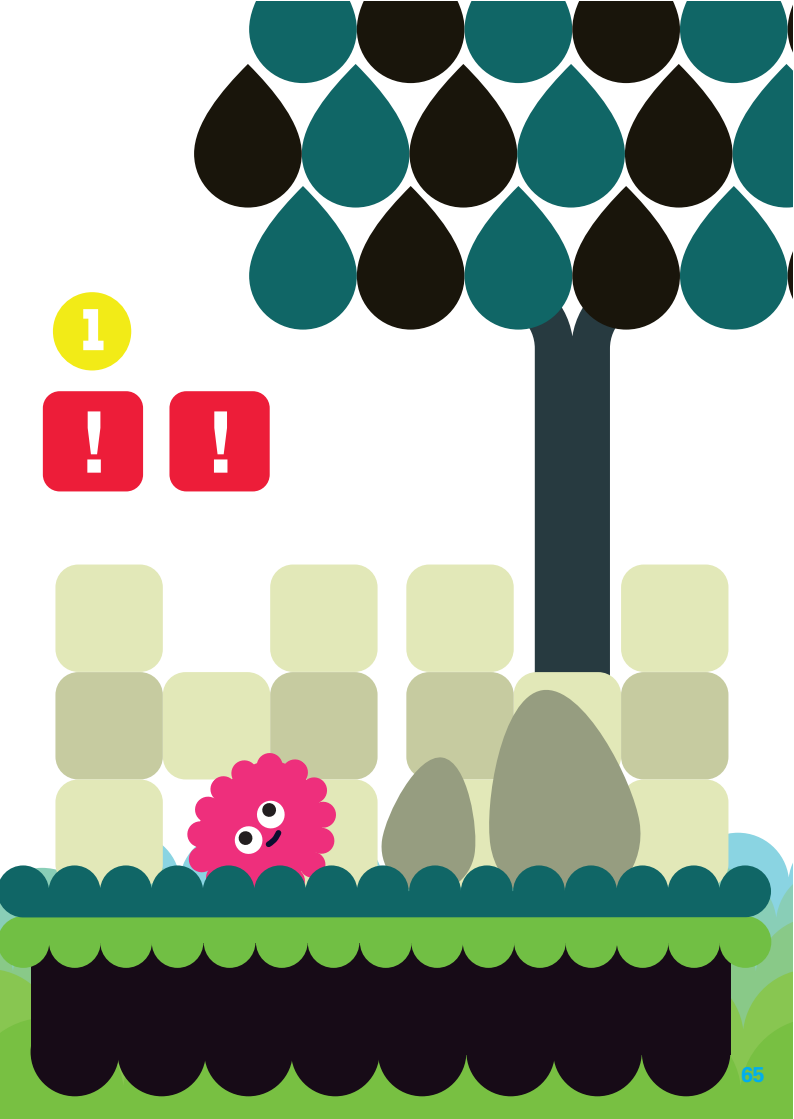
The Wireless Markup Language WML which was used by that time was another reason why the expected breakthrough did not happen as fast as some were thinking: While the different internet browsers (Internet Explorer and Netscape) allowed a „dirty“ html, WML requested a valid XML structure and proper UTF-8 encoding. The usage of images was limited to black and white pictures which were a lot less sexy than what the users knew from their home computers.

The first tries to lift the mobile internet to the next level were made by NTT DoCoMo by inventing CHTML and Microsofts Windows CE which worked best with HTML/3.2. But both of them were just interludes which were replaced by XHTML/MP 1.0.

Current Situation

Today, 99% of all mobile browsers are still supporting XHTML/MP 1.0, for many low-end devices this language still delivers the best results. On the other hand, the so-called Full Web Browsers have to support HTML/4.0, but they still have their differences when it comes to content-type or doctype and choose different views for distinguishing between mobile and desktop websites.





How to adapt content for the mobile user

Static Version

Of course you can simply ignore the possibilities of automatic optimization and leave it up to the user: Create different versions of your content, let the user start with the most low-level version and let him decide manually which variation is the most comfortable one for his device and usage patterns in a next step.

But since you are dealing with a user who is on the move and does not want to spend a lot of time with finding out how to see properly what you are delivering, this is probably not the best way to go...

Automatic adaption technologies

To adapt the content to different devices, you basically need two components: One logic that detects the device, knows about its browser and its characteristics — a device database. The second component interpretes these characteristics and adjusts the content accordingly.

The most popular Open Source device database is the WURFL project¹ which offers comprehensive information via XML. A lot of tools deliver APIs for detecting the browser, delivering its properties and adjusting the content, the markup and the images. One example for an Open Source project which adjusts content is MyMobileWeb², financed by public funds and Telefonica.

1) wurfl.sf.net

2) mymobileweb.morfeo-project.org



But of course there are also purely commercial providers. Again, some are concentrating on device data and detection, others are focusing on offering software platforms which adjust the content accordingly.

Examples for commercial device data and device detection providers:

- dotMobi¹ offers several APIs to access their device database DeviceAtlas² and intelligent detection
- DetectRight³

Some commercial content adaptation software providers:

- Sevenval⁴, a platform independent technology which works via HTTP and markup. This solution is available as a Service or can be installed on Linux systems.
- Volantis Mobility Server⁵ is running 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 Server⁶ is running on BEA, IBM WebSphere, JBoss, Tomcat and Caucho Resin.

1) www.mtld.mobi

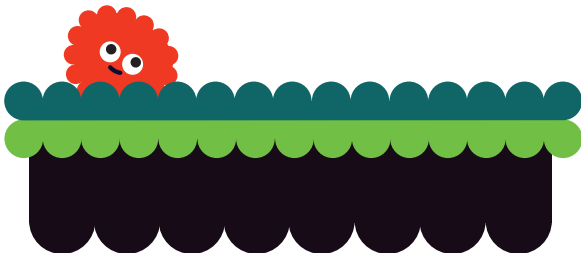
4) www.sevenval.com

2) www.deviceatlas.com

5) www.volantis.com

3) www.detectright.com

6) www.mobileaware.com



Satisfy the Interpreter of Your Content: The Browser

Markup

Of course it would be great if there would be 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:** validator.w3.org
- **W3C mobileOK checker:** validator.w3.org/mobile
- **dotMobi testing tool:** mobiready.com
- **Korean MobileOK Test & Validation Service:** v.mobileok.kr

As a general advice, it is always good to stick to UTF-8 encoding. You certainly can also consider going low-level and create 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, better use percentage values. Even when using device databases, the browsers still have different display methods (fullscreen, landscape, portrait) and only some allow 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 flatrate, so do not use images too 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 the possibility to easily scale your images¹.

¹) www.imagemagick.org

When thinking about the image format, GIF and JPEG are still the most solid choice. Of course PNG offers more flexibility, but when it comes to transparency, you cannot always be sure to what degree it is supported. And some operators still use image proxies which might not be able to handle this.

Tables

For stationary web, tables are no longer used for webdesign. 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" for not displaying unmeant spaces. CSS is simply not capable to assure this with all browsers.

CSS

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

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

Fonts

Do not deal too arbitrarily with fonts: all browsers just dispose of a limited set of font-types. Better focus on the font size. Use relative values (small/medium/large) rather than fixed values (pixels) — the browser knows best how many pixels to use for displaying a large, a medium or a small font.

Cookies

You can of course use cookies and should do so, but only when really needed. And never trust too much in them: although it might work fine during one session, the cookie might get lost afterwards. This is

why you should always offer alternatives like an URL based parameter or a personalised bookmark for permanent settings.

Script / AJAX

According to www.device-trends.com, 93% of the mobile web users who visited the associated websites, were using a browser which supports JavaScript, 70% were able to handle AJAX. So it would be wrong not to use the possibilities of these technologies. But you should make sure that your side works fine without them as well.

Let's do some pigeonholing: Device categories

You have no idea what kind of devices your visitors mostly use? Then you should start low-level and implement a tool which detects the devices who 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 collected data which give you an idea of which devices are actually using the mobile web:

- **AdMob** metrics.admob.com
- **mobiForge** mobiforge.com/designing/story/effective-design-multiple-screen-sizes
- **Opera** www.opera.com/smw
- **Sevenval** www.slideshare.net/sevenval/tag/devicetrends

When using this data, keep in mind where they come from and how they have been generated: In which region have they been collected, are they really reflecting the users' mobile web usage or just the pure general market share of the devices? Does the data maybe only reflect the user base of certain operators or technology platform? Is it about page impressions, visits or unique users?

Never trust any report blindly. Use several sources and make your own picture out of it.

Simple Devices

Probably the vast majority of your users will access your site with a basic handset like described by the W3C in the Default Delivery Context¹ or by Luca Passani in his baseline device description².

To fit their needs, these are our recommendations:

- XHTML MP/1.0
- screensize: 176x144
- GIF or JPEG image format
- max. 20kb page size (incl. images, CSS)
- basicTableSupport, without „rowspan“, „colspan“-nonested tables
- minimal CSS (WCSS 1.0)

Of course even these basic characteristics exceed the possibilities of many older handsets, but people who tend to use the mobile internet usually own handsets who should properly display pages that follow the guidelines above. Surfing the internet with much older devices is no fun anyway.

1) www.w3.org/TR/mobile-bp/#ddc

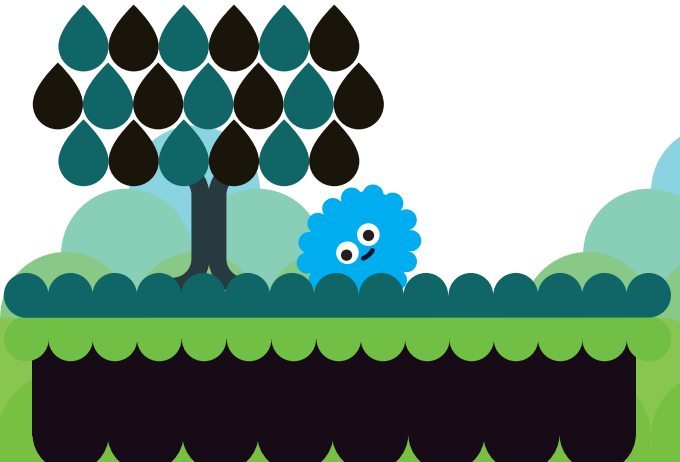
2) www.passani.it/gap/#baseline

Full Web Browser-supported Devices

Full Web Browsers are characterized by their capability to display any website. These are all webkit based browsers, Opera Mobile, Microsoft

Internet Explorer, Netfront (version 3.4 and later), UCWEB, Nokia Web Browser or Fennec. Devices who use a Full Web Browser do not necessarily require any technological concessions regarding website design, but we suggest to follow these guidelines:

- HTML/4.0 Strict
- screensize: 240x320
- GIF or JPEG Images, PNG without alpha transparency
- max. 50kb page size (incl. images, CSS)
- basic table support, without „rowspan“, „colspan“ - no nested tables
- keep CSS simple, tables are not bad
- script is allowed, but not needed (avoid unneeded script or animation effects – think of the battery)
- if available, use AJAX to get content



Touch Devices

Here it is referred to modern devices with a powerful browser like the iPhone, Android or Symbian handsets. The touch-sensitive UI offers the possibility to move more freely within a website. At the same time, it makes it more complicated to choose one line from a list or any other smaller element within a page.

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

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



Use GPS in the Browser

Devices like the iPhone (firmware version 3 and later), the Blackberry (firmware 4.2. and later) and browsers with Google Gears allow you to use the GPS information from within the browser. The W3C published a specification about this¹, but unfortunately this is not supported by all devices. Nevertheless you can still use a simple Java API which is laid over the different implementations².

For further information about how to use GPS location information for web-based Apps and services, check out mobiforge³.

Testing your Mobile Website

There are several ways to test your mobile website:

User-Agent / Browser

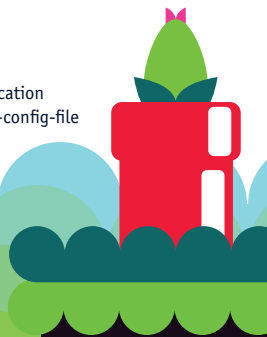
Several desktop browsers offer the opportunity to change the user-agent and emulate a device detection for automatic adaptation. For Firefox users, the User-Agent Switcher is available on addons.mozilla.org/en-US/firefox/addon/59. mobiForge offers a configuration file for this Add-On which contains some properties of mobile handsets⁴.

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

4) www.mobiforge.com/developing/blog/user-agent-switcher-config-file



Emulators / SDKs

The better option to test your mobile site are emulators or SDKs of the manufacturers. We made good experiences with 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:** developer.symbian.org/main/tools_and_kits

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

Remote and Real Device Testing

As mentioned in the other chapters in this guide, the best is always to test your product on as many real devices as possible under real-life conditions. The alternatives are always remote testing (by using services like www.deviceanywhere.com or Nokias RDA¹ and crowdsourcing the testing with the help of www.mob4hire.com for example).

¹⁾ apu.nbhub.net

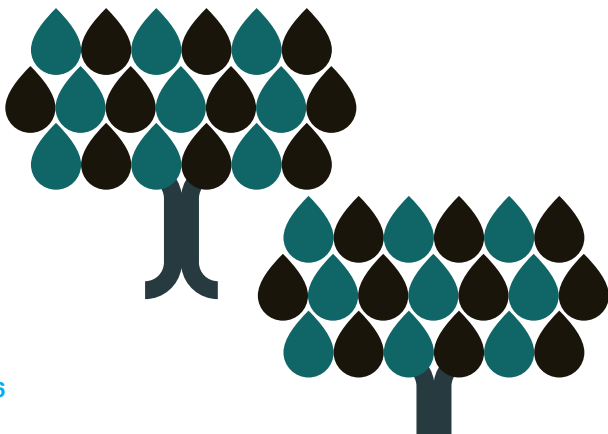
Learn More — On the Web

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:

The mobile website is not the small website, it is the mobile one! Keep it simple.



Mobile Developer's Guide

Implementing Rich Media

„So many handsets, so many standards”— Again this is true for the list of supported media formats on mobile phones. The majority of them can be found in the table on the next page, however multiple variations amongst handsets even within one vendor or phone type firmware.

Streaming

Depending on your choice of offering streaming or downloadable contents, there are some restrictions. To offer streaming content, the easiest setup is to use Apple's open source Darwin streaming server¹ which can serve streaming video and audio with highest compatibility (except for Windows Mobile) and reliable Realtime Streaming Protocol (RTSP). For Windows Mobile, Windows Media Server² is the easiest choice (available as a free download). 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. RTSP is a protocol which is designed for low-bandwidth mobile devices. If you need a PC player to read 3gp rtsp streams, use QuickTime or Real player.

Progressive download

To avoid setup of streaming servers, a good alternative is to offer progressive downloads, for which your media files can be served from any web server if you hint your files. Hinting is the

1) developer.apple.com/darwin/projects/streaming

2) en.wikipedia.org/wiki/Windows_Media_Services

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 first 15 seconds).

So far the most reliable open source hinting software found is Mp4box¹. Use `mp4box -hint -latm` for mobile phone compatible hinting (note: `-latm` is only needed by AAC audio codecs). mp3 files do not need to be hinted.

Ringtones

To use audio as a ringtone or within a J2ME application, make sure that the audio format is supported by your target devices. Within J2ME `.wav` or `.mid` is the best choice for maximum compatibility, however the file size might restrict you as it is uncompressed. There are more than 10 file formats for ringtones, but we suggest to use the MP3 format, 64 or 128kbps, max 30 seconds and a file size of between 300kb and 500kb or AAC/AAC+ 24 kbps (mono) or 48kbps (stereo) on high-end phones to ensure maximum compatibility.

Flash

Flash is very popular on the many mobile devices today and it can be really fun to integrate Flash animations into the mobile applications. Flash animations can be used to create “eye-popping” graphics in 2D user interfaces (UI).

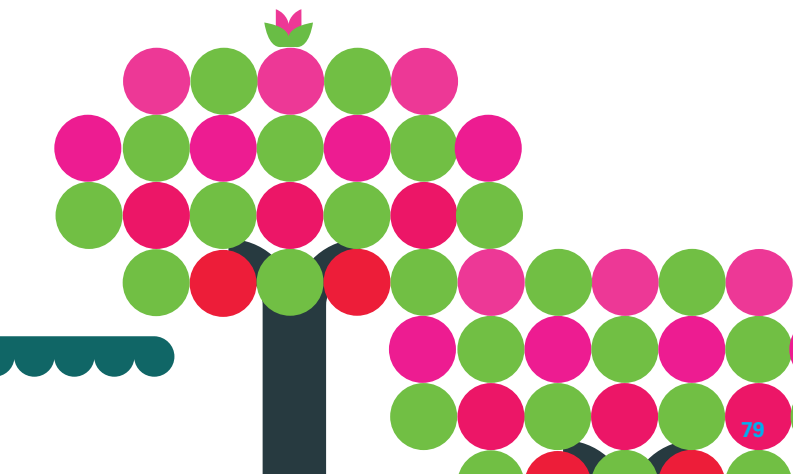
Flash animations are based upon Scalable Vector Graphics (SVG). This means the developer or designer can create a Flash object once, save it in the SWF file format, and that Flash object can be rendered to any size display.

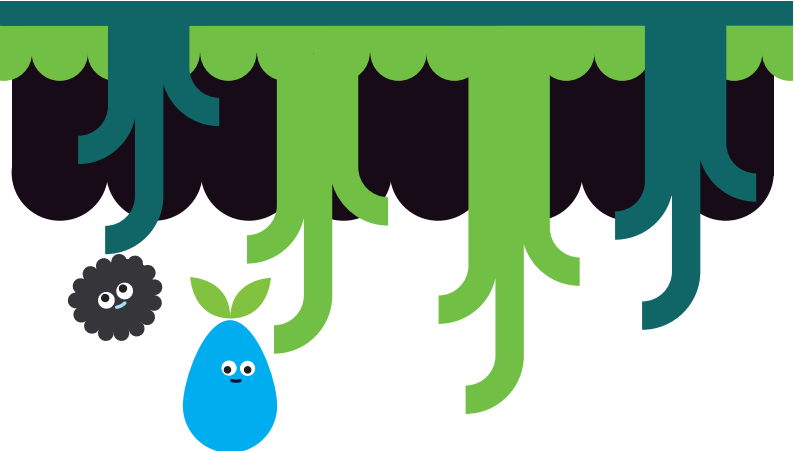
¹⁾ gpac.sourceforge.net



The same Flash animation will render with high quality on a mobile device or on a 1080p HD display screen.

Flash games and applications are very popular, and in some cases, Flash can be used for UI menus and icons. Flash-based rich media UIs can scale to any display screen resolution automatically. This sharply contrasts with implementing standard 2D bitmap graphical UIs which require separate development for rendering to typical display resolutions, and resizing those UI elements at runtime results in pixilated (jagged) images as the bitmap is simply being stretched pixel by pixel. An application developer can utilize Flash animations to call attention to various UI elements within an application including prompting a user selection and showing intensity levels. Examples could include using Flash animations to demonstrate the draining of a device's battery level, show an incoming call, or to prompt the user for a selection or action, like responding to a calendar alarm or reminder.





OS/Brand	Container	Protocol	Video Codec	Audio Codec	Typical Resolution
Windows Mobile	.wmv or .mp4	http://	H264	MP3 or AAC	320x240 176x220
Symbian, Nokia	.3gp or .mp4	rtsp://	H263	AMR_nb or AAC	176x144 320x240
iPhone/iPod	.mp4	http://	H264	AAC or AAC+	320x240 320x480
Android	.mp4	http://	H264	AAC lc	320x480
Blackberry =<OS 4.2	.avi / .mp4	http://	MPEG4	AAC lc	480x320
Blackberry =>OS 4.3	.avi / .mp4/ .3gp	rtsp://	MPEG4	AAC lc	480x320
Palm OS	.flv / .mp4	http://	MPEG4	ADPCM or MP3	352x288
Other 3G Devices	.3gp	rtsp://	H263	AMR_nb	176x144

To display video streams on a mobile device, the video player needs to connect to the streaming server. This sounds easier than it usually is.

First of all, for many players a special APN (Access Point Name) needs to be configured. If this configuration is not correct, the videoplayer will not be able to connect with the server.

But even if the APN is configured well, some operators are blocking the RTSP protocol. This leads to the situation that the streaming will work within the operator portal and probably even on youtube, but your own server cannot be reached. Actually you cannot really do anything about this, you are totally depending on the user's know-how: He needs to manually set the APN to an open Internet connection.

But even if you do not use RTSP, there are strange things that might happen when streaming video on the mobile handset: If a Blackberry device processes video via WiFi, even large videos usually are displayed consistently. But if GPRS is used for data transmission, the content is processed by the RIM servers which define maximum values that may vary from server to server¹. As a consequence, content might behave differently even if the same device, same operator and same APN are used.

And now? Don't panic! Offer video to your users if you like to, but inform them if something is going wrong and if you use a device database, give them the option to chose manually.

1) www.blackberry.com/btsc/search.do?cmd=displayKC&docType=kc&externalId=KB10264

Media converters

To convert existing media of any kind to mobile phone compatible formats and adjust the frame rate, bitrate and channels at the same time, FFMPEG is a must have (open source) media format converter¹.

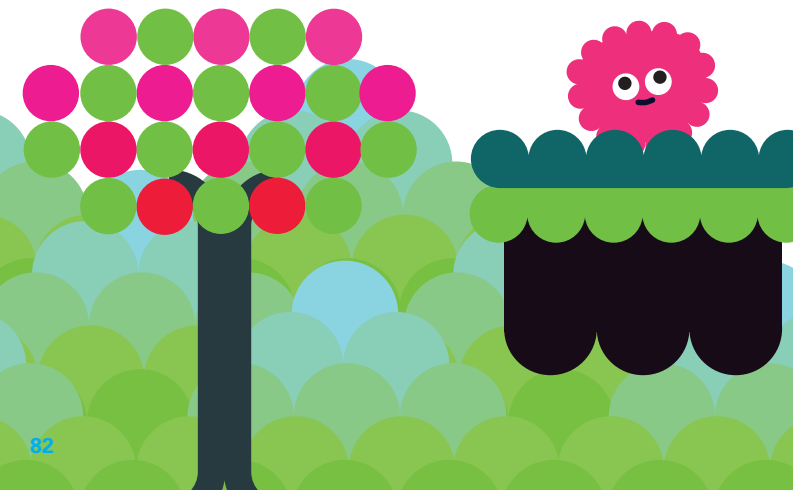
Make sure you build or get the binary with H263, AAC and AMR encoder support included. For MAC users,

QuickTime pro (paid version) is a good alternative to encode and hint 3gp files. A good alternative for converting video is “Super” from eRightSoft². If you are looking for a complete server solution with a Java / Open Source Background, check out Alembik³.

1) www.ffmpeg.org

2) www.erightsoft.com/SUPER.html

3) alembik.sourceforge.net



Mobile Developer's Guide

Implementing Location-Based Services

An interesting area for mobile development are location aware services, where the delivered information is adapted to the user's location. Applications could be anything from finding parking spaces nearby, analyze pollen reports for your area, find friends at the trade fair et cetera. There are two basic ways to go for it; either process geo information on the server-side or obtain location data for a device to use it directly. Doing everything on the server-side allows thinner clients, while fatter clients are more self-sufficient without constant network access.

How to obtain location data

Basically there are four ways to determine the user's position:

- Manual input
The user states the phone where he is, by choosing a location on a map, an area code, a city on a list. This is often an overlooked method.
- Cell tower (GSM triangulation) or WiFi-based positioning
The phone locks to a base station, and its unique ID can be looked up from a list that the operator keeps. There are also more advanced cell-based methods based on GSM/UMTS traffic, which requires a relationship with operators. Alternatively the received WiFi signal patterns can be used for determining the user's location. In both cases, the accuracy depends largely on the cell / net

work density. In urban areas a higher accuracy is obtained than in rural areas.

- **GPS positioning**
The built-in GPS module in the phone (or an external one) gives you a reading of 5-50 meters, depending on the terrain, canopy and wall materials.
- **Short range positioning**
Systems based on sensors, like NFC (near field communication) marketed by Nokia and others, where active or passive sensors are placed in the near proximity to points of interest, such as a museum or a shopping mall. Good for smaller, confined areas.

How to obtain mapping material

There are several public map sources out there and the static map type (distributed as bitmap tiles) is the most widespread: Free maps like Open Street Map or CloudMate and also maps that require partnerships and/or money like Navteq or Microsoft. Google Maps is released free of use at the time of this writing. Interestingly enough, several of these sources share a similar map format based on an origin (e.g. the North Pole), numbering like three tiles down and five to the left, and a scale. This allows the map sources to be interchangeable.

- **Open Street Map:** wiki.openstreetmap.org/wiki/Slippy_map_tilenames
- **Cloudmade:** developers.cloudmade.com/projects
- **Navteq:** developer.navteq.com
- **Google Maps:** code.google.com/apis/maps/
- **Microsofts Bing API:** www.microsoft.com/maps/developers

Implementing location support on different platforms

Location API for J2me is probably the most wide-spread method, outside of using the web browser. In there you find criteria for accuracy, response time, altitude, and speed, allowing even advanced location services. Other platforms like Symbian, Windows Phone and Android have similar offerings, even though they sometimes required signing of the application to access the location functions. A special case is the Apple iPhone SDK, offering an integrated support for location, even though there are restrictions on what map sources are used and how the location data is generated. The maps are often overlaid with geodata, which is available in a number of formats. One of the simplest is called geoRSS¹, and could look like this:

```
<item>
  <title>Alex's favorite fishing place</title>
  <description>Wonderful place for salmon
</description>
  <georss:point>18.256 59.92</georss:point>
</item>
```

It represents a point of interest described as its latitude and longitude, along with some metadata. There are many more formats for geodata, but the basic idea is the same, and more and more sources are harmonizing their data streams for interoperability.

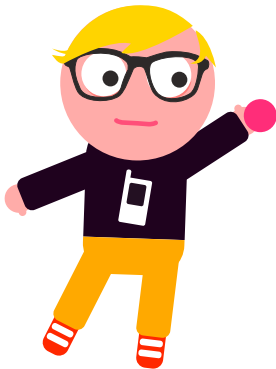
1) www.georss.org



Tools for making map apps

Several players in the industry provide developer-friendly tools and APIs as a value added service, such as map data providers, handset manufacturers and third-party developers. Using these dramatically speeds up the development and deployment of location-aware services. Each tool normally focuses on one or a few mobile platforms.

- **Nutiteq:** www.nutiteq.com
- **Spime:** www.spime.com
- **Mobile Gmaps:** www.mgmaps.com
- **Garmin Mobile XT SDK:** developer.garmin.com
- **Keymap SDK:** www.mapsdk.com
- **iPhone SDK:** developer.apple.com

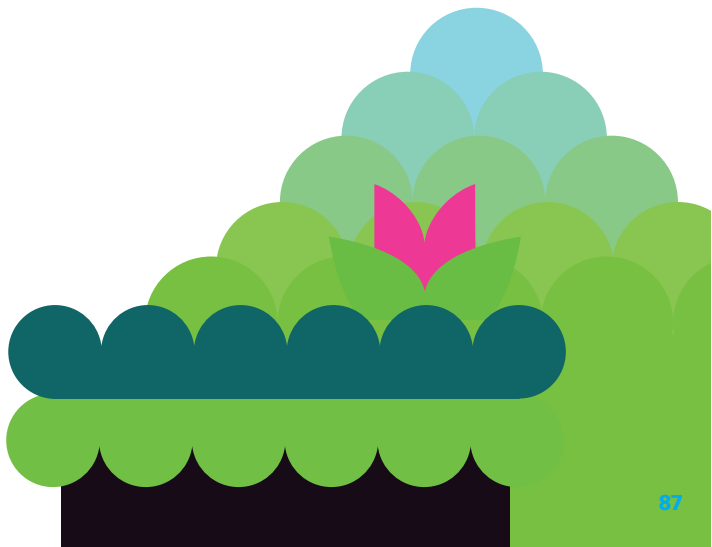


Mobile Developer's Guide

Now what - which Environment Should I Use?

The short and crystal clear answer: it depends.

The longer answer: think about your target users, about their needs, about their devices and their dataplans. And about your vision and the requirements for your idea. Remember that you are not necessarily restricted to a single application environment. A practical approach is to use the environment that you are most comfortable with and then move on to other environments for increasing the market reach of your app. Sometimes it also makes sense to combine different environments, for example by providing a mobile website for casual users and a J2ME app for your power users.



The following table provides a very rough overview about the individual strengths and limitations of each application environment. **Green** stands for good coverage, **yellow** for limited and **red** for bad coverage of the respective topic:

	Market Reach	Interactivity	Online / Offline	Availability of Developers	Device Tools	Performance	Fragmentation
SMS	Green	Red	Yellow	Red	Red	Red	Green
Bada	Red	Green	Green	Red	Yellow	Green	Yellow
Windows Phone	Red	Green	Green	Red	Yellow	Green	Yellow
BREW	Red	Green	Green	Red	Yellow	Green	Yellow
Flash Lite	Yellow	Green	Green	Green	Yellow	Yellow	Yellow
Widgets	Red	Green	Green	Yellow	Yellow	Yellow	Green
Native Blackberry	Yellow	Green	Green	Green	Green	Green	Yellow
Native Symbian	Yellow	Green	Green	Red	Yellow	Green	Yellow
iPhone	Yellow	Green	Green	Yellow	Green	Green	Yellow
Windows Mobile	Yellow	Green	Green	Yellow	Green	Green	Yellow
Android	Red	Green	Green	Green	Green	Green	Yellow
J2ME	Green	Green	Green	Green	Green	Green	Red
Web	Green	Yellow	Red	Green	Yellow	Red	Yellow



You can also compare application development with web development in a more general manner:

	App Development	Web Development
Approval process	2–8 weeks in app stores	Instant updates
Portability	Complex	Easy
Complexity	Complex	Easy
Monetization	Sale, Advertisements, in app purchase	Advertisements

Web development beats application development in many but one crucial area: monetization. Internet services are mostly free and the only reliable business model is often advertisements. In contrast, app stores shine in this area. All stores support charging for application, the iPhone app store nowadays also supports in app purchases. And of course you can embed advertisements in your application as well. If you need to earn money today, it will be easier with an application unless you want to earn money purely with ads.



Mobile Developer's Guide

Epilogue

Thanks for reading this fifth release of our mobile developer's guide. We hope you've enjoyed this and we helped you to clarify your options. Don't be put down 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!



Mobile Developer's Guide

About the Authors

Robert Virkus / Enough Software

Robert works in the mobile space since 1998. He experienced the Java fragmentation first hand by developing and porting a mobile sports betting client on the Siemens SL42i phone, which happened to be the first mass market phone with an embedded Java Virtual Machine. With this experience he launched the Open Source J2ME Polish project in 2004 that aims to overcome the device fragmentation barrier. He is the founder and CEO of Enough Software, the company behind J2ME Polish.

www.enough.de www.j2mepolish.org

Roland Gülle / Sevenval

Roland joined the mobile industry in 2001. At Sevenval he is responsible for the development of the adaptation technology and the FITML Platform which allows developers to create mobile internet portals. Roland is an active member of the the Mobile Web Initiative (MWI) and several open source projects.

www.sevenval.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

Wolfram Kriesing / uxebu

Wolfram has more than twelve years professional experience in IT. With two equivalently experienced experts he founded uxebu, a software consulting company focused on mobile cross platform solutions and web2.0, AJAX-oriented front-end engineering. He has been an active open source contributor on multiple projects and is currently a member of the the Dojo Toolkit project.

www.uxebu.com

Friedger Müffke / OpenIntents

Friedger is following the development of Android since the first announcement of the Open Handset Alliance in late 2007. As lead developer, CEO and co-founder of OpenIntents he promotes the building block philosophy of Android and tries to connect developers. He also organizes the Android conference droidcon.

www.openintents.org

Michel Shuqair / 24access Solutions

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 is now leading the mobile social network m.wauwee.com with over 850,000 members and supported by a team of Symbian, iPhone, Blackberry and Android specialists with headquarter in Amsterdam.

www.24access.nl

Alexander Repty / Enough Software

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 and written a series of articles on iPhone development. He is working as an iPhone developer for Enough Software since October 2008.

www.enough.de 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 focuses on mobile web development, together with two friends. Today, the team consists of 10 people focused mainly on usability optimization of mobile websites, social network applications and widgets.

www.inserteffect.com

Marco Tabor / Enough Software

Marco is responsible for PR, Sales and a lot more at Enough Software. He is coordinating this project as well and takes care of finding sponsors and of merging all the input we get from the mobile community.

www.enough.de

Alex Jonsson / MoSync

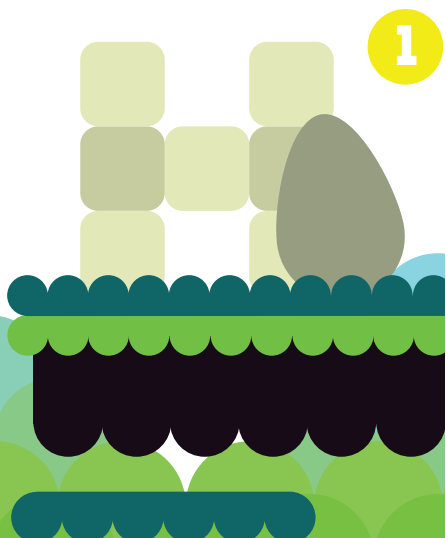
Alex likes anything mobile, both apps and web technology and connecting physical stuff to digital stuff. He holds a doctors degree in on-line publishing and distributed education. Behind this tech surface lies an eclectic urge to create new value by exploiting aspects of communication and media to bring people closer 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 New Zealand 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.





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

An initiative by:



www.enough.de



www.wipconnector.com



»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*

»Congratulations! A well written, very interesting little book with lots of good references and addresses. Fun to read.«
— *Jean-Marc Jobin, R&D, Datamars RFID Systems*

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

»A handy introduction and basic reference for the mobile development world.«
— *Kevin Farnham, java.net Blog Editor, O'Reilly Media*

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