

# Don't Panic

## Mobile Developer's Guide to the Galaxy



# Mobile Developer's Guide

## Table of Contents



### 7 Introduction

### 10 An Overview of Application Platforms

10 Native Applications

13 J2ME / Java ME

13 Flash Lite and Alternative Flash-compatible Platforms

14 BREW

14 Widgets

16 Websites

16 SMS Text Messaging

### 17 Programming Android Apps

18 Prerequisites

19 Implementation

21 Testing

22 Signing

22 Distribution

### 23 Programming bada Apps

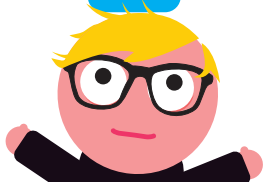
23 Prerequisites

24 Implementation

25 Testing

26 Distribution

1UP  
1UP  
1UP



- 27 Programming Native Blackberry Apps**
- 27 Prerequisites
- 29 Coding Your Application
- 29 Services
- 30 Testing
- 30 Porting
- 31 Signing
- 31 Distribution
  
- 32 Programming Flash Apps**
- 33 Prerequisites
- 34 Tips & Tricks
- 36 Testing
- 37 Packaging and distribution
  
- 38 Programming iPhone Apps**
- 39 Prerequisites
- 42 Implementation
- 42 Testing
- 44 Distribution
- 45 Books
- 46 Community
  
- 48 Programming J2ME / Java ME Apps**
- 49 Prerequisites
- 50 Implementation
- 51 Testing
- 52 Porting
- 55 Signing
- 56 Distribution
  
- 58 Programming Qt Apps**
- 60 Prerequisites
- 60 Creating Your Application
- 62 Testing



- 63 Packaging
- 64 Signing
- 65 Distribution
  
- 67 Programming Native Symbian Apps**
- 68 Prerequisites
- 68 Carbide.c++
- 69 Symbian/S60 Software Developer Kits (SDKs)
- 69 Porting to Symbian
- 70 Testing
- 70 Signing
- 71 Distribution
  
- 73 Programming WebOS Apps**
- 74 Prerequisites
- 74 Implementation
- 75 Testing
- 78 Distribution
  
- 80 Programming Windows Phone 7 Apps**
- 80 Development
- 84 Functions and Services
- 85 Distribution
- 85 Resources
- 85 Testing
  
- 86 Programming Mobile Widgets**
- 87 Widget Characteristics
- 89 Prerequisites
- 90 Writing Your Code
- 92 Testing
- 92 Signing
- 93 Distribution



- 94**    **Developing Accessible Apps**
- 94    Built-In Accessibility Features
- 96    Developing Accessible iOS Apps
- 96    Developing Accessible BlackBerry Apps
- 97    Developing Accessible Symbian / Qt Apps
- 97    Developing Accessible Android Apps
  
- 98**    **Programming With Cross-Platform Tools**
- 99    Limitations and Challenges of Cross Platform Approaches
- 108**    Cross Platform Solutions
  
- 112**    **Creating Websites for Mobile Usage**
- 115    A History on the Mobile Web
- 116    Content adaptation
- 118    Satisfy the Browser
- 122    Device Categories
- 126    Use GPS in the Browser
- 126    Testing your Mobile Website
- 128    Hybrid Apps
- 129    Learn More – On the Web
  
- 130**    **Implementing Rich Media**
- 132    Streaming vs download
- 134    Progressive download
- 134    Ringtones
- 134    Media Converters
- 135    Flash (Lite)
- 136    HTML5
  
- 137**    **Implementing Location-based Services**
- 137    How to obtain Positioning Data
- 139    How to obtain Mapping Services
- 140    Implementing Location Support on Different Platforms
- 141    Tools for LBS Apps

- 142** **Testing Your Application**
- 142** Testability: the Biggest Single Win
- 143** Headless Client
- 143** Separate the generic from specific
- 144** Test-Driven Development
- 144** Physical Devices
- 145** Remote Control
- 146** GUI Test Automation
- 146** Beware of Specifics
- 146** Crowd Sourcing
- 147** Web-Based Content and Applications
- 147** Next Steps
  
- 148** **Now what — Which Environment Should I Use?**
  
- 152** **Epilogue**
  
- 153** **About the Authors**
- 159** **Imprint/Contact**

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



# Mobile Developer's Guide

## Introduction

Welcome to the 7th edition of our Mobile Developer's Guide To The Galaxy. When we started this project in 2009, a lot of people thought the mobile ecosystem's fragmentation was a temporary phenomenon and that by 2011 a guide, such as this one, would be needed no longer. They thought that one, maybe two, platforms would dominate the market. The opposite is happening: New platforms continue to be introduced, others are evolving and spreading to new device classes so fast that it would be naive to assume one platform will dominate anytime soon.

We think that the need for this guide is therefore greater than ever. We hope that it will help you make the right decision about entering the mobile market and guide you through the first steps on the platform of your choice.

For this edition, we updated almost all the content and added new chapters on BlackBerry and webOS development.

This means that all the principal mobile platforms are covered in this guide.

With so many different platforms for developers to consider, one approach is becoming increasingly popular: Cross-platform development. Another new chapter covers this topic; it discusses the potentials of the technologies and how to determine if this approach is right for your development. Furthermore you find a new chapter about how to create accessible mobile applications. We would like to thank all writers and sponsors, our special thanks go out to Richard Bloor this time.

We are looking forward to receiving your feedback and input at [developers@enough.de](mailto:developers@enough.de) and hope to welcome many new contributors for the forthcoming editions.

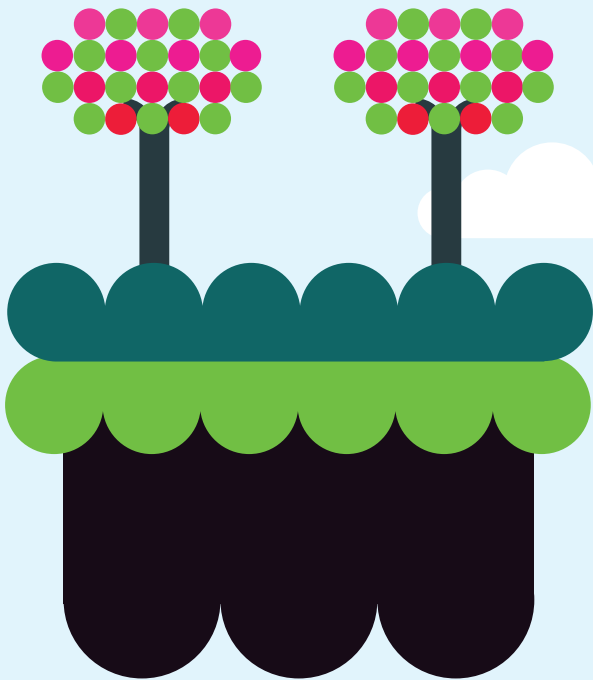
1

1

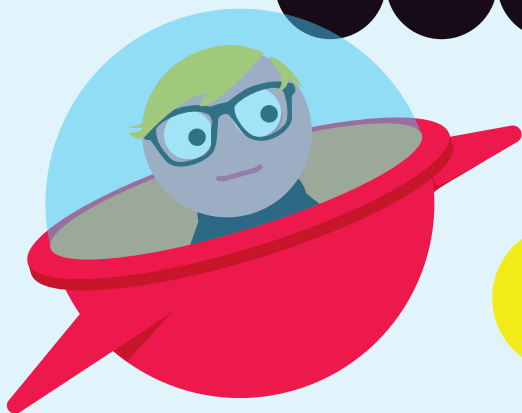
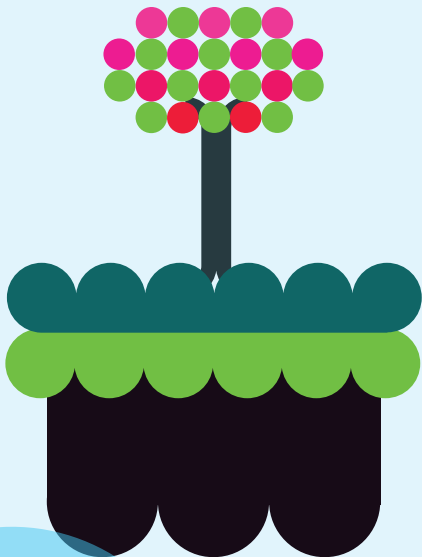
1

1

1



IUP  
IUP  
IUP



# Mobile Developer's Guide

## An Overview of Application Platforms

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

### Native Applications

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

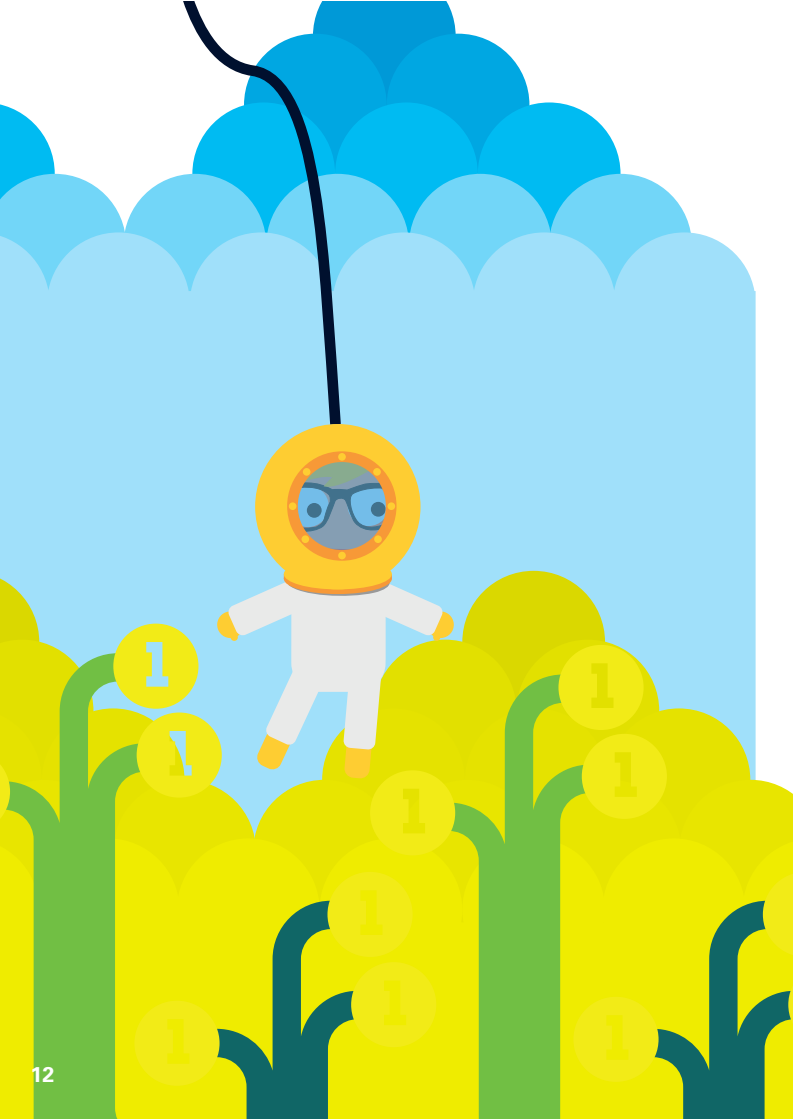
The main benefits of programming apps natively include better integration with the platform's features and often better performance. Typical drawbacks are the effort and complexity of supporting several native platforms (or limiting your app to one platform).

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

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

Platform	Language(s)	Remarks
<b>Android</b>	Java, C, C++	Open Source OS (based on Linux) developer.android.com
<b>bada</b>	C, C++	Samsung's mobile platform running on Linux or RealTime OS developer.bada.com
<b>BlackBerry</b>	Java	J2ME compatible, extensions enable tighter integration na.blackberry.com/eng/developers
<b>iOS</b>	Objective-C, C	Requires Apple Developer Account developer.apple.com/iphone
<b>MeeGo</b>	Qt, Web Apps, C++, others	Intel and Nokia guided open source OS (based on Linux) meego.com/developers
<b>Symbian</b>	C, C++, Java, Qt, WebApps, others	Open source OS built from the ground up for mobile devices www.forum.nokia.com/symbian
<b>webOS</b>	HTML, CSS, JavaScript, C	Supports widget style programming, (based on Linux) developer.palm.com
<b>Windows Mobile</b>	C#, C	.NET CF or Windows Mobile API, most devices ship with J2ME compatible JVM developer.windowsmobile.com
<b>Windows Phone</b>	C#, VB.NET	Silverlight, XNA frameworks create.msdn.com





## J2ME / Java ME

Around 80% of all mobile handsets worldwide support the mobile Java standard (J2ME/Java ME), making it 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's the source of many fragmentation problems.

On many feature phones, Java ME is the only way to realize client side applications. With the increasing penetration of smartphones, J2ME has lost some of its importance, at least in the US and Europe. However, for many emerging markets it remains the main option to target the mass market.

## Flash and Alternative Flash-compatible Platforms

Historically, Flash Lite was the mobile edition of Flash, an older version of Adobe's web Flash product with ActionScript 2.0 support. Adobe is phasing out Flash Lite for mobile and simply using the full version of Flash. Flash is favored by many designers, since they know the tools already and it can be used to create engaging, powerful user interfaces (UIs). It's relatively easy to code thanks to the ActionScript language, which is very similar to JavaScript. The drawbacks of Flash on mobile devices used to be poor performance, suboptimal integration into host devices and small market share in comparison to Java ME. However, all these things are improving: There are millions of feature phones supporting Flash today and many smartphones and tablets can support some Flash content including MeeGo, Symbian, iOS, Android and BlackBerry devices.

# BREW

The Binary Runtime Environment for Wireless (BREW) is a programming environment promoted by Qualcomm<sup>1</sup>.

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

## Widgets

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

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

1) [www.brewmp.com](http://www.brewmp.com)

2) [developer.brewmp.com](http://developer.brewmp.com)

The following table provides an overview of popular widget frameworks:

Environment	Language(s)	Remarks
<b>Web Runtime Widgets on Symbian</b>	XML, HTML, CSS, JavaScript	Standard web technology based widgets, with a proprietary packaging standard. JavaScript APIs offer high degree of access to platform features. <a href="http://www.forum.nokia.com/Develop/Web">www.forum.nokia.com/Develop/Web</a>
<b>Skylight</b>	HTML, CSS, JavaScript	Open Source widget platform for featurephones and smartphones, still under development. <a href="http://www.skylightmobile.org">www.skylightmobile.org</a>
<b>WAC / JIL</b>	XML, HTML, JavaScript, CSS	A joint initiative by Vodafone, China Mobile and other companies are pushing the W3C widget standard <a href="http://www.jil.org">www.jil.org</a>
<b>Samsung</b>	XML, HTML, CSS, JavaScript	<a href="http://innovator.samsungmobile.com">innovator.samsungmobile.com</a>
<b>PhoneGap</b>	HTML, CSS, JavaScript	Cross platform widget platform <a href="http://www.phonegap.com">www.phonegap.com</a>
<b>Sony Ericsson WebSDK</b>	HTML, CSS, JavaScript	Based on PhoneGap <a href="http://developer.sonyericsson.com">developer.sonyericsson.com</a>
<b>BlackBerry</b>	HTML, CSS, JavaScript	<a href="http://na.blackberry.com/eng/developers">na.blackberry.com/eng/developers</a>



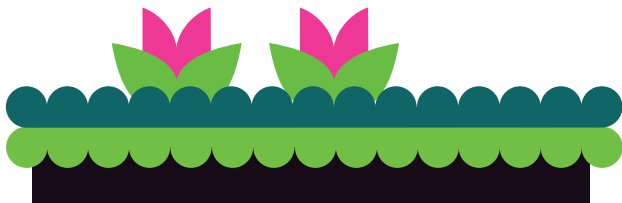
## Websites

The browsing of webpages is supported by most phones, so in principle this should be the environment of choice to get the widest possible reach (after SMS texting). However, the sheer number of browsers and their varying feature sets can make this approach challenging. Some browsers are very powerful and support CSS as well as JavaScript, others are less sophisticated and support XHTML only. Thankfully the old WAP standard with its WML pages doesn't play any significant role nowadays. The main drawback of web pages is that they are available when the device is online only and their access to device features is extremely limited.

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

## SMS Text Messaging

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



# Mobile Developer's Guide

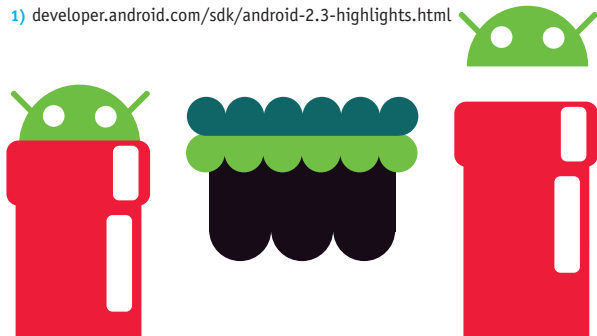
## Programming Android Apps

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

Android is an operating system and an application framework with complete tooling support and a variety of preinstalled applications. In late 2010, Google announced that every day 300,000 Android devices are shipped to end users. Since the platform is supported by many hardware manufacturers, it is the fastest growing smartphone operating system.

Additionally, Android is used (or planned to be used) for tablets, media players, setup boxes, desktop phones and car entertainment systems. It is also growing very fast when it comes to the platform features: Each new release usually includes many new features. For example, Android 2.3 (so-called "Gingerbread") introduced NFC and VOIP communication, better game development and a lot more<sup>1</sup>, Android 3.0 ("Honeycomb") has been especially designed for deployment on tablets and other devices with larger screens.

1) [developer.android.com/sdk/android-2.3-highlights.html](http://developer.android.com/sdk/android-2.3-highlights.html)



# Prerequisites

The main programming language for Android is Java. But beware, only a subset of the Java libraries is supported and there are lots of platform specific APIs. You find answers to your What and Why questions in the Dev Guide<sup>1</sup> and to your How questions in the reference documentation<sup>2</sup>.

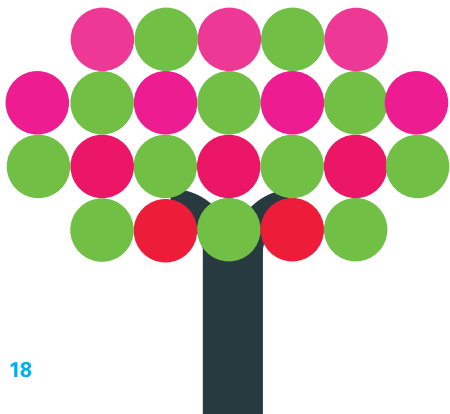
To get started, you need the Android SDK<sup>3</sup>, which is available for Windows, Mac OS X, and Linux. It contains tools to build, test, debug and analyse applications. You will probably also want a good Java IDE. Eclipse or IntelliJ seem a good choice as there is good support for development, deployment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

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

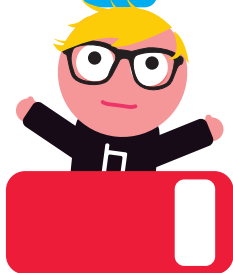
1) [developer.android.com/guide](http://developer.android.com/guide)

2) [developer.android.com/reference](http://developer.android.com/reference)

3) [developer.android.com/sdk](http://developer.android.com/sdk)



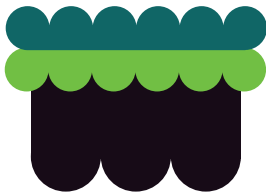
IUP  
IUP  
IUP



## 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 broadcast 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. An application may consist of several of these components, for instance an activity for the UI and a service for long running tasks.

Communication between the components is done by intents. An intent bundles data like the user's location or an URL with an action. These intents trigger behaviours in the platform. For instance, the intent of showing a web page will open the browser activity. The nice thing about this building-block philosophy is that functionality can be replaced by other applications and the Android system will use the preferred application for a specific intent.



For example the intent of sharing a web page triggered by a news reader app can open an email client or a text messaging app depending on the user's preference and on the applications installed. Any application that declares the sharing intent as their interface can be used.

To aid development, you have a lot of tools from the SDK at your disposal, the most important ones are:

- **android:** Create an initial project or manage virtual devices and versions of the SDK.
- **adb:** Query devices, connect and interact with them (and virtual devices) by moving files, installing apps etc.
- **emulator:** Start it with a virtual device and it will emulate the defined features. It takes a while to start so do it once and not on every build.
- **ddms:** Look inside your device or emulator, watch log messages, and control emulator features like network latency and GPS position. View memory consumption or simply kill processes. If this tool is running, you can also connect the Eclipse debugger to a process running in the emulator.

These tools and more – e.g. to analyze method trace logs, inspect layouts, or to test apps with random events or backup functionality – can be found in the tools directory of the SDK.

The user interface of an application is separated from the code in Android-specific xml layout files. Different layouts can be created for different screen sizes, country locales and device

features without touching Java code. To this end, localized strings and images are organized in separate resource folders and the IDE plugins 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<sup>1</sup>. Therefore, testing on a physical device is paramount.

### Automated Testing

To automate testing, the Android SDK comes with some capable and useful testing and instrumentation<sup>2</sup> tools. Tests can be written using the standard JUnit format with some Android mock objects that are contained in the SDK.

The Instrumentation classes can monitor the UI, send system events like key presses, et cetera. You can test for the status of your application after these events occur. The automated tests can be run on physical devices or in a virtual device. Open-source testing frameworks such as Robotium<sup>3</sup> can complement your other automated tests; it can even be used to test binary apk files, if the source is not available. A maven plugin<sup>4</sup> and a helper for the continuous integration server Hudson may also assist your testing<sup>5</sup>.

1) *For an overview see e.g.* [www.androidfragmentation.com](http://www.androidfragmentation.com)

2) [developer.android.com/guide/topics/testing/testing\\_android.html](http://developer.android.com/guide/topics/testing/testing_android.html)

3) [code.google.com/p/robotium](http://code.google.com/p/robotium)

4) [code.google.com/p/maven-android-plugin/](http://code.google.com/p/maven-android-plugin/)

5) [wiki.hudson-ci.org/display/HUDSON/Android+Emulator+Plugin](http://wiki.hudson-ci.org/display/HUDSON/Android+Emulator+Plugin) and [hudson-labs.org/content/getting-started-building-android-apps-hudson](http://hudson-labs.org/content/getting-started-building-android-apps-hudson)



## 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 a good place to reach both customers and developers of the Android platform, to browse for new exciting apps, and to sell your own apps. It is used by other app portals as a source for app meta data. Furthermore, it is part of the Google Backup Service<sup>1</sup>, the Cloud to Device Messaging (C2DM)<sup>2</sup> and the License Verification Library (LVL)<sup>3</sup>. To upload your application to the Android Market, start at [market.android.com/publish](http://market.android.com/publish).

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

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

- 1) [code.google.com/android/backup/index.html](http://code.google.com/android/backup/index.html)
- 2) [code.google.com/android/c2dm/index.html](http://code.google.com/android/c2dm/index.html)
- 3) [developer.android.com/guide/publishing/licensing.html](http://developer.android.com/guide/publishing/licensing.html)
- 4) [www.wipconnector.com/index.php/appstores/tag/android](http://www.wipconnector.com/index.php/appstores/tag/android)

# Mobile Developer's Guide

## Programming bada Apps

bada is Samsung's own mobile platform introduced in late 2009 and released to endusers in June 2010. bada can run either on a Linux kernel for high-end devices or on real-time OS kernels for low-end devices. Samsung promotes bada handsets as the "smartphones for everybody" aiming to replace feature-phones with bada handsets.

The UI is based on TouchWiz, already known from Samsung's Android handhelds. Over 5 million bada phones were sold between June and December 2010. For the future, Samsung plans to sell about 20 million bada-phones per year. Currently there are 5 bada-based devices available with the Wave S8500 being the flagship.

### Prerequisites

In order to start developing for bada you need to register at [developer.bada.com](http://developer.bada.com) and download the latest bada SDK. The SDK includes the bada IDE (based on eclipse CDT), a simulator and a GNU toolchain to provide developers with a popular and known developing system. Also the IDE offers a GUI Tool, for designing the UI of your app.

Furthermore there is a bada Widget SDK included, which supports BONDI JavaScript libraries.



# Implementation

Inside the IDE you will find a lot of code examples, which you can copy with one click into your own workspace. These examples provide a really good entry point for beginners on the platform. Native bada apps are developed in C++, but Samsung has implemented some restrictions. For example, the API does not use exceptions. Instead, return values and a combination of macros are used for error handling. Also RTTI is turned off, so for example `dynamic_cast` does not work on bada.

When creating bada apps, you need to understand memory management basics, because the API often leaves this to the developer. The API only uses some parts of STL, but Samsung claims that you could use STL in your own code.

Note that bada's STL version is missing some parts, so you might need to use STLPort for full STL support. Porting other modern C++ Libraries like boost to work on bada is possible, but the lack of RTTI and exceptions can make it hard.

The bada API is wrapped in a number of namespaces. The API offers UI Control and Container classes, but there are no UI Layout classes, so all your UI must be positioned by hand or within the code. You need to provide an UI layout for the landscape and/or the portrait perspective. The API also provides most standard classes for XML, SQL or Network and resembles a pretty complete framework. Also make use of the callbacks for important phone events in your application class, like low battery level or incoming calls.



If you want to write games for bada, the SDK supports OpenGL ES 1.1 and 2.0. Also the SDK wraps parts of OpenGL for use in its own classes, so you can also easily port existing OpenGL code to bada. The bada API has some easy patterns, like that you have to delete any pointer returned by a method ending in 'N'. You should also make sure that each of your own new variables has its delete:

```
"MyType* var = new MyType(); // call delete
MyType* array 0 new MyType[10]; // call delete[]
MyType type, stackarray[];
// variable on stack will be destroyed by
//scope, no delete"
```

The central resource for bada developers is [developer.bada.com](http://developer.bada.com). The biggest independent bada website and forum is currently [BadaDev.com](http://BadaDev.com) where you will find great tutorials about coding for bada. There is an IRC channel #bada at [irc.freenode.net](http://irc.freenode.net), and of course there are groups for bada on most social networks.

## Testing

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

Otherwise you can never be sure how the app performs and it also might turn out that the code that worked perfectly on the simulator will not do so on the handset.

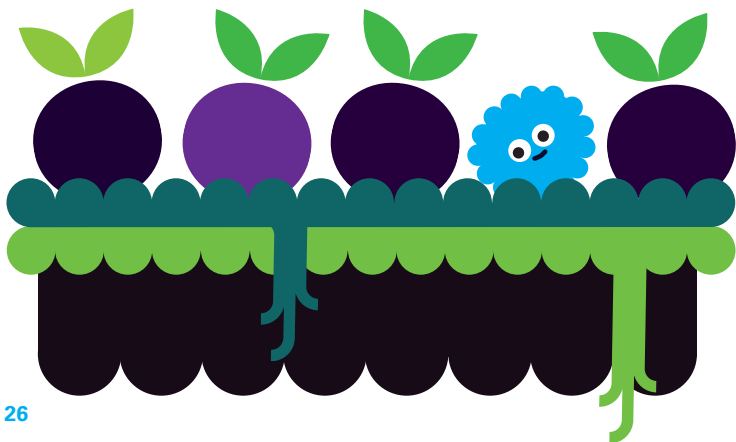
Samsung provides the bada Remote Test Lab (RTL) which is available for all registered developers and can be installed as Eclipse-plugin.

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

## Distribution

The bada ecosystem is a closed platform with the Samsung App Store as its only distribution channel. Similar to Apples AppStore, there are quite strict acceptance rules applied.

Check them out in the „Samsung Apps Publisher Guide“, downloadable after you registered at the Samsung Apps Seller Office. Once your app has made it to the App Store, you will get 70% of the revenue. Since January 2011, Samsung also allows you to include third party ad network contents in your bada application.



# Mobile Developer's Guide

## Programming Native BlackBerry Apps

The BlackBerry platform is developed by the Canadian company Research In Motion (RIM)<sup>1</sup> and was launched in 1999. The popularity of BlackBerry devices arose because they were traditionally equipped with a full keyboard for comfortable text input (which spawned a condition named BlackBerry Thumb<sup>2</sup>) and their long battery life. Add PDA applications such as address book, mail, calendar, tasks and memopad to these features and you will understand why the platform is very popular among business users. Over the last couple of years, BlackBerry increased its popularity among mainstream users, especially in the US where it's the second most popular smartphone platform behind iOS<sup>3</sup>.

### Prerequisites

Depending on the type and nature of your planned project you need to choose an approach for developing a BlackBerry application. For mid-sized to large applications native Java development is the first choice. Small apps can also be developed with the BlackBerry Widget SDK.

The next generation BlackBerry OS is based on QNX Neutrino Realtime OS (RTOS) which will support Adobe AIR Flash programming, Java and most likely (but not yet announced) widget development. However, this chapter focuses on Java development, for more information on widget and Flash programming, please see the respective chapters in this guide.

1) [www.rim.com](http://www.rim.com)

2) [en.wikipedia.org/wiki/Blackberry\\_thumb](http://en.wikipedia.org/wiki/Blackberry_thumb)

3) [gs.statcounter.com](http://gs.statcounter.com)

## Java SDK

As for all Java-driven applications and development, you need the Java SDK<sup>1</sup> (not the Java Runtime Edition).

## IDE

For native Java development, you first need to decide which IDE to use. The first option is one of the BlackBerry IDEs. BlackBerry devices run on different OS versions, ranging from version 4.0 to 6.0. For each of these versions, there is a BlackBerry Java Development Environment (JDE) available<sup>2</sup>. These JDEs are complete environments enabling you to write, compile, package and sign your applications. Device simulators are included as well.

To compensate for shortcomings in these JDEs (such as the lack of syntax checking and code completion), the currently preferred option is the BlackBerry Plugin for the Eclipse IDE<sup>3</sup>. To use this plugin, you obviously need to install the Eclipse IDE first.

## Desktop Manager

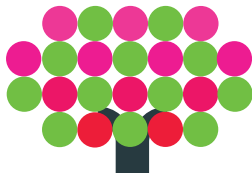
You should download and install the BlackBerry Desktop Manager<sup>4</sup> also. It enables you to deploy your app package on a device for testing. For faster deployment, you might also use the tool javaloader that comes with the JDE.

1) [www.oracle.com/technetwork/java](http://www.oracle.com/technetwork/java)

2) [us.blackberry.com/developers/javaappdev/javadevdev.jsp](http://us.blackberry.com/developers/javaappdev/javadevdev.jsp)

3) [us.blackberry.com/developers/javaappdev/javaplugin.jsp](http://us.blackberry.com/developers/javaappdev/javaplugin.jsp)

4) [us.blackberry.com/apps-software/desktop/](http://us.blackberry.com/apps-software/desktop/)



# Coding Your Application

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

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

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

## Services

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

The push service<sup>3</sup> is useful mainly in mail, messaging or news applications. Its benefit is that the device waits for the server to push updates to it, instead of the device continuously polling the server to find out if updates are available and then pulling the updates from the server. This reduces network traffic, battery usage and – for users on metered data plans – costs.

1) [www.blackberry.com/developers/docs/6.0.0api/index.html](http://www.blackberry.com/developers/docs/6.0.0api/index.html)

2) [www.j2mepolish.org](http://www.j2mepolish.org)

3) [us.blackberry.com/developers/platform/pushapi.jsp](http://us.blackberry.com/developers/platform/pushapi.jsp)

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

## Testing

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

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

## Porting

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

- Some classes and functionalities are available on specific OS versions only. For example the FilePicker that is used to choose a file is available on OS 5.0 onwards only.

- You need to handle different screen resolutions and orientation modes (landscape and portrait).
- You need to handle touch and non-touch devices. In addition, the Storm devices use a touchscreen that is physically clickable, so there is a distinction between a touch and a click on these devices. BlackBerry's more recent touch devices don't use this technology anymore.

Porting to other Java platforms such as J2ME and Android is complicated as it's not possible to port the BlackBerry UI. Self-written classes for server communication and storage among others might be reused on J2ME and Android in certain cases. In general, cross-platform portability strongly depends on how frequently your app uses native BlackBerry components. For example it is not possible to reuse BlackBerry push services classes on other platforms.

## Signing

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

## Distribution

BlackBerry's own distribution channel is called App World<sup>2</sup> where you can publish your apps. For paid applications, you'll get a 70% revenue share. In addition, GetJar<sup>3</sup> is a well known independent website that publishes BlackBerry apps.

1) [us.blackberry.com/developers/javaappdev/codekeys.jsp](http://us.blackberry.com/developers/javaappdev/codekeys.jsp)

2) [appworld.blackberry.com](http://appworld.blackberry.com)

3) [www.getjar.com](http://www.getjar.com)

# Mobile Developer's Guide

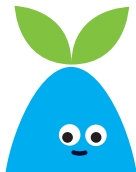
## Programming Flash Apps

Developing Flash applications and Flash animations for use within mobile websites, widgets and applications for mobile devices doesn't differ significantly from developing browser-based Flash applications for desktop computers. You must, however, be aware of the requirements of the target device.

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

The Adobe Flash players installed within the firmware of mobile devices are not upgradable in the field. This means that you must author Flash content to match the Flash player in the device. Flash-compatible engines and players from alternative Flash-compatible SDK and tool vendors vary in the functionality provided.

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 GPUs with OpenGL ES graphics acceleration. Until then, you have to find a way to deal with this fragmentation: Be sure to save your Flash animation in a format that is compatible with your target device's software.



Many feature phones have support for Flash Lite. The Flash Lite players provided are compatible with Flash 3, 6 or 8 depending on when the device was manufactured. These are perfect for simple Flash games (e.g., puzzle and card games). Some smartphones and tablets also have a Flash player pre-installed: Full Flash support has been announced for Android-based devices and RIM's BlackBerry devices. BlackBerry's tablet OS fully supports Flash and native UI elements and events allow you to integrate visually nicely. For the iPhone, Adobe has released a packager that enables Adobe AIR applications to run on iOS devices.

You should be aware of the restrictions of each platform and of the fact that there are a number of alternative Flash-compatible products that can enable Flash to operate on a number of these devices – often with better performance.

## Prerequisites

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 (CS) software. CS 5 supports ActionScript 3 and Flash 10.1 offering the full 3D and 2D feature set on some smartphones and tablets. If you want to utilize features such as 3D and ActionScript 3 compatibility, using the CS5 package and tools is the way to go.

However, as stated earlier, one potential drawback is poor performance: Large binary files may run slowly on less powerful devices resulting in a poor user experience. Adobe CS 3, 4 and 5 can be used to author Flash content that runs on Alternative Flash-compatible SDKs, engines and players.

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

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

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

There are also open source versions including Gnash<sup>1</sup> and GameSWF<sup>2</sup> that are designed for desktop systems. Many of the alternative Flash platforms run outside the browser environment, working directly with a device's native APIs.

## Tips & Tricks

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

1) [www.gnashdev.org](http://www.gnashdev.org)

2) [tulrich.com/geekstuff/gameswf.html](http://tulrich.com/geekstuff/gameswf.html)

In general, you should think carefully about whether you need ActionScript 3: It's a completely different scripting language to ActionScript 2 and requires a lot more development know-how and experience to implement efficiently.

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

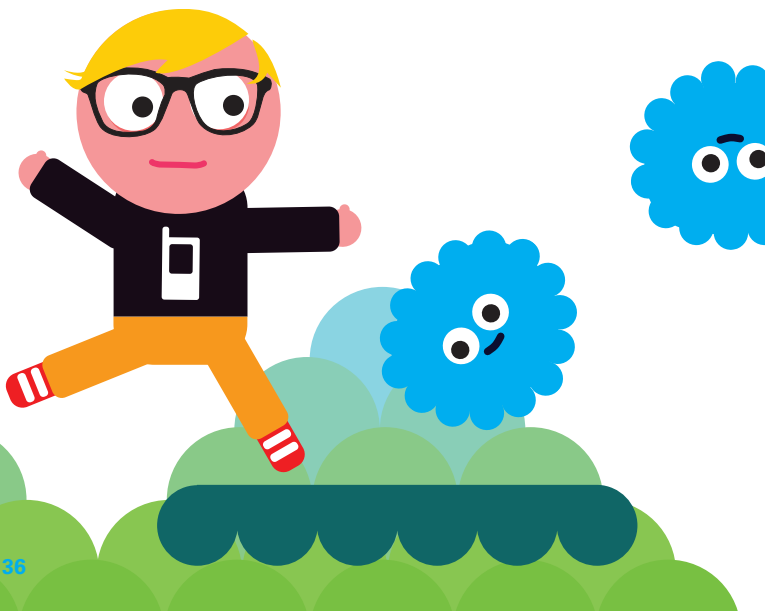
- Avoid sliding a Flash object across the screen, unless you know it performs well. Redrawing every pixel multiple times in a frame without the support of a GPU is a performance killer. Select a SDK toolkit that minimizes CPU utilization to preserve battery life. If the Flash animation is not changing,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

## Testing

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

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

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



## Packaging and distribution

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

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

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

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

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

While some developers will do this manually, Nokia does provide an online packaging service that does the heavy lifting for you<sup>2</sup>. Generally, when the packaging seems complex, it can often be simplified by using the platform's widget option to package and deploy the content.

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

1) [bit.ly/aqEmvv](http://bit.ly/aqEmvv)

2) [esitv008song.itlase.com/sispack](http://esitv008song.itlase.com/sispack)

# Mobile Developer's Guide

## Programming iPhone Apps

The iPhone, along with the iPod touch and iPad, is a highly interesting and very popular development platform for many reasons, a commonly named one being the App Store. When it was introduced in July 2008, the App Store took off like no other marketplace did before. Now there are far more than 300,000 applications in the App Store, and the number is growing daily. This reflects the success of the concept but it also means that it is getting more and more difficult to stand out in this mass of applications.

Users have downloaded more than 8 billion iOS apps until end of 2010, and with device sales reaching new all-time highs just about each quarter, there is no sign of the current volume of about half a billion downloads per month slowing down. Over 120 million sold devices paired with the users' willingness to try apps and pay for content makes the App Store the most economically interesting target for mobile app development.

The iOS SDK offers high-level APIs for a wide range of tasks which helps to cut down on development time 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, skills acquired in iPhone development can be used in iPad development. A single binary can even contain different versions for both platforms with large parts of the code being shared. Since iOS version 4.2 was released in November 2010, all currently sold iOS devices use a common firmware version, which makes development of universal apps for multiple device classes even more feasible.

# Prerequisites

## Apple's iOS SDK

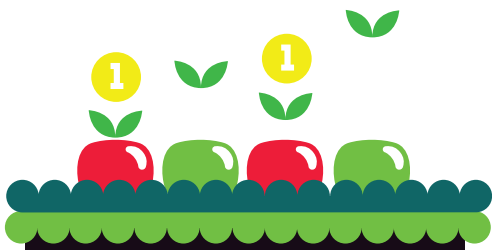
In order to develop iPhone (and iPod Touch and iPad) apps, you will need the iOS SDK, which can be downloaded at [developer.apple.com/iphone](http://developer.apple.com/iphone). This requires a membership, which is available for free. If you plan to test your apps on your device or distribute your apps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

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

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

The iOS 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](http://developer.apple.com/iphone/library/navigation).



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

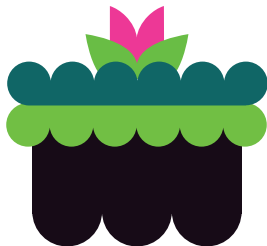
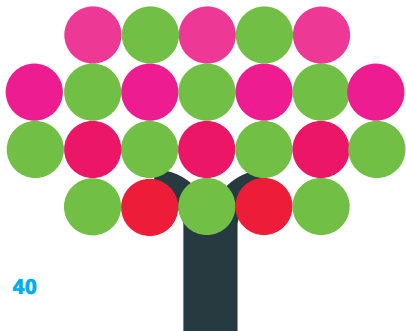
- **Cocoa Touch**, which consists of the UI libraries, various input methods such as multi-touch and accelerometer.
- **Media frameworks**, such as OpenAL, OpenGL ES, Quartz, Core Animation and various audio and video libraries.
- **Core Services**, such as networking, SQLite, threading and various other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

The list of available frameworks constantly grows with each major release of the iOS firmware, which usually happens once a year in June or July.

### Alternative 3rd party development environments

Since Apple relaxed their App Store distribution guidelines, development using other tools than Objective-C, Cocoa Touch and Xcode is officially permitted again and most commonly used in game development, for example using the Unreal Development Kit<sup>1</sup>, which Epic released for iOS to much fanfare in December 2010.

1) [www.udk.com](http://www.udk.com)



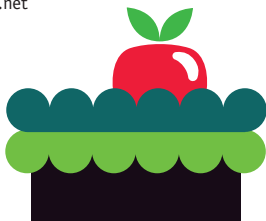
Using third party development environments and languages for iOS development offers a number of advantages and disadvantages compared to the official way of producing iOS apps. The major advantage being that it is easy to support multiple platforms from a single code base without having too much of a maintenance burden. However, as experience with desktop software has shown, cross-platform software development rarely produces outstanding quality and in most cases has to concentrate on the lowest common denominator, resulting in a product which doesn't feel like it really belongs on any of the platforms.

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

There are, however, third party development environments which focus solely on iOS development, such as MonoTouch<sup>1</sup>. The platform allows developers to build iOS apps using C# and .NET while taking advantage of iOS APIs, making it the alternative that comes closest to what the original SDK has to offer and still allowing code re-use, for example when creating similar Windows Phone 7 apps.

Some of those alternative IDEs might, however, carry additional fees in addition to Apple's yearly \$99 for access to the develop program and Apple's 30% cut of all sales. As the perceived quality of mobile apps comes in large part from a usable and beautiful user interface, cross-platform development using third party IDEs makes the most sense for games, which can share almost all their code between different platforms.

1) [www.monotouch.net](http://www.monotouch.net)



# Implementation

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

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

If you are used to other object-oriented languages such as C++ or Java, Objective-C's syntax might take some time getting used to, but is explained in detail at [developer.apple.com](http://developer.apple.com)<sup>1</sup>. What separates Objective-C most from these languages is its dynamic nature, lack of namespace support and the concept of message passing vs. method calls.

A great way to get you started is Apple's guide "Your First iPhone Application", which will explain various concepts and common tasks in an iPhone developer's workflow<sup>2</sup>. Also check out some of the sample code that Apple provides online<sup>3</sup> to find out more about various APIs available to you.

## Testing

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

1) [developer.apple.com/iphone/manage/overview/index.action](http://developer.apple.com/iphone/manage/overview/index.action)

2) [developer.apple.com/iphone/manage/distribution/distribution.action](http://developer.apple.com/iphone/manage/distribution/distribution.action)

3) [developer.apple.com/iphone/library/navigation/SampleCode.html](http://developer.apple.com/iphone/library/navigation/SampleCode.html)

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

Testing on real devices is also important because touch-based input is completely different from a pointer-driven UI model. You can distribute builds of your application to up to 100 testers through Ad-Hoc Provisioning, which you can set up in the Program Portal<sup>1</sup>. Each iPhone (and iPad/ iPod touch) has a unique identifier (UDID – universal device identifier), which is a string of 40 hex characters based on various hardware parts of the device.

If you choose to test using Ad-Hoc-Provisioning, simply follow Apple's detailed set-up instructions<sup>2</sup>. Every single step is vital to success, so make sure that you execute them all correctly.

With iOS 4.0, Apple has introduced the possibility for developers to deploy Over-The-Air (OTA) Ad-Hoc builds of their apps to beta testers. There are open source projects<sup>3</sup> to facilitate this new feature, as well as commercial services<sup>4</sup>.

Google Toolbox for Mac<sup>5</sup> runs the test cases using a shell script during the build phase, while GHUnit<sup>6</sup> runs the tests on the device (or in the simulator), allowing the developer to attach a debugger to investigate possible bugs. In version 2.2 of the SDK Apple included OCUnt; an example of how to create the unit tests is available online<sup>7</sup>.

- 1) [developer.apple.com/iphone/library/referencelibrary/GettingStarted/Learning\\_Objective-C\\_A\\_Primer/index.html#//apple\\_ref/doc/uid/TP40007594](http://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](http://developer.apple.com/iphone/library/documentation/iPhone/Conceptual/iPhone101/Articles/00_Introduction.html)
- 3) [github.com/therealkerni/hockey](http://github.com/therealkerni/hockey)
- 4) [www.testflightapp.com](http://www.testflightapp.com)
- 5) [code.google.com/p/google-toolbox-for-mac](http://code.google.com/p/google-toolbox-for-mac)
- 6) [github.com/gabriel/gh-unit](http://github.com/gabriel/gh-unit)
- 7) [www.mobileorchard.com/ocunit-integrated-unit-testing-in-xcode](http://www.mobileorchard.com/ocunit-integrated-unit-testing-in-xcode)

In version 4 of iOS Apple introduced a new tool, UIAutomation which aims to automate the testing of your application by scripting touch events. UIAutomation tests are written in JavaScript and a full reference is available in the iOS Reference Library<sup>1</sup>. Several other third party testing automation tools for iPhone applications are available as well, including FoneMonkey<sup>2</sup> and Squish<sup>3</sup>.

## Distribution

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

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

After Apple has approved your application, which usually shouldn't take more than 2 weeks, your app will be available to customers in the App Store. Due to several rejections in the past, the approval process receives more complaints than any other aspect of the iPhone ecosystem. A list of common rejection reasons can be found on [www.apprejections.com](http://www.apprejections.com). Recently, Apple has released their full App Store testing guidelines in order to give developers a better chance to estimate their app's success of being approved. Also, the restrictions were relaxed and apps which were previously rejected were approved after being re-submitted.

- 1) [developer.apple.com/library/ios/#documentation/DeveloperTools/Reference/UIAutomationRef/\\_index.html](http://developer.apple.com/library/ios/#documentation/DeveloperTools/Reference/UIAutomationRef/_index.html)
- 2) [www.gorillalogic.com/fonemonkey](http://www.gorillalogic.com/fonemonkey)
- 3) [www.froglogic.com/products](http://www.froglogic.com/products)

# Books

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

## Beginner books

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

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

## Intermediate books

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

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

## Professional books

If you already have some good knowledge of the iOS SDK, one of these books is sure to increase even your skills.

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

## Companion books

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

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

## Community

One of the most important aspects of iOS development is the community. A lot of iOS developers are very outspoken and open about what they do, and how they did certain things. This is even more visible ever since Twitter and Github gained momentum and became widely-known.

Search for iPhone, iPad or any other related search terms on Github.com and you'll find a lot of source code, frameworks, tutorials, code snippets and complete applications — most of them with very liberal licenses which even allow commercial usage.

Practically all of the most important and most experienced iOS developers use Twitter to share their thoughts about the platform with others. There are many comprehensible lists of iOS developers out there, a notable and well-curated one being Robert Scoble's list<sup>1</sup>. Following a list like that means staying up to date on current issues and generally interesting things about iOS development.

1) [www.twitter.com/Scobleizer/iphone-and-ipad](http://www.twitter.com/Scobleizer/iphone-and-ipad)

What makes the community especially interesting is that many iOS developers pride themselves on taking an exceptional interest in usability, great user experience and beautiful user interfaces. You can usually find out about the most interesting trends on blog aggregators such as [CocoaHub.de](http://CocoaHub.de) and [PlanetCocoa.org](http://PlanetCocoa.org)



# Mobile Developer's Guide

## Programming J2ME / Java ME Apps

Here's the naked truth: compared to native mobile platform APIs, such as those on Android, iPhone and Symbian, J2ME offers a less powerful API, often runs on less powerful hardware and tends to generate less money for the developer.

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

With over 80% of phones worldwide supporting it, J2ME (which is officially called Java Mobile Edition or Java ME) is miles ahead of the competition in this regard. If your business model relies on access to as many potential customers as possible, or on providing extra value to existing customers via a mobile application, then J2ME is a great choice.

However, if your business model relies on direct application sales, you might want to consider targeting a different platform (such as Android, Blackberry, iPhone or Symbian).

That being said, it should be noted that J2ME's capabilities are constantly improving thanks to the Java Community Process<sup>1</sup> that standardizes new APIs. In addition, J2ME-compatible hardware is becoming more powerful and less expensive all the time. Overall the platform is evolving and changing for the better, though admittedly at a slower pace than the competition.

1) [www.jcp.org](http://www.jcp.org)



# Prerequisites

To develop a J2ME application, you'll need a couple of things:

- The Java SDK<sup>1</sup> (not the Java Runtime Environment) and an IDE of your choice, such as Eclipse Pulsar for Mobile Developers<sup>2</sup>, NetBeans<sup>3</sup> with its Java ME plug-in or IntelliJ<sup>4</sup>.
- An Emulator, such as the Wireless Toolkit<sup>5</sup>, the Micro Emulator<sup>6</sup> or a vendor specific SDK or emulator.
- Depending on your setup you may need an obfuscator like ProGuard<sup>7</sup>. If you build applications professionally you will probably want to use a build tool like Maven<sup>8</sup> or Ant<sup>9</sup> also.
- You may want to check out J2ME Polish, the open source framework for building your application for various devices<sup>10</sup>.

1) [www.oracle.com/technetwork/java/javame/downloads/index.html](http://www.oracle.com/technetwork/java/javame/downloads/index.html)

2) [www.eclipse.org](http://www.eclipse.org)

3) [www.netbeans.org](http://www.netbeans.org)

4) [www.jetbrains.com](http://www.jetbrains.com)

5) [www.oracle.com/technetwork/java/download-135801.html](http://www.oracle.com/technetwork/java/download-135801.html)

6) [www.microemu.org](http://www.microemu.org)

7) [www.proguard.sourceforge.net](http://www.proguard.sourceforge.net)

8) [maven.apache.org](http://maven.apache.org)

9) [ant.apache.org](http://ant.apache.org)

10) [www.j2mepolish.org](http://www.j2mepolish.org)



A complete installation and setup guide is beyond the scope of this guide, please refer to the respective tools' documentation. Beginners often like to use NetBeans, with the Java ME plug-in installed. Also download and read the JavaDocs for the most important technologies and APIs: You can download most JavaDocs from [www.jcp.org](http://www.jcp.org). There are a couple of useful vendor specific APIs that should be tracked down manually from the vendor's pages (such as the Nokia UI API and Samsung APIs).

## Implementation

The J2ME platform is fairly straight-forward: it's formed by the Connected Limited Device Configuration (CLDC) and the Mobile Internet Device Profile (MIDP), which are both quite easy to understand.

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

- **Highlevel LCDUI components:** you use standardized UI components, such as Form and List.
- **Lowlevel LCDUI:** you manually control every pixel of your UI using low-level graphics functions.
- **SVG:** you draw the UI in scalable vector graphics then use the APIs of JSR 226 or JSR 287<sup>1</sup>.

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

1) [www.jcp.org/en/jsr/detail?id=287](http://www.jcp.org/en/jsr/detail?id=287)

There are also tools that can help you with the UI development. All of them use low-level graphics to create better looking and more powerful UIs than are possible with the standard high-level LCDUI components.

- **J2ME Polish**<sup>1</sup>: compatible with the highlevel LCDUI framework. This tool separates the design in CSS and you can use HTML for the user interface.
- **LWUIT**<sup>2</sup>: a Swing inspired UI framework.
- **Mewt**<sup>3</sup>: Uses XML to define the UI.
- **TWUIK**<sup>4</sup>: A powerful “Rich Media Engine”.

There is a rich open source scene in the J2ME sector. Interesting projects can be found via the blog on [opensource.ngphone.com](http://opensource.ngphone.com). You will also find interesting projects on the Mobile and Embedded page of [java.net](http://java.net)<sup>5</sup>, for example the Bluetooth project Marge<sup>6</sup>.

## Testing

Because of the fragmentation in the various implementations of Java ME, testing your applications is vital. Test as early and as often as you can on a mix of devices. Some emulators are quite good (personal favorites are Black-Berry and Symbian) but there are some things you have to test on devices. Thankfully vendors like Nokia and Samsung provide subsidized or even free remote access to selected devices<sup>7</sup>.

1) [www.j2mepolish.org](http://www.j2mepolish.org)

2) [lwuit.dev.java.net](http://lwuit.dev.java.net)

3) [www.mewt.sourceforge.net](http://www.mewt.sourceforge.net)

4) [www.tricastmedia.com/index.php?s=twuik](http://www.tricastmedia.com/index.php?s=twuik)

5) [mobileandembedded.dev.java.net](http://mobileandembedded.dev.java.net)

6) [marge.dev.java.net](http://marge.dev.java.net)

7) [www.forum.nokia.com/rda](http://www.forum.nokia.com/rda)

*and* [innovator.samsungmobile.com/bbs/lab/view.do?platformId=2](http://innovator.samsungmobile.com/bbs/lab/view.do?platformId=2)

## Automated Testing

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

## Porting

One of the strengths of the mobile Java environment is that it's backed by a standard, so it can be implemented by competing vendors. The downside is that the standard has to be interpreted, and this interpretation process can cause differences in individual implementations. This results in all kinds of bugs and non-standard behaviours. In the following 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 adapt your application to each platform's UI. At the same time your development costs will skyrocket. We advise the use of another strategy first, until your application idea has proved itself to be successful.

1) [www.j2meunit.sourceforge.net](http://www.j2meunit.sourceforge.net)

2) [www.momeunit.sourceforge.net](http://www.momeunit.sourceforge.net)

3) [snapshot.pyx4me.com/pyx4me-cldcunit](http://snapshot.pyx4me.com/pyx4me-cldcunit)

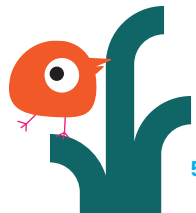
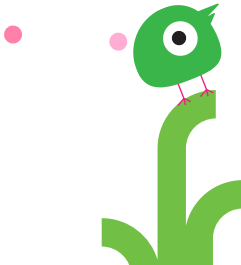
4) [www.code.google.com/p/jinjector](http://www.code.google.com/p/jinjector)

## Lowest Common Denominator

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

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

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



## Porting Frameworks

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

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

In the J2ME world there are various frameworks to choose from:

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

For more information about cross-platform development and available toolsets, please see the respective chapter.

Good frameworks enable 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 porting framework doesn't hide device fragmentation, but makes the fragmentation more manageable.

1) [www.mobile-distillery.com](http://www.mobile-distillery.com)

2) [www.metismo.com](http://www.metismo.com)

3) [www.enough.de](http://www.enough.de)

4) [www.mosync.com](http://www.mosync.com)

# Signing

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

On one phone the user might be asked once to enable this functionality, on another they'll be asked every time the feature is used and on a third device they 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 feature they desire. Applications signed with a carrier certificate are on a similar level.

Applications signed by JavaVerified<sup>1</sup>, Verisign<sup>2</sup> or Thawte<sup>3</sup> are on the lowest security level. To make matters worse, not every phone carries all the necessary root certificates. And some well known device vendors have even stripped away all root certificates in the past. The result is something of 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. However, in some cases an app store may offer to undertake the signing for you, as Ovi store does.

Another option is to consider using a testing and certification service provider and leaving the complexity to them. Intertek<sup>4</sup> is probably the largest such supplier.

1) [www.javaverified.com](http://www.javaverified.com)

2) [www.verisign.com](http://www.verisign.com)

3) [www.thawte.com](http://www.thawte.com)

4) [www.intertek.com/wireless-mobile](http://www.intertek.com/wireless-mobile)

# Distribution

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

- Handmark<sup>1</sup> and Mobile Rated<sup>2</sup> provide carrier and vendor independent application stores.
- GetJar is one of the oldest distributors for free mobile applications<sup>3</sup> – not only for Java applications.
- LG distributes apps on [www.lgapplication.com](http://www.lgapplication.com)
- Ovi store<sup>4</sup> targets Nokia users worldwide and provides a revenue share to the developer at 70% from credit card billing and 60% from operator billing
- Carriers are in the game also, such as Orange<sup>5</sup> or O2<sup>6</sup>. Basically almost everyone has announced an app store.

1) [store.handmark.com](http://store.handmark.com)

2) [www.mobilerated.com](http://www.mobilerated.com)

3) [www.getjar.com](http://www.getjar.com)

4) [www.publish.ovi.com](http://www.publish.ovi.com)

5) [www.orangepartner.com/site/enuk/mobile/application\\_shop/p\\_application\\_shop.jsp](http://www.orangepartner.com/site/enuk/mobile/application_shop/p_application_shop.jsp)

6) [www.o2litmus.com](http://www.o2litmus.com)

An overview of the available app stores (not those selling J2ME apps alone) can be found in the appstore Wiki on wipwiki.com. Furthermore there are various vendors who provide solutions for provisioning of Java applications over a Bluetooth connection, including Waymedia<sup>1</sup> and Futurlink<sup>2</sup>.

1) [www.waymedia.it](http://www.waymedia.it)

2) [www.futurlink.com](http://www.futurlink.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 devices based on Symbian, Maemo and, in the near future, MeeGo, an open source platform initiated by Nokia and Intel. In fact, Qt can now be thought of as a platform in its own right — you will create a Qt application and deploy it to devices utilizing a number of different underlying operating systems.

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

Qt uses standard C++ but makes extensive use of a special pre-processor (called the Meta Object Compiler, or moc) to deal with many of the challenges faced in standard C++ development. As a consequence Qt is able to offer powerful features that are not burdened by the usual C++ housekeeping. For example, instead of callbacks, a paradigm of signals and slots is used to simplify

the communication between objects<sup>1</sup>; the output from one object is a “signal” that has a receiving “slot” function in the same or another object.

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

Many developers who have used Qt report that applications can be written with fewer lines of code and with greater in-built reliability when compared to coding from scratch in C++. 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 contributing to the content and quality of the Qt APIs. Should you wish to get involved the source code is made available over Gitorious<sup>2</sup>.

1) [doc.qt.nokia.com/4.7-snapshot/signalsandslots.html](http://doc.qt.nokia.com/4.7-snapshot/signalsandslots.html)

2) [qt.gitorious.org](http://qt.gitorious.org)



## Prerequisites

Setting up a development environment to create applications for Nokia's Symbian devices is 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 for Symbian and Linux computer for Maemo.

The Nokia Qt SDK installs everything you need to create, test, and debug applications for Symbian and Maemo from a single package. It's also future proofed for MeeGo apps development. All versions offer tools for compiling Symbian and Maemo apps, with Symbian apps being compiled in the Linux and Apple Mac versions using the Remote Compiler service.

## Creating your application

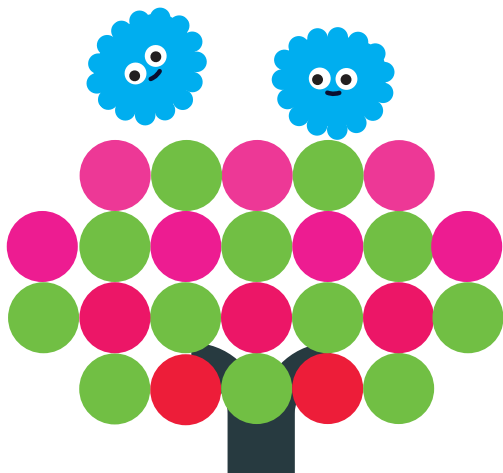
The Qt SDK is built around the Qt Creator development tool. Using Qt Creator you define most of your application visually and then add the specific program logic through a code editor that offers full code completion support and integrated help. One of the neat features of Qt is QML, a language for declarative UI definition. While QML generally simplifies UI development, its biggest advantage is that the tools within Qt Creator enable the UI to be defined by graphic designers who do not have to be aware of the technical programming aspects.

In the past, one of the challenges with cross platform applications for mobile has been accessing platform features: Anytime you want to find the device's location or read a contact record it's been necessary to revert back to the platform's native APIs. This is where the Qt Mobility APIs come in. The APIs provided by Qt Mobility offer a common interface to device data such as

contacts, location, messages, 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();
}
```

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



If you are already familiar with C++ development on the desktop, creating Qt applications for Symbian or 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. 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's also worth pointing out that, because Qt applications are compiled to the platform they will run on, they deliver very good performance and usability also. For most applications the levels of performance will be comparable to that previously achieved by hardcore native applications only.

## Testing

The Nokia Qt SDK includes a lightweight simulator enabling applications to be tested and debugged on the development PC. The simulator includes tools that enable device data, such as location or contacts records, to be defined so that the application's functionality can be fully tested. The simulator does not, however, eliminate the need for on device testing.

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

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

## Packaging

For a Qt application to run on a mobile device the Qt API framework has to be present. The Nokia N900 has the Qt APIs built in. In addition, Maemo and MeeGo devices provide an update mechanism that will install the necessary framework components should there be newer versions needed by the app.

For Symbian devices the situation is a little different. Symbian^3 devices have the APIs built in. However, Symbian doesn't include a built in mechanism to add the APIs to earlier devices. The solution is Smart Installer, which is included automatically in Symbian apps built with the Nokia Qt SDK. As an app is installed on a Symbian device, Smart Installer checks for the presence of the necessary Qt packages and, if they are not there, downloads and installs them. Using this mechanism, Qt apps can be easily targeted at almost all recent S60 and Symbian devices.



# Signing

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

The process is straightforward and described in full in the Distribute section of the Forum Nokia website<sup>2</sup>, but in summary:

- You sign up as an Ovi publisher
- You provide up to five device IMEIs and request a UID for your application
- The Ovi team provides you with a “developer certificate” and a UID for your app
- Now you create your app with the UID provided, sign your app during development to run it on one of the five devices elected and test it to ensure it complies with the Symbian Signed Test Criteria<sup>2</sup>
- Once tested, you submit an unsigned copy of the app to the Ovi publishing portal

1) [www.forum.nokia.com/Distribute/Packaging\\_and\\_signing.xhtml](http://www.forum.nokia.com/Distribute/Packaging_and_signing.xhtml)

2) [www.symbiansigned.com/app/page/overview/testcriteria](http://www.symbiansigned.com/app/page/overview/testcriteria)

## Distribution

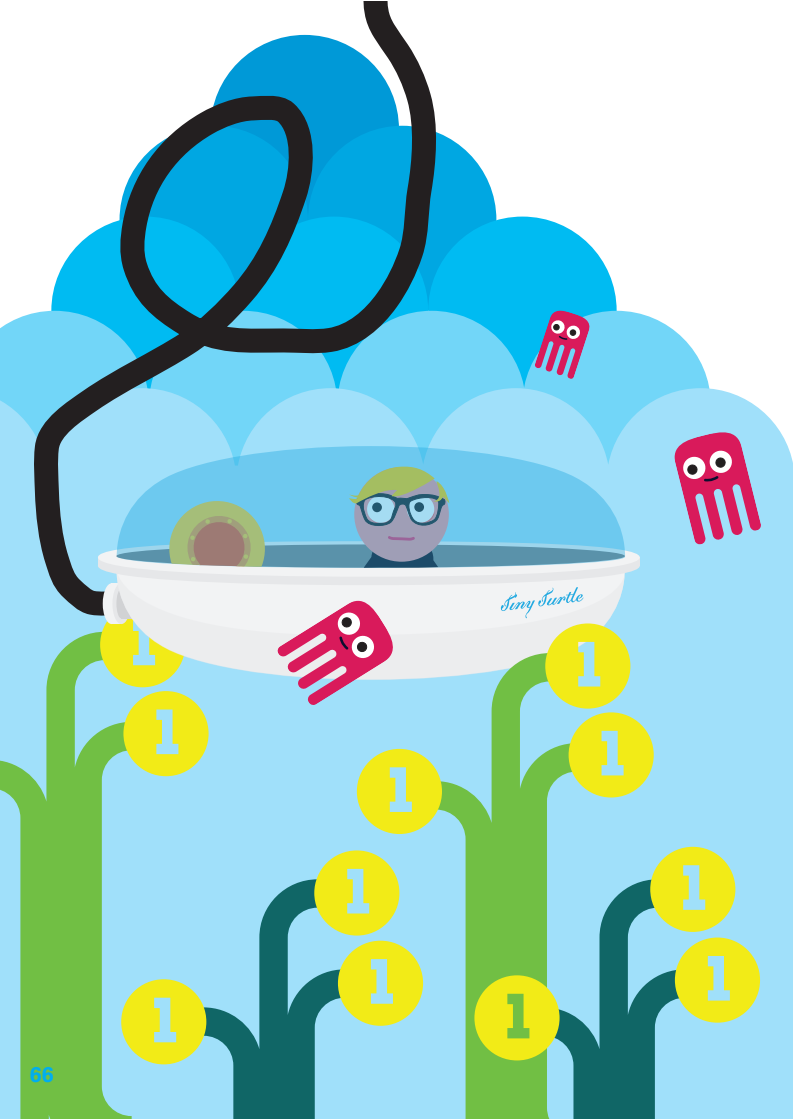
Ovi Store is the latest iteration of the Nokia app store solution, with a history stretching back to 2003. Importantly, once an application has met their quality requirements — beyond removing indecent or illegal applications — there is no restriction on the applications that can be hosted.

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 €1 fee — registration is open to both companies and individuals. When your application starts selling the revenue depends on how the user makes the purchase:

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

While the reduced revenue from purchases made through operator billing may seem a disadvantage, where operator billing is available for each \$1 in credit card revenue on average developers will receive over \$10 — so operator billing can be the most lucrative option. (Developers who really don't like the idea of losing the 10% can opt to sell apps through credit card purchases only.)



# 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). Its development is stewarded by Nokia.

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.

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

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

That said, native Symbian C++ APIs provide the most comprehensive access to device features and enable rich application development. They provide fine-grained control over all aspects of the operating system, including memory, performance and battery life; and deliver a consistent performance advantage over other runtimes. For these reasons Symbian C++ is still of interest to many developers.

1) [www.symbian.org](http://www.symbian.org)

2) [developer.symbian.org/wiki/index.php/Symbian\\_C%2B%2B\\_Quick\\_Start](http://developer.symbian.org/wiki/index.php/Symbian_C%2B%2B_Quick_Start)

# Prerequisites

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

1. Symbian^3 SDK for Nokia devices<sup>1</sup> and installing it
2. Installing ActivePerl version 5.6.1.638<sup>2</sup> from the SDK
3. Installing Carbide.c++<sup>3</sup>

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

## Carbide.c++

Carbide.c++ is designed for developers who wish to create applications that run on production phones - that is “on top” of the Symbian platform. Typical users include professional application and games developers, professional service companies, hobbyist developers, students and research groups. Carbide.c++ is intended for use with one or more SDKs.

- 1) [www.forum.nokia.com/Develop/Other\\_Technologies/Symbian\\_C++/Tools](http://www.forum.nokia.com/Develop/Other_Technologies/Symbian_C++/Tools)
- 2) found in the Symbian^3 SDK at `epoc32\tools\distrib\ActivePerl-5.6.1.635-MSWin32-x86.msit`
- 3) [www.forum.nokia.com/Develop/Other\\_Technologies/Symbian\\_C++/Tools/](http://www.forum.nokia.com/Develop/Other_Technologies/Symbian_C++/Tools/)
- 4) [www.forum.nokia.com](http://www.forum.nokia.com)

Based on Eclipse, Carbide.c++ includes the GCC compiler, a debugger that enables debugging on both the emulator and production devices, analysis tools, and more.

## Symbian/S60 Software Developer Kits (SDKs)

The Symbian and older S60 SDKs contain the libraries and header files that enable you to develop applications. These SDKs provide 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 SDKs for the Symbian/S60 versions you wish to build for, you can use the built-in application wizard to build, debug and run your first native application, and download it to a Symbian phone, without 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. This means that Developers can benefit from an open-source codebase 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. To access these features you need to use the Symbian APIs or take advantage of the APIs offered by Qt and Qt Mobility.

## Testing

For automated unit testing, googletest<sup>1</sup> works on Symbian, and other Mobile C++ platforms.

## Singing

Symbian uses a trust-based platform security model. This means some APIs are protected by platform security “capabilities”. If you use APIs protected by capabilities, your application will need to be signed before it can be distributed. In addition it’s necessary to sign an application during development in order to install it to a device: This is done using a “development certificate”. For most applications, signing and the provision of “development certificates” will be provided for free as part of the submission to Ovi store (see the chapter on Programming Qt applications for more information). For a limited number of applications using more advanced APIs, it will be necessary to obtain Certified Signed through the Symbian Signed website<sup>2</sup>.

1) [www.code.google.com/p/googletest](http://www.code.google.com/p/googletest)

2) [www.symbiansigned.com](http://www.symbiansigned.com)

## Distribution

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





# Mobile Developer's Guide

## Programming WebOS Apps

WebOS is a multitasking mobile operating system based on a Linux kernel introduced January 2009 by Palm. They introduced the Palm Pre and Palm Pixi which are webOS driven. In Spring 2010, Palm was bought by Hewlett Packard who are now introducing webOS based tablets to increase the market share (in December 2010, only 1,3% of US smartphones supported webOS<sup>1</sup>).

Since early 2011 the prototype-based WeOS framework “Mojo” is replaced by “Enyo”. Enyo improves the performance of your apps and enhances the layout handling. Probably the biggest advantage of the platform is that there are very few prerequisites needed and that it is very easy to get started, especially when you have a little experience with web developing:

WebOS applications can be built using common web standards like JavaScript, html and CSS. For implementing advanced technologies like 3D graphics, you will code in C language.

1) [blog.nielsen.com/nielsenwire/online\\_mobile/us-smartphone-battle-heats-up](http://blog.nielsen.com/nielsenwire/online_mobile/us-smartphone-battle-heats-up)



# Prerequisites

First step of implementation is downloading the HP webOS SDK (which also includes a PDK installer) and VirtualBox (needed for the Palm Emulator) from [developer.palm.com](http://developer.palm.com). The SDK/PDK is available for Windows, Mac and Linux.

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

# Implementation

Before you start developing you need to be aware of which Development Kit answers your needs.

SDK (Software Development Kit)	PDK (Plug-in Development Kit)
Javascript, CSS, Prototype based framework (Enyo)	C, C++
No compiling needed (no compiling errors)	Porting applications from other platforms
Easy developing for web developers	Implementing 3D graphics with OpenGL and SDL
Ares <sup>1</sup> , an online interface builder via drag&drop and a built-in editor	Integrate C/C++ code into apps built with web technologies
Easily swap between Ares and hand-built workspace (since Enyo)	Eclipse plugin
Source is viewable for users	Pre-configured Xcode template with headers and libraries for MAC machines, source is not viewable for users

1) [ares.palm.com](http://ares.palm.com)

For using Ares you need to sign up for a free account at [developer.palm.com](http://developer.palm.com). The source code is saved online so you can access it and continue your work from anywhere you want.

Some major Ares features:

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

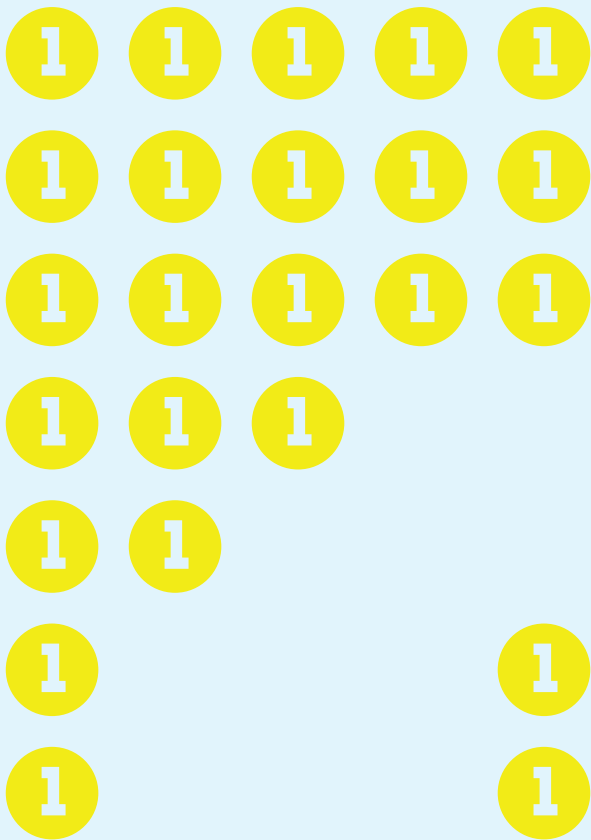
## Testing

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

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

The debugging tools depend on which Development Kit you decide to use:

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

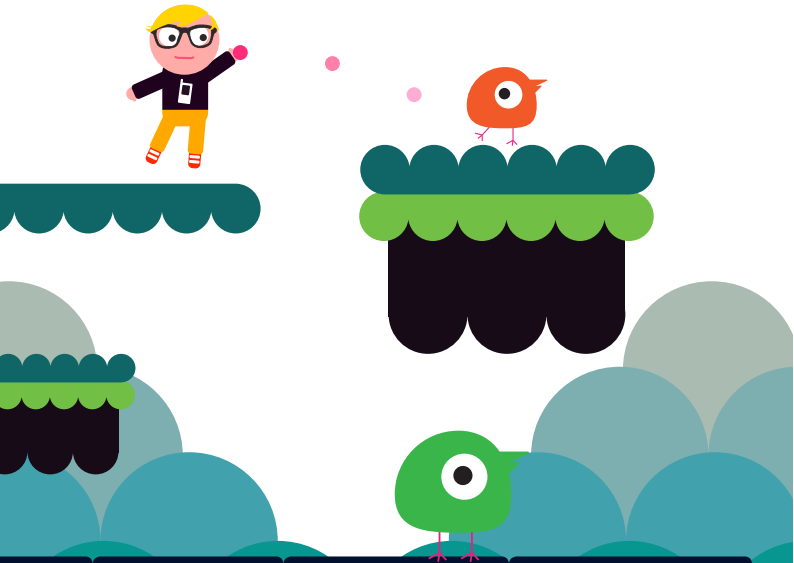


## Distribution

Be sure to have a look into the app submission checklist on [developer.palm.com](http://developer.palm.com) to ensure you are following the HP guidelines. There is a SDK and a PDK app submission checklist related to the application type you want to submit. Before you can submit any application you need to create a developer account. There are three different types, all of them are free.

Since you cannot change the membership type afterwards, choose carefully:

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



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

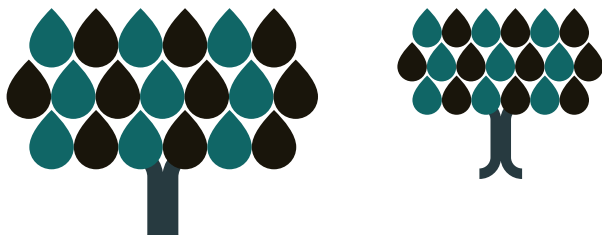
# Mobile Developer's Guide

## Programming Windows Phone 7 Apps

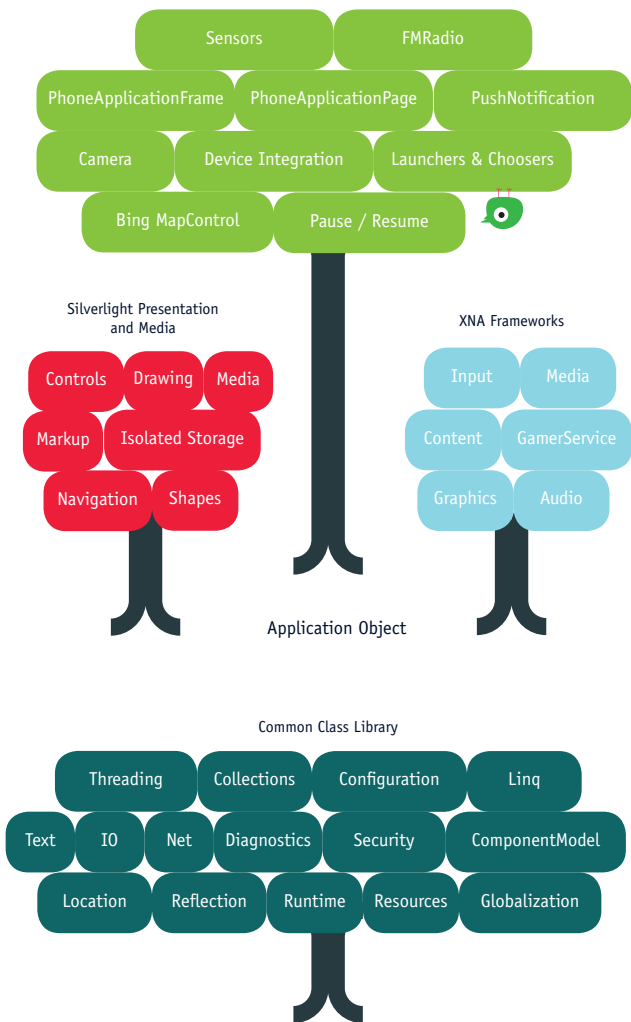
Microsoft is making a fresh start with the Windows Phone 7 platform. The former Windows Mobile operating system was declining in both user acceptance and market share, so the need for innovation was clearly felt. The OS is geared towards consumers rather than business users, and has a simple user interface that is focused on typography and content. A marketing budget of 500 million USD has been spent to promote the new platform and 1.5 million handsets have been sold in the first six weeks since launch. Although Apple sold more devices of the iPhone 4 within just three days, Windows Phone 7 shall still be considered as a relevant player.

## Development

Windows Phone 7 development is done in C# or VB.NET, using the Microsoft Visual Studio IDE. Depending on the type of application you are developing, you will use one of two platforms – Silverlight for event-driven applications, or XNA for games driven by a “game loop”. Both platforms are present in the Windows Phone 7 OS layer as shown in the following figure:



## Windows Phone 7 Series Framework



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

**Use Silverlight if...**

...you want to create an event-driven application.

...you want to use standard Windows Phone 7 controls.

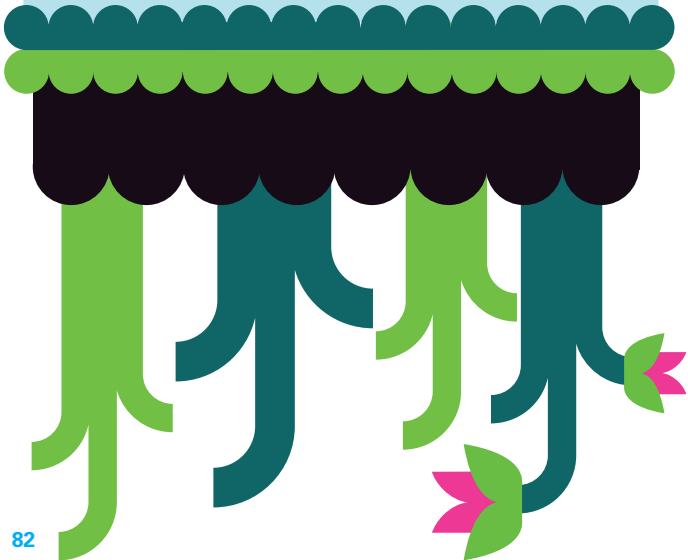
...you want to target both Windows Phone 7 and the web, re-using some code.

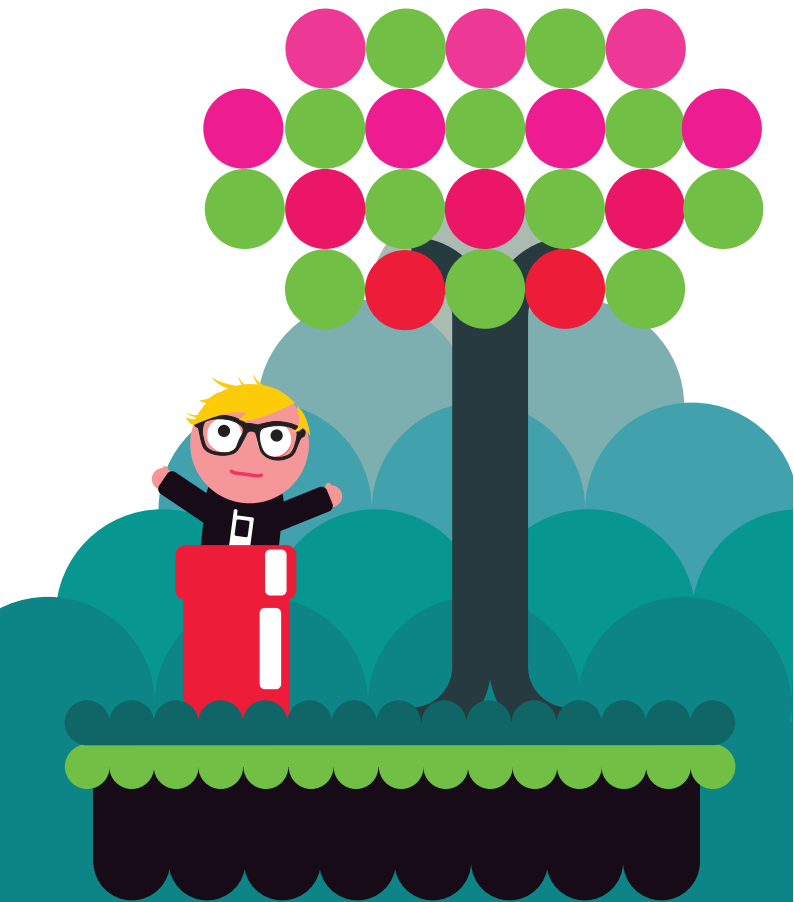
**Use XNA if...**

...you want to create a 2D or 3D game.

...you want to manage art assets such as models, meshes, sprites, textures, or animations.

...you want to target Windows Phone 7, Windows, and Xbox 360, re-using lots of code.





## Functions and Services

Windows Phone 7 applications have access to input such as location, multi-touch screen, accelerometer, and microphone. Available services include media playback and push notifications that can update application tiles (widgets that reside on the start page of the phone).

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

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

There are currently several native controls that are not included in Silverlight, such as pivot, panorama, and context menu. At [phone.codeplex.com](http://phone.codeplex.com) you can download an intermediate solution if you want to use such controls.



## Distribution

Applications for Windows Phone 7 are distributed through a single endpoint, Microsoft's Marketplace service. While application content is reviewed and restricted similar to the Apple App Store, Microsoft provides fairly comprehensive guidelines for submission, available at [create.msdn.com](http://create.msdn.com).

## Resources

Visit [create.msdn.com](http://create.msdn.com) for news, developer tools, and forums. The development team posts on their blog [windowsteamblog.com/windows\\_phone](http://windowsteamblog.com/windows_phone).

## Testing

You can unit test applications using the Windows Phone Test Framework<sup>2</sup> or the Silverlight Unit Test Framework<sup>3</sup>. Check the web for several public articles on unit testing Silverlight and Windows Phone applications<sup>4</sup>.

- 1) [www.wptestlib.codeplex.com](http://www.wptestlib.codeplex.com)
- 2) [www.silverlight.codeplex.com](http://www.silverlight.codeplex.com)
- 3) [live.visitmix.com/mix10/sessions/cl5](http://live.visitmix.com/mix10/sessions/cl5) and [dotnet.dzone.com/news/test-driven-development](http://dotnet.dzone.com/news/test-driven-development) and [www.smartypantscoding.com/a-cheat-sheet-for-unit-testing-silverlight-apps-on-windows-phone-7](http://www.smartypantscoding.com/a-cheat-sheet-for-unit-testing-silverlight-apps-on-windows-phone-7)

# 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 characterized 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, on Symbian or Blackberry devices widgets are installed and accessed in the same way as native applications are.
- Widgets can look like native applications: Some widget environments include features that replicate the device's native menus and UI. Widgets that behave like native applications are much easier to use than websites.
- Widgets can run on a device's home screen: Some widget environments, such as Symbian, are able to provide summary views users can add to their device's homepage while others, such as Samsung's Touchwiz can incorporate arbitrarily sized widgets.
- Widgets can use device data: The ability to use device data, such as location or contact records, enables widgets to offer information that has context, such as identifying social network contacts based on the entries in the device's address book.

- Widgets can generate revenue: They can be packaged and distributed via application stores so you can sell them just like native applications.

If there is a challenge in creating widgets, it's the the lack of universal support for a common standard. W3C, together with WAC/ JIL is pushing forward with the definition of standards. This standardization is still underway and information on its progress can be found in the W3C Wiki<sup>1</sup>.

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:

- **Blackberry (v5.0 or later):**  
[bit.ly/blackberry-widgets](http://bit.ly/blackberry-widgets)
- **Nokia WRT (Selected S60 3rd Edition, Feature Pack 2, S60 5th Edition and Symbian^3 devices):**  
[bit.ly/nokia-wrt](http://bit.ly/nokia-wrt)
- **Vodafone360:** [bit.ly/vf-widgets](http://bit.ly/vf-widgets)
- **WAC/ JIL:** [www.jil.org](http://www.jil.org)
- **Windows Mobile (v6.5):** [bit.ly/winmo-widgets](http://bit.ly/winmo-widgets)

To port your widget over to for example iPhone and Android, you can use tools like PhoneGap<sup>2</sup>, Titanium from Appcelerator<sup>3</sup>, and Rhomobile<sup>4</sup> among others. Of these options, PhoneGap offers a solution that is closest to the W3C approach.

1) [www.w3.org/2008/webapps/wiki/WidgetSpecs](http://www.w3.org/2008/webapps/wiki/WidgetSpecs)

2) [www.phonegap.com](http://www.phonegap.com)

3) [www.appcelerator.com](http://www.appcelerator.com)

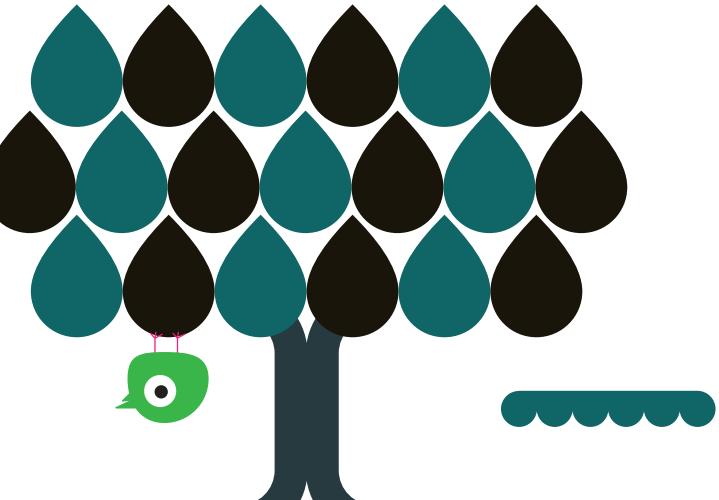
4) [www.rhomobile.com](http://www.rhomobile.com)

# 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.

There are also a number of tools specifically designed for developing widgets. These may be delivered as plug-ins to web authoring tools, such as the Nokia WRT plug-ins for Aptana Studio; Dreamweaver; and Visual Studio, or as standalone tools like BlackBerrys Widget SDK<sup>1</sup>. These tools provide template projects, preview environment, validation, packaging, and deployment features.

1) [us.blackberry.com/developers/browserdev/widgetsdk.jsp](http://us.blackberry.com/developers/browserdev/widgetsdk.jsp)



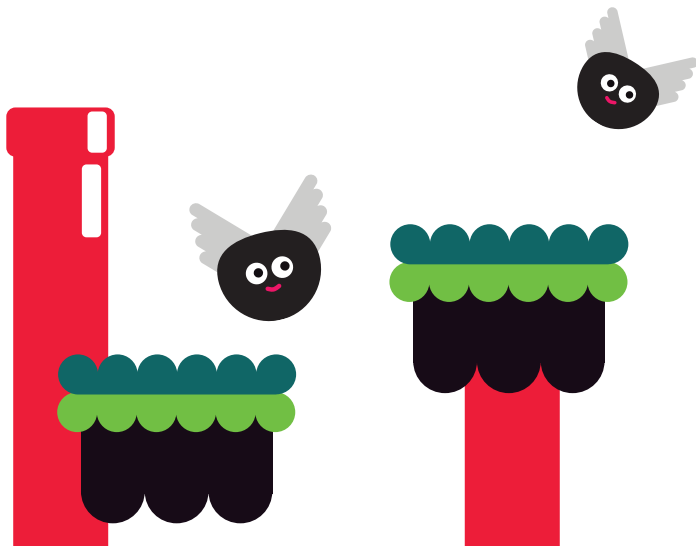
## Writing your code

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

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

You can start by simply:

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



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

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

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

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

serviceObj.getCurrentPosition(success_callback,
error_callback,positionOpts);
...
function success_callback(result){
    Lat = result.coords.latitude;
    Long = result.coords.longitude;
}
```

While standards are still to be 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 finish desktop testing, final testing on your own phone will be essential. The way a widget looks and behaves can only be fully assessed on a real screen, under realistic lighting conditions, and in a real network.

## Signing

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

# Distribution

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

However, some widget environments enable you to add a link to a widget on a website so that the widget installs directly into the 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 store<sup>1</sup>, but by packaging your widgets appropriately you can upload them into Nokia Ovi store<sup>2</sup>, the Windows Marketplace<sup>3</sup> or RIM Blackberry AppWorld<sup>4</sup>. 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](http://widget.vodafone.com)

2) [store.ovi.com](http://store.ovi.com)

3) [www.windowsmarketplace.com](http://www.windowsmarketplace.com)

4) [appworld.blackberry.com/webstore](http://appworld.blackberry.com/webstore)

# Mobile Developer's Guide

## Developing Accessible Apps

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

Many of your potential users have a range of disabilities which make it more difficult for them to use mobile technology. These disabilities include but are not limited to various levels of sight or hearing impairments, Cognitive disabilities, dexterity issues or things like technophobia.

The challenge then, is to design your apps accessibly to enable as many people to use/purchase your application as possible.

### Built-In Accessibility Features

Depending on the platform you are using, there are various accessibility features built-in to handsets to help both the user and the developer to make best use of your apps. For example, the iOS devices include:

- **VoiceOver**, which speaks the objects and text on screen, helping blind users who cannot see the screen.
- **Zoom**. This magnifies the entire contents of the display screen.
- **White on Black**. This inverts the colours on the display, which helps many people who need the contrast of black and white but find a white background emits too much light.

- **Captioning and subtitles**, for people with hearing loss.
- **Audible, visible and vibrating alerts**, to enable people to choose what works best for them.
- **Voice Control**. This enables users to make phone calls and control music playback using voice commands.

Users often rely on 3rd party applications such as Talkback on the Android platform available from the market place or Talks from nuance for the Symbian platform which provides screen reading and screen magnification features.

Some of the platforms have Accessibility APIs to help you enable your app to work along side assistive technology that may be running on the user's phone along side your app. In general however, the following guidelines will help your apps to be more accessible.

- Use standard rather than custom UI elements where possible
- Follow the standard UI guidelines on your platform.
- Label all images with a short description of what the image is such as "Play" for a play button etc.
- Avoid using colour as the only means of differentiating an action. For example a colour-blind user won't be able to correct the red form fields.
- Use the Accessibility API for your platform if there is one. This will enable you to make custom UI elements more accessible and will mean less work on your part across your whole app.
- Test your app on the target device with assistive technology such as Voiceover on the iPhone.

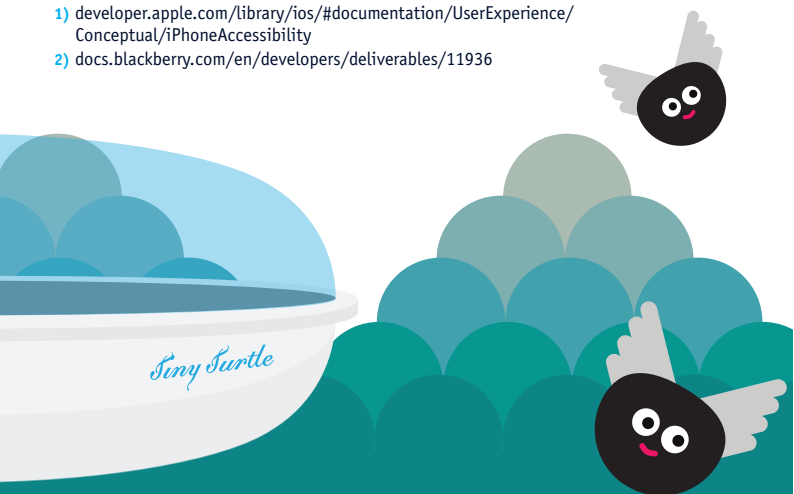
## Developing Accessible iOS Apps

iOS has good support for accessibility. However, it only works well if the accessibility guidelines<sup>1</sup> have been followed. This document details the API and provides an excellent source of hints and tips for maximizing the user experience with your apps.

## Developing Accessible BlackBerry Apps

BlackBerry also provides good and extensive information about the use of the accessibility API and many hints on accessible UI design on their website for developers<sup>2</sup>.

- 1) [developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/iPhoneAccessibility](http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/iPhoneAccessibility)
- 2) [docs.blackberry.com/en/developers/deliverables/11936](http://docs.blackberry.com/en/developers/deliverables/11936)



## Developing Accessible Symbian / Qt Apps

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

When developing Native Symbian apps your best chance of developing an accessible app is to use the standard UI controls where possible. If you are developing using QT, then at the time of writing the Symbian port of Qt does not include, Qt’s accessibility API. However this is likely to change soon. Please check the web for details<sup>1</sup>.

## Developing Accessible Android Apps

Android has several accessibility features including an accessibility API. Again, when developing Android apps you should use standard views where possible and make sure users can navigate your app via the trackball. This will give your app the best chance of being rendered accessibly by the likes of Talk-back and other assistive technology applications. For more information about Android accessibility including how to use the text to speech API, go to [code.google.com/p/eyes-free](http://code.google.com/p/eyes-free). For more information about making your custom views accessible, please see [developer.android.com/reference/android/view/accessibility/package-summary.html](http://developer.android.com/reference/android/view/accessibility/package-summary.html)

1) [doc.qt.nokia.com/qq/qq24-accessibility.html](http://doc.qt.nokia.com/qq/qq24-accessibility.html)

# Mobile Developer's Guide

## Programming with Cross-Platform Tools

So many platforms, so little time. This accurately sums up the situation that we have in the mobile space. There are plenty of platforms to choose from nowadays: Android, bada, BlackBerry, iOS, MeeGo, Symbian, webOS and Windows Phone are the most important smartphone platforms; while Java ME and Brew MP dominate on feature phones.

Decision makers have to evaluate the market reach, market potential and cost of development for each platform and weigh those against the drawbacks of cross platform frameworks. Don't confuse the market size of a platform with the market potential for your application — while Android and iPhone appear to have the biggest market places at the beginning of 2011, you will also need the biggest marketing effort to get noticed. So concentrating on seemingly smaller platforms might be a smart choice for some apps.

However, to quote Queen's famous lyrics: 'I want it all, I want it all, I want it all ...and I want it now!' In that case you can either throw money at the problem or adopt a cross platform strategy.

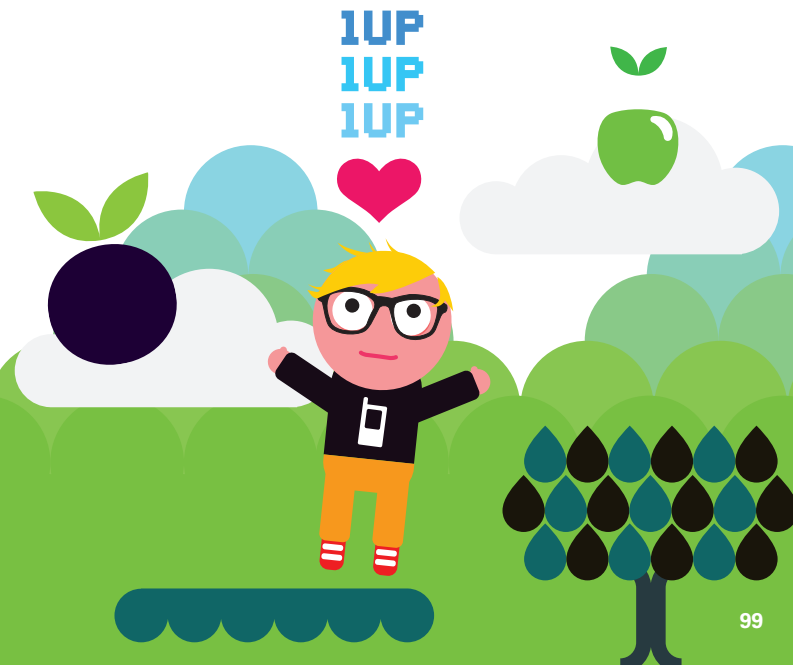
By the way, we are not talking about app stores here, this is a different market fragmentation problem. The more than 100 app stores, from operators, manufacturers and independent companies create challenges of their own. This is a topic for a different chapter that may come in a future edition.

# Limitations and Challenges of Cross Platform Approaches

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

## Native Programming Languages

By now you will certainly have noticed that most mobile platforms release their own SDKs, which enable you to develop apps in the platform's supported programming languages.



However, these languages tend to belong to one of a few families of root languages and the following table provides an overview of these and the platforms they are supported on:

Language	1st Class Citizen <sup>1</sup>	2nd Class Citizen <sup>2</sup>	Target Platforms
C, C++	5	3	First class: bada, Brew MP, MeeGo, Symbian, Windows Phone Classic Second class: Android (partly, using the NDK), iOS (partly), webOS (partly)
C	1	0	Windows Phone and Windows Phone Classic (formerly Windows Mobile)
Java	3	2	First class: Android, BlackBerry, Java ME Second class: Symbian, Windows Phone Classic
JavaScript	1	2	First class: webOS Second class: BlackBerry (Widget), Nokia (WRT)
Objective-C	1	0	iOS

- 1) Supported natively by the platform, e.g. either the primary or only language for creating applications
- 2) Supported as an option by the platform, e.g. can be used as an alternative to the native language but generally won't provide the same level of access to platform features.



Cross platform frameworks can overcome the programming language barrier by different means:

- **Web Technologies**

On most platforms you have direct support for web technologies by embedding so called webviews. Along with HTML and CSS this approach supports JavaScript also.

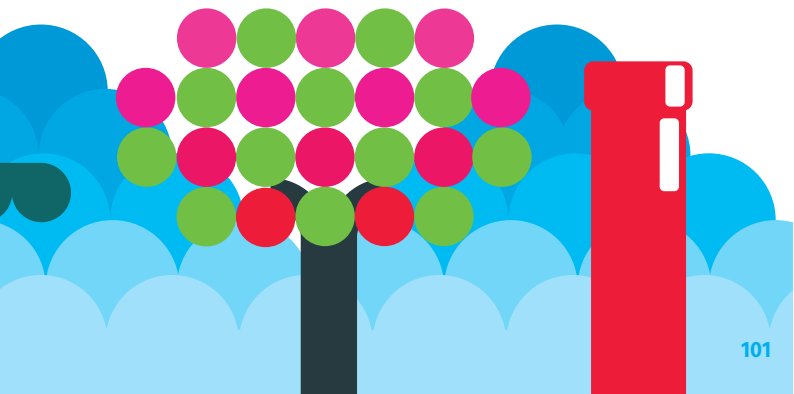
- **Interpretation**

Frameworks can also choose to interpret any language themselves. In the gaming world Lua scripting is quite popular, for example.

- **Cross Compilation**

The holy grail of cross platform frameworks, cross compilation is the most complex technical solution to this problem. It enables you to write an app in one language and have it transcoded into a different language, offering runtime native speed.

Most frameworks also provide a set of cross platform APIs that enable you to access certain features, such as the handset's geolocation capabilities, in a common way.



## UI + UX

A difficult hurdle for cross platform approaches is created by the different User Interface (UI) and User eXperience (UX) patterns that prevail on individual platforms. It's relatively easy to create a nice looking UI that works the same on several platform. Such an approach, however, might miss important UI components that are available on a single platform only and could improve the user experience drastically. The other challenge with a cross-platform UI is that it can behave differently to the native UI users are familiar with, resulting in your application failing to "work" for users. Customizing and tailoring the UI and UX to each platform can be a large part of your application development effort and is arguably the most challenging aspect of a cross platform strategy.

## Desktop Integration

Integration of your application into the desktop varies a lot between the platforms; on iOS you can only add a badge with a number to your app's icon, on Windows Phone you can create live tiles that add any simple information to the desktop, while on Android and Symbian you can add a full-blown desktop widget that may display arbitrary data and use any visuals. Using such integration might improve the interaction with your users drastically.

## Multitasking

Multitasking enables background services and several apps to run at the same time. Multitasking is another feature that is realized differently among operating system. On Android, BlackBerry, Symbian and MeeGo there are background services and you can run several apps at the same time; on Android it's not possible for the user to exit apps as this is handled automatically by the OS when resources run low. On iOS we

have a limited selection of background tasks that may continue to run after the app's exit. And then there is a freezing and unfreezing mechanism on Windows Phone. So if background services can improve your app's offering, you should evaluate cross platform strategies carefully to ensure it enables full access to the phone's capabilities in this regard.

### **Battery Consumption and Performance**

Closely related with multitasking is the battery usage of your application. While CPU power is roughly doubled every two years (Moore's law says that the number of transistors is doubled every 18 months), battery capacity is doubled around every seven years. This is why some smartphones like to spend so much time on their charger. The closer you are to the platform in a cross platform abstraction layer, the better you can control the battery consumption and performance of your app. As a rule of thumb, the longer your application runs in one go, the less abstraction you can afford.

### **Push Services**

Push services are a great way to give the appearance that your application alive even when it's not running. In a chat application you can, for example, send incoming chat messages to the user using a push mechanism. The way push services work and the protocols they use, again, can be realized differently on each platform.

This section outlines some of those strategies you can employ to implement your apps on different platforms.

## Direct Support

You can easily support several platforms by having a specialized team for each and every target platform. While this can be resource intensive, it will most likely give you the best integration and user experience on each system. An easy entry route is to start with one platform and then progress to further platforms once your application proves itself in the real world.

## Asset Sharing Levels

When you maintain several teams for different platforms you can still save a lot of effort when you share some application constructs:

- **Concept and Assets**

Mostly you will do this automatically: share the ideas and concepts of the application, the UI flow, the input and output and the design and design assets of the app (but be aware of the need to support platform specific UI constructs).

- **Data Structure and Algorithm**

Go one step further by sharing data structures and algorithms among platforms.

- **Code Sharing of Business Model**

Using cross platform compilers you can also share the business model between the platforms. Alternatively you can use an interpreter or a virtual machine and one common language across a variety of platforms.

## — Complete Abstraction

Some cross platform tools enable you to completely abstract the business model, view and control of your application for different platforms.

## Player and Virtual Machine Concepts

Player concepts typically provide a common set of APIs across different platforms. Famous examples include Flash, Java ME and Lua. This approach makes development very easy. You are dependent, however, on the platform provider for new features and the challenge here is when those features are available in one platform only.

Often player concepts tend to use a 'common denominator' approach to the offered features, to maintain commonality among implementations for various platforms.

Generator concepts carry the player concept a step further, they are often domain specific and enable you to generate apps out of given data. They often lack flexibility compared to programmable solutions.

## Cross Language Compilation

Cross language compilation enables to code in one language that is then transformed into a different, platform specific language. In terms of performance this is often the best cross platform solution, however there might be performance differences when compared to native apps. This can be the case, for example, when certain programming constructs cannot be translated from the source to the target language automatically. There are three common approaches to cross language compilation: direct source to source translation, indirectly by translating the source code into an intermediate abstract language and direct compilation into a platform's binary format. The indirect approach typically produces less readable code.

This is important when you would like to continue the development on the target platform and use the translated source code as a starting point.

### (Hybrid) Web Apps

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

- 1) Websites are not listed in the app stores, so users don't find them and monetization is difficult. (Although you could create a simple application or widget that opens your website and submit that to a store, but this will not help with monetization.)
- 2) Websites only work online.
- 3) Websites have an inferior user experience compared to native apps.

There are a couple of web application frameworks that are listed in the following table. With these frameworks you can create web apps that behave almost like real apps, including offline capabilities. Typically you have no access to hardware features and native UI elements, so in our opinion they don't count as 'real' cross platform solutions: these solutions are therefore not listed in the table at the end of this article. Web apps have some advantages over traditional websites:

- 1) While you cannot put a web app directly in an app store, you can use web based tools such as PhoneGap in combination with a web app solution to create a better experience that is available in the app stores.
- 2) Web apps also work offline.
- 3) Web apps can look and behave similar to native apps, however there are often slight — albeit annoying — differences compared with their native counterparts.

Web App Solution	License	Target Platforms
<b>jQuery Mobile</b> www.jquerymobile.com	MIT and GPL	Android, bada, BlackBerry, iOS, MeeGo, Symbian, webOS, Windows Phone
<b>JQTouch</b> www.jqtouch.com	MIT	iOS
<b>iWebkit</b> iwebkit.net	LGPL	iOS
<b>iUI</b> code.google.com/p/iui	BSD	iOS
<b>Sencha Touch</b> www.sencha.com/products/touch	GPL	Android, iOS

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

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

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

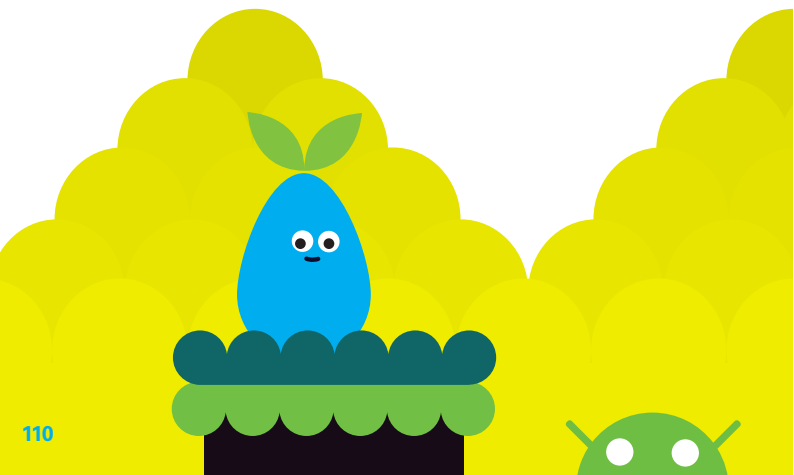
# Cross Platform Solutions

There are many solutions available now, so it's hard to provide a complete overview. You may call this fragmentation, I call it competition. Word of warning: we don't know about all solutions here, if you happen to have a solution on your own that is publicly available, please let us know about it at [developers@enough.de](mailto:developers@enough.de)

Solution	License	Input	Output
<b>Airplay</b> www.airplaysdk.com (Ideaworks Labs)	Commercial	C++	Android, bada, brew, iOS, Symbian, webOS, Windows Phone Classic
<b>appMobi</b> www.appmobi.com	Commercial	HTML, CSS, JavaScript	Android, iOS
<b>Bedrock</b> www.metismo.com (Metismo)	Commercial	Java ME	Android, bada, BlackBerry, brew, Consoles, iOS, PC, webOS, Windows Phone, Windows Phone Classic
<b>Celsius</b> mobile-distillery.com (Mobile Distillery)	Commercial	iOS Java ME	Android, BlackBerry, brew, iOS, Symbian, Windows Phone Classic
<b>Corona</b> www.anscamobile.com (Anasca Software)	Commercial	JavaScript	Android, iOS
<b>EDGELIB</b> www.edgelib.com (elements interactive)	Commercial	C++	Android, iOS, PC, Symbian

Solution	License	Input	Output
<b>id Tech 5</b> www.idsoftware.com (id)	Commercial	C++	Consoles, iOS, PC
<b>Irrlicht</b> irrlicht.sourceforge.net gitorious.org/irrlicht-android	Open Source	C++	Android & iOS with OpenGL-ES version, PC
<b>J2ME Polish</b> www.j2mepolish.org (Enough Software)	Open Source + Commercial	Java ME, HTML, CSS, JavaScript	Android, BlackBerry, iOS, J2ME, PC, Windows Phone Classic
<b>Flash</b> adobe.com/devnet/devices.html (Adobe)	Commercial	Flash	Android, iOS, PC
<b>Mono Touch</b> monotouch.net (Novell)	Commercial	C#	iOS
<b>MoSync</b> www.mosync.com (MoSync)	Open Source + Commercial	C	Android, J2ME, Moblin, Symbian, Windows Phone Classic
<b>PhoneGap</b> www.phonegap.com (Nitobi)	Open Source	HTML, CSS, JavaScript	Android, BlackBerry, iOS, Symbian, webOS
<b>Qt</b> qt.nokia.com (Nokia)	Open Source + Commercial	C++	MeeGo, PC, Symbian, and Windows Phone Classic as well as desktop Windows, Apple and Linux OS.
<b>Rhodes</b> rhomobile.com/products/rhodes (rhomobile)	Open Source + Commercial	Ruby, HTML, CSS, JavaScript	Android, BlackBerry, iOS, Symbian, Windows Phone Classic

Solution	License	Input	Output
<b>SI02</b> sio2interactive.com (sio2interactive)	Commercial	C	iOS, other announced
<b>Titanium</b> www.appcelerator.com	Open Source	JavaScript	Android, Consoles, iOS, PC
<b>Unity3</b> unity3d.com (Unity Technologies)	Commercial	C#	Android, iOS, PC
<b>Unreal</b> www.unrealtechnology.com (Epic Games)	Commercial	UnrealScript, C++	Consoles, iOS, PC
<b>Whoop</b> www.whoop.com (Whoop)	Commercial	Drag and Drop	Android, BlackBerry, iOS, J2ME, Windows Phone Classic
<b>XML VM</b> xmlvm.org	Open Source + Commercial	Java, .NET, Ruby	C++, Java, JavaScript, .NET, Python



Here are some questions that you should have in mind when evaluating cross platform tools. Not all of them might be relevant to you, so weight the answers appropriately.

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



# Mobile Developer's Guide

## Creating Websites for Mobile Usage

“Don’t make me think!”- this title of one of the most popular books about usability<sup>1</sup> summarizes what usability is all about very well. You should always remember that it does not matter how fancy a mobile website might be: **The user should be able to use it easily and intuitively.**

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

1. Mobilize, don't miniaturize: Create a concept that utilizes the possibilities of the technology. You won't satisfy many people by simply offering a smaller version of your classic website.
2. Keep all paths open: Leave it up to the user to access either the mobile or desktop version of your website.
3. Keep it simple: Avoid complex navigation structures, the users won't dig that deep anyway while they are on the go.
4. Avoid text input wherever possible: Text input on mobiles sucks. If you really need the users to enter text, use wide input boxes so that they see what they are typing.

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 and alike to be displayed properly on the handset (check the corresponding chapter in this guide: “Implementing Rich Media” for more information). Avoid formats such as .doc and pdf.
- 6.** The user is a creature of habit; respect that: Adapt usage patterns from classic websites such as linking logos to the home page or offering corrections to mistyped search requests.
- 7.** Think of stubby fingers: When optimizing your content for touch screen phones don't use clickable areas smaller than 1cm x 1cm.
- 8.** Use sharp contrasts: Fonts and background colors that guarantee legibility in any surrounding, including bright sunlight.
- 9.** Reflect continuously: Ask yourself if you would use the implemented features yourself. Ask your friends and colleagues as well before realizing your ideas.
- 10.** Again: Don't make the users think. Try to implement intuitive navigation, don't force them to make decisions more often than necessary. Users will often click the first link in a list anyway.

## Mobile usage patterns and their consequences

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

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

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

Other things to consider during design are that the mobile user has no mouse, often they have no real keyboard and the size of the screen is very limited. This means the content of a mobile website has to be arranged accordingly: Images should not be too large, all relevant elements should be easily accessible as the user is not able to move a cursor freely around the site.

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

In addition to factors like markup, image formats and navigation, you should never forget about the most valuable feature to a mobile user: battery life. Complex websites with many JavaScript, CSS and Flash elements need a lot of processing power which means battery power.

# A History of the Mobile Web

## WAP: The stone age of mobile internet

From its very beginning, the mobile internet was expected to be a cash cow. WAP made it possible to send limited text along with black and white images in the WBMP format to the monochrome displays of user's cellphones. The industry charged the user for that by inventing complicated data transfer rates and strange subscription models. Many players expected to get rich by selling information that was already available for free on the internet.

The Wireless Markup Language WML, which was used at that time was another reason why the expected breakthrough didn't happen quickly: 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 use of images was limited to black and white pictures which were a lot less appealing when compared to the images users could see on their home computers.

The first attempts to lift the mobile internet to the next level were made by NTT DoCoMo with the invention of CHTML; and Microsoft's Windows CE that worked best with HTML/3.2. Both of these technologies were eventually replaced by XHTML/MP 1.0.

## Current Situation

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

The next big thing is HTML5 which is still under development. However, some of the latest browsers already support parts of this new standard: Scripting, GPS access, CSS3 elements, CSS animations and the possibility of offline (on-device) data storage. At the same time, the W3C Device APIs and Policy Working Group (DAP) is defining client-side APIs. These APIs will make it possible to develop web applications and web widgets that interact with device services such as calendar, contacts, and-camera among others.

As such websites might be expected to progressively evolve to offer the features of web widgets (for more information on widgets, see the corresponding chapter in this guide).

## Content adaptation

### Static Version

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

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



## Automatic adaption technologies

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

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

But of course there are also commercial providers. Again, some concentrate on device data and detection while others focus on offering software platforms that adjust the content accordingly.

Examples of commercial device data and device detection providers:

- **dotMobi**<sup>3</sup> offers several APIs to access their device database DeviceAtlas<sup>4</sup> and intelligent detection
- **DetectRight**<sup>5</sup>

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



Some commercial content adaptation software providers:

- **Sevenval**<sup>1</sup>, 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 Server**<sup>2</sup> 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 Server**<sup>3</sup> is running on BEA, IBM WebSphere, JBoss, Tomcat and Caucho Resin.

## Satisfy the Browser

### Markup

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

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

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

1) [www.sevenval.com](http://www.sevenval.com)

2) [www.volantis.com](http://www.volantis.com)

3) [www.mobileaware.com](http://www.mobileaware.com)

## Page width

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

## Images

Not everybody has a mobile data flat rate, so do not use images 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 functionality to easily scale your images<sup>1</sup>.

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.

## Tables

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

1) [www.imagemagick.org](http://www.imagemagick.org)

## CSS

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

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

## Fonts

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

## Cookies

You can use cookies and should do so, but when really needed only. However, don't put too much trust in cookies: Although it might work fine during one session, the cookie might not longer be available at the start of the next.

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

## Script / AJAX

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

## Meta Settings

You can influence the rendering mode by a number of meta settings<sup>1</sup>:

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

<sup>1</sup> Please compare [learnthemobileweb.com/tag/handheldfriendly](http://learnthemobileweb.com/tag/handheldfriendly)

## Device categories

Do you know what kinds of devices your visitors use most? If not, you should start at the low-level and implement a tool that detects the devices that are accessing your site.

Several providers are offering this kind of analysis software:

- **AdMob** [www.admob.com](http://www.admob.com)
- **Bango** [www.bango.com](http://www.bango.com)
- **Sevenval** [www.device-trends.com](http://www.device-trends.com)
- **TigTags** [www.tigtags.com/mobile\\_analytics/mobile\\_site\\_tracking](http://www.tigtags.com/mobile_analytics/mobile_site_tracking)

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

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

When using this data, bear 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 general market share of various devices? Is it possible that the data reflect the user base of certain operators or technology platform only? Are these data for page impressions, visits or unique users?

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

### Simple Devices

It's likely the majority of your users will access your site with a basic handset such as described by the W3C in the Default Delivery Context<sup>1</sup> or by Luca Passani in his baseline device description<sup>2</sup>.

To fit their needs, these are our recommendations:

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

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

<sup>1)</sup> [www.w3.org/TR/mobile-bp/#ddc](http://www.w3.org/TR/mobile-bp/#ddc)

<sup>2)</sup> [www.passani.it/gap/#baseline](http://www.passani.it/gap/#baseline)

## Full Web Browser-supported Devices

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

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

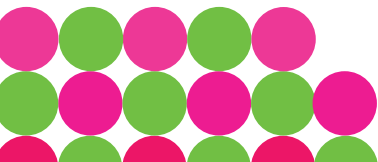
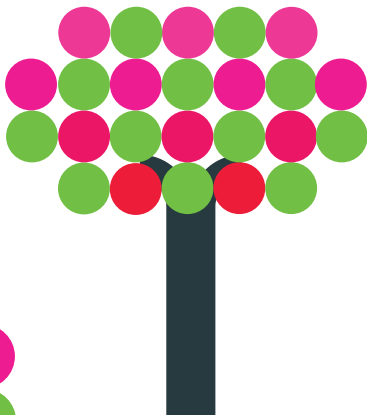
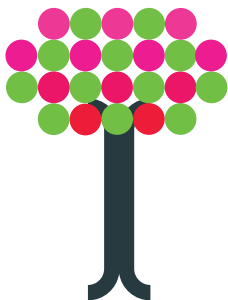
## Touch Devices

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



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

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



## Using GPS in the Browser

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

For further information about how to use GPS location information for web-based apps and services, check out [mobiforge](#)<sup>3</sup>.

## Testing your Mobile Website

There are several ways to test your mobile website:

### User-Agent / Browser

Several desktop browsers offer the ability to change the user-agent and thereby enable the emulation of device detection for the testing of automatic adaption. For Firefox users, the User-Agent Switcher is available on [addons.mozilla.org/en-US/firefox/addon/59](https://addons.mozilla.org/en-US/firefox/addon/59).

[mobiForge](#) offers a configuration file for this Add-On which contains some properties of mobile handsets<sup>4</sup>.

1) [dev.w3.org/geo/api/spec-source.html](http://dev.w3.org/geo/api/spec-source.html)

2) [code.google.com/p/geo-location-javascript](http://code.google.com/p/geo-location-javascript)

3) [www.mobiforge.com/developing/blog/location-location-location](http://www.mobiforge.com/developing/blog/location-location-location)

4) [www.mobiforge.com/developing/blog/user-agent-switcher-config-file](http://www.mobiforge.com/developing/blog/user-agent-switcher-config-file)

## Emulators / SDKs

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

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

Additional emulators can be found on [mobiforge.com/emulators/page/mobile-emulators](http://mobiforge.com/emulators/page/mobile-emulators)

## Remote and Real Device Testing

As mentioned in the other chapters in this guide, it's best to test your product under real-life conditions on as many devices as possible. Remote testing is an alternative, particularly where your access to handsets is limited, consider services like [www.deviceanywhere.com](http://www.deviceanywhere.com) or [www.perfectomobile.com](http://www.perfectomobile.com).

Crowdsourcing testers, with the help of [www.mob4hire.com](http://www.mob4hire.com), is also worth considering.

# Hybrid Apps

The term “hybrid” is used in many technological fields. The auto industry combines two technologies and utilizes the advantages of one system where the other has its drawbacks. The concept is the same behind the so-called hybrid apps.

In the online world you find the classic web browser which is combined with the typical native features. The possibilities for the mobile world are versatile:

- a completely native app that is controlled by the down stream content (yeah, something like that already exists and it is called a browser)
- a client that in addition to its existing function, displays content in a web view format
- applications that give a website a framework to reflect either lack of browser functions or just to take you to an App Store



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

And always remember:

**A mobile website is not simply a small website; it is the one for mobile phone users.  
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 is found in the table below, however multiple variations are possible amongst handsets even within one vendor or version of firmware.

OS/Brand	Container	Protocol
Windows Mobile	.wmv or .mp4	http://
Windows Phone 7	.wmv or .mp4	http://
Symbian, Nokia	.3gp or .mp4	rtsp://
iPhone/ iPod Touch	.mp4	http://
iPad	.mp4	http://
Android	.mp4	http:// for audio rtsp:// for video
ePad	.mp4	http:// for audio rtsp:// for video
Blackberry <=OS 4.2	.avi / .mp4	http://
Blackberry >=OS 4.3	.avi / .mp4 / .3gp	rtsp://
webOS	.flv / .mp4	http://
Other 3G devices	.3gp	rtsp://
Bada	.3gp or .mp4	rtsp://



	Video Codec	Audio Codec	Typical Resolution
	H264	MP3 or AAC	320x240, 176x220, 480x800
	MP4, H263, WMV	MP3, WMA, AAC	800x600
	H263	AMR_nb or AAC	176x144, 320x240, 352x416, 640x480
	H264	AAC or AAC+	320x240/320x480, 640x960
	H264	AAC or AAC+	1024x768
	H264	AAC lc, HE-AAC v2	320x480, 480x800, 480x854
	H264	AAC lc, HE-AAC v2	800x480, 800x854
	MPEG4	AAC lc	480x320
	MPEG4	ADPCM or MP3	352x288
	MPEG4	ADPCM or MP3	352x288
	H263	AMR_nb	176x144, 320x240
	H263, H264, MPEG	AAC, AMR_nb, MP3	800x480

## Streaming vs download

To offer streaming content, Apple's open source Darwin streaming server<sup>1</sup> which can serve streaming video and audio with highest compatibility (except for Windows Mobile) and reliable RTSP combined with FFMPEG<sup>2</sup> would be your preferred choice. For Windows Mobile, Windows Media Server<sup>3</sup> is the easiest choice (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 and 600 kbps for Wi-Fi. RTSP is a protocol which is designed for low-bandwidth mobile devices. If you need a PC player to test and display 3gp RTSP streams, use QuickTime or Real player.

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 video player will not be able to connect with the server. But even if the APN is configured well, some operators block 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. There's not much you can do directly you depend on the user's know-how: He needs to manually set the APN to an open Internet connection. From the device point of view, playing the media will be handled automatically by using the built-in players. Some phones have one player to handle both music and video, some have separate players with separate configurations.

1) [en.wikipedia.org/wiki/Darwin\\_Streaming\\_Server](http://en.wikipedia.org/wiki/Darwin_Streaming_Server)

2) [www.ffmpeg.org](http://www.ffmpeg.org)

3) [en.wikipedia.org/wiki/Windows\\_Media\\_Services](http://en.wikipedia.org/wiki/Windows_Media_Services)

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 Wi-Fi, 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 server. As a consequence, content might behave differently even if the same device, same operator, and same APN are used.

And now? Don't panic! If you use a device database, give them the option to chose manually; else you can offer video to your users if you wish, but inform them about possible problems they may encounter.

When streaming is not available on the phone or blocked by the carrier (which is almost impossible to detect upfront automatically) it's common practice to also provide the user with a download link to the complete file. This is as easy as linking to a download on the regular web, however mobile might be more strict in checking the correct mime types. Use `audio/3gp` for 3gp files and `video/mp4` for mp4 files.

Some mobile phones only use the file extensions for detection, so when using a script like `download.php` a well known trick is to add a parameter like `download.php?dummy=.3gp` to allow correct processing of the media. Some phones cannot play 3gp audio without video, so a simple trick is than to include an empty video track in the file or a still image of the album cover.

## 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 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 Mp4box use `mp4box -hint -latm` for hinting that is compatible with mobile phones. Note that a mp3 file doesn't need and cannot 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. For 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 nowadays 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.

## Media converters

To convert a wide variety of existing media to mobile phone compatible formats FFMPEG is a must have (open source) media format converter. It can and adjust the frame rate, bit rate and channels at the same time. Make sure you build or get the binary with H263, AAC and AMR encoder support included. 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 eRightSoft. If you are looking for a complete server solution with a Java/Open Source Background, check out Alembik.

## Flash (Lite)

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

Early handsets supported Flash Lite, a stripped down version of Flash, but due to lack of enough Flash Lite devices and high power processors in modern devices, full flash format is now supported on most phones with a ‘full internet browser’, exceptions are iPhone and iPad in favor of their development tools for native objective C.

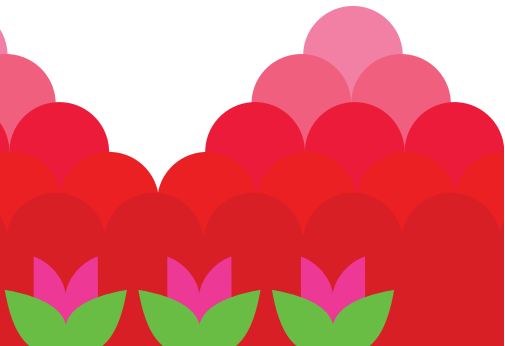
Flash games and applications are very popular, and in some cases, Flash can be used for UI menus and icons. Flash-based rich media UIs can scale to any display screen resolution automatically. This sharply contrasts with implementing standard 2D bitmap graphical UIs which require separate development 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 highlight various UI elements within an application including prompting a user selection and showing color animated menu's. 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.

## HTML5

The more operating systems exist, the more time-consuming it is to build an app for each of the market leaders. In the past Nokia already launched it's web run-time (WRT) system, which allows developers to use HTML but include rich GUI components, just like native apps, and standard functions to integrate into the phone address book. This brings me to HTML5, a standard which is supposed to solve all today's problems with cross-browser and plug-in requirements. HTML5 is in fact an extension to the standard tags and attributes to allow more interaction, richer GUI components and closer integration to the desktop or mobile operating system. Most interesting part is the new <video> tag, which allows to embed a video without the overhead of a Flash plug-in, however early reports show that it's not yet easily supported<sup>1</sup> on all phones.

1) [html5test.com](http://html5test.com)



# Mobile Developer's Guide

## Implementing Location-based Services

An interesting area for mobile development is location aware services, where the information delivered is adapted to the user's location. Applications could be anything from finding parking spaces nearby, analyzing pollen reports for your area, to finding friends at the trade fair et cetera. While the device has to provide the location information you don't need to do all the processing there. There are two basic options for creating location aware data: creating local data sets on a server or on the device. Doing everything on the server-side allows for thinner clients, while fatter clients are more self-sufficient as they are less reliant on constant network access.

### How to obtain positioning data

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

- **Manual input**

The user identifies where his or her phone is located, by choosing a location on a map, an area code, a city on a list. This is often an overlooked method.

- **Network positioning**

The phone locks to a base station, and its unique ID can be looked up from a list provided by the operator. 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 each

case, the accuracy depends largely on the cell / net work density. Higher accuracy is obtained in urban areas than in rural areas.

- **GPS positioning**

The built-in GPS module in the phone (or an external one) gives you an accuracy 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 services

There are several public map sources available. The static map type (distributed as bitmap tiles) is the most widespread, however vector based mapping data solutions, such as Nokia Ovi maps, are becoming available and are likely to become more so as network data capacity issues increase.

Free maps include Open Street Map or CloudMate. Maps that require partnerships and/or payment of a usage fee include NAVTEQ and Microsoft. Some solutions, such as Google Maps, are free when your application is free, while providers are likely to charge for use in paid-for applications or services. Interestingly, several of these sources share similar map formats and therefore are interchangeable.

- **Cloudmade:** [developers.cloudmade.com/projects](http://developers.cloudmade.com/projects)
- **Google Maps:** [code.google.com/apis/maps/](http://code.google.com/apis/maps/)
- **Microsoft Bing API:**  
[www.microsoft.com/maps/developers](http://www.microsoft.com/maps/developers)
- **NAVTEQ:** [www.nn4d.com](http://www.nn4d.com)
- **Nokia:** [www.forum.nokia.com/Develop/Web/Maps/](http://www.forum.nokia.com/Develop/Web/Maps/)
- **Open Street Map:**  
[wiki.openstreetmap.org/wiki/Slippy\\_map\\_tilenames](http://wiki.openstreetmap.org/wiki/Slippy_map_tilenames)

A common type of service is to offer map data in the form of static bitmap tiles. By posting a center coordinate (its latitude and longitude) along with a zoom level, the service returns the appropriate tiles forming a client map view for either desktop or handheld devices. Other services can send map data in vector form, which requires much less network data throughput. On the other hand, the client software needs to be able to handle vector data and display it properly, which is more CPU intense than just displaying bitmap images.

## Implementing location support on different platforms

Location API for J2ME is probably the most widespread 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 [georSS<sup>1</sup>](http://www.georss.org), 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.

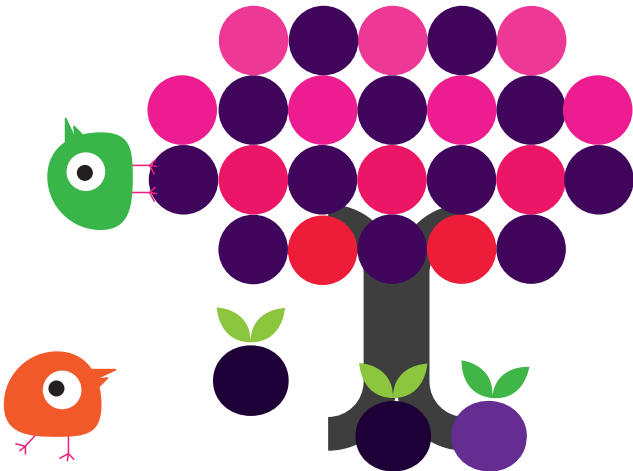
1) [www.georss.org](http://www.georss.org)



## Tools for LBS apps

Several players in the industry provide developer-friendly tools and APIs as a value added service. Using these dramatically speeds up the development and deployment of location-aware services. Each tool normally focuses on one or a few mobile platforms.

- **Garmin Mobile XT SDK:** [developer.garmin.com](http://developer.garmin.com)
- **iPhone SDK:** [developer.apple.com](http://developer.apple.com)
- **Keymap SDK:** [www.mapsdk.com](http://www.mapsdk.com)
- **Mobile Gmaps:** [www.mgmaps.com](http://www.mgmaps.com)
- **NAVTEQ:** [www.nn4d.com](http://www.nn4d.com)
- **Nutiteq:** [www.nutiteq.com](http://www.nutiteq.com)
- **Qt Maps/Navigation API:**  
[qt.nokia.com/products/qt-addons/mobility](http://qt.nokia.com/products/qt-addons/mobility)
- **Spime:** [www.spime.com](http://www.spime.com)



# Mobile Developer's Guide

## Testing your application

After all your hard work creating your application how about testing it before unleashing it on the world? Testing mobile applications used to be almost entirely manual, thankfully automated testing is now viable for many of the mobile platforms. This chapter covers the general topics; testing for specific platforms is covered in the relevant chapter.

### Testability: the biggest single win

If you want to find ways to test your application effectively and efficiently then start designing and implementing ways to test it; this applies especially for automated testing. For example, using techniques such as Dependency Injection in your code enables you to replace real servers (slow and flaky) with mock servers (controlable and fast).

Separate your code into testable modules. Several years ago, when mobile devices and software tools were very limited, developers chose to 'optimize' their mobile code into monolithic blobs of code, however the current devices and mobile platforms mean this form of 'optimization' is unnecessary and possibly even counter-productive.

Provide ways to query the state of the application, possibly through a custom debug interface. You, or your testers, might otherwise spend lots of time trying to fathom out what the problems are when the application doesn't work as hoped.

## Headless client

The user-interface (UI) of a modern mobile application can constitute over 50% of the entire codebase. If you limit your testing to testing using the GUI designed for users you may needlessly complicate your testing and debugging efforts. One approach is to create a very basic UI that's a thin wrapper around the rest of the core code (typically this includes the networking and business layers). This 'headless' client may help you to quickly isolate and identify bugs e.g. related to the device, carrier, and other environmental issues.

Another benefit of creating a headless client is that it may be simpler to automate some of the testing e.g. to exercise all the key business functions and/or to automate the capture and reporting of test results.

You can also consider creating skeletal programs that 'probe' for essential features and capabilities across a range of phone models e.g. for a J2ME application to test the File Handling where the user may be prompted (many times) for permission to allow file IO operations. Given the fragmentation and quirks of mature platforms such probes can quickly repay the investment you make to create and run them.

## Separate the generic from specific

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

Consider platform-specific test automation once the generic code has good automated tests.

## Test-Driven Development

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

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

## Physical devices

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

Here are some examples of areas to test on physical devices

- **UI:** Navigating the UI – for instance, can users use your application with one hand? Effects of different lighting conditions: the experience of the user interface can differ in real sunlight when you're out and about. It's a mobile device – most users will be on the move.

- **Location:** if you use location information within your app: move – both fast and slowly. Go to locations with patchy network and GPS coverage to see how your app behaves.
- **Multimedia:** support for audio, video playback and recording facilities can differ dramatically between devices and their respective emulators.
- **Internet connectivity:** establishing an internet connection can take an incredible amount of time. Connection delay and bandwidth depend on the network, its current strength and the number of simultaneous connections.

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

## Remote Control

If you have physical devices to hand, use them to test your application. However when you don't, or if you need to test your application on other networks, especially abroad and for other locales, then one of the 'remote device services' might help you. For instance they can help extend the breadth and depth of your testing at little or no cost.

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

You can even create a private repository of remote devices, e.g. by hosting them in remote offices and locations. Beware of privacy and confidentiality when using shared devices.

## GUI Test Automation

GUI test automation is one of the elixirs of the testing industry, many have tried but few have succeeded in creating useful and viable GUI test automation for mobile applications.

Several commercial companies tried to provide automated testing 'solutions'; with one exception these have been mothballed. Tampere University in Finland have had some success creating automated tests for various mobile platforms, including Android and Nokia's Series 60 phones. See [tema.cs.tut.fi](http://tema.cs.tut.fi) for more information on their work.

## Beware of specifics

Platforms, networks, devices, and even firmware, are all specific. Any could cause problems for your applications. Test these manually first, provided you have the time and budget to get fast and early feedback.

## Crowd-sourcing

There are billions of users with mobile phones across the world. Some of them are professional software testers, and of these, some work for professional out-sourced testing service companies such as uTest and mob4hire. They can test your application quickly and relatively inexpensively, compared to maintaining a larger dedicated software testing team.

These services can augment your other testing, we don't recommend using them as your only formal testing. To get good

results you will need to devote some of your time and effort to defining the tests you want them to run, and to working with the company to review the results, etc.

## Web-based content and applications

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

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

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

## Next steps

There's more material available on testing your mobile applications at [tr.im/mobtest](http://tr.im/mobtest)

1) [googletesting.blogspot.com](http://googletesting.blogspot.com)

2) [googletesting.blogspot.com/2009/05/survival-techniques-for-web-app.html](http://googletesting.blogspot.com/2009/05/survival-techniques-for-web-app.html)

# Mobile Developer's Guide

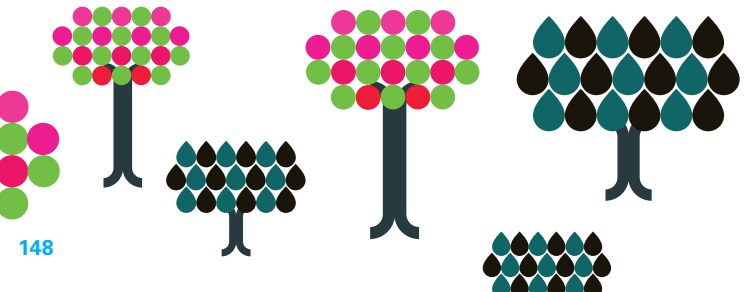
## Now what - which Environment Should I Use?

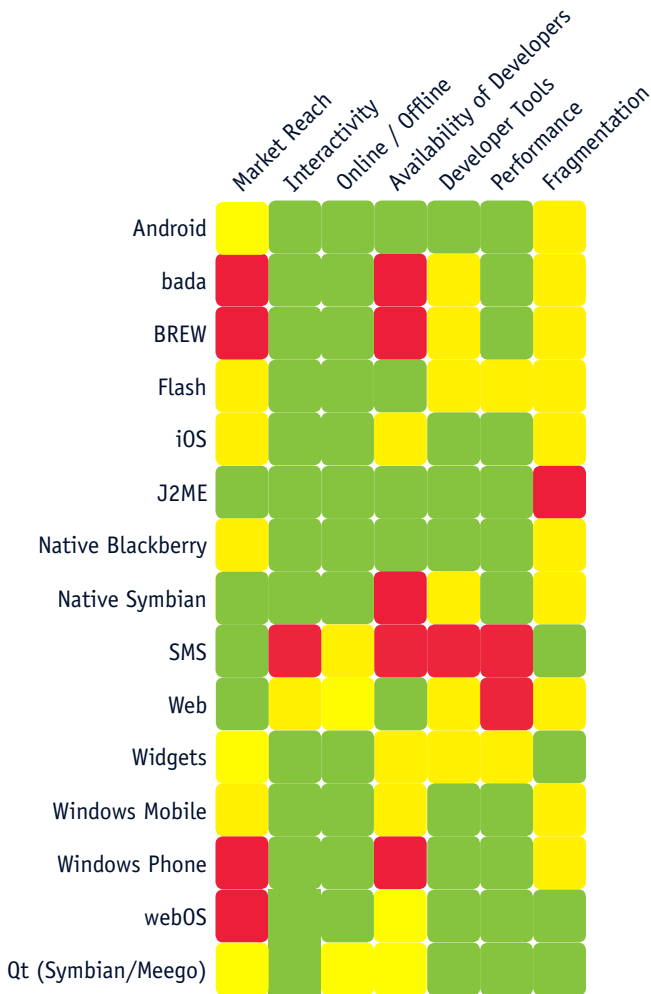
The short answer: it depends.

The longer answer: think about your target users, about their needs, about their devices and their dataplans. Then consider your vision and the requirements for your application. Remember that you are not necessarily restricted to a single application environment.

A pragmatic approach is to start with the environment that you are most comfortable with or which you expect to be the preferred platform of your target group. In a next step you can then move on to other environments to increase the market reach. Sometimes it also makes sense to combine different environments, for example by providing a mobile website for your casual users, a native smartphone application for your power users in certain countries and a J2ME app for other regions.

The following table provides a very rough overview of the strengths and limitations of each application environment without taking into account the big regional differences. **Green** indicates good coverage or support, **yellow** for limited and **red** for bad coverage of the respective topic:



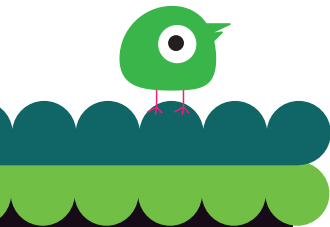


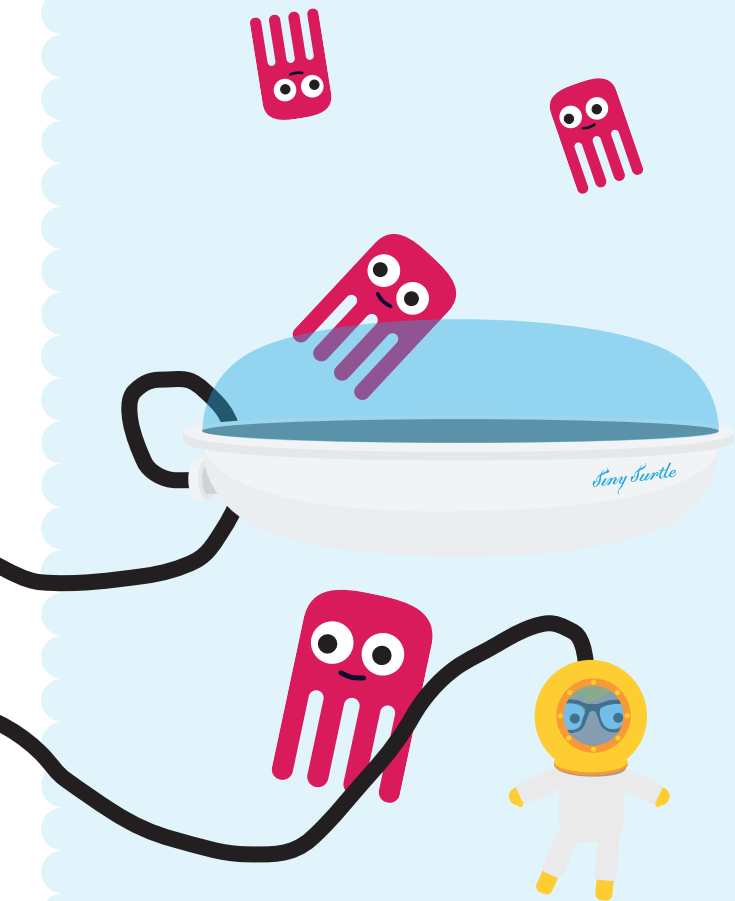
You can also compare application development with web development in a more general manner:

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

Web development beats application development in all but one crucial area: monetization. Internet services are mostly free and the only reliable revenue 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 7th release of our Mobile Developer's Guide. We hope you've enjoyed reading it and that we helped you to clarify your options. Don't be put off by the difficulties in entering the mobile arena — once you're in the water, you can and will swim.

Would you like to contribute to this guide or sponsor upcoming editions? Please send your feedback to [developers@enough.de](mailto:developers@enough.de)



# Mobile Developer's Guide

## About the Authors

### Robert Virkus / Enough Software

Robert is working in the mobile space since 1998. He experienced Java fragmentation first hand by developing and porting a mobile client on the Siemens SL42i, the first mass market phone with an embedded Java VM. With this experience he launched the Open Source J2ME Polish project in 2004 that helps developers to overcome device fragmentation barriers. He is the founder and CEO of Enough Software, the company behind J2ME Polish and many mobile apps.

[www.enough.de](http://www.enough.de)   [www.j2mepolish.org](http://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](http://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](http://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](http://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](http://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](http://www.openintents.org)

## Michel Shuqair / AppValley

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

[www.appvalley.nl](http://www.appvalley.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 and iPad developer for Enough Software since October 2008.

[www.enough.de](http://www.enough.de)   [www.alexrepty.com](http://www.alexrepty.com)

## Benno Bartels / InsertEFFECT

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

## Marco Tabor / Enough Software

Marco is responsible for PR, sales and much more at Enough Software. He coordinates this project as well taking responsibility for finding sponsors and merging the input provided by the mobile community. [www.enough.de](http://www.enough.de)

## Alex Jonsson / MoSync

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

## Richard Bloor / Sherpa Consulting Ltd

Richard has been writing about mobile applications development since 2000. He contributes to popular websites, such as AllAboutSymbian.com, and assists companies in creating resources for developers. Richard brings a strong technical background to his work, having managed development and testing on a number of major IT projects, including the Land Information NZ integrated land ownership and survey system. When not writing about mobile development, Richard can be found regenerating the native bush on his property north of Wellington.

## Jens Weller / Code Node

Jens started his career at Vodafone in 2002 as an apprentice, and joined the Vodafone test & innovation center in 2005 as a software engineer. In 2007 he founded his own company Code Node Ltd. Since then he works in the industry as a specialist for C++ , Qt and bada. He likes to dance Salsa in his free time.  
[www.codenode.de](http://www.codenode.de)

## Julian Harty / eBay

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

## André Schmidt / Enough Software

André is developing mobile applications since 2001. He joined Enough Software in 2007 where he heads the development of Open Source products for mobile developers and mobile applications of any kind. He is mainly developing for J2ME, Android and Blackberry. [www.enough.de](http://www.enough.de)

## Michael Koch / Enough Software

Michael joined the development team at Enough Software in 2005. He has not only headed the development of numerous mobile projects (mainly for Windows Mobile and Blackberry), but is also an expert on server technology. And of course he is an open source enthusiast, just like everybody at Enough Software. [www.enough.de](http://www.enough.de)

## Gary Johnson / Hyland Software, Inc.

Gary has been working as a software developer for Hyland Software, Inc. since 2005. He works primarily in Silverlight and WPF, and has a strong passion for UX and mobile development. As a hobbyist, he is heavily involved in Windows Phone 7 development. [www.hyland.com](http://www.hyland.com)

## Oliver Graf / Enough Software

Oliver is coding software for several platforms since 2000. He is working as a multi-platform developer for Enough Software and writes about mobile development for several magazines. Oliver was among the first registered developers for bada. As one of the Samsung developer advocates, he connects developers with Samsung (and vice-versa) to improve the bada ecosystem. [www.enough.de](http://www.enough.de) [www.dm-graf.de](http://www.dm-graf.de)

## Ovidiu Iliescu / Enough Software

After developing desktop and web-based applications for several years, Ovidiu decided mobile software is more to his liking. He's been doing J2ME and Blackberry development for Enough Software since 2009. He gets excited by anything related to efficient coding, algorithms and computer graphics. [www.enough.de](http://www.enough.de) [www.oviduiulescu.com](http://www.oviduiulescu.com)

## Gary Readfern-Gray / RNIB

Gary is an accessibility specialist working for the UK Royal National Institute of the Blind. Located in the innovation unit, he has a passion for the mobile space and particularly for enabling accessible app development across a range of platforms by engaging with developer communities. [www.rnib.org.uk](http://www.rnib.org.uk)



published by  
**Enough Software GmbH + Co. KG**  
Sögestrasse 70  
28195 Bremen  
Germany  
[www.enough.de](http://www.enough.de)

An initiative by:



[www.enough.de](http://www.enough.de)



[www.wipconnector.com](http://www.wipconnector.com)

Printing sponsor:

**NOKIA**  
Connecting People

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

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

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

»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*

