How To Develop iPhone Apps on Windows

<http://geeknizer.com/how-to-develop-iphone-apps-on-windows/>

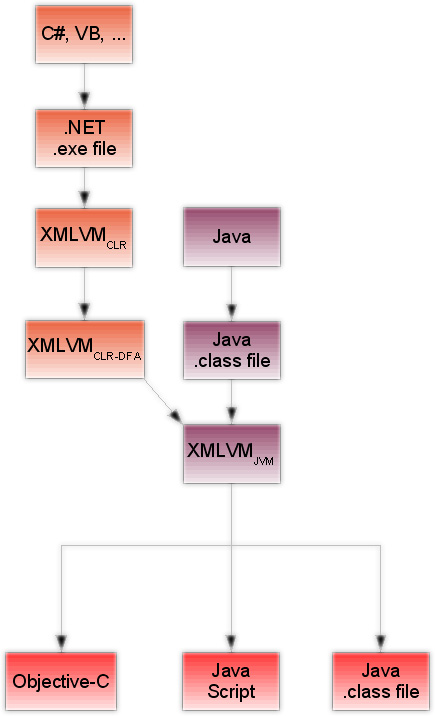
**Top 5 Ways:**

**Method 5.** *Toolchains***: -🡪 NO !**

There are several toolchains available (like [winChain](http://code.google.com/p/winchain/)) that actually lets you write and build iPhone applications on windows. There are several associated tutorials to build the Objective C code on Windows. But there is a problem, the apps hence developed will work on [Jailbroken iPhones](http://geeknizer.com/blog/tag/jailbreak) only. We’ve seen few hacks to get over that and make it to App Store, but as Apple keeps on updating SDKs, toolchains need regular updates. It’s a hassle to make it up all the time. That’s why this is the least of the recommended methods.

**Method 4.** Use other Languages instead of Objective-C **-🡪 NO !**

***(i)* Code in Java: For Java developers, there is a workaround:** [**XMLVM**](http://www.xmlvm.org/overview/).

XMLVM is an extensible cross-compiler toolchain which instead of cross-compiling on a source code level, XMLVM cross-compiles byte code instructions from Sun’s [Java](http://geeknizer.com/blog/tag/java) virtual machine and [Microsoft](http://geeknizer.com/blog/tag/microsoft)‘s Common Language Runtime (CLR). And the Result: The byte code instructions are easier to cross-compile and the difficult parsing of a high-level programming language is left to a regular compiler and you get to write apps in different language and then compile and convert to a different one. The diagram below shows an abstract idea: 

Without laying much stress on it, I`ll share my experience. The project is a great piece of Innovation but is still in it’s early phases. On one side, I was able to use their API and develop fairly well application (Simple game, Travel app), but when it comes to complex graphics, features, this method looked pretty immature. However, over time this should change and we could see the project doing almost everything the original SDK does. And yes, you can test your apps on the Java based simulator and deploy on [jailbroken](http://geeknizer.com/blog/tag/jailbreak) iPhone.

There are several other frameworks (like  [Appcelerator](http://www.appcelerator.com/" \t "_blank)`s [Titanium](http://www.appcelerator.com/products/download/)) that let you code iPhone apps in Java, but the limitations are similar thought they are all worth giving a look for most day-to-day apps.

**Update: *(ii):* Code in C/C++ -🡪 DragonFireSDK --- Good !**

DragonFireSDK**:** Say no to Objective C, say no to forced-Mac and yes to C/C++, Windows. This founds the base for [DragonFireSDK](http://www.dragonfiresdk.com/) that uses Microsoft Visual C++ to develop, test iPhone apps.

Apps, Games created with DragonFireSDK can be completely written and debugged in Windows and are also fully compliant for distribution and sales at the [Apple](http://geeknizer.com/tag/apple) iPhone [App Store](http://geeknizer.com/tag/app-store).

There is a [quick Starter Guide](http://www.dragonfiresdk.com/getstarted.htm) available that help you kick start writing your first iPhone app and run it inside the emulator that ships with it. The API is quiet simple to use and is available [here](http://www.dragonfiresdk.com/help/DragonFireSDKHelp.html). One of the Apps: Un Stacker developed using this SDK is already available on App Store [[link](http://itunes.apple.com/WebObjects/MZStore.woa/wa/browserRedirect?url=itms%253A%252F%252Fitunes.apple.com%252FWebObjects%252FMZStore.woa%252Fwa%252FviewSoftware%253Fid%253D337731072%2526cc%253Dus%2526mt%253D8)]. In addition, [5 Sample Apps](http://www.dragonfiresdk.com/sampleapps/sampleapp.htm) are demonstrated and explained with code.

**Method 3.** Hackintosh: **-🡪 Good !**

This is one of the effective ways of doing it: Install Mac on PC and then run the Native iPhone SDK. This is already a popular practice among [OSx86](http://geeknizer.com/blog/tag/osx86) communities. The only limitation is that it could get tricky and time consuming for the newbees. You can refer to our Hackintosh Guides:

* [How to Install Snow Leopard on PC Dual Boot](http://geeknizer.com/blog/how-to-install-mac-os-x-snow-leopard-on-pc-dual-boot)
* [Install Mac OS X Leopard on PC using Rebel EFI](http://geeknizer.com/blog/install-mac-os-x-leopard-osx86-on-pc-dual-boot-windows-7-vista-in-9-easy-steps)
* [Install Snow Leopard on PC using USB Easily](http://geeknizer.com/blog/install-snow-leopard-on-pc-easy) [Retail]
* [How to Install SnowOSX Universal](http://geeknizer.com/blog/how-to-install-snowosx-universal)
* [Install Snow Leopard on **VMWARE** Windows, Linux](http://geeknizer.com/how-to-install-snow-leopard-vmware-workstation-windows)
* [Install Snow Leopard on **VirtualBox**](http://geeknizer.com/install-snow-leopard-virtualbox)

**Method 2:** Cross compilation of Adobe apps: **Flash CS 5 🡪 Good !**

You can write your apps in Flash ActionScript 2, ActionScript 3 or Adobe AIR, Flex and then cross compile it to [ARM](http://geeknizer.com/blog/tag/arm) binary that is executable on iPhone. This can be done installing [Project Sprouts](http://github.com/lukebayes/project-sprouts) for which sample Flex [applications source is available here](http://onflash.org/ted/2009/10/source-to-4-flash-iphone-apps.php).

Here is a video on how this is done:  


**Method 1.** Flash CS 5: This is in fact the most effective and easiest way to make it to App store doing all the “legal stuff”.

Flash CS 5 introduces new Feature that let’s you develop [iPhone](http://geeknizer.com/blog/tag/iphone) native [applications](http://geeknizer.com/blog/tag/apps) just like you develop Adobe AIR apps. Recently, Adobe announced support for [Multitouch, Accelerometer, GPS support in Flash 10.1 for phones](http://geeknizer.com/blog/adobe-flash-10-1-mobile-brings-full-flash-multitouch). CS5 adds new APIs that lets developers leverage these modern Phone features and hence develop application not just for iPhone but for all Phones that support Flash.

So the Horizon is quiet big, and CS5 with ActionScript could find a great way to develop applications on iPhone. If you are familiar with a scripting language, say, Javascript, learning ActionScript is as easy as an ApplePie.

Already, App Store has a number of Apps built based on [Flash](http://geeknizer.com/blog/tag/flash) (I believe they are using Crosscompilation): you can checkout few full blown apps [here](http://labs.adobe.com/technologies/flashcs5/appsfor_iphone/#examples).

Only bad part of this method is, it’s still unavailable. However This is what [Official adobe site](http://labs.adobe.com/technologies/flashcs5/) has to say about it:

**When will the Flash Professional CS5 beta be available for download?**

The beta will be available for download from Adobe Labs before the end of 2009.

You can develop, build and test in native Flash debugger, however, soon we should see a simulator for mobile devices, especially for the iPhone.

I needed more clarity whether the final step, signing of Apps would be possible on Windows. I contacted Adobe on this. Alexander MacDonald said “Once you have created your content it is compiled into an iphone executable, then signed by our ADT tool and then zipped to create an ipa—the only thing you need from apple is your developer certificate. The crypto algorithms used by Apple to sign iPhone apps are all industry standard ones which anyone can implement on any platform they wish,” which in the case of Flash CS5, also includes Windows.

The app hence created can be installed to iPhone via iTunes for testing to substitute absence of simulator for the mean time. So all in all, everything would be legal, and will work great.

However, here is the demo of how applications will be created in Flash CS5:

[Get the lastest version of the Flash Player](http://www.macromedia.com/go/getflashplayer) to see This Video

Today, it doesn’t support everything SDK supports, but it would soon do. With Flash opening up a way to iPhone development, Adobe is adding millions of new developers to the iPhone App store contributors.

We write latest and greatest in [Tech Guides](http://geeknizer.com/tag/guide), [Hackintosh](http://geeknizer.com/tag/hackintosh), [Apple](http://geeknizer.com/tag/apple), [iPhone](http://geeknizer.com/tag/iphone), [Tablets](http://geeknizer.com/tag/tablet), [Android](http://geeknizer.com/tag/android),  [Open Source](http://geeknizer.com/tag/open-source), Latest in Tech, subscribe to us[**@geeknizer** on Twitter](http://twitter.com/geeknizer) OR on [Facebook Fanpage](https://www.facebook.com/geeknizer)

Read more: <http://geeknizer.com/how-to-develop-iphone-apps-on-windows/#ixzz2E7jbJQgj>

## [Comparing Titanium and PhoneGap](http://developer.appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap.html)

<http://developer.appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap.html>

Published on May 12, 2012 by [Kevin Whinnery](http://developer.appcelerator.com/redirect/user/7f26f56eec87ba129abbc01706185126) (Platform Evangelist)

A common question I get asked at developer events and conferences is how [Titanium](http://www.appcelerator.com/download) compares to [PhoneGap](http://www.phonegap.com). I thought I would take some time to explain how each technology works at a high level, and assess how the two technologies compare to one another.

From 10,000 feet, PhoneGap and Titanium appear to be similar. They both provide tools for cross-platform mobile development. Both also require the use of JavaScript and web technologies in some capacity. Both Titanium and PhoneGap are open source software with permissive licenses (the Titanium Mobile SDK is released under the [Apache 2.0 license](http://www.apache.org/licenses/LICENSE-2.0) – PhoneGap, which might also be called a “distro” of the [Apache Software Foundation-governed project “Cordova”](http://incubator.apache.org/cordova/), is similarly licensed).

But that’s really where the similarities end. While both technologies exist to enable cross-platform mobile development, the philosophies and approaches to solving this problem have very little in common. Also, the business goals driving each project from the perspective of the sponsoring companies ([Adobe](http://www.adobe.com) for PhoneGap and [Appcelerator](http://www.appcelerator.com) for Titanium) are very different. I will attempt, from my perspective, to describe these technical, philosophical, and business model differences in some detail in the text to follow.

Also, if you weren’t already aware, I am a long time Appcelerator contributor and employee. That said, I have worked hard to keep my technical and philosophical assessments based in technical fact and the explicitly expressed goals of the teams involved. If you feel I have made any points that are factually incorrect or misleading in some way, please let me know in the comments and I will update this post as appropriate.

I will first describe at a high level how both technologies work. I will also describe how both technologies are extended with additional native functionality. For each technology, I will also summarize the key strengths and weaknesses with their chosen approach to cross-platform. The technical differences will quickly become obvious, but after these overviews and comparisons, I will also describe what I feel are the philosophical and strategic differences between the platforms and where they are going.

Let’s start by exploring PhoneGap and how it works.

### What is PhoneGap Trying To Accomplish?

The purpose of PhoneGap is to allow HTML-based web applications to be deployed and installed as native applications. PhoneGap web applications are wrapped in a native application shell, and can be installed via the native app stores for multiple platforms. Additionally, PhoneGap strives to provide a common native API set which is typically unavailable to web applications, such as basic camera access, device contacts, and sensors not already exposed in the browser.

At a higher level, PhoneGap might be considered the vanguard of the emerging [W3C Device API standards](http://www.w3.org/2009/dap/), as they attempt to bring that future to web developers in the present. Today, no platform makes web applications first class citizens, though [Mozilla’s promising Boot To Gecko platform](http://www.mozilla.org/en-US/b2g/) has a chance to change that. Microsoft is also making interesting strides for Windows 8 with regard to first-class API access to web applications. But the goal of PhoneGap is to seize a subset of these rights for web applications today.

### End User Workflow, Tooling and Interface for PhoneGap

To develop PhoneGap applications, developers will create HTML, CSS, and JavaScript files in a local directory, much like developing a static website. In fact, some PhoneGap developers cite as a bonus of the tool that they can develop in a desktop web browser most of the time, without needing the native toolchain at all.

To run a PhoneGap application on a native emulator/simulator, developers will generate a project for each of the native platforms they wish to support, configure that project’s “web root” directory in Xcode, Eclipse, or whatever native toolchain is needed, and then run the project using that tool. The precise steps are [outlined in their getting started guides, per platform](http://phonegap.com/start). Often, symbolic links are used to route the “www” folder across multiple native projects to a common directory location.

Installing a native-wrapped PhoneGap application to a device requires a similar workflow. However, to augment that process and alleviate the need to have native SDKs installed locally, Nitobi (recently acquired by Adobe) had created a service called [PhoneGap Build](https://build.phonegap.com/), which will generate installable applications in the cloud. Functionality to support PhoneGap build deployment has recently been integrated into Adobe’s Dreamweaver tool.

The tools used with PhoneGap are the standard tools of web development, such as Firebug, Web Inspector, and your text editor of choice. There is also an emerging tool for remote debugging [known as Weinre](http://people.apache.org/~pmuellr/weinre/) that is becoming more commonly used. Overall, the fact that you are developing a native application at all is mostly abstract during the development process.

### How PhoneGap Works

As we mentioned previously, a PhoneGap application is a “native-wrapped” web application. Let’s explore how the web application is “wrapped”.

Many native mobile development SDKs provide a web browser widget (a “web view”) as a part of their UI framework ([iOS](http://developer.apple.com/library/ios/" \l "documentation/uikit/reference/UIWebView_Class/" \t "_blank) and [Android](http://developer.android.com/reference/android/webkit/WebView.html), for example). In purely native applications, web view controls are used to display HTML content either from a remote server, or local HTML packaged along with the native application in some way. The native “wrapper” application generated by PhoneGap loads the end developer’s HTML pages into one of these web view controls, and displays the resulting HTML as the UI when the application is launched.

If JavaScript files are included in a page loaded by a web view, this code is evaluated on the page as normal. However, the native application which creates the web view is able to (in different ways, depending on the platform) asynchronously communicate with JavaScript code running inside of the web view. This technology is usually referred to as “the bridge” in the context of PhoneGap architecture – the “bridge” means something slightly different in Titanium, as we will see later.

PhoneGap takes advantage of this to create a JavaScript API inside a web view which is able to send messages to and receive messages from native code in the wrapper application asynchronously. The way the bridge layer is implemented is different per platform, but on iOS, when you call for [a list of contacts](http://docs.phonegap.com/en/1.7.0/cordova_contacts_contacts.md.html#contacts.find), your native method invocation goes into a [queue of requests to be sent over the bridge](https://github.com/apache/incubator-cordova-ios/blob/master/CordovaLib/javascript/cordova.ios.js#L994). PhoneGap will then create an iframe which loads a URI scheme (“gap://”) that the native app is configured to handle, at which point [all the queued commands will be executed](https://github.com/apache/incubator-cordova-ios/blob/master/CordovaLib/Classes/CDVViewController.m#L461). Communication back into the web view is done by evaluating a string of JavaScript in the context of the web view from native code.

There is much more to PhoneGap than that, but the messaging from web view to native code via the bridge implementation is the key piece of technology which allows local web applications to call native code.

### Extending PhoneGap

Writing native extensions for PhoneGap requires that you:

1. Write a JavaScript interface for your extension which will use PhoneGap’s API to queue up messages to be sent to native code.
2. Register your extension with the native project in some way – on iOS this is done in the [Cordova.plist file](http://wiki.phonegap.com/w/page/36753496/How%20to%20Create%20a%20PhoneGap%20Plugin%20for%20iOS#EditCordovaplist).
3. Write native code that PhoneGap will route requests to from the web view, and implement any native code needed

Basically, developers can participate in the same asynchronous messaging system which powers the core PhoneGap native APIs.

### Strengths of the PhoneGap Approach

In my estimation, PhoneGap’s primary architectural strength is that it is so small and simple. It does what it does, and it does that well. The PhoneGap team has intentionally implemented only the lowest common denominator of native APIs for the web browser-based app. Because the native API set is so small, it has been relatively easy to port PhoneGap to many different environments. Basically any native platform that supports a web view or web runtime can be a PhoneGap platform.

Non-visual native extensions in PhoneGap are also very simple. The requirements for registering native code to receive messages from the web view are very modest. Simple native extensions can be developed rapidly. This plug-in architecture was also well executed in my opinion.

There is also strength in the fact that native APIs and native app development are almost completely abstract to the end developer. Anyone who can write HTML, CSS, and even a small bit of JavaScript can wrap up a web page in a native app and distribute it as such. The barrier to entry in using PhoneGap to package web pages as native apps is extremely low.

### Weaknesses of the PhoneGap Approach

The quality of the user interface in a PhoneGap application will vary based on the quality of the web view and rendering engine on the platform. The Webkit-based rendering engine on iOS is strong, and provides the best performance. The Android web view is functional, [but has some notable limitations](http://simonmacdonald.blogspot.com/2012/02/android-issues-all-phonegap-developers.html). On other platforms, the web view performance can be suspect depending on the OS version.

There are also the standard cross-browser issues web developers have always had to deal with. UIs will need to employ progressive enhancement, media queries, and that entire bag of tricks to remain usable on multiple platforms. It helps that many mobile platforms are adopting Webkit, but there are [still significant differences even in Webkit based environments](http://westcoastlogic.com/slides/debug-mobile/#/17).

Mobile browsers are getting better all the time, which will help mitigate those problems. But approaching native-quality UI performance in the browser is a non-trivial task – [Sencha](http://www.sencha.com) employs a large team of web programming experts dedicated full-time to solving this problem. Even so, on most platforms, in most browsers today, reaching native-quality UI performance and responsiveness is simply not possible, even with a framework as advanced as Sencha Touch. Is the browser already “good enough” though? It depends on your requirements and sensibilities, but it is unquestionably less good than native UI. Sometimes much worse, depending on the browser.

PhoneGap also cannot be extended with native user interface. The end developer’s application its self lives inside a web view, and user interface is rendered in HTML. One can message to native code and create native UI that goes on, over, above, or adjacent to the web view, but it’s difficult or impossible to integrate a dynamic, HTML DOM-based UI with native UI components. [Appcelerator would know](https://github.com/appcelerator/titanium_mobile/tree/0_8_X) – we tried to associate native UI with DOM elements early on, and needed to scrap that effort as the results were unpredictable and of insufficient quality.

There is also the other edge of the “lowest common denominator” sword. Very few native APIs are exposed to PhoneGap applications by default, which makes platform integration limited. There are a [variety of plug-ins that exist to plug some of these holes](https://github.com/phonegap/phonegap-plugins), but in my personal experience they have varied in quality and maintenance. This could very well continue to improve over time though – there is a strong community around PhoneGap.

We’ll dive more into the philosophical aspects of PhoneGap soon, but let’s explore these same technical areas for Titanium first.

### What is Titanium Trying to Accomplish?

The goal of Titanium Mobile is to provide a high level, cross-platform JavaScript runtime and API for mobile development (today we support iOS, Android, and the browser, with BlackBerry 10 and Windows Phone coming soon and eventually, respectively). Titanium actually has more in common with MacRuby/Hot Cocoa, PHP, or [node.js](http://nodejs.org) than it does with PhoneGap, Adobe AIR, Corona, or Rhomobile. Titanium is built on two assertions about mobile development:

* There is a core of mobile development APIs which can be normalized across platforms. These areas should be targeted for code reuse.
* There are platform-specific APIs, UI conventions, and features which developers should incorporate when developing for that platform. Platform-specific code should exist for these use cases to provide the best possible experience.

So for those reasons, Titanium is not an attempt at “write once, run everywhere”. We think there are great, user-experience enhancing features across multiple platforms that developers should be using. We think that native apps should, where appropriate, take advantage of familiar, high-performance native UI widgets. However, we think it is unnecessary that native developers need to learn platform-specific APIs to draw a rectangle, or make an HTTP request.

Titanium is an attempt to achieve code reuse with a unified JavaScript API, with platform-specific features and native performance to meet user expectations. When you write a Titanium application, you are writing a native application in JavaScript. Titanium should be considered a framework for writing native apps, versus an abstraction from the actual platform you are targeting.

### End User Workflow, Tooling, and Interface for Titanium

To [develop native applications with Titanium](http://docs.appcelerator.com/titanium/2.0/index.html#!/guide/Quick_Start), the developer is required to install the native tool chains for iOS and Android. After those tools are installed, however, the developer usually only interacts with the Titanium SDK’s scripting interface (today Python based). This is done either directly through the command line or (more commonly) through Titanium Studio, our Eclipse-based IDE.

Using the Titanium tool set, you will generate an application project directory which contains a configuration file, localization files, and a directory to contain the images, assets, and JavaScript source you will be writing to power your application. You will not, by default, be editing HTML and CSS files, unless you intend to create a hybrid-type application which contains [both native and HTML-based UI](http://docs.appcelerator.com/titanium/2.0/index.html#!/guide/Communication_Between_WebViews_and_Titanium). Titanium applications can and often do employ a “hybrid” (native and web) UI, like Facebook’s native application for instance. In this way, one could actually implement PhoneGap with Titanium, but that’s out of scope for this discussion.

Using this toolchain, your application is run using the actual em/simulators for the platforms you’re targeting. Titanium Studio also provides step-through debugging, code completion, and other IDE-level features.

Installing to a device for testing is also typically done using our build system. In Studio we provide a wizard interface to configure any code-signing dependencies, and then handle the deployment of your application to a connected device. You can also use the native toolchains to deploy or package your applications, if that is your preference.

When it comes time to ship your application to the stores, our build system will handle the creation of the final application packages for you. This is done locally on the developer’s machine using the native toolchains. The upload process will be the same as it is for native-only developers.

While developing a Titanium application, the underlying tool chains are mostly abstract. They must be present for development, but the end developer is rarely required to use them directly. The fact that native apps are being developed, however, is not abstract. User interfaces are created with cross-platform AND platform-specific components, and your applications should be dealing with things like background services, local notifications, app badges, configuration, activities/intents (on Android)… all things that are exposed via the Titanium JavaScript API.

### How Titanium Works

There’s quite a bit happening behind the scenes in a Titanium application. But basically, at runtime, your application consists of three major components – your JavaScript source code (inlined into a Java or Objective-C file and compiled as an encoded string), the platform-specific implementation of the Titanium API in the native programming language, and a JavaScript interpreter that will be used to evaluate your code at runtime ([V8 (default)](https://github.com/appcelerator/v8_titanium) or [Rhino](http://www.mozilla.org/rhino/) for Android, or [JavaScriptCore](https://github.com/appcelerator/tijscore) for iOS). Except in the browser, of course, where the built-in JavaScript engine will be used.

When your application is launched, a JavaScript execution environment is created in native code, and your application source code is evaluated. Injected into the JavaScript runtime environment of your application is [what we call “proxy” objects](http://docs.appcelerator.com/titanium/2.0/index.html#!/guide/iOS_Module_Development_Guide-section-29004946_iOSModuleDevelopmentGuide-Step2%3ABasicModuleArchitecture) – basically, a JavaScript object which has a paired object in native code. Colloquially we will often refer to “JavaScript land” and “native land” in a Titanium application, as they are kind of parallel universes to one another. The proxy object exists both in JavaScript land and native land, and serves as the “bridge” between the two.

In your JavaScript code, when you call a function on the global Titanium or Ti object, such as var b = Ti.UI.createButton({title:'Poke Me'});, that will invoke a native method that will create a native UI object, and create a “proxy” object (b) which exposes properties and methods on the underlying native UI object to JavaScript.

UI components (view proxies) can be arranged [hierarchically to create complex user interfaces](http://docs.appcelerator.com/titanium/2.0/index.html#!/guide/User_Interface_Fundamentals). Proxy objects which represent an interface to non-visual APIs (like filesystem I/O or database access) execute in native code, and synchronously (or asynchronously for APIs like network access) return a result to JavaScript.

Hopefully this helps directly address two common misconceptions about Titanium – at no point does Titanium require the use of a web view component. The developer can create a web view as a native UI widget, but the web view is not used to evaluate Titanium source code. Nor is JavaScript code cross-compiled to Objective-C or Java in Titanium. Your JavaScript source is evaluated at runtime.

### Extending Titanium

Titanium is extensible with both non-visual and UI capabilities in native code. By implementing a Proxy and/or View Proxy interface in native code, developers can create new native functionality for Titanium applications exposed in JavaScript. We expose the same interface we use to create Titanium’s own internal interface to module developers both on [iOS](http://docs.appcelerator.com/titanium/2.0/index.html#!/guide/iOS_Module_Development_Guide) and [Android](http://docs.appcelerator.com/titanium/2.0/index.html#!/guide/Android_Module_Development_Guide).

### Strengths of the Titanium Approach

Since the goal of Titanium is to provide a higher level API for native mobile development across platforms, you will get access to a wide array of native features and functionality out of the box, from user interface components to socket interfaces to notification system integration. The goal of Titanium is to reduce the functionality gap between Titanium and pure native apps to something approaching zero. We’re likely to never support an entire platform’s API out of the box, but we want to cover 90% of the most common use cases and provide a platform where the other 10% can be added by people that need it.

Since Titanium can be extended with visual components that plug into the same view hierarchy as the rest of the application, you’re able to (ultimately) implement any user interface that is possible on the underlying native platform. Need a TableView to scroll at 60fps with special native code? You can do that. Want to seamlessly integrate an [OpenGL drawing surface for a game, and keep the logic for the run loop in JavaScript](http://code.google.com/p/quicktigame2d/)? You can do that. You can integrate these UI extensions directly into the rest of your application built with the core Titanium APIs.

The look and feel of a Titanium application, when using common UI widgets, is also a strength of the platform. There is no visual emulation going on (either through the application of CSS, or rendering of UI widgets using OpenGL or Flash). When you create a [NavigationGroup](http://docs.appcelerator.com/titanium/2.0/index.html#!/api/Titanium.UI.iPhone.NavigationGroup), it is backed by an actual UINavigationController on iOS. The animations and behavior match what a native app user will expect, because you’re using the same UI control.

Since Titanium provides a high level native programming API in JavaScript, the barrier to entry for native programming is significantly reduced for anyone who has used an ECMAScript based language (which is a lot of developers). [Atwood’s Law is alive and well through Titanium](http://www.codinghorror.com/blog/2007/07/the-principle-of-least-power.html).

### Weaknesses of the Titanium Approach

The scope of the Titanium API makes the addition of new platforms difficult – implementing the Titanium API on a new native platform is a massive undertaking. For that reason, the Titanium platform is only available on what have been deemed the most critical mobile platforms at present: iOS, Android, and the web.

Our mobile web browser support is not yet of GA quality – we are continuing to work on the performance and feel of our UI widget set, as well as rounding out the implementation of our core Titanium APIs.

Because the layer of abstraction provided by Titanium is large, sub-optimal API implementations remain in our own internal framework. Some user interface components do not yet perform as well as their native counterparts under some circumstances, such as very large table views with highly customized layouts. Optimizing our core user interface components remains the primary engineering task for our team. As we fix bugs and hardware improves, we are seeing this become less of an issue. We also find that information architecture, especially for large data sets, needs to be applied in many cases.

Also owing to the ambitiousness of the Titanium platform, extending Titanium is non-trivial. A good working knowledge of Titanium’s architecture and the environment is necessary to effectively integrate a new native control or API. The [developer experience, API docs, and high level guides for module developers](http://docs.appcelerator.com/titanium/2.0/index.html#!/guide/Extending_Titanium_Mobile) were improved a lot with our latest 2.0 release, but remain an area of focus for us.

### Philosophical Differences

By now, I would hope that the technical differences between PhoneGap and Titanium are pretty clear. But beyond those differences, the goals and direction of each project are different as well. The stated goal of the PhoneGap project is to, eventually, [cease to exist](http://phonegap.com/2012/05/09/phonegap-beliefs-goals-and-philosophy/). As stated earlier, PhoneGap is intended to be the leading implementation of emerging browser standards around device APIs. In theory, once browser vendors implement the features of PhoneGap, the platform will no longer be necessary. PhoneGap its self isn’t intended to be a platform – it’s a shim to add native app-like functionality to web applications. The web is intended to be the platform.

PhoneGap’s new sponsoring organization, Adobe, is also very much interested in the advancement of the web as a platform. In recent months, Adobe has been aggressively building out tools to enable the development of HTML 5/CSS 3 web applications. It seems obvious to me (and many others) that Adobe sees a diminishing role for Flash as standard web technologies evolve.

At it’s core, Adobe is a tools business. Platforms are a channel through which Adobe can sell tools. Once, that platform was Flash. Now, that platform is the web browser (in addition to Flash). I don’t know precisely how PhoneGap factors into Adobe’s product roadmap, but in a lot of ways it serves a similar purpose as Flash. PhoneGap is an attempt to create an abstract runtime environment to enable cross-platform deployment.

If Adobe can sell tools to develop for the web, and the web can be used to develop more types of applications, then that’s a clear win for Adobe. Which is fine, by the way – nothing wrong with selling tools.

It’s worth noting, however, that Adobe is not the governing body of the Cordova project, on which PhoneGap is now based. That project is owned and governed by the Apache Software Foundation. It remains to be seen what the interplay is going to be between the two projects, but my gut instinct is that they won’t diverge much. I think their goals will remain philosophically aligned.

Appcelerator is also interested in and supportive of the advancement of the web as a platform. Everyone wins when the web gets stronger as an application platform. The difference is that we view the web as one great platform among others, with a unique character and set of strengths and weaknesses. We don’t expect the web to become the only mobile application platform. We think that platforms like iOS, Android, BlackBerry, Windows Phone, and the like will continue to be influential, and will provide great experiences for users. That choice and competition will be a good thing for consumers, but will remain a problem for developers.

What we expect to provide for developers through Titanium is a way to target the web and native platforms from a single codebase, while retaining the features, performance, and tight platform integration that the users of that platform expect. We expect to build an enduring platform for mobile client development, with services and tools to [speed up that process](http://www.appcelerator.com/products/appcelerator-cloud-services). We are not a tools company – we are a platform company, and our success will be linked to the success of developers on top of our platform. Over time, we hope to build an open source platform company in the spirit of [Red Hat](http://www.redhat.com/) and other giants in that space.

Which tool or approach is right for you? Like all things in software development, it depends. There are no silver bullets. But hopefully this description and comparison will help you make the right choice for your situation.

1. [May 15, 2012 at 12:18 pm](http://developer.appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap.html/comment-page-1#comment-94215)

Well balanced article:) i have used both platforms for quite some time  
and find myself using titanium for meatier projects due to it’s  
“native” like performance

1. Jack says:

[May 18, 2012 at 6:04 am](http://developer.appcelerator.com/blog/2012/05/comparing-titanium-and-phonegap.html/comment-page-1#comment-94766)

If you start using Titanium, you have only 2 choises in future:  
-rewrite everything on native;  
-change framework (who need 3MB hello world?).

If you start using PhoneGap you have no limits – just have to “pay” little for speed. jQuery Mobile, Dojo mobile, jqMobi…so on and more. You can make website and app in same time – welcome to jQuery mobile. App Size will be MUCH LESS then with appcelerator, speed – same, development – smarter (lets be honest – titanium coding approach not so good).

And you can see that – every real business company stay away from appcelerator

-🡪 If your aim is to build apps with the highest level of quality, performance, and native integration on your target platforms, then Titanium IS for you.

Mobileweb SDK RC will allow you, using the same Titanium API, to build native apps for android, ios, and blackberry, while simultaneously building HTML5 compliant web apps. So basically, that extensive reach that was reserved for web-based platforms is now within the reach of Titanium developers.

1. We’ve done since now two apps based on titanium. One of them used an hybrid approach. We stated that there’s clearly a price to pay for the hybrid solution(performance and fluency of the UI).

Our work is mainly digital apps and books for children. This mean using all multimedia capabilities, voice recording, images, sounds, animations , … in a very quick response UI.

Since I’ve followed this battle between mobile cross-platforms I’ve always stated that the native approach of titanium was the better solution.

After reading @Laszlo post and @Tony reply I’ve question about what are the graphics and gaming limits, above witch titanium is not the best solution, or have some significant drawbacks

????Titanium and MoSync.

1. had a good look at mosync vs appcelerator and have to say on first look went with MoSync. I am a C/C++ coder and it was mainly familiarity. I then went on the extend MoSync and found very simple c libraries and some common ones (sqlite for example) would just not compile. So for me the attractiveness of a c compiler for lost. As for the MoSync wormhole technology, it’s nice, but performance is rubbish and the resulting UI is nowhere near native.

As for the Vendor support and services. This is something the Appcelerator team could really learn from. Appcel JIRA is full of really old bugs or features requests that have no comment or timeline (take TIMOB-1309 for example). The Q&A section seems to be rarely visited by Appcel staff and the only support is incredibly expensive

[Mobile HTML5: PhoneGap vs Appcelerator Titanium](http://www.universalmind.com/mindshare/entry/mobile-html5-phonegap-vs-appcelerator-titanium)

And the winner is: PhoneGap! The reasons why are actually simple.

PhoneGap and Appcelerator Titanium are both very popular open-source JavaScript frameworks for packaging and deploying mobile applications. At Universal Mind, we have clients that leverage both and clearly both frameworks have countless successful implementations on numerous platforms. However, there are enough significant differences between the two products that it doesn’t really make that much sense to compare them directly. The fundamental difference is that PhoneGap is a web-based solution where Appcelerator Titanium is a pure JavaScript API that creates native code. At the core, they serve different functions for an organization.

At this stage we see many more PhoneGap implementations than Appcelerator Titanum for a few simple reasons. Appcelerator allows developers to utilize JavaScript to write a mobile application and compile down to native code for deployment. PhoneGap allows you to utilize HTML5 standards to write your mobile applications while also providing a JavaScript SDK to access native device capabilities. This clear differentiator is not one to be taken lightly. Simply put, if you want to re-use the same code that you deploy with your HTML5 enterprise desktop apps across other tablet, mobile and connected TV platforms, PhoneGap is the right choice.

PhoneGap is a web app that runs in a native web browser view. It lets you utilize HTML5, CSS, and JavaScript, as well as all of the frameworks, like jQuery Mobile, Sencha, etc. described above. Appcelerator is pure JavaScript that compiles to Native Code. PhoneGap supports more platforms but Appcelerator may give you better performance in specific instances.

I have dropped literally 30+ HTML5/CSS/JavaScript applications into PhoneGap, which run without issue on the desktop, and deployed them to Android, iOS, and Google TV without changing one line of code. PhoneGap clearly promotes the most re-use and provides the easiest transition from desktop to mobile.

Appcelerator is like writing a native application with JavaScript and the Titanium SDK. Thus, if your goal is to re-use your HTML5 code base across mobile, tablet, desktop and TV then PhoneGap has to be the natural choice. Appcelerator touts improved performance because they compile to native code but truthfully we don’t see many use cases where this is a differentiating factor. Organizations usually choose Native iOS or Native Android over a “Native JavaScript” option. PhoneGap also supports deployments to more device platforms than Titanium because it doesn’t need to compile to a native form. It is probably a more natural analysis to compare Native Platform development to Appcelerator.

PhoneGap:

* JavaScript API that provides access to Native Functions
* Supports HTML5/CSS3
* Supports Web Standards & Re-use Across Enterprise Apps
* Supports DOM based JavaScript Libraries/Frameworks
* Supports the most platforms

Appcelerator Titanium:

* JavaScript API that provides access to Native Functions
* Compiles to Native Code
* Could provide better performance.

***Cross-Platform Developer Tools 2012***

Cross-Platform Developer Tools 2012 is the seminal report on the landscape of 100+ cross-platform developer tools with an analysis of key vendors and the metrics of developer experience.  
Cross-Platform Tools 2012 is the first major report that analyses the complex cross-platform tools landscape, presents the key tools vendors and maps out the trends in one of the hottest markets in mobile. This report, which is supported by [webinos](http://www.webinos.org/), presents developer perceptions about a wide range of key issues, from incentives and deterrents to cross-platform tool use to the most-wanted tool features. Our research profiles 15 well-known cross-platform vendors and tools with regard to their technology, positioning and market traction.

Features in-depth profiles on Adobe (PhoneGap), Adobe (Air / Flex), Ansca Mobile (Corona), Appcelerator (Titanium), Seregon (DragonRad), IBM (Worklight), Ideaworks 3D Ltd (Marmalade), MoSync, Motorola Solutions (RhoMobile), Netbiscuits, RunRev (LiveCode), Qt (Nokia), Sencha, Unity, Xamarin (MonoTouch and Mono for Android)

This report includes:

* Mindshare of cross-platform tools – which are being adopted or abandoned?
* Breakdown of cross-platform technologies, languages and target platforms
* Market trends – how do cross-platform tools affect the industry?
* Developer perceptions – incentives and deterrents to tool use
* Competitive scoring – how do the top tools rank in terms of developer experience?
* Analysis of 15 major cross-platform tool vendors, with in-depth profiles

## [New report] Cross-Platform Developer Tools 2012

[We ‘re proud to announce the launch of Cross-Platform Tools 2012 - the free, industry-first report on cross-platform developer tools. You can [download a free copy here](http://goo.gl/npsnP). Cross-platform tools (CPTs) allow developers to create applications for multiple platforms with a small incremental cost. Their impact is both tactical in allowing developers to target more platforms, but also strategic in having the potential to disrupt the Apple/Google duopoly in mobile ecosystems.]

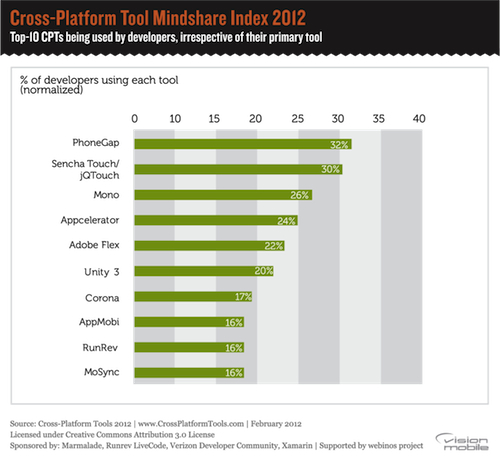
[](http://goo.gl/npsnP)

Our report is based on a 6-month project, comprising a large-scale online developer survey (nearly 2,500 respondents) combined with meticulous research, vendor interviews and analysis of this complex market of over 100 tools vendors. This report would not have been possible without the support of Marmalade, RunRev, Verizon Developer Communities, Xamarin and the many other companies behind this multi-sponsored project.

Cross-platform tools (CPTs) solve real challenges today; they allow developers to create applications for multiple platforms – usually mobile, but increasingly tablets or TV screens – from almost the same codebase or from within the same design tool. CPTs reduce the cost of platform fragmentation and allow developers to target new platforms at a small incremental cost. More importantly, cross-platform tools allow software companies targeting multiple platforms to reuse developer skills, share codebases, synchronise releases and reduce support costs.

### Early leaders in the cross-platform tools space

Our survey revealed that PhoneGap and Sencha lead in terms of mindshare, as they are currently used by 32% and 30% of cross-platform developers, irrespective of their primary tools. Completing the top-5 ranking of our Mindshare Index are Xamarin’s MonoTouch / Mono for Android, Appcelerator and Adobe (Flex). The second half of the top-10 CPTs in terms of current use are Unity, Corona, AppMobi, RunRev and MoSync.

[](http://goo.gl/npsnP)

PhoneGap (23%), Xamarin Mono (22%) and Unity (22%) are the tools most developers plan to adopt, irrespective of their primary tool. This market is in constant flux, with developers experimenting and trying out new tools – for example PhoneGap is a stepping stone to cross-platform development as it leads Mindshare, IntentShare, but also comes third in the tools being abandoned. The most widely used CPT accounts for just half of the Mindshare seen in the iOS and Android platforms in our Developer Economics 2011 [report](http://www.visionmobile.com/blog/2012/02/crossplatformtools/www.DeveloperEconomics.com).

### Cross-platform tools challenge the Apple/Google duopoly

The real impact of cross-platform tools is strategic. Just as the Apple/Google duopoly began to look impenetrable in 2011, a major disruption is flattening the playing field for competitors like Microsoft’s WP7, RIM’s BlackBerry OS and Samsung’s Bada: cross-platform tools are letting developers target multiple platforms with low incremental costs and high levels of code reuse.

2012 marks an inflexion point in the war of mobile ecosystems where the network effects built by Apple and Google are being challenged by an unsuspected new entrant. Cross-platform tools (CPTs) make it easier for example for an iPhone developer to reach Android and Windows Phone 7 users. CPTs dilute network effects by allowing other ecosystems to compete not just in terms of the number of apps listed, but also the availability of top apps, the time-to- market (an app rarely appears at the same time across all platform app stores) and the overall app quality.

Moreover, **cross-platform tools reduce barriers to entry and democratise app development**, by allowing developers from any language (HTML, Java, C++), any background (hobbyist, pros, agencies, corporates) and any skill level (visual designer to hard-core developer) to build mobile apps. The dozens of CPTs available cater to every developer segment, from creative designers to C++ gurus to hobbyist website enthusiasts to Fortune-500 CIOs. The result could be termed a “democratisation” of software development (in the words of Unity’s Dan Adams), in that mobile platforms may be opened up to all types of developers.

### Mergers, financings and the survival of the strongest

**We have identified over 100 cross-platform developer tools,** in a market that’s booming with new players in 2011. Cross-platform tools have passed the “early adopter” phase, and are now moving into mainstream. For example vendor Sencha counts 1.6 million SDK downloads, Corona apps have reportedly been downloaded 35 million times in 2011, Unity reports 200,000 developers active each month, while Appcelerator boasts 35,000 apps published using the tool and deployed on 40 million devices.

Since 2011, cross-platform tool vendors have raised major VC funding, have been acquired, or achieved major releases. In the CPT space we have tracked 10 acquisitions, and over US$ 200 million in funding rounds. This is a market that takes cash to survive: CPT vendors are subsidizing their entry to market with free products, based on ample VC funding. For example OpenPlug ceased operations as it failed to find a monetisation model, with its key challenge being the conversion of freemium users into paying customers for its support and professional services. CPT vendors without a compelling free product will be washed out by the competition.

### Cross-platform tools are taking HTML further than browsers can

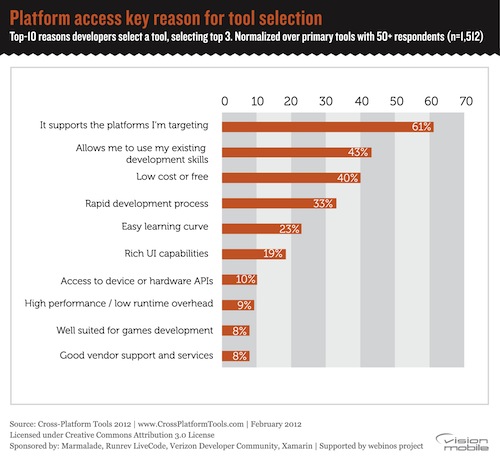
The purpose of HTML5 has been to extend the capabilities of web apps (those developed using HTML and JavaScript) to more closely match the capabilities of native apps. Despite performance disadvantages and fragmentation across different browser versions, HTML5 has emerged as the most widely supported authoring technology for cross-platform apps. Cross-platform tools are taking HTML further than web browsers can, by allowing web developers to create native smartphone apps. **In other words, CPTs are taking HTML5 much further by unifying the authoring side- rather than the runtime side – of the app across platforms**.

Moreover, CPTs are paving the way for HTML5 to become not a platform, but the mainstream development technology for smartphone apps. **Cross-platform tools are already triggering an influx of web developers**; We found that 60% of CPT users, irrespective of their primary tool, have more than five years experience in web development. Indeed, cross-platform tools have triggered an influx of web developers into mobile.

Android and Windows Phone have been constantly evolving, adding hundreds of new APIs from each major version to the next. Due to the rapid advancement of platforms, tools vendors will always be one or two steps behind in terms of features and access to the complete set of device capabilities. Developers that create demanding applications like 3D games or apps requiring intense user interaction, exceptionally deep user experience, or apps relying on specific features not available on all platforms will need to be developed using the native SDK. **Cross-platform tools will therefore be complementary to native SDKs**.

### Cross platform tools will become “business as usual”

As the platform landscape remains fragmented for the foreseeable future, cross-platform tools will become “business as usual” The future of mobile development is multi-platform – fewer and fewer developers will be able to afford to be confined to a single platform with the limited user reach and monetisation opportunities that implies. The adoption of cross-platform tools is driven by the ability to reach masses of users, which is the primary consideration for most developer segments. Cross-platform tools are indeed the only cost-effective vehicle for these developers to reach a wide mass of users, and we expect CPT usage to become commonplace a result.

[](http://goo.gl/npsnP)

### Multi-screen and the evolving points of competition

At the onset of 2012, CPT developer selection criteria are heavily skewed towards the breadth of platforms supported by each tool. This picture will change considerably as cross-platform tools vendors advance their products to cover all the major mobile platforms. We expect that by mid-2013, the platforms covered by a CPT will move from a point of differentiation to a point of parity. In that timeframe, we expect the points of competition to move to later stages of the app lifecycle, with vendors offering component marketplaces, end-to-end workflow tools, device adaptation tools, app publishing services and post-download services.

In the sea of 100+ cross-platform tools, vendors are beginning to differentiate by targeting three distinct developer segments: those working on games, enterprise or media apps. Developers in these three segments face distinctly different challenges, work in distinctly different environments and as such need very different CPT solutions. As tool vendors try to survive in the “red ocean” of dozens of cross-platform tools, we expect CPTs to emerge for the financial sector, media publishers and the healthcare/medical sector.

Multi-screen is the next frontier. The battle of the software ecosystems is raging across many screens – mobile, tablet, PC and soon smart TV devices – and multi-screen will be the next frontier for cross-platform tools. Already in our survey, 27% of respondents noted that they also target Windows PC and 24% target Mac desktops with their main cross-platform tool. However, the complexities of cross-platform development in a multi-screen environment are growing exponentially and beyond the simple sharing of the code between multiple platforms. Different screen types have different interaction models, input methods, screen sizes, go-to-market channels and pricing models, while developers working on different screens have use varying tool-chains, development cycles and collaboration processes. **With the proliferation of users who own more than one connect screen, the next frontier for cross-platform tools will be multi-screen**.

### Lessons to be learned

Cross platform tools have previously faced criticism, most notably from Steve Jobs in his infamous open letter “Thoughts on Flash”. The next generation of tools are however rapidly coming to market or maturing with abundant backing from the financial and developer community. The cross-platform tools market is in a state of abundant volatility and we see continual flux, as developers try a tool, and then churn to a different one. This is a market with no clear winners or losers. It’s a market where there is little developer loyalty, and perceptions are still being formed. Now is the time for well-funded vendors with great tools to prove themselves and establish a firm beachhead.

- Seth  
Follow us on Twitter – [@visionmobile](https://twitter.com/#!/visionmobile)