Tam Hanna

PhoneGap 3

Apps für iOS, Android, Windows Phone & Co. entwickeln

- Mit JavaScript plattformübergreifende Apps erstellen
- Ein Code für alle mobilen Betriebssysteme
- Plattformübergreifend auf die Hardware zugreifen

FRANZIS

Tam Hanna

PhoneGap 3

Tam Hanna

PhoneGap 3

Apps für iOS, Android, Windows Phone & Co. entwickeln

- Mit JavaScript plattformübergreifende Apps erstellen
- Ein Code für alle mobilen Betriebssysteme
- Plattformübergreifend auf die Hardware zugreifen

Bibliografische Information der Deutschen Bibliothek

Die Deutsche Bibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie;
detaillierte Daten sind im Internet über http://dnb.ddb.de abrufbar.

Programmleitung: Markus Stäuble
Satz: DTP-Satz A. Kugge, München
art & design: www.ideehoch2.de
Druck: C.H. Beck, Nördlingen
Printed in Germany

ISBN 978-3-645-60153-5

Inhaltsverzeichnis

1 Einleitung

Die Entwicklung der Programmiersprache JavaScript ist zweifellos eine der größten Erfolgsgeschichten der modernen Informatikgeschichte. Ursprünglich als Werkzeug zum besseren Animieren von Webseiten vorgesehen, eroberte die Sprache im Laufe der Jahre immer mehr und mehr Anwendungsfelder, die einst klassischen Sprachen wie C++ und Java vorbehalten waren.

Spätestens seit dem Erscheinen des iPhones der ersten Generation wurde klar, dass HTML5 und JavaScript auch zur Entwicklung von Anwendungen für Mobilcomputer taugen – anfangs war das Erfolgsgerät von Apple sogar ausschließlich durch seinen Webbrowser programmierbar.

Leider zeigte die von Desktop-Browsern hinreichend bekannte Problematik der nicht standardisierten Schnittstellen auch am Handy ihre Zähne. Jeder Hersteller zimmerte seine eigene API zusammen, die mit den Angeboten der anderen Hersteller inkompatibel war. Aufgrund der großen Marktbreite entstanden schon bald diverse Frameworks, die die verschiedenen Plattformen zu »einen« suchten. PhoneGap ist das mit Abstand erfolgreichste Resultat dieser Entwicklung – seine genauere Betrachtung ist die Aufgabe der folgenden Seiten.

Für PhoneGap spricht neben der extremen Aktivität der dahinterstehenden Community auch, dass das Produkt Eigentum der sehr finanzstarken Adobe Systems Corporation ist. Aus diesem Grund fehlt es den Entwicklern nicht an Ressourcen – ein Vorteil, der sich insbesondere bei der enormen Anzahl der unterstützten Plattformen manifestiert.

In diesem Kapitel wollen wir uns einen kleinen Überblick über die momentane Situation verschaffen. Auch gehen wir ein wenig darauf ein, wie Sie das vorliegende Buch optimal zu Ihrem eigenen Vorteil nutzen.

Für das Verständnis der folgenden Kapitel sind die hier abgedruckten Ausführungen nicht von eminenter Bedeutung. Trotzdem ist es hilfreich, ein wenig »über den Tellerrand« hinauszusehen – gehen wir es an.

1.1 Wieso Cross-Plattform-Entwicklung?

Kaum ein Bereich der IT ist so unübersichtlich wie die Mobilcomputerindustrie. Selbst für Analysten mit zehnjähriger Diensterfahrung gibt es immer wieder überraschende, verwirrende oder schlichtweg undurchschaubare Vorfälle.

Auch wenn sich die Betriebssystemlandschaft mittlerweile ein wenig bereinigt hat – anders als im Bereich der Workstations gibt es im Mobilcomputerbereich (noch) keinen klaren Marktführer, der alle anderen Plattformen verdrängt.

Das Messen der Marktanteile von Betriebssystemen für Smartphones ist eine Wissenschaft für sich. Die in Abbildung 1.1 gezeigte Entwicklung ist mit Sicherheit nicht auf das Zehntelprozent genau, gibt aber nach Meinung des Autors einen guten Überblick über die Marktbewegungen.

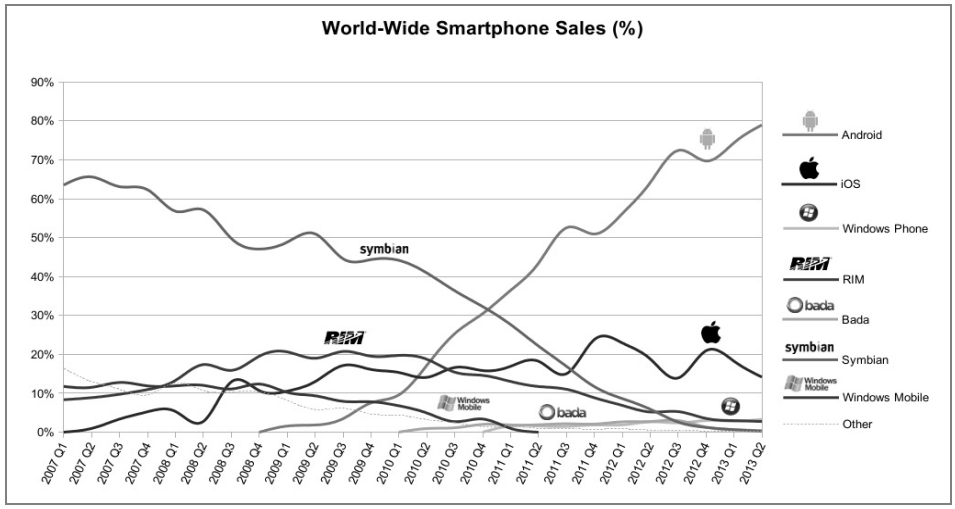


Bild 1.1: Die Marktanteile ändern sich rasend schnell (bearbeitet nach *http://en.wikipedia.org/wiki/File:World_Wide_Smartphone_Sales_Share.png*).

Geübten Wirtschaftlern fällt sofort auf, dass sich die Positionen der einzelnen Marktteilnehmer ungewöhnlich schnell ändern. Es kommt außerhalb des Mobilcomputermarkts nur sehr selten vor, dass ein Marktführer binnen weniger Monate so gut wie komplett von der Bildfläche verschwindet und durch ein anderes System ersetzt wird. Übrigens ist der Absturz von Symbian nicht das erste derartige Ereignis in der Industrie – auch Palm OS starb einen relativ schnellen und sehr schmerzhaften Tod.

Im selben Zeitraum ist eine Vielzahl neuer Plattformen auf den Markt gekommen. Das liegt unter anderem daran, dass sich die Hersteller von Handcomputern durch das Forcieren eines hauseigenen Betriebssystems Vorteile im Konkurrenzkampf erhoffen.

Der wichtigste Effekt eines proprietären Systems ist die Investitionssicherheit beziehungsweise gesteigerte Kundenbindung. Wer sein ThinkPad satt hat, kauft sich ein Elitebook. Beide laufen unter Windows, die auf dem Lenovo-Computer installierten Anwendungsprogramme lassen sich (in der Regel) auch am neuen Rechner weiterverwenden.

Im Mobilbereich ist die Lage völlig anders. Wenn ein Hersteller erfolgreich ein proprietäres Betriebssystem samt eigenem Ökosystem auf den Markt bringt, entkommt ihm der unzufriedene User nicht mehr ohne Weiteres.

Der Grund dafür ist die Vielzahl von Applikationen, Spielen und sonstigen Inhalten. Die im Samsung App Seller Office oder im Nokia Ovi Store gekaufte Lizenz von TouchCalc lässt sich (selbst bei einem hoch kooperativen Entwickler) unmöglich unter Windows

Phone 8 verwenden oder auf den iTunes App Store bringen – der Autor ist selbst im Bereich der Anwendungsentwicklung tätig und hatte das Problem schon mehrfach.

Daraus folgt eine enorme »Klebrigkeit«. Wer im Fall eines Herstellerwechsels seine gesamte Musik und alle Apps verliert, verzeiht zähneknirschend so manche Frechheit – in Cupertino gibt es ein Unternehmen, das unter anderem davon lebt.

Mindestens ebenso nützlich ist eine proprietäre Plattform beim Ausnutzen der neudeutsch als Convergence bezeichneten Effekte. Dabei handelt es sich um das Zusammenwachsen von verschiedenartigen Geräten – klassisch ist das Übertragen von am Telefon aufgenommenen Fotos in Richtung des Fernsehgeräts. Allerdings sind, zumindest im universitären Bereich, schon ganz andere Konzepte »unterwegs« – so ist zum Beispiel eine Waschmaschine denkbar, die das zeitkritische Trocknen von bügelfreien Hemden erst auf Anweisung per Smartphone anwirft.

Auch an dieser Stelle sind all jene Hersteller empfindlich im Vorteil, die eine eigene Plattform haben. Wer eine Waschmaschine der ACME Corp. hat und diese fernsteuern möchte, braucht ein Smartphone aus demselben Haus.

Convergence ist im Übrigen keine Zukunftsmusik. Abbildung 1.2 entstand auf einem Presseevent in Wien – der Anlass war die Ankündigung einer Kooperation zwischen dem Netzbetreiber Hutchison (Drei) und LG. Das Unübliche daran war, dass sich die beiden Firmen nicht auf eine Kooperation im Bereich der Handys einigten … hier ging es um den Vertrieb von TV-Geräten.

Bild 1.2: Hutchison und LG verkaufen gemeinsam Fernsehgeräte.

In Summe lässt sich also postulieren, dass im Mobilcomputerbereich »der Bär steppt« und dass es immer wieder etwas Neues zu tun und/oder zu sehen gibt. Diese Volatilität ist für Pressejournalisten hilfreich, da sie immer wieder Stoff für neue Sensationsberichte bietet. Leider hat man als Entwickler eher wenig davon: Wenn ein Betriebssystem untergeht, muss meist auch ein Gutteil der eigenen Software dran glauben.

Allein aus diesem Grund ist es sinnvoll, die eigenen Applikationen möglichst plattformunabhängig zu gestalten. Seit Jahren greifen Entwickler zum Erreichen dieses hehren Ziels auf eine Vielzahl von Methoden zurück, die teilweise sehr kompliziert sind und die Codebasis der erstellten Applikation wesentlich »verfetten«.

Zudem erfüllt die derzeitige Marktlage erstmals das Gesetz vom Duopol – es gibt zwei große Plattformen, die den Markt untereinander aufteilen. Aus dieser Sicht wäre es eine gewagte (aber durchaus wirtschaftlich begründbare) Prognose, dass sich die Verteilung der Betriebssysteme nicht mehr wesentlich ändert.

Aus dieser Situation folgt, dass einige Systeme im Laufe der Zeit vom Markt verschwinden werden. Die Benutzer von derartigen Geräten flüchten zwangsweise in Richtung anderer Plattformen – aus der Erfahrung ist bekannt, dass diese Flucht sehr langsam erfolgt.

Der Grund dafür liegt unter anderem in der auf 24 Monate angelegten Vertragsbindung der diversen Handyanbieter. Jedes zweite Jahr bekommt der durchschnittliche User ein neues Telefon geschenkt – warum also vorher upgraden?

Während des »Todes« einer Plattform treten hochinteressante Effekte auf. Da immer mehr Entwickler die Wartung ihrer Produkte für das System aufgeben, sehen sich die verbleibenden User mit einer immer geringeren Auswahl an Produkten konfrontiert. Dadurch wird es für Entwickler immer leichter, die Aufmerksamkeit dieser User zu gewinnen.

Menschen sind Gewohnheitstiere. Aus diesem Grund besteht eine hohe Wahrscheinlichkeit, dass die migrierenden User sich ihrer »alten Freunde« erinnern (siehe Abbildung 1.3). Für diese User konkurriert Ihre App nicht mit den 500.000 anderen Produkten in den Stores von Apple und Google – solange der Produktname gleich bleibt, sind Sie der Primus inter Pares.

Das hat längerfristig ziemlich interessante Effekte im Bereich der Userbindung. Sowohl Google als auch Apple »bevorzugen den Habenden« – in der Praxis bedeutet das, dass Entwickler mit hohen Userzahlen und guten Bewertungen leichter an Feature Spaces kommen als unerprobte Anbieter.

Bild 1.3: Die Beschreibung spricht User an, die von Symbian zu bada wechseln.

Natürlich ist es ökonomischer Unsinn, nur aus diesem Grund eine komplette und aufwendige Portierung anzustoßen. Wenn es sich dabei aber nur um eine Rekompilation handelt, sieht die Sache anders aus – und genau das leisten Cross-Plattform-Frameworks wie PhoneGap.

1.2 Für wen dieses Buch geschrieben ist

Wie Sie schon an der Einleitung bemerken, wendet sich dieses Werk nicht nur an handcomputererfahrene Entwickler. Natürlich ist auch für diese Zielgruppe jede Menge an interessanten Inhalten dabei – es ist aber nicht unbedingt erforderlich, Erfahrung im Bereich der Programmierung von Mobil-Apps zu haben.

Da PhoneGap auf JavaScript aufsetzt, ist es in jedem Fall sinnvoll, gute Kenntnisse in der Anwendung dieser Programmiersprache zu haben. Wer – wie der Autor – bisher eher mit C++ beziehungsweise Java arbeitet, findet im Anhang »OOP mit JavaScript« einige wertvolle Hinweise zum effizienten Einsatz.

Viel wichtiger ist in der Praxis das Mitbringen von ausreichenden Kenntnissen im Bereich der Erstellung von ansprechenden Webseiten. PhoneGap-Applikationen sind im Prinzip nichts anderes als Webpages mit interaktivem Code im Hintergrund – aus diesem Grund ist es dem Framework nicht wirklich möglich, Ihnen bei der Erstellung der grafischen Elemente zur Hand zu gehen.

Allerdings ist Webdesign eines der bestdokumentierten Themen im Bereich der Informatik. Literatur ist ohne Probleme erhältlich, Hunderte Grafiker warten nur darauf, für (oft überraschend) wenig Geld wunderbare Entwürfe zu liefern.

Dabei kommt Ihnen die soeben erwähnte Eigenschaft des Frameworks sehr entgegen. Es ist nämlich nicht notwendig, Ihrem Designer Zugriff auf die Programmlogik zu gewähren. Stattdessen kommt der in Abbildung 1.4 gezeigte Workflow zum Einsatz.

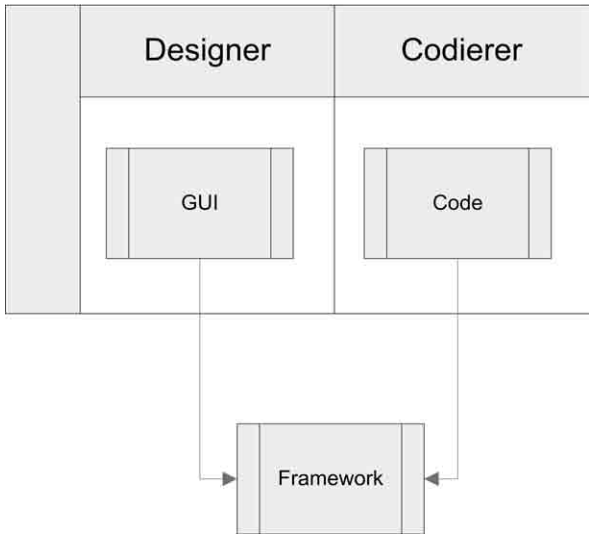


Bild 1.4: Entwickler und Designer bearbeiten separate Teile des Produkts.

Der Webdesigner bearbeitet ausschließlich die Layoutdaten, die in den HTML-Dateien liegen. Mit den die Programmlogik enthaltenden JavaScript-Files hat er nichts zu tun – diese verbleiben auf Ihrer Maschine und sind zum Erstellen der Designs nicht erforderlich.

Ähnlich wie bei QML profitieren Sie auch hier vom Wegfallen des fehlerträchtigen und höchst lästigen »Übersetzungsschritts« zwischen dem Design und der praktischen Applikation.

1.3 Wie Sie dieses Buch lesen sollten

Dank immer schneller werdender Produktzyklen gibt es kaum einen Mobilcomputerprogrammierer, der über zu viel Freizeit klagt. Aus diesem Grund ist dieses Buch so aufgebaut, dass Sie es nicht zur Gänze lesen müssen. Allerdings spricht nichts dagegen, es zu tun.

Wenn Sie mit JavaScript noch keine Erfahrung haben, sollten Sie nach der Lektüre dieses Kapitels sofort mit dem Anhang zur objektorientierten Programmierung in JavaScript beginnen. Die dort vorgestellten Patterns erleichtern Ihnen die Arbeit mit den Codebeispielen – außerdem verhindern sie das Erstellen von unwartbarem Spaghetticode.

Zum »Schnellstart« empfiehlt es sich, Kapitel 2, »Erste Schritte mit PhoneGap«, sowie die Beschreibung der einzelnen Namespaces in Kapitel 6, »Die PhoneGap-API«, zu

lesen. Danach wissen Sie genug über die Grundlagen von PhoneGap, um erste Entscheidungen zur Struktur Ihrer Anwendungen zu treffen.

Wenn Sie PhoneGap 3.0 verwenden, sollten Sie unbedingt das Kapitel 7 zu den Kommandozeilenwerkzeugen (CLIs) lesen.

Wenn Sie bisher keine Erfahrung mit der Entwicklung von mobilen Apps mit HTML und CSS haben, empfiehlt sich eine sorgfältige Beschäftigung mit dem Kapitel 5, »Oberflächen für Cross-Plattform-Apps«, das sich auch mit geeigneten Templates und Frameworks beschäftigt. Diese ersparen Ihnen unter Umständen sehr viel Arbeitszeit und sorgen für ein passables Aussehen der Applikation.

Da jedes Betriebssystem eigene »Parameter« hat, lesen Sie dazu selektiv Kapitel 3 zur Konfiguration der jeweiligen Entwicklungsumgebungen und Kapitel 10 zu nativen Plug-ins.

1.4 Beispiele für PhoneGap-Projekte

Oftmals werden die Funktionen eines Produkts erst dann verständlich, wenn man einige Beispiele für die damit erreichbaren Resultate sieht.

Die folgenden Beispielanwendungen entstammen allesamt der »Usecase-Sammlung« des PhoneGap-Projekts. Aus Platzgründen konnten hier nur einige interessante Apps ausgewählt werden – die volle Liste findet sich unter *http://phonegap.com/case*.

LogiTech SqueezeBox Controller

Die von LogiTech angebotenen Medienverteiler aus der SqueezeBox-Serie sind seit einiger Zeit auf dem Markt – sie erfreuen sich enormer Beliebtheit.

Weniger bekannt ist, dass die in Abbildung 1.5 gezeigte Steueranwendung fast komplett auf PhoneGap basiert. Nur die Kommunikation zwischen Endgerät und Telefon erfolgt durch nativen Code, der über ein Plug-in in die Programmierumgebung eingebunden wurde.

Interessant ist auch, dass der Entwickler im Interview (siehe *http://phonegap.com/case/2011/07/20/turn-on-tune-in-turn-up-logitechs-squeezebox-controller-app-turns-phones-and-tablets-into-remote-controls/*) besonders auf das GUI-Framework Sencha Touch hinweist. Und ein weiteres Thema ist die Verwendung von Desktop-Browsern zum Debugging.

Bild 1.5: LogiTech nutzt PhoneGap für eine Gerätesteuerungsapplikation (Bildquelle: *http://phonegap.com/case/2011/07/20/turn-on-tune-in-turn-up-logitechs-squeezebox-controller-app-turns-phones-and-tablets-into-remote-controls/*).

IGN Dominate

Die von der amerikanischen Spiele-Website IGN entwickelte App ist ein klassisches Beispiel für eine App, die für die Realisierung mit PhoneGap prädestiniert ist.

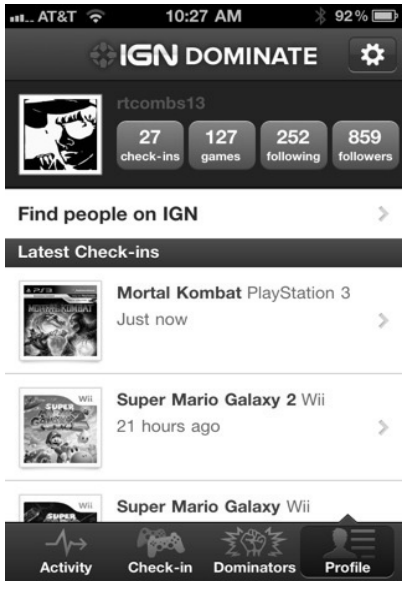


Bild 1.6: Auch IGN nutzt PhoneGap (Bildquelle: *http://phonegap.com/case/2011/07/29/check-in-and-dominate-phonegap-score-with-ign-for-playing-well-with-others/*).

Die Aufgabe des Programms ist vergleichsweise simpel: Es erlaubt Spielern, ihre in diversen Titeln erspielten Ergebnisse in soziale Netzwerke zu stellen. Dazu ist kein nativer Code erforderlich, deshalb konnte die Entwicklung komplett in JavaScript erfolgen.

Übrigens kam auch hier Sencha als GUI-Stack zum Einsatz.

HyperWallet

Der kanadische Anbieter von Zahlungsdiensten benutzte PhoneGap, um eine in Objective-C geschriebene Anwendung für iOS zu ersetzen. Dabei kam ein hauseigener GUI-Stack zum Einsatz, der das Anpassen der Anwendung an die diversen Kunden wesentlich erleichtert (Stichwort CSS).

Besonders interessant ist im Fall von HyperWallet, dass das Unternehmen auch die mobile Webseite aus der PhoneGap-Codebasis heraus generiert.

Bild 1.7: PhoneGap ist auch für M-Payment-Applikationen geeignet (Bildquelle: *http://phonegap.com/case/2011/06/03/hyperwallet-adds-mobile-apps-to-the-mix-with-phonegap/*).

1.5 Danksagung

Am Ende der Einleitung eines Buchs sollte die Danksagung nicht fehlen – es gibt kaum ein Buch, an dessen Erstellung wirklich nur der Autor allein beteiligt war.

Erst durch Ihr Vertrauen, lieber Leser, bekam meine Arbeit einen Sinn. Deshalb gebührt der erste Platz auf dieser Liste Ihnen. Ihr Kauf zeigt Wertschätzung in meine Arbeit, ich hoffe, dass das vorliegende Werk Ihre Wünsche voll und ganz zufriedenstellt.

Meine Lebensgefährtin verdient besonderen Dank dafür, dass sie mir den Rücken während der Arbeiten an diesem Buch stets freigehalten hat – sei es durch das Abwimmeln lästiger Besuche oder das Anliefern von Nahrungsmitteln und Dokumenten. Deshalb: milujem ťa, Macky! Srdečná vďaka...

Fast genauso wichtig war mein langjähriger Wegbegleiter Markus Stäuble vom Franzis Verlag. Es ist immer wieder eine große Freude, mit einem Experten dieses Kalibers

zusammenzuarbeiten – ohne sein Feedback und seine Hilfe (Stichwort: Apple) wäre dieses Werk nie so komplett, wie es ist.

Ohne Herrn Dipl.-Ing. Hubert Illibauer wäre ich heute nicht da, wo ich bin. Danke dafür, dass Sie da waren, als ich allein war.

Zu guter Letzt sei allen Partnern und Kunden der Tamoggemon gedankt. Ohne das Vertrauen all dieser Personen wäre das Unternehmen heute nicht da, wo es steht. Egal ob als Käufer einer Applikation oder als Bezieher von Dienstleistungen – diese Passage gehört nur ihnen. Danke, danke, danke!

2 Erste Schritte mit PhoneGap

Nun ist es an der Zeit, erste Schritte in die Welt der Entwicklung von PhoneGap-Anwendungen zu gehen.

Außerdem werfen wir einen kleinen Blick zurück auf die Geschichte und die lizenzrechtlichen Bedingungen dieses Frameworks.

In diesem Kapitel betrachten wir die Struktur einer PhoneGap-Anwendung im Detail für eine Android-App. Auch wenn Sie selbst nicht für Android entwickeln möchten, empfiehlt sich die Lektüre – in den späteren Kapiteln wird ein Verständnis der hier besprochenen Grundstruktur vorausgesetzt.

2.1 Die Geschichte von PhoneGap

Die Entwicklung von PhoneGap begann auf dem iPhoneDevCamp in San Francisco – im August 2008 stellten sich dort einige Entwickler die Frage, ob man iOS-Anwendungen nicht auch ohne Objective-C erstellen könnte. Hinter diesen Überlegungen stand die Problematik, dass die native Programmiersprache des iPhones alles andere als leicht zu erlernen ist – ein Faktum, das auf JavaScript nicht zutrifft.

Das Produkt erfreute sich binnen kürzester Zeit großer Beliebtheit. Die Unterstützung für Android und BlackBerry begann schon im Oktober des Jahres 2008, auf der Web 2.0 Expo gewann das Produkt den LaunchPad-Preis.

Im Laufe der Zeit schlossen sich die Entwickler in einem Unternehmen namens Nitobi zusammen. Dieses finanzierte sich unter anderem durch den Verkauf von Beratungsdienstleistungen.

Adobe – besser bekannt für Flash, Photoshop und das PDF-Format – steckte in dieser Zeit in einer ärgerlichen Fehde mit Apple. Der Grund dafür war, dass Apple den Vertrieb eines Flash-Players für das iPhone untersagte und Entwickler stattdessen in Richtung offener Standards trieb.

Die darauffolgende Medienkampagne war gigantisch. Mit Sicherheit hatte Adobe die Mehrheit der User auf seiner Seite und hätte sein Flash-Plug-in einfach als Applikation für »Jailbroken Devices« auf den Markt bringen können. Dies wäre ein mittleres Desaster für Apple gewesen, da das DRM-Konzept von iTunes somit ad absurdum geführt worden wäre. Außerdem hätten Third-Party-App-Stores wie Cydia in den Massenmarkt eindringen können – Adobe wäre zumindest nach Ansicht des Autors mit Sicherheit als der Gewinner dieses Konflikts hervorgegangen.

Trotzdem beschloss das Unternehmen, »mit der Zeit zu gehen« und in Zukunft auf die von Apple propagierten offenen Standards zu setzen. Da sich PhoneGap in der Zwischenzeit einen durchaus attraktiven Marktanteil erkämpft hatte, kaufte Adobe das Unternehmen Nitobi am 4. Oktober 2011 kurzerhand auf.

Zu Imagezwecken und aus Gründen der leichteren Wartbarkeit beschloss man, den Quellcode des Produkts weiter als Open Source zu halten. Was auf den ersten Blick widersprüchlich klingt, ergibt auf den zweiten Blick ökonomisch Sinn – eine aktive Entwicklercommunity spart hauseigenes Personal und hilft mit, die Kosten zu minimieren. Außerdem lässt sich am Vertrieb von Infrastruktur und Consulting-Dienstleistungen oft mehr verdienen als am eigentlichen Verkauf von (leicht piratierbaren) Lizenzen.

Die Apache Foundation – am besten bekannt durch den gleichnamigen Webserver – betreut seit langer Zeit diverse andere quelloffene Projekte. Adobe lagerte den Quellcode deshalb in Richtung von Apache aus, wo man – der Name PhoneGap ist ja eine eingetragene Marke von Adobe – einen neuen Namen für das nun quelloffene Kernprodukt suchte.

Nach einigen Überlegungen (Stichwort Apache Callback) landete die Community bei dem Namen Cordova, das Projekt selbst ist unter *http://cordova.apache.org/* beheimatet. Der Grund für die Wahl war, dass die Büros von Nitobi seinerzeit in der Cordova Street in Vancouver beheimatet waren.

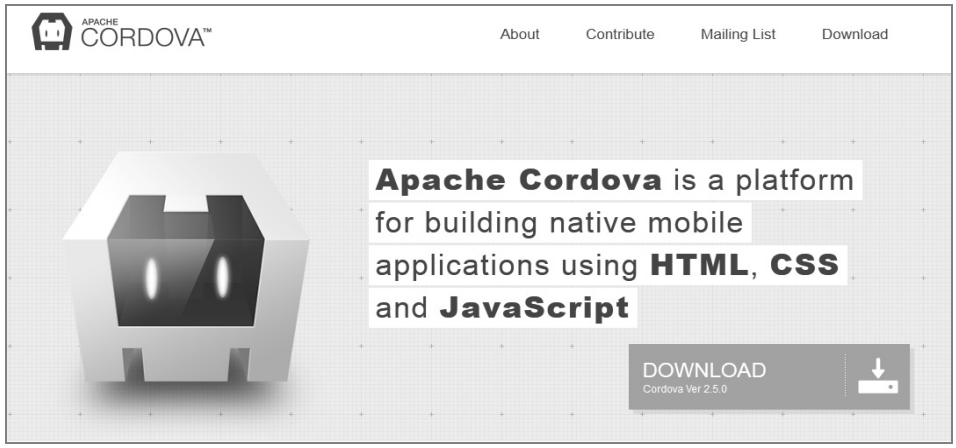


Bild 2.1: Apache hat Cordova mittlerweile komplett in die eigene Website eingebaut.

In der Praxis sieht die Struktur des resultierenden Projekts wie die in Abbildung 2.2 gezeigte aus.

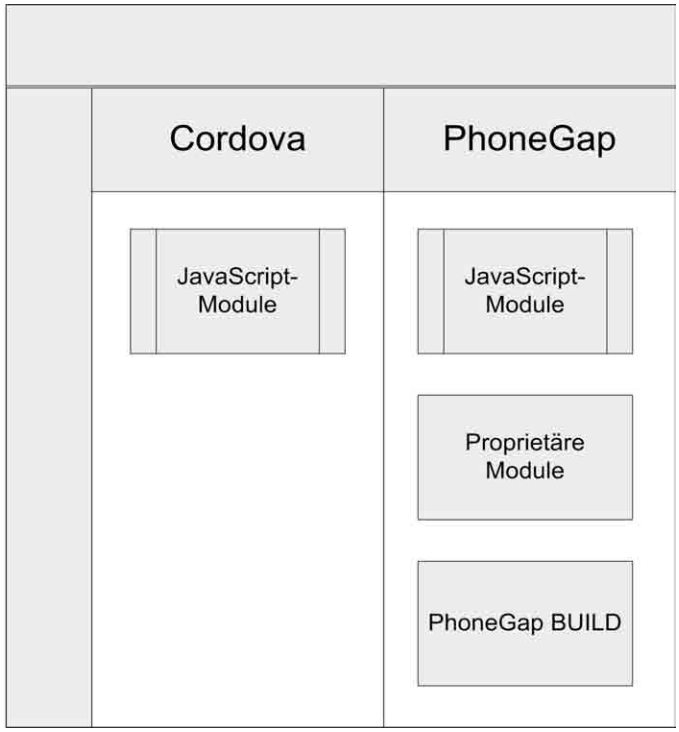


Bild 2.2: PhoneGap besteht aus Cordova plus einigen proprietären Modulen.

Angemerkt sei, dass Adobe derzeit noch keine proprietären Module in die PhoneGap-Distribution einpflegt. Allerdings ist davon auszugehen, dass sich das – ganz analog zu den speziellen Modulen in der kommerziellen Distribution von Qt – im Laufe der Zeit ändert.

Solange Sie kein Interesse daran haben, an der eigentlichen Entwicklung von PhoneGap mitzuwirken, ist diese Unterscheidung rein philosophischer Natur. Als »Endanwender-Entwickler« laden Sie einfach die aktuellste Version von PhoneGap von der Website herunter und verwenden sie. Fehlerberichte und eigene Patches sind bei Apache willkommen.

2.2 Lizenzrechtliches

Da Cordova – wie soeben festgestellt – von der Apache Foundation betreut wird, ist es naheliegend, dass die Codebasis unter einer der hinreichend bekannten quelloffenen Lizenzen aus dem gleichen Haus steht.

Speziell handelt es sich dabei um die Version 2.0 der Apache License:

The PhoneGap code was contributed to the Apache Software Foundation (ASF) under the name Apache Cordova. *Through the ASF, future PhoneGap development will ensure open stewardship of the project. It will always remain free and open source under the Apache License, Version 2.0. –via http://phonegap.com/about/license/*

Es ist also – anders als bei der klassischen GPL – erlaubt, Apache-Code in ein kommerzielles und nicht quelloffenes Produkt einzubauen und dieses weiterzuverkaufen.

Die Apache Foundation weist explizit darauf hin, dass dadurch keine Verpflichtung zur Freigabe des eigenen Codes entsteht:

It does not require you to:

- *include the source of the Apache software itself, or of any modifications you may have made to it, in any redistribution you may assemble that includes it;*
- *submit changes that you make to the software back to the Apache Software Foundation (though such feedback* is *encouraged). –via* http://www.apache.org/foundation/license-faq.html#WhatDoesItMEAN

Wichtig zu wissen ist, dass durch die Verwendung einige Verpflichtungen entstehen. Im Speziellen sind Sie verpflichtet, eine Kopie der jeweiligen Apache-Lizenz in den EULA-Vertrag Ihres Programms einzubinden und den Benutzer darauf hinzuweisen, dass das Produkt Elemente enthält, die der Apache Foundation gehören.

Streng verboten ist es indes, ein auf Basis eines Apache-Projekts entwickeltes Produkt als Apache-Produkt zu bezeichnen. Ebenfalls ist es verboten, sich selbst als Urheber des Apache-Produkts auszugeben.

2.3 Wie funktioniert PhoneGap?

Nach diesen einleitenden Bemerkungen ist es an der Zeit, sich näher mit der Struktur von PhoneGap und den damit erstellten Applikationen zu befassen. Im Prinzip unterscheidet man im Bereich der Mobilcomputerapplikationen zwischen nativen, hybriden und klassischen webbasierten Applikationen. Jede dieser drei hat ihre spezifischen Stärken und Schwächen, die in der Tabelle zusammengefasst sind:

	Web-App	*Hybride Applikation*	*Native Applikation*
Geschwindigkeit	Langsam	Gut	Schnellstmöglich
Portabilität	Hoch	Hoch	Je nach Architektur: schlecht bis minimal
Entwicklungskosten	Gering	Gering	Hoch
Zugriff auf native Funktionen	Nein	Fast alles	Alles
Distribution im App-Store	Nein	Ja	Ja
Natives Aussehen	Bedingt	Bedingt	Je nach Framework: bedingt bis komplett

Hybride Applikationen erreichen die ihnen eigenen Vorteile, indem sie auf einen kleinen Kniff zurückgreifen. Statt im Browser des Systems werden sie in einem eigenen »Loader« ausgeführt. Dabei handelt es sich um eine Stand-alone-Applikation, die – zusammen mit den *.html*-, *.js*- und *.css*-Dateien mit ihrer Geschäftslogik – gemeinsam in

den App-Store marschieren und auch von dort wie ein normales natives Programm heruntergeladen werden können.

In einem »Verpackungsdiagramm« würde dieser Sachverhalt wie in Abbildung 2.3 gezeigt aussehen.

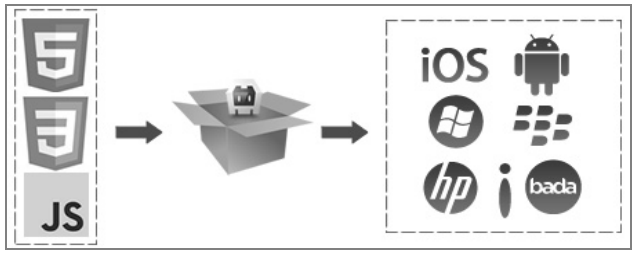


Bild 2.3: Die fertige Applikation besteht aus Loader und Quellcode (Quelle: *http://phonegap.com/2012/05/02/phonegap-explained-visually/*).

Böse Zungen würden den soeben genannten Loader als Webbrowser auf Steroiden bezeichnen. Diese Beschreibung ist nicht einmal falsch – das PhoneGap-Projekt liefert in der offiziellen Dokumentation die folgende Abbildung aus, die das Konzept veranschaulicht:

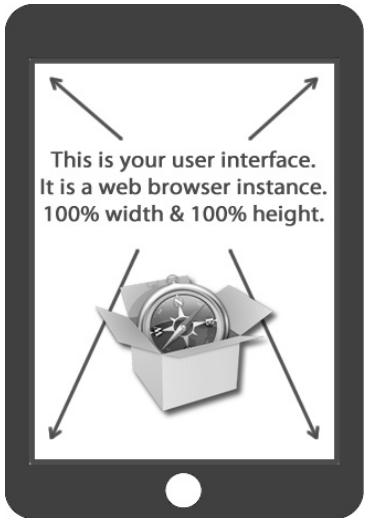


Bild 2.4: Der Loader von PhoneGap ist im Prinzip ein Webbrowser.

Diese an sich nicht allzu falsche Beschreibung hat nur einen kleinen Haken. Anders als der normale Webbrowser ist der Loader in der Lage, auf den ausgeführten JavaScript-Code Einfluss zu nehmen. Auf diese Art und Weise ist es möglich, den von Haus aus unterstützten Namespace um zusätzliche Module und APIs zu erweitern.

PhoneGap nutzt diese Möglichkeit, indem es die in Kapitel 6 im Detail beschriebene PhoneGap-API einschreibt. Diese ermöglicht den Zugriff auf eine Vielzahl von nativen Funktionen des jeweiligen Endgeräts (siehe dafür auch die Liste im Abschnitt 2.6 »Was funktioniert auf welcher Plattform?«).

Zusätzlich gibt es in PhoneGap die Möglichkeit der Erstellung von nativen Plug-ins. Dabei handelt es sich um eigene JavaScript-Erweiterungen, die aber in der nativen Programmiersprache der jeweiligen Plattform erstellt werden. Das erlaubt das »Ansprechbarmachen« von in der PhoneGap-API nicht vorgesehenen Teilen des Betriebssystems und lässt sich – zumindest theoretisch – auch verwenden, um eine existierende und in C/C++ vorliegende Geschäftslogik à la QML zu recyceln und mit einem neuen Benutzer-Interface zu versehen.

Abbildung 2.5 zeigt, wie sich also in Summe die hinter PhoneGap stehende Architektur präsentiert.

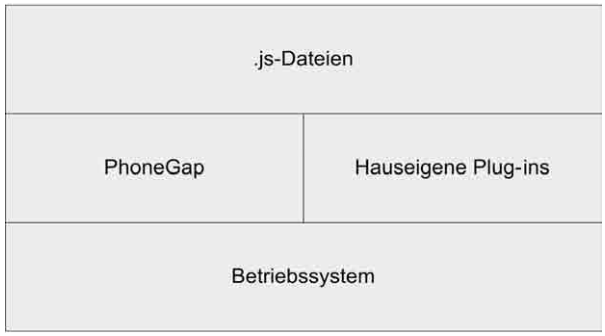


Bild 2.5: PhoneGap-Applikationen können auch native Plug-ins enthalten.

Da PhoneGap keine API zum Erstellen von GUIs mitbringt, ist es in der Praxis nicht unüblich, zusätzlich auf ein GUI-Framework zurückzugreifen. Allerdings ist PhoneGap völlig GUI-agnostisch – es wäre theoretisch möglich, die eigene Anwendung nur aus HTML und CSS zusammenzubauen.

Die Mehrheit der Entwickler setzt dabei auf jQuery Mobile, da es als leicht zu erlernen gilt – aufgrund der hohen Verbreitung gibt es für diese Konfiguration auch die meisten Beispiele. Sencha Touch kommt ebenfalls zum Einsatz, gilt aber – insbesondere aufgrund des nur kommerziell erhältlichen WYSIWYG-Designers – als ärgerlich komplex.

2.3.1 High-Level-Architektur

An sich sind PhoneGap-Applikationen – wie soeben beschrieben – nichts anderes als Webseiten mit einigen speziellen JavaScript-APIs. Deshalb gelten die für das Design von normalen Web-2.0-Anwendungen gültigen Patterns so gut wie uneingeschränkt auch für den Entwickler einer mobilen Applikation.

In der Praxis wird PhoneGap häufig zur Realisierung sogenannter Data-Driven Applications eingesetzt. Dabei handelt es sich um Programme, die den Zugriff auf die in einer Datenbank liegenden Daten ermöglichen.

Das PhoneGap-Team empfiehlt, den eigentlichen Zugriff auf die Datenbank nicht direkt aus der mobilen Applikation heraus abzuwickeln. Stattdessen wird das Dazwischenschalten eines Applikationsservers dringend empfohlen.

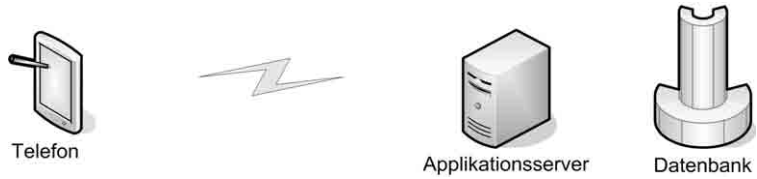


Bild 2.6: Das Telefon arbeitet als »Thin Client«.

Dieser handhabt die Geschäftslogik und die Kommunikation mit dem Server. Der auf dem Telefon befindliche Client nimmt die Daten über REST, JSON, SOAP oder als klassische HTTP-Requests entgegen und verarbeitet sie lokal weiter.

Auf Clientseite ist es sehr empfehlenswert, das als Single-Page Application bekannte Design-Pattern umzusetzen. Es beschreibt eine Applikation, die nur aus einer einzigen Webseite besteht – das Ein- und Ausblenden der Inhalte erfolgt zur Laufzeit durch den Einsatz von JavaScript.

Tipp: Der Begriff Single-Page Application geht auf Steve Yen zurück, der das Pattern im Jahr 2005 erstmals formalisierte. Allerdings ist er nicht der Erste, der eine derartige Webseite umsetzte – die im Jahr 2002 von Stuart Morris realisierte und unter *http://slashdotslash.com/* einsehbare »Self-Contained Website« zeigt Grundzüge der Technologie in Anwendung.

Übrigens spricht per se nichts dagegen, eine PhoneGap-Anwendung in mehrere *.html*-Dateien beziehungsweise Formulare aufzuteilen. Ärgerlich ist, dass die globalen Variablen beim »Weiterscrollen« verloren gehen – das erschwert das Erstellen einer produktiv nutzbaren Applikation ungemein, da der Programmzustand permanent zwischengespeichert werden muss.

2.4 Was kann PhoneGap?

Wer erstmals von PhoneGap hört, denkt oft an ein Cross-Plattform-Framework nach dem Schema von Qt. Leider ist diese Annahme grundfalsch – PhoneGap ist, wie der Name schon sagt, ein Werkzeug zur Überbrückung von Lücken (Gaps) in den Webbrowsern der diversen Telefone.

Auf gut Deutsch bedeutet dies, dass das Framework darauf spezialisiert ist, in den JavaScript-Runtimes fehlende APIs nachzurüsten. Ein gutes Beispiel dafür ist das Storage-Modul, das Ihrer Webanwendung beim Aufbewahren von größeren Datenmengen assistiert.

PhoneGap ist ideal, wenn Sie Ihre Anwendung eigentlich unter Verwendung von Webtechnologien realisieren möchten, Ihnen aber aufgrund technischer Beschränkungen einige kritische Funktionen fehlen. In diesem Fall ist die Wahrscheinlichkeit hoch, dass das Framework das fehlende native Feature von Haus aus mitbringt.

Ist das nicht der Fall, hilft meist die Verwendung eines nativen Plug-ins. Bei dieser in Kapitel 10 dieses Buchs beschriebenen Technik erstellen Sie in der nativen IDE des jeweiligen Betriebssystems ein kleines Modul, das die JavaScript-Engine des Browsers um die notwendige Funktion ergänzt.

Eine weitere Stärke von PhoneGap ist das schon weiter oben besprochene Verpacken von Webanwendungen. Nach der Verarbeitung durch das jeweilige Platform-SDK ist Ihre Anwendung nicht von einem voll nativen Programm zu unterscheiden – dem Hochladen des Resultats steht also nichts im Weg. Alle gängigen App-Stores akzeptieren mit PhoneGap realisierte Produkte.

Zu guter Letzt sei noch darauf hingewiesen, dass die resultierenden Anwendungen selbstverständlich auch ohne Internetverbindung lauffähig sind – anders als auf Webservern gehostete Programme funktioniert Ihre PhoneGap-App auch im Tunnel.

2.5 Was kann PhoneGap nicht?

Die mit Abstand wichtigste Limitierung von PhoneGap ist, dass das Framework keinen eigenen GUI-Stack mitbringt. Anders als in Qt und QML gibt es in PhoneGap keinerlei Möglichkeit, Einfluss auf das Aussehen der erstellten Benutzeroberflächen zu nehmen.

Das ist allerdings kein allzu großes Problem: Wer sich mit CSS und HTML gut auskennt und Zeit investiert, kann auch auf Basis dieser Technologien attraktive User-Interfaces entwerfen.

Alternativ gibt es natürlich immer die Möglichkeit, ein GUI-Toolkit wie das hier im Detail besprochene jQuery Mobile einzusetzen. Es bringt eine Vielzahl von attraktiv aussehenden Steuerelementen mit, die Sie nur noch in Ihr Projekt einbauen müssen.

Eine weitere Einschränkung ist, dass das Framework nicht wirklich zur Realisierung von »hybriden« Applikationen gedacht ist. Native Plug-ins erlauben Ihnen zwar, empfindliche oder performancekritische Passagen Ihrer Applikation zu »nativisieren«, allerdings ist das nicht der primäre Einsatzzweck von PhoneGap. Das Framework ist dazu vorgesehen, Webapplikationen mit »nativen Teilen« zu realisieren – alles andere ist eher eine »unintendierte Nebennutzung«.

Kurz gefasst: Nutzen Sie PhoneGap bitte nicht zur Realisierung von System-Utilitys oder für einen Defragmentierer des internen Speichers …

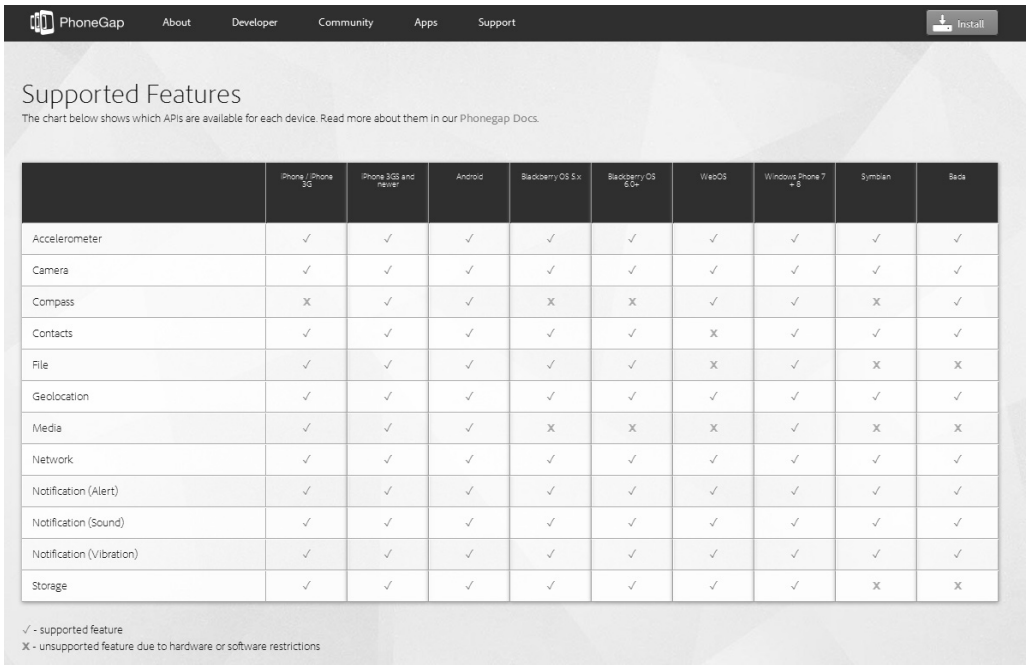
2.6 Was funktioniert auf welcher Plattform?

Da sich die Betriebssysteme untereinander wesentlich unterscheiden, gibt es für die Anbieter von Cross-Plattform-Frameworks eigentlich nur zwei Vorgehensweisen.

Entweder beschränkt man sich auf die minimale Anzahl der auf allen Systemen verfügbaren Funktionen. Alternativ ist es auch möglich, einige Features nur auf jenen Syste-

men anzubieten, auf denen sie lauffähig sind – der Entwickler muss dann selbst entscheiden, wie er weiter vorgehen möchte.

PhoneGap arbeitet nach der zweiten Methode. Abbildung 2.7 zeigt, welche Module auf welchen Betriebssystemen unterstützt werden:



	iPhone / iPhone 3G	iPhone 3GS and newer	Android	Blackberry OS 5.x	Blackberry OS 6.0+	WebOS	Windows Phone 7 + 8	Symbian	Bada
Accelerometer	✓	✓	✓	✓	✓	✓	✓	✓	✓
Camera	✓	✓	✓	✓	✓	✓	✓	✓	✓
Compass	x	✓	✓	x	x	✓	✓	x	✓
Contacts	✓	✓	✓	✓	✓	x	✓	✓	✓
File	✓	✓	✓	✓	✓	x	✓	x	x
Geolocation	✓	✓	✓	✓	✓	✓	✓	✓	✓
Media	✓	✓	✓	x	x	x	✓	x	x
Network	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Alert)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Sound)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Notification (Vibration)	✓	✓	✓	✓	✓	✓	✓	✓	✓
Storage	✓	✓	✓	✓	✓	✓	✓	x	x

Bild 2.7: Hier können Sie entnehmen, was unter PhoneGap alles möglich ist (*http://phonegap.com/about/feature*).

2.7 PhoneGap 2 vs. PhoneGap 3

Zur Zeit der Fertigstellung dieses Buchs hat Adobe eine neue Hauptversion von Cordova beziehungsweise PhoneGap auf den Markt gebracht. Diese unterscheidet sich von ihrem Vorgänger aus programmiertechnischer Sicht nicht oder nur minimal, bringt aber in logistischer Hinsicht einige Änderungen mit sich.

Der wichtigste Unterschied zur Vorgängerversion liegt darin, dass die im Abschnitt 2.9, »Projektskelett«, beschriebene manuelle Erstellung der Projektstruktur in den Hintergrund rückt – wenn es nach Wunsch von Adobe und Apache geht, arbeiten Entwickler in Zukunft bei der Projekterstellung und dem Debugging mit den in Kapitel 7 zu den CLIs beschriebenen Werkzeugen und nutzen die nativen IDEs nur für spezielle Bearbeitungsschritte.

Zudem brachte die neue Version auch im Bereich der Plug-ins einige Änderungen mit sich – aufgrund der neuen API verweigern die meisten für PhoneGap 2.x geschriebenen

nativen Erweiterungen unter PhoneGap 3.x den Dienst. Zudem wurden die Namespaces ebenfalls in Plug-ins ausgelagert, die vor der Verwendung über ein Werkzeug namens PlugMan ins Projekt kopiert werden müssen.

Aus Gründen der effizienteren Weiterentwicklung wurden einige Betriebssysteme abgekündigt. Wenn Sie Anwendungen für bada oder Symbian entwickeln möchten, müssen Sie notgedrungen auf die ältere Version setzen.

Aufgrund der tief greifenden Änderungen wird Adobe PhoneGap 2.x noch »einige Zeit« weiter unterstützen. In diesem Buch werden beide Vorgehensweisen »gleichberechtigt« vorgestellt – wählen Sie einfach die aus, die Ihnen für Ihr Projekt angemessener erscheint.

2.8 PhoneGap für Android installieren

Wir haben schon in der Einleitung festgestellt, dass die Entwicklung von PhoneGap-Anwendungen in der nativen Entwicklungsumgebung der jeweiligen Plattform erfolgt.

Die von Google gepushte Android-Plattform ist das derzeit am weitesten verbreitete Mobilcomputersystem. Da die (sehr leistungsfähige) native IDE kostenlos angeboten wird und Endgeräte mit dieser Plattform preiswert sind, wollen wir die ersten Schritte mit diesem Betriebssystem erledigen.

Android-Anwendungen entwickelt man in der Regel in einer Arbeitsumgebung namens *Eclipse*. Diese steht unter der URL *http://www.eclipse.org/downloads/* zum Download bereit – für Android-Entwickler ist die in Abbildung 2.8 hervorgehobene Version »Eclipse for Mobile Developers« ideal geeignet.

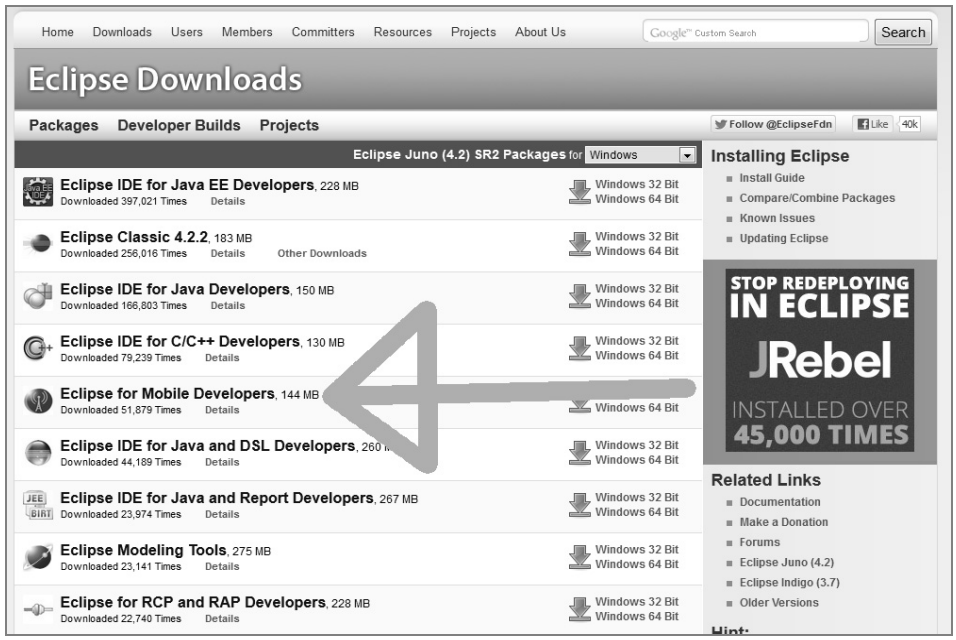


Bild 2.8: Eine passende Version der IDE verbirgt sich hinter diesem Link.

Entpacken Sie das heruntergeladene Archiv nach dem Download in einen beliebigen Ordner. Der Autor verwendet in den folgenden Schritten stets den Pfad *C:\Program Files\Android\eclipse*. Starten Sie die Entwicklungsumgebung durch einen Doppelklick auf die Datei *eclipse.exe*.

Im nächsten Schritt benötigen wir das Android-SDK. Dieses ist höchst modular aufgebaut, daher laden wir im ersten Schritt nur die SDK-Tools von *http://developer.android.com/sdk/index.html* herunter. Leider ist der Download etwas versteckt – Abbildung 2.9 zeigt, welche Sektion der Seite aufgeklappt werden muss.

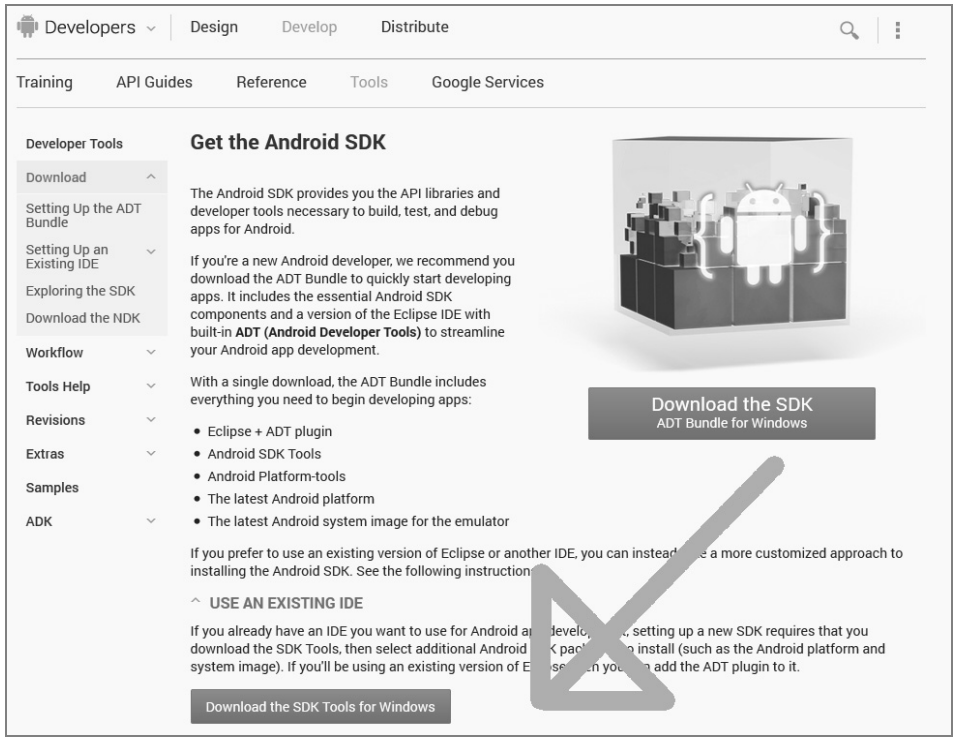


Bild 2.9: Der Download des passenden Moduls ist gut versteckt.

Führen Sie die heruntergeladene *.exe*-Datei aus, um die Installation der Platform-Tools anzuwerfen.

Tipp: Googles SDK ist hoch modular aufgebaut. Das hat den Vorteil, dass Sie keine unnötigen Dateien auf Ihre Maschine herunterladen müssen – andererseits ist die Installation der einzelnen Module etwas aufwendiger.

Im nächsten Schritt müssen Sie das Android-Plug-in für Eclipse nachinstallieren. Es ermöglicht Ihnen, die IDE zum Debuggen von Android-Applikationen einzusetzen, und rüstet auch einige andere Spezialfunktionen nach.

Erfreulicherweise bringt Eclipse einen Wizard zur Nachinstallation von Plug-ins und sonstigen Erweiterungen mit. Er findet sich im Menü unter *Help -> Install new Software*. Setzen Sie – wie in Abbildung 2.10 gezeigt – ins Feld *Work with:* die URL *https://dl-ssl.google.com/android/eclipse/* ein und drücken Sie Enter.

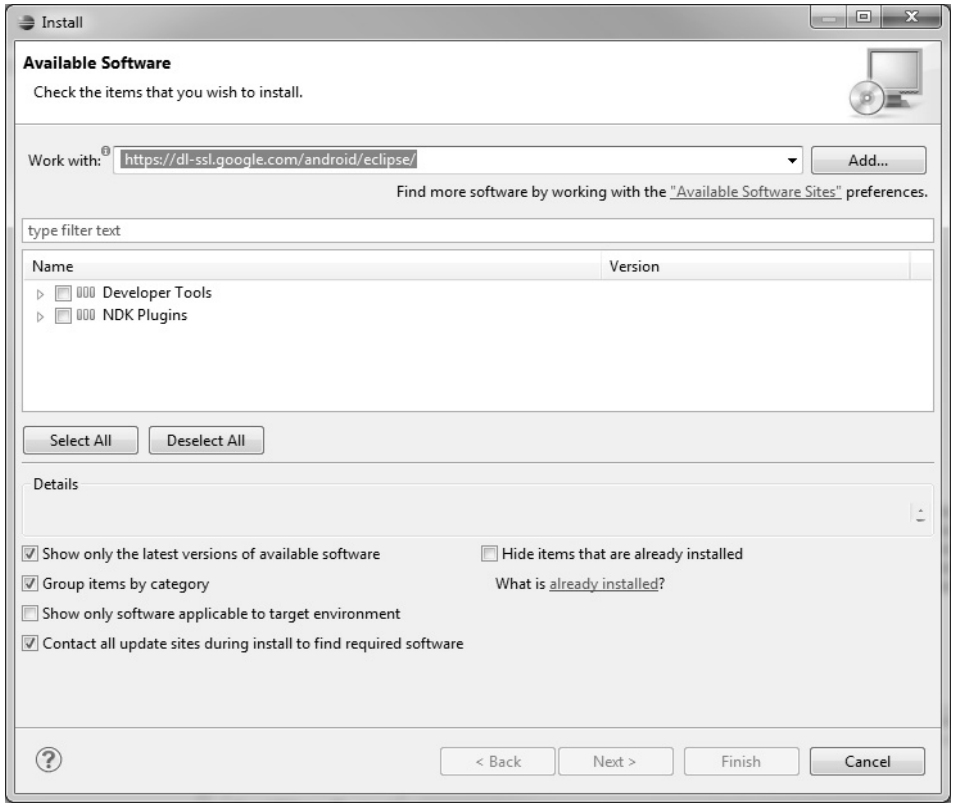


Bild 2.10: Eclipse hat die auf dem Server verfügbaren Module für Sie aufgelistet.

Markieren Sie alle in der Kategorie *Developer Tools* angebotenen Komponenten. Der Download des *OpenGL ES-Tracer* ist nicht erforderlich, sofern Sie keine 3-D-Applikationen erstellen möchten. Die Installation des NDK ist ebenfalls empfehlenswert.

Bevor Sie eine Anwendung entwickeln können, müssen Sie noch ein Platform-SDK herunterladen. Dabei handelt es sich um die Software, die die eigentliche Entwicklungsarbeit leistet – sie ist je nach Betriebssystemversion verschieden.

Der Download von Plattformen erfolgt über den Android SDK Manager, der sich über die in Abbildung 2.11 gezeigte Schaltfläche in Eclipse öffnen lässt.

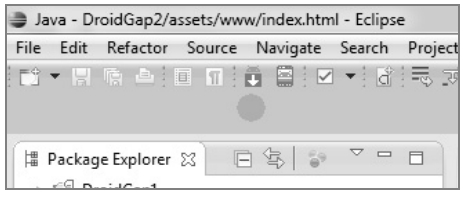


Bild 2.11: Nach der Installation des Plug-ins lässt sich der Android SDK Manager mit einem Klick öffnen.

Bei bestehender Internetverbindung zeigt er Ihnen sodann eine Liste der verfügbaren Plattformen an. Diese bestehen aus mehreren Komponenten – Abbildung 2.12 zeigt die vom Autor für dieses Kapitel verwendete Version 4.2 in ihren Einzelteilen.

Tipp: PhoneGap unterstützt relativ viele Android-Versionen ab 2.0. Die Abkündigung der einzelnen Versionen erfolgt anhand der von Google veröffentlichten Daten über den Marktanteil – sobald dieser unter 5 % fällt, betrachtet das Team die Plattformversion als »deprecated«.

Zum Zeitpunkt der Drucklegung sieht die Unterstützungsliste so aus:

x) Android 2.1 (Deprecated May 2013)

x) Android 2.2

x) Android 2.3

x) Android 3.x (Deprecated May 2013)

x) Android 4.x

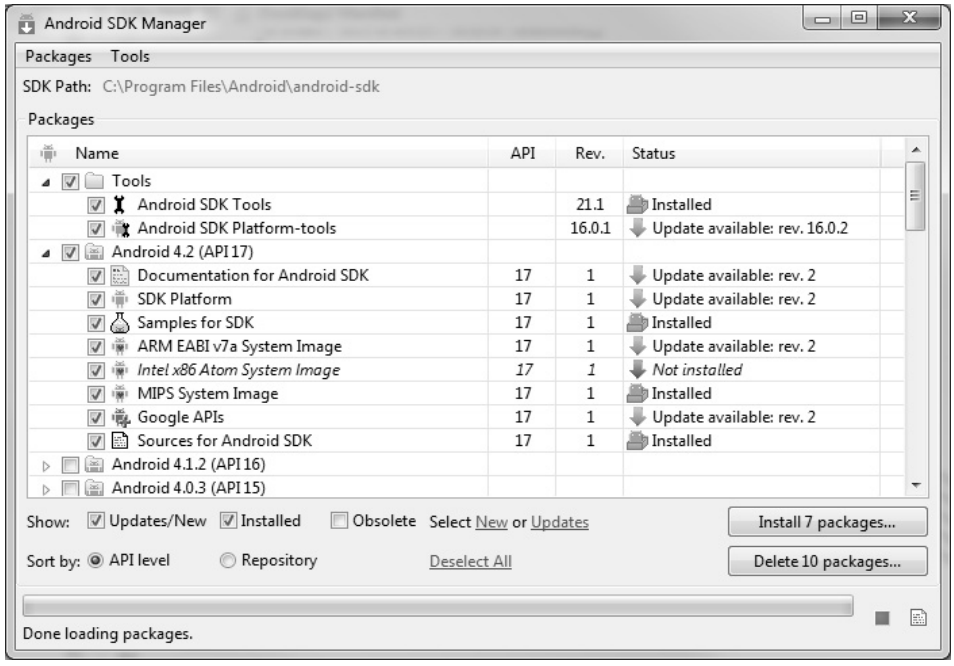


Bild 2.12: Alle mit dem Haken markierten Pakete sollten auf Ihre Maschine wandern.

Am wichtigsten ist die SDK-Plattform, die die Header, Bibliotheken und sonstigen für die Entwicklung von Anwendungen erforderlichen Komponenten enthält. Wenn Sie Ihre Programme auch im Emulator testen möchten, sollten Sie die System-Images

ebenfalls herunterladen. Google-APIs, Documentation, Samples und Sources sind nicht unbedingt erforderlich.

Nachdem Sie die zu installierenden Komponenten mit einem Häkchen markiert haben, klicken Sie auf *Install packages*. Im nächsten Schritt fragt der SDK Manager – wie in Abbildung 2.13 gezeigt – nach Ihrer Zustimmung zu den Lizenzbedingungen.

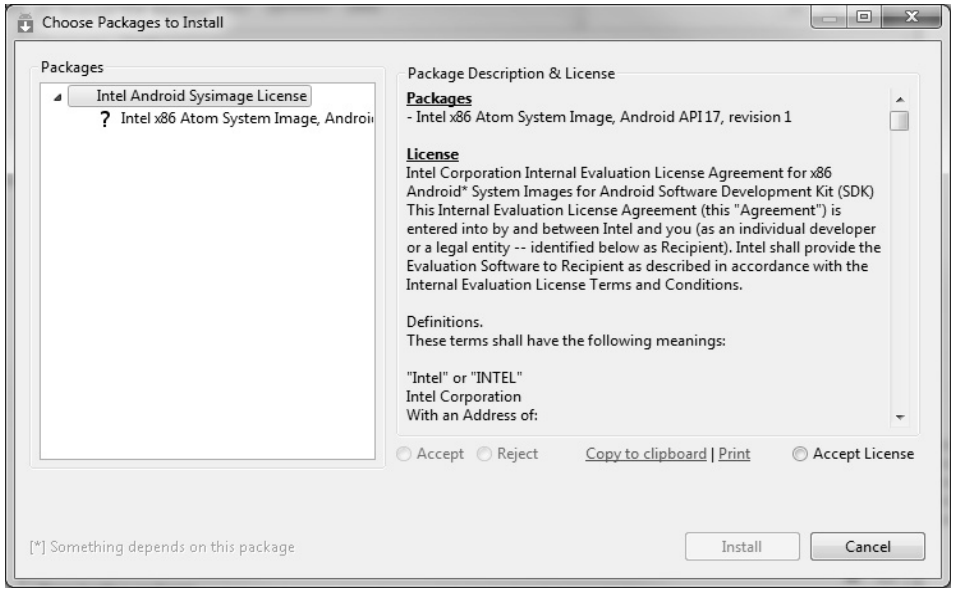


Bild 2.13: Hier müssen Sie den Lizenzbedingungen vor der Installation des Pakets zustimmen.

Sind alle Lizenzen abgenickt, beginnt der eigentliche Download. Dieser kann – je nach Geschwindigkeit Ihrer Internetverbindung – schon einmal einige Minuten Zeit in Anspruch nehmen. Danach sollten Sie den SDK Manager schließen und Eclipse neu starten – die Entwicklungsumgebung ist einsatzbereit.

Tipp: Unter Windows Vista/7/8 startet der SDK Manager manchmal nicht und gibt als Begründung fehlende Zugriffsrechte an. In diesem Fall hilft es, den SDK Manager aus dem Startmenü heraus per Rechtsklick mit Administratorrechten anzuwerfen.

Die Installation von PhoneGap ist vergleichsweise simpel. Besuchen Sie die Webseite des Projekts unter *http://phonegap.com/download/* und klicken Sie auf den prominent platzierten Download-Button.

Extrahieren Sie das Archiv (in diesem Kapitel kommt die Version 2.3 zum Einsatz) an eine für Sie leicht aufzufindende Stelle im Dateisystem Ihrer Workstation. Damit ist die Installation von PhoneGap auch schon abgeschlossen, und die Entwicklung von Anwendungen kann beginnen.

2.9 Projektskelett

PhoneGap-Anwendungen sind – zumindest auf den ersten Blick – Android-Anwendungen mit einigen besonderen Komponenten und Projekteinstellungen. Da das manuelle Erstellen der Projektskelette in Arbeit ausartet, gibt es unter PhoneGap 2.x ein spezielles Generatorprogramm.

Tipp: PhoneGap 3.0 ersetzt das hier besprochene manuelle Generatorprogramm durch die in Kapitel 7 zu den CLIs besprochenen Entwicklungswerkzeuge.

Zur Ausführung des Programms sind zwingend sowohl Ant als auch das soeben installierte Android-SDK erforderlich. Zusätzlich müssen die beiden Komponenten über Umgebungsvariablen ansprechbar sein.

Da das Android-SDK seit einiger Zeit ohne Ant ausgeliefert wird, führt der nächste Schritt zur Webseite des Build-Automatisierers (*http://ant.apache.org*). Dort klicken Sie auf *Download -> Binary Distribution*, um eine Binärdatei des Programms herunterzuladen.

Klicken Sie auf den in Abbildung 2.14 gezeigten Link neben dem Menüpunkt *.zip archive*, um die Datei herunterzuladen. Extrahieren Sie ihren Inhalt danach in einen für Sie gut erreichbaren Ordner (der Autor verwendet für die folgenden Schritte den Pfad *C:\Program Files\Android\apache-ant-1.8.4*).

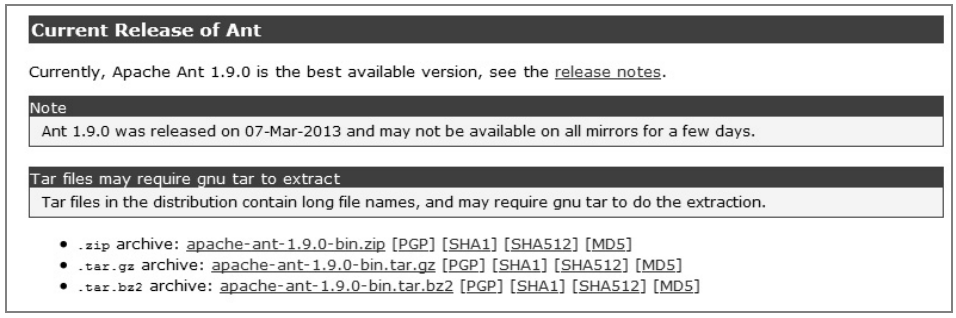


Bild 2.14: Der Download-Link für Ant ist hinter der Archivbezeichnung zu finden.

Der Generator lässt sich am einfachsten aus der Kommandozeile heraus bedienen. Dazu öffnen Sie im Startmenü *cmd.exe* mit Administratorrechten und navigieren danach in das Unterverzeichnis */lib/android/bin* des PhoneGap-Archivs.

Leider muss er vor der eigentlichen Ausführung noch parametriert werden. Dazu öffnen Sie die Datei *create.bat* in einem Texteditor und ergänzen die `PATH`-Deklarationen im folgenden Listing:

```
@ECHO OFF

SET PATH=%PATH%;c:\Program Files\Android\android-sdk\platform-tools\
```

```
SET PATH=%PATH%;c:\Program Files\Android\android-sdk\tools\
SET PATH=%PATH%;c:\Program Files\Android\apache-ant-1.8.4\bin\
SET ANT_HOME=c:\Program Files\Android\apache-ant-1.8.4\bin\
set PATH=%PATH%;C:\Program Files\Java\jdk1.7.0_09\bin\
set JAVA_HOME=C:\Program Files\Java\jdk1.7.0_09\bin\


IF NOT DEFINED JAVA_HOME GOTO MISSING
FOR %%X in (java.exe javac.exe ant.bat android.bat) do (
    SET FOUND=%%~$PATH:X
    IF NOT DEFINED FOUND GOTO MISSING
)
cscript "%~dp0\create.js" %*
GOTO END
:MISSING
ECHO Missing one of the following:
ECHO JDK: http://java.oracle.com
ECHO Android SDK: http://developer.android.com
ECHO Apache ant: http://ant.apache.org
EXIT /B 1
:END
```

Die mit `SET` beginnenden Zeilen haben die Aufgabe, den Generator an die Gegebenheiten auf Ihrer lokalen Maschine anzupassen. Je nach Systemkonfiguration sind die Pfade etwas anders, prüfen Sie daher die Korrektheit der Pfade am besten mit dem Windows Explorer von Hand nach.

Sobald alle Pfade korrekt festgelegt sind, dürfen Sie die Erstellung eines neuen Projekts mit folgendem Befehl anweisen:

```
C:\Program Files\Android\phonegap-2.3.0\lib\android\bin>create C:\Cordova
com.tamoggemon.franzispg.example1 DroidGap1
Microsoft (R) Windows Script Host Version 5.8
Copyright (C) Microsoft Corporation. All rights reserved.

Creating new android project...
Copying template files...
Copying js, jar & config.xml files...
Copying cordova command tools...
Updating AndroidManifest.xml and Main Activity...
```

Aufrufe von *create.bat* folgen immer der gleichen Syntax. Als ersten Parameter verlangt das Werkzeug den Pfad, in dem das Projektskelett gespeichert werden soll. Der zweite Parameter ist der Paketname, der später als »Identifier« der fertigen Anwendung dient. Parameter Nummer drei ist der Projektname.

Tipp: Android-Anwendungen sind im Prinzip nichts anderes als »komplexe Java-Pakete«. Sie werden durch ihre Paket-ID identifiziert – es ist üblich, diese nach normalen Java-Konventionen aus der umgekehrten Domain (in diesem Fall *com.tamoggemon*) und dem Projektnamen zusammenzubauen.

Nachdem Sie den oben angegebenen Befehl in einer mit Administratorrechten gestarteten Kommandozeile ausgeführt haben, finden Sie im Verzeichnis *C:\Cordova* ein Projektskelett.

Starten Sie im nächsten Schritt Eclipse über die Datei *eclipse.exe* – im Rahmen des Startprozesses wird der in Abbildung 2.15 gezeigte Bildschirm erscheinen, in dem die IDE nach dem Arbeitsordner fragt.

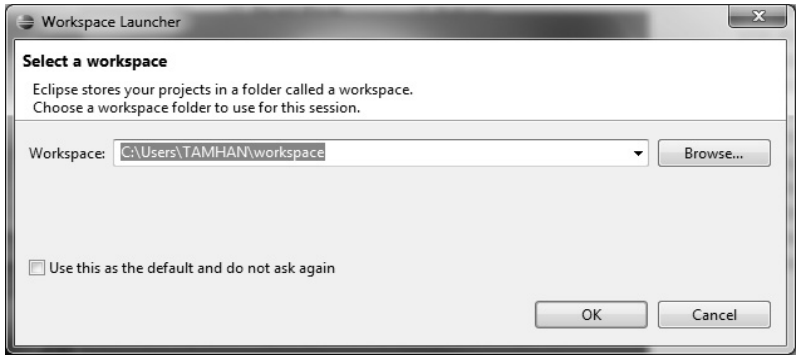


Bild 2.15: Eclipse verrät sein Arbeitsverzeichnis bereitwillig.

Notieren Sie diesen Ordner. Kopieren Sie danach den Ordner *Cordova* von *C:* in den Workspace der Entwicklungsumgebung. Im nächsten Schritt starten Sie den Wizard zur Erstellung von neuen Projekten mit *File -> New -> Project…* Er fragt, wie in Abbildung 2.16 gezeigt, nach dem Typ des zu erstellenden Projekts.

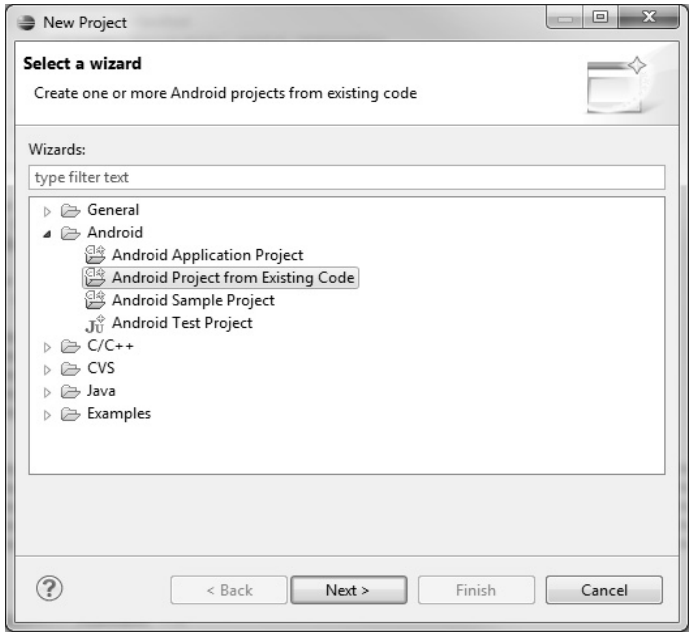


Bild 2.16: Unser Projekt soll aus existierendem Code entstehen.

Wählen Sie *Android Project from Existing Code*, um den eigentlichen Importassistenten zu starten. Danach navigieren Sie in den Ordner *Cordova* und aktivieren die Checkbox *Copy projects into workspace*. Vergewissern Sie sich, dass der Dialog wie der in Abbildung 2.17 aussieht – danach genügt ein Klick auf *Finish*, um das Projekt in die IDE zu laden.

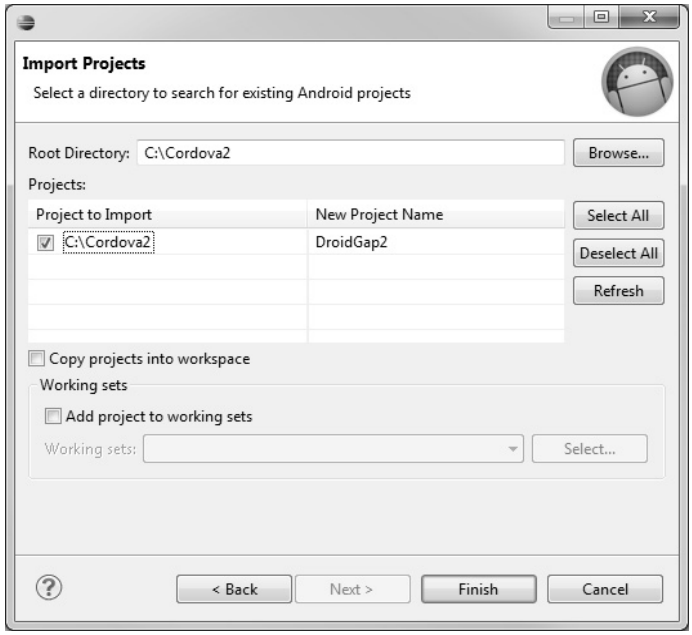


Bild 2.17: Unser neues Projekt wird vom Importwizard erkannt.

In der Regel läuft der Import ohne weitere Probleme ab. Leider treten manchmal seltsame Fehler auf, die sich aber so gut wie immer durch die Menüoption *Project -> Clean…* beheben lassen.

An sich handelt es sich beim vom Generator erstellten Programm um ein normales Android-Projekt. Der erste Unterschied liegt darin, dass die Konfiguration um ein *.jar*-File namens *cordova-2.3.0.jar* erweitert ist. Dieses enthält die Intelligenz des Frameworks samt dem vorher besprochenen Loader.

Die Activity *DroidGap1.java* hat die Aufgabe, den von PhoneGap bereitgestellten »Webbrowser« mit Inhalten zu füttern. Das ist klar im Quellcode der Datei *DroidGap1.java* zu sehen.

```
package com.tamoggemon.franzispg.example1;

import android.app.Activity;
import android.os.Bundle;
import org.apache.cordova.*;

public class DroidGap1 extends DroidGap
{
    @Override
    public void onCreate(Bundle savedInstanceState)
    {
        super.onCreate(savedInstanceState);
        super.loadUrl("file:///android_asset/www/index.html");
    }
}
```

An dieser Stelle sehen wir einige Passagen, die insbesondere für erfahrene Android-Entwickler alles andere als gewohnt sind. Die Klasse `DroidGap` wurde vom Generatorwerkzeug für unser Projekt vorkompiliert und enthält die eigentliche Intelligenz des Frameworks. Der Aufruf von `super.loadUrl` weist sie an, die im angegebenen Pfad befindlichen Dateien zu laden.

Tipp: Die Layoutdatei *main.xml* wird bei der Erstellung des Formulars nicht aufgeblasen – sie enthält nur irrelevanten Dummycode.

Das Herz unserer Anwendung findet sich, wie in Abbildung 2.18 gezeigt, im *assets*-Ordner unseres Projekts. Dort gibt es neben der im Quellcode oben referenzierten Datei *index.html* auch einige Ordner mit diversen anderen Dateien, die zur Realisierung von HTML5-Anwendungen erforderlich sind.

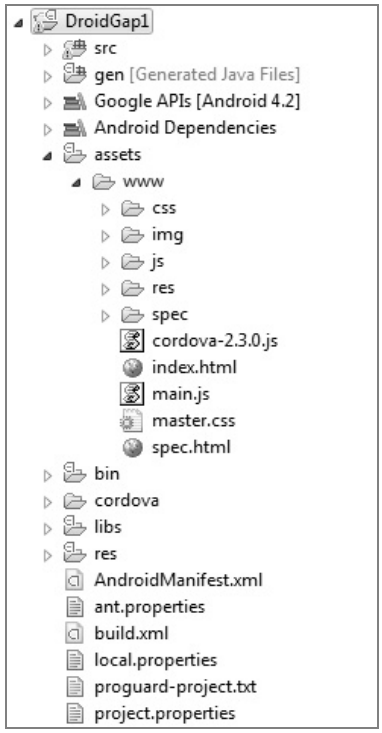


Bild 2.18: Die eigentliche »Intelligenz« ist im *assets*-Ordner zu finden.

2.9.1 index.html

Als Erstes wollen wir uns der Datei *index.html* zuwenden. Klicken Sie sie im Projektexplorer mit der rechten Maustaste an und wählen Sie danach die Option *Open With -> Text Editor*, um sie zur Bearbeitung zu öffnen.

Der Beginn der Datei entspricht weitgehend einer normalen HTML-Datei mit einigen Metadeklarationen:

```
<html>
    <head>
        <meta http-equiv="Content-Type" content="text/html; charset=UTF-8" />
        <meta name="format-detection" content="telephone=no" />
        <meta name="viewport" content="user-scalable=no, initial-scale=1,
        maximum-scale=1, minimum-scale=1, width=device-width, height=device-
        height, target-densitydpi=device-dpi" />
        <link rel="stylesheet" type="text/css" href="css/index.css" />
        <title>Hello World</title>
    </head>
```

Da wir unsere Anwendung im Single-Page-Pattern realisieren, folgt im nächsten Schritt der »Korpus« der Applikation. Zusätzlich enthält er einen `<div>` und zwei `<p>`-Abschnitte, die die jeweiligen »Formulare« enthalten werden.

```
    <body>
        <div class="app">
            <h1>Apache Cordova</h1>
            <div id="deviceready" class="blink">
                <p class="event listening">Connecting to Device</p>
                <p class="event received">Device is Ready</p>
            </div>
        </div>
```

Zu guter Letzt lädt die HTML-Datei einige zum Framework gehörende Skriptdateien und führt die Methode `app.initialize()` aus. Diese ist in *index.js* deklariert und initialisiert das Formular.

```
        <script type="text/javascript" src="cordova-2.3.0.js"></script>
        <script type="text/javascript" src="js/index.js"></script>
        <script type="text/javascript">
            app.initialize();
        </script>
    </body>
</html>
```

Tipp: Von Haus aus öffnet Eclipse *.html*- und *.js*-Dateien nach einem Doppelklick im Webbrowser.

2.9.2 index.js

Die zur soeben beschriebenen HTML-Datei gehörende Intelligenz findet sich in der Datei *index.js*, die im Unterordner */js/* des *assets*-Ordners liegt.

In unserem Beispiel beginnt sie mit der Deklaration des globalen App-Objekts über ein klassisches JSON-Konstrukt. Sein Konstruktor hat nur die Aufgabe, die Methode `bindEvents` aufzurufen:

```
var app = {
    // Application Constructor
    initialize: function()
    {
        this.bindEvents();
    },
```

`bindEvents` hat die Aufgabe, die Bindung zu den diversen Events herzustellen. In diesem Fall interessieren wir uns nur für das `deviceready`-Event, dem hier der Handler `onDeviceReady` zugewiesen wird.

```
    // Bind Event Listeners
    //
    // Bind any events that are required on startup. Common events are:
    // 'load', 'deviceready', 'offline', and 'online'.
```

```
    bindEvents: function()
    {
        document.addEventListener('deviceready', this.onDeviceReady, false);
    },
```

Im eigentlichen Handler finden wir eine kleine Besonderheit. Während der Ausführung des Codes verweist die `this`-Variable auf das eingehende Ereignis. Aus diesem Grund müssen wir `app` vor den Aufruf von `receivedEvent` schreiben – nur so ist sichergestellt, dass die Methode `receivedEvent` unserer Applikation aufgerufen wird:

```
    // deviceready Event Handler
    //
    // The scope of 'this' is the event. In order to call the
    // 'receivedEvent'
    // function, we must explicity call 'app.receivedEvent(...);'
    onDeviceReady: function()
    {
        app.receivedEvent('deviceready');
    },
```

Die eigentliche Intelligenz hinter der Eventverarbeitung findet sich in `receivedEvent`. Die Methode beginnt damit, einige Elemente einzusammeln – speziell sucht sie nach dem Elternelement, das für das jeweilige Element zuständig ist (in unserem Fall ist es das `<div>`-Tag in *index.html*).

```
    // Update DOM on a Received Event
    receivedEvent: function(id)
    {
        var parentElement = document.getElementById(id);
```

Im nächsten Schritt sucht die Methode nach Unterelementen, auf die die jeweiligen Parameter zutreffen – in diesem Fall sind das die beiden `<p>`-Tags. Jedem der beiden wird dann ein Stil aus der *.css*-Datei zugewiesen, zu guter Letzt erfolgt eine Meldung in die Kommandozeile:

```
        var listeningElement = parentElement.querySelector('.listening');
        var receivedElement = parentElement.querySelector('.received');

        listeningElement.setAttribute('style', 'display:none;');
        receivedElement.setAttribute('style', 'display:block;');

        console.log('Received Event: ' + id);
    }
};
```

2.9.3 cordova-2.3.0.js

Die Datei *cordova-2.3.0.js* enthält den eigentlichen Code des Frameworks, sie wird in der Regel vom Entwickler nicht modifiziert.

2.9.4 main.js

Die *main.js* enthält einige Beispielroutinen, die die PhoneGap-API grob vorführen. Wenn Sie eine der Methoden testen wollen, müssen Sie sie die Datei in *index.html* einbinden.

Außerdem müssen Sie einen Button realisieren, der die jeweilige Methode aufruft. Manche Routinen verlangen zusätzlich nach einigen Steuerelementen im Formular, in denen die errechneten Informationen ausgegeben werden.

```
var deviceInfo = function() {
    document.getElementById("platform").innerHTML = device.platform;
    document.getElementById("version").innerHTML = device.version;
    document.getElementById("uuid").innerHTML = device.uuid;
    document.getElementById("name").innerHTML = device.name;
    document.getElementById("width").innerHTML = screen.width;
    document.getElementById("height").innerHTML = screen.height;
    document.getElementById("colorDepth").innerHTML = screen.colorDepth;
};

var getLocation = function() {
    var suc = function(p) {
        alert(p.coords.latitude + " " + p.coords.longitude);
    };
    var locFail = function() {
    };
    navigator.geolocation.getCurrentPosition(suc, locFail);
};

var beep = function() {
    navigator.notification.beep(2);
};

var vibrate = function() {
    navigator.notification.vibrate(0);
};

function roundNumber(num) {
    var dec = 3;
    var result = Math.round(num * Math.pow(10, dec)) / Math.pow(10, dec);
    return result;
}

var accelerationWatch = null;

function updateAcceleration(a) {
    document.getElementById('x').innerHTML = roundNumber(a.x);
    document.getElementById('y').innerHTML = roundNumber(a.y);
    document.getElementById('z').innerHTML = roundNumber(a.z);
```

```
}

var toggleAccel = function() {
    if (accelerationWatch !== null) {
        navigator.accelerometer.clearWatch(accelerationWatch);
        updateAcceleration({
            x : "",
            y : "",
            z : ""
        });
        accelerationWatch = null;
    } else {
        var options = {};
        options.frequency = 1000;
        accelerationWatch = navigator.accelerometer.watchAcceleration(
                updateAcceleration, function(ex) {
                    alert("accel fail (" + ex.name + ": " + ex.message + ")");
                }, options);
    }
};

var preventBehavior = function(e) {
    e.preventDefault();
};

function dump_pic(data) {
    var viewport = document.getElementById('viewport');
    console.log(data);
    viewport.style.display = "";
    viewport.style.position = "absolute";
    viewport.style.top = "10px";
    viewport.style.left = "10px";
    document.getElementById("test_img").src = data;
}

function fail(msg) {
    alert(msg);
}

function show_pic() {
    navigator.camera.getPicture(dump_pic, fail, {
        quality : 50
    });
}

function close() {
    var viewport = document.getElementById('viewport');
    viewport.style.position = "relative";
```

```
    viewport.style.display = "none";
}

function contacts_success(contacts) {
    alert(contacts.length
            + ' contacts returned.'
            + (contacts[2] && contacts[2].name ? (' Third contact is ' +
contacts[2].name.formatted)
                    : ''));
}

function get_contacts() {
    var obj = new ContactFindOptions();
    obj.filter = "";
    obj.multiple = true;
    navigator.contacts.find(
            [ "displayName", "name" ], contacts_success,
            fail, obj);
}

function check_network() {
    var networkState = navigator.network.connection.type;

    var states = {};
    states[Connection.UNKNOWN]  = 'Unknown connection';
    states[Connection.ETHERNET] = 'Ethernet connection';
    states[Connection.WIFI]     = 'WiFi connection';
    states[Connection.CELL_2G]  = 'Cell 2G connection';
    states[Connection.CELL_3G]  = 'Cell 3G connection';
    states[Connection.CELL_4G]  = 'Cell 4G connection';
    states[Connection.NONE]     = 'No network connection';

    confirm('Connection type:\n ' + states[networkState]);
}

var watchID = null;

function updateHeading(h) {
    document.getElementById('h').innerHTML = h.magneticHeading;
}

function toggleCompass() {
    if (watchID !== null) {
        navigator.compass.clearWatch(watchID);
        watchID = null;
        updateHeading({ magneticHeading : "Off"});
    } else {
        var options = { frequency: 1000 };
```

```
        watchID = navigator.compass.watchHeading(updateHeading, function(e)
{
            alert('Compass Error: ' + e.code);
        }, options);
    }
}

function init() {
    // the next line makes it impossible to see Contacts on the HTC Evo since
    // it doesn't have a scroll button
    // document.addEventListener("touchmove", preventBehavior, false);
    document.addEventListener("deviceready", deviceInfo, true);
}
```

Die genaue Beschreibung der Routinen erfolgt in den folgenden Kapiteln, die auf die einzelnen Teile des Frameworks im Detail eingehen.

2.10 Anwendung testen

Nach diesen theoretischen Betrachtungen ist es an der Zeit, unsere Applikation zu testen. Dazu stehen unter Android – wie auch auf den meisten anderen Mobilplattformen – zwei verschiedene Vorgehensweisen zur Verfügung.

Als Erstes gibt es einen Emulator, der das Programm auf der Arbeitsstation in einer virtuellen Maschine ausführt. Die zweite Option – in Entwicklerkreisen auch als ODD (kurz für On-Device-Debugging) bezeichnet – führt die Applikation direkt auf einem dafür konfigurierten Telefon aus.

Tipp: Je nach Plattform ist die eine oder die andere Vorgehensweise zielführender. Da der Android-Emulator sehr langsam arbeitet, ist hier das Verwenden von realer Hardware vorzuziehen.

2.10.1 Test im Emulator

Die preisgünstigste und einfachste Methode zum Testen einer Android-Applikation führt mit Sicherheit über den zur Plattform gehörenden Emulator. Er verwendet System-Images, die Sie, wie im Abschnitt zur Installation beschrieben, herunterladen und dann nutzen können.

Eclipse assistiert Ihnen bei der Verwaltung von virtuellen Maschinen. Der Android Virtual Device Manager lässt sich aus Eclipse über das in Abbildung 2.19 gezeigte Icon öffnen.

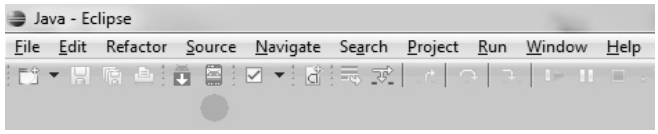


Bild 2.19: Auch die Erstellung von virtuellen Maschinen beginnt in Eclipse.

Ist das Snap-in geladen, präsentiert es eine Liste der verfügbaren virtuellen Maschinen. Um eine neue anzulegen, klicken Sie auf den *New*-Button.

Die IDE reagiert daraufhin mit dem in Abbildung 2.20 gezeigten Bildschirm.

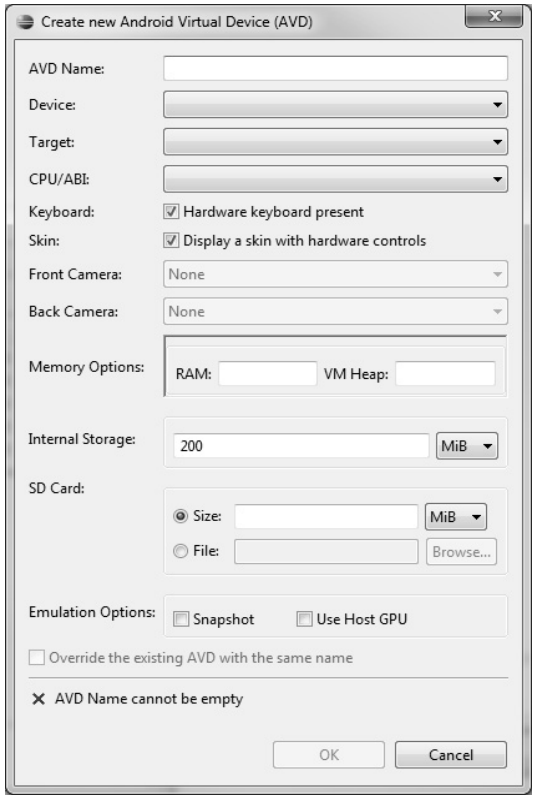


Bild 2.20: Dieser Dialog dient der Erstellung einer neuen Virtual Machine (VM).

Das Feld *AVD Name* dient nur zum Festlegen eines Namens für die virtuelle Maschine. Dieser sollte sprechend sein, sodass Sie die VM später wiederfinden – für das System selbst ist er ohne tieferen Belang.

In der Rubrik *Device* legen Sie fest, welcher Telefontyp emuliert werden soll. Darunter findet sich die *Target*-Checkbox, in der sie das zu verwendende Betriebssystem-Image auswählen. Wichtig ist, dass Sie die Version mit der Google-API auswählen.

In der Rubrik *CPU/ABI* können Sie entweder *ARM* oder *x86* auswählen – die angebotenen Optionen hängen davon ab, welche Images Sie auf Ihrer Maschine installiert haben.

Tipp: Wenn Sie auf einem X86-Prozessor mit Virtualisierungsfunktionen arbeiten, ist ein X86-Image aus Geschwindigkeitsgründen die beste Wahl.

Die Felder *RAM*, *VM Heap* und *Internal Storage* können Sie nach Belieben auswählen. Für RAM sind 300 MB ausreichend, 200 MB Internal Storage sind in der Regel ebenfalls genug. Schließen Sie danach sowohl den Dialog als auch den (modalen!) Android Virtual Device Manager, um zur IDE zurückzukehren.

Tipp: Zum Zeitpunkt der Drucklegung behindert die Option *Use Host GPU* den Webbrowser der virtuellen Maschine. Da PhoneGap-Applikationen in ebendiesem ablaufen, ist die Option unbedingt zu deaktivieren!

Eclipse arbeitet mit Debugger-Konfigurationen. Dabei handelt es sich um spezielle Profile, die die für eine Anwendung zu nutzenden Einstellungen für Debugger und Emulation zusammenfassen.

Leider hat unser Projektskelett noch keine derartige Konfiguration. Aus diesem Grund öffnen wir im nächsten Schritt eine beliebige Datei des Projekts und klicken danach auf das *Debug*-Icon in der Toolbar der IDE.

Eclipse präsentiert Ihnen daraufhin eine Liste mit verschiedenen Debugger-Konfigurationen. Wählen Sie *Android Application* und klicken Sie auf *OK* – die IDE startet daraufhin automatisch den Emulator und lädt die Beispielapplikation. Dieser Vorgang dauert in der Regel einige Minuten.

2.10.2 Test am realen Endgerät

Wenn Sie Ihre Anwendung lieber auf einem echten Android-Handy ausprobieren möchten, so ist auch das kein Problem. Im ersten Schritt müssen Sie dazu den Developer Mode aktivieren – dieser Prozess erfolgt je nach Gerät unterschiedlich.

Wir werden hier aus Anschauungsgründen die Vorgehensweise auf einem unter Android 4.0 laufenden Endgerät betrachten. Die meisten Systeme verhalten sich weitgehend ähnlich, im Zweifelsfall hilft Google weiter.

Trennen Sie die USB-Verbindung zwischen Ihrem Rechner und dem Telefon. Öffnen Sie danach am Handy die Anwendung *Einstellungen* und navigieren Sie dort in die Rubrik *Entwickleroptionen*. Dort finden Sie die in Abbildung 2.21 gezeigte Option *USB-Debugging*.

Bild 2.21: *USB-Debugging* muss aus Sicherheitsgründen von Hand aktiviert werden.

Nach der Aktivierung erscheint auf den meisten Telefonen eine Sicherheitswarnung. Quittieren Sie diese, um das Handy in den Entwicklermodus zu versetzen – danach schließen Sie es wieder an Ihre Arbeitsstation an.

Tipp: Ab Android 4.2 sind die Entwickleroptionen versteckt, um User daran zu hindern, ihr Telefon aus Versehen in den Developer Mode zu versetzen. Das Menü erscheint, nachdem Sie in der Rubrik *About Phone* der *Einstellungen* siebenmal auf den Unterpunkt *Build Number* geklickt haben.

Im nächsten Schritt klicken Sie wieder auf *Debug*. Sofern Sie noch keine Debugger-Konfiguration erstellt haben, folgt nun der weiter oben beschriebene Prozess. Danach überträgt Eclipse Ihr Programm automatisch auf das Telefon, wo Sie es testen können.

2.11 Anwendung ausliefern

Als offene Plattform ist Android – anders als die meisten Systeme – nicht an den Store des Plattformentwicklers gebunden. Android-Apps kommen als *.apk*-Datei auf das Handy des Users – wo die Datei herkommt, ist dem System vergleichsweise egal. Wichtig ist nur, dass die Datei mit einem weltweit einzigartigen Schlüssel signiert ist – er hat die Aufgabe, dem Betriebssystem das Zuordnen von zukünftigen Updates zu ermöglichen.

Die Erstellung eines derartigen Schlüssels bewerkstelligen Sie mit dem Programm *keytool.exe*, das normalerweise im Rahmen der Installation des Java Development Kit auf Ihre Maschine kommt und deshalb schon verfügbar sein sollte – suchen Sie es notfalls mit der Suchen-Funktion Ihres Betriebssystems.

Zur Erstellung des Schlüssels rufen Sie es dann mit den folgenden Parametern auf. Das Programm fragt Sie nach einigen Informationen und generiert danach die angewiesene Schlüsseldatei – in unserem Beispiel heißt sie *xample.keystore*:

```
keytool -genkey -v -keystore xample.keystore -alias publishingdoc -keyalg
RSA -keysize 2048 -validity 10000
```

Tipp: Die von `keytool` erstellte Datei darf keinesfalls verloren gehen, da Sie sonst keine Updates mehr ausliefern können. Allerdings sollten Sie sie auch nicht ins Internet stellen – wer das File samt dem dazugehörenden Passwort besitzt, kann Programme in Ihrem Namen ausliefern.

Bevor Sie Ihr Programm in die diversen Stores hochladen, sollten Sie es noch um ein Applikations-Icon ergänzen – das von PhoneGap erstellte Icon sticht nicht aus der Masse heraus und sieht dementsprechend unprofessionell aus.

Das Anpassen des Programmsymbols ist nicht allzu schwer. Je nach Auflösung des Bildschirms kommt eine andere Version der Datei *icon.png* zum Einsatz. Durchsuchen Sie Ihr Projekt einfach nach allen Vorkommnissen von *icon.png* und ersetzen Sie diese durch ein von Ihnen generiertes Symbol mit der jeweils passenden Auflösung.

Nachdem Sie die Symboldateien ausgetauscht haben, müssen Sie das Programm für die Auslieferung verpacken. Dabei assistiert Ihnen der Exportwizard der IDE, den Sie im Project Explorer durch einen Rechtsklick auf den Projektnamen und die Option *Android Tools -> Export Signed Application Package* finden. Er fragt Sie in den folgenden Schritten nach Informationen und Einstellungen und generiert danach eine auslieferbare Version Ihrer Applikation.

Tipp: Sollten Ihnen die hier besprochenen Informationen zum Signiersystem nicht ausreichen, finden Sie weitere Daten unter *http://developer.android.com/tools/publishing/app-signing.html*.

Als Resultat liefert Ihnen der Signierwizard eine *.apk*-Datei. Diese ist in so gut wie jedem Store ein gern gesehener Gast – weitere Informationen zu den Veröffentlichungsprozessen entnehmen Sie bitte der Dokumentation des jeweiligen Distributors.

2.12 Zusammenfassung

Nach den Betrachtungen in diesem Kapitel sollten Sie einen guten Überblick über die Rolle der PhoneGap-Plattform haben. Auch wenn wir uns hier nur mit Android beschäftigt haben, lassen sich die gezeigten Konzepte auch auf alle anderen von PhoneGap unterstützten Plattformen übertragen.

Stichwortverzeichnis

Tam Hanna

PhoneGap 3

Apps für iOS, Android, Windows Phone & Co. entwickeln

Der Traum eines jeden Mobile-Entwicklers: Eine Codebasis für alle Plattformen! Mit PhoneGap und einigen Kenntnissen in JavaScript ist dies möglich! Das Framework ist dabei nicht nur ein Werkzeug zur Verpackung von Web-Apps für die unterschiedlichen Betriebssysteme. Lernen Sie in diesem Buch, für welche Hardwarefunktionen PhoneGap Schnittstellen bereitstellt und wie Sie mit dem Einbau eines nativen Plug-ins Ihrer App alle gewünschten zusätzlichen Funktionen zugänglich machen.

Volle Portabilität

Es gibt kaum ein Betriebssystem, das sich mit PhoneGap nicht versteht. Dieses Buch zeigt Ihnen, wie Sie Ihrer PhoneGap-App mit minimalem Aufwand zu maximaler Reichweite verhelfen.

Die PhoneGap-API

Nutzen Sie alle in PhoneGap enthaltenen Namespaces aus, um Ihrer App neue und innovative Funktionen zu verpassen. Dieses Buch enthält eine kompakte Übersicht, die Ihnen alle wichtigen Module anhand praktischer Beispiele vorstellt.

PhoneGap 3

Die neue Version PhoneGap 3 bringt Kommandozeilenwerkzeuge mit, die Ihnen das Hantieren mit der nativen IDE ersparen. Mit PhoneGap Build wird App-Entwicklung zum Kinderspiel!

Diverse Tools

Ein Werkzeug kommt selten allein. Richtig Spaß macht die Entwicklung von PhoneGap-Apps erst im Zusammenspiel diverser Hilfsmittel – wir stellen Ihnen die besten vor!

Aus dem Inhalt:

Über den Autor:

Tam Hanna lebt in der Slowakei und leitet dort die Tamoggemon Holding k.s. Das Unternehmen beschäftigt sich mit Consulting, Anwendungsentwicklung und dem Verfassen von Fachtexten für die IT-Industrie. Seit 2004 liegt der Schwerpunkt von Tams Tätigkeit im Bereich der Mobilcomputer. Er verfolgt die Industrie seit dem Palm IIIc, schreibt für diverse Magazine und hält Vorträge auf Kongressen. Tam bloggt regelmäßig auf heise.de.

Auf www.buch.cd

Der komplette Quellcode des Buchs

9 783645 601535

30,– EUR [D] **/ 30,90 EUR** [A]

ISBN 978-3-645-60153-5

Besuchen Sie unsere Website **www.franzis.de**

FRANZIS