Markus Spiering / Sven Haiges

Know-how ist blau.

2. aktualisierte Auflage

HTML5-Apps
für iPhone und Android

> So gestalten Sie das „Look and Feel“ für iPhone- und Android-Apps mit HTML5 und CSS

> Bauen Sie mit JavaScript aktuelle Geodaten, Videos und Grafiken in Ihre Apps ein!

> Wandeln Sie Ihre Apps mit PhoneGap in native Applikationen für das iPhone und Android-Smartphones um!

FRANZIS

Markus Spiering / Sven Haiges

HTML 5-Apps für iPhone und Android entwickeln

Markus Spiering / Sven Haiges

HTML 5-Apps

für iPhone und Android entwickeln

2. aktualisierte Auflage

Mit 214 Abbildungen

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.

Herausgeber: Ulrich Dorn
Satz: DTP-Satz A. Kugge, München
art & design: www.ideehoch2.de
Druck: Bercker, 47623 Kevelaer
Printed in Germany

Einführung

Mit der Einführung des Apple iPhones im Jahr 2007 hat sich die mobile Welt komplett geändert: Internetdienste werden von einer großen Masse wie selbstverständlich genutzt, und der mobile Markt hat sich, anstatt sich auf eine oder einige wenige Plattformen zu konsolidieren, in Höchstgeschwindigkeit weiter fragmentiert. Apple hat mit dem iPhone und dem iPhone OS ein Erfolgsmodell geschaffen und ein unvergleichliches Wettrennen der mobilen Plattformen eingeläutet.

Einige Jahre nach dem ersten iPhone haben wir folgende Situation: Die Verbindung von iPhone und iTunes hat ein Verkaufsmodell für mobile Anwendungen geschaffen, das Hunderttausende von Anwendungen für die iOS-Plattform hervorgebracht hat und Entwickler begeistert. Microsoft ist fast schon in der mobilen Bedeutungslosigkeit verschwunden und versucht, mit Windows Phone 7 wieder Marktanteile zu gewinnen. Handy-Weltmarktführer Nokia hat sich jahrelang auf seine Symbian-Plattform verlassen, unterstützt diese zwar weiter, entwickelt aber auch neue Multimedia-Telefone auf der Linux-basierten MeeGo-Plattform. BlackBerry hat keine andere Plattform lizenziert, sondern setzt weiterhin auf sein eigenes System. Palm, schon vor einigen Jahren tot geglaubt, hat zahlreiche Apple-Designer und -Techniker abgeworden und mit WebOS ein unglaublich performantes und zukunftweisendes mobiles Betriebssystem auf den Markt gebracht. Leider wird es bislang nur von Palm selbst benutzt, wobei der neue Eigentümer Hewlett-Packard hier sicher für weitere Verbreitung sorgen kann. Trotz dieser Plattformvielfalt gab Google im November 2007 bekannt, gemeinsam mit 33 Mitgliedern der sogenannten Open Handset Alliance ein weiteres mobiles Betriebssystem namens Android zu entwickeln.

Ohne die hausgemachten Systeme von Samsung, LG und anderen Herstellern, besonders aus dem asiatischen Raum, zu erwähnen, kommen wir mit dem Apple iOS, Android, Windows Phone 7, BlackBerry, Symbian, MeeGo, Web OS und mobilen Linux-Versionen auf acht verschiedene nennenswerte Handyplattformen.

Zur gleichen Zeit hat sich mit dem iPhone die Handynutzung verändert: Während man Telefone zuvor größtenteils zum Telefonieren, Verfassen von Textnachrichten oder zum Checken von E-Mails verwendete, wurde das iPhone als erstes Telefon serienmäßig mit einem Browser ausgeliefert, der dem eines Browsers auf dem PC in nichts nachsteht und mit dem App-Model und der iTunes-Anbindung den angestaubten Markt von mobilen Anwendungen auf den Kopf gestellt hat. Ein Verkaufsargument für Handys ist heute nicht mehr nur der bessere Bildschirm oder die bessere Kamera, sondern auch, welche Art von Diensten ich auf dem Gerät bzw. der Plattform nutzen kann. Kann ich das Telefon mit iTunes verbinden und meine Musik synchronisieren? Kann ich meinen Facebook-Status aktualisieren, meine Fotos direkt auf Flickr laden und mit meinen Freunden und Bekannten direkt vom Telefon aus teilen? Wie viele Programme und auch Spiele gibt es eigentlich für mein Telefon?

Und hier beginnt genau das Problem für Handyhersteller, für Anwender und auch für Anbieter von Programmen und Diensten. Alle Hersteller, außer derzeit Apple, haben Probleme, im großen Rahmen Entwickler für ihre Plattform zu finden und zu begeistern. Waren mobile Softwareentwickler vor ein paar Jahren noch kaum beachtet, so sind sie jetzt heiß umworbene Dienstleister. Alle Systemhersteller betreiben einen riesigen Aufwand, um Entwickler davon zu überzeugen, mit ihrem System zu arbeiten und Programme und Dienste zu schreiben.

Gleichzeitig müssen sich Anbieter von Diensten und Programmierer die Frage stellen, auf welche Plattformen sie sich konzentrieren sollen. Alle Plattformen sind grundverschieden, und Code, der für das iPhone geschrieben wird, kann nicht einfach für Android oder das Symbian-Betriebssystem wiederverwendet werden. Die Entwicklung von mobilen Diensten ist für Firmen notwendig, da der mobile Markt viel verspricht, aber bislang nur wenig abwirft. Gleichzeitig ist die Entwicklung von mobilen Programmen sehr teuer, sollte von mobilen Softwarespezialisten vorgenommen werden und verursacht allein durch das Testen auf verschiedenen Telefonen hohe Kosten. Keine Firma wird so einfach bereit sein, gleichzeitig in alle acht Plattformen zu investieren, sondern sich höchstens zwei oder maximal drei unterschiedliche Versionen leisten. Doch es gibt Hoffnung: Neben großen, hochauflösenden Bildschirmen und tollen Kameras werden alle Handys in naher Zukunft eine Gemeinsamkeit aufweisen: unwahrscheinlich gute Webbrowser, die sich an die neuesten Standards richten. Der neueste Webstandard, der von den gängigen Webbrowsern mehr und mehr unterstützt wird, ist HTML5.

Glücklicherweise wird sich HTML5 von der HTML 4-Spezifikation insbesondere dadurch unterscheiden, dass sich mit der neuen Version Webdienste erstellen lassen, die nicht nur wie richtige Programme aussehen, sondern sich auch wie Programme verhalten. Auf Inhalte einer Webseite kann auch dann zugegriffen werden, wenn keine Internetverbindung vorhanden ist, auf das Dateisystem kann vom Internetbrowser aus zugegriffen werden, und die aktuelle Position des Benutzers ist ebenfalls ab sofort für eine Internetseite kein Geheimnis mehr. Alle diese Funktionen konnten früher nur von richtigen Programmen, aber nie von Internetseiten verwendet werden.

Mit den genannten Problemen und der starken Fragmentierung des mobilen Markts, aber auch durch die herausragenden Möglichkeiten, die moderne, mobile Webbrowser und der HTML5-Standard bieten, muss man annehmen, dass insbesondere im mobilen Bereich HTML5 einen enormen Einfluss nehmen wird.

Für Dienstanbieter ist dies ein Glücksfall: Sie können auf vorhandene Webprogrammierer zurückgreifen, mit einer mobilen Webanwendung in HTML5 sicherstellen, dass sie einen sehr guten Dienst auf allen mobilen Plattformen anbieten, und sie können sich immer noch entscheiden, eine native Anwendung für beispielsweise den iPhone App Store oder den Android Market programmieren zu lassen. Und das ist auch gar nicht schwer: Mittlerweise gibt es mit dem PhoneGap (*www. phonegap.com*) und dem Appcelerator Titanium Mobile (*www.appcelerator.com*) Technologien, die aus HTML5-Webanwendungen native Anwendungen für die gängigsten mobilen Plattformen bauen.

Insbesondere mit PhoneGap lassen sich im Handumdrehen richtige Anwendungen erstellen, die sich nicht nur in den App Stores verkaufen, sondern auch mit Funktionen bereichern lassen, die im Browser gar nicht zur Verfügung stehen. Die Bewegungssensoren, die Kamera, der Vibrationsalarm, der Kompass, das Adressbuch und vieles

mehr lassen sich über ein paar simple JavaScript-APIs direkt vom PhoneGap-HTML-, CSS- und JavaScript-Code aus steuern.

All diese Technologien und Ansätze möchten wir Ihnen auf den nächsten gut 300 Seiten vorstellen. Am Ende des Buchs haben Sie nicht nur ein umfassendes Wissen über HTML5, sondern auch das Zeug, Ihre erste iPhone App oder Android-Anwendung in iTunes bzw. dem Android Market zu veröffentlichen.

Worum geht es hier?

Dieses Buch ist kein Programmierbuch im klassischen Sinn. Es gibt Ihnen einen Einblick in Designkonzepte von mobilen Webdiensten und hilft Ihnen, erste Webanwendungen mittels HTML, CSS und JavaScript zu erstellen. Diese Webanwendungen werden bereits ein richtiges »iPhone-Look & Feel« besitzen. Ein großer Teil des Buchs beschäftigt sich mit den neuen Funktionen und Möglichkeiten von HTML5 und zeigt diese im mobilen Kontext eindrucksvoll und verständlich an vielen Beispielen. Der letzte Teil des Buchs beschreibt Schritt für Schritt, wie Sie eine HTML5-Webanwendung in echte iPhone- oder Android-Anwendungen umwandeln und dabei Telefonhardwarefunktionen wie die Kamera oder den Bewegungssensor eines iPhones oder Android-Smartphones ausnutzen. Abschließend erfahren Sie, wie Sie Ihre frisch gebaute Anwendung dann auch in den Apple Store und in den Android Market bekommen.

Die Twitter-Updates zum Buch
Folgen Sie *@html5togo* auf Twitter – *http://twitter.com/html5togo* –, um neueste Updates und interessante Artikel zum Thema HTML5 und Mobile direkt von den Autoren des Buchs zu erhalten und mit diesen in Kontakt zu treten.

HTML5 – die Spezifikation und der aktuelle Stand

Streng genommen, ist HTML5 die sich noch in Entwicklung befindliche neue Version des HTML-Standards. Wie auch HTML 4.1 und XHTML 1.1 ist HTML5 eine standardisierte Markup-Sprache, die dazu dient, Webinhalte zu strukturieren und zu präsentieren. Rein technisch gesehen, ist HTML5 aber eben HTML und nicht JavaScript und auch nicht CSS.

Doch die Masse versteht unter HTML5, speziell unter dem Begriff »HTML5 Web Apps«, etwas anderes. HTML5 ist zum Schlagwort einer (begrüßenswerten) Entwicklung geworden und beschreibt Anwendungen, die mit Webstandards, also HTML, CSS und JavaScript, geschrieben werden und in den Browsern auf unterschiedlichen Plattformen laufen. HTML5 steht für Webseiten, die sich wie Anwendungen anfühlen, wie Anwendungen gestartet werden können und weit in den Bereich nativer Anwendungen reichen. Wir versuchen in diesem Buch, die richtige Balance zu finden zwischen neuen Funktionen, die sich wirklich auf HTML5 als Markup-Sprache beziehen, und den Möglichkeiten, die durch die Vermischung von HTML, JavaScript und CSS entstehen und eben auch als »HTML5« bezeichnet werden.

Auch wenn Sie überall schon von HTML5 hören, heißt das noch nicht, dass die neue Version des HTML-Standards bereits fertig ist. Im Gegenteil: HTML5 ist derzeit noch in der »Working Draft«-Phase, einem Stadium, in dem immer noch aktiv an der Definition der Sprache gearbeitet wird. Allerdings werden nicht alle Komponenten, also nicht alle Tags, gleichzeitig bearbeitet, und bestimmte Teile der Sprache sind bereits weiter vorangeschritten als andere. Komponenten wie beispielsweise das `canvas`-Tag sind sehr stabil und nahezu »fertig«, während andere Teile der Sprache noch gar nicht definiert wurden.

Der Entwicklungszyklus einer neuen HTML-Definition bewegt sich im Bereich von 20 Jahren. Mit der HTML5-Spezifizierung wurde 2004 unter dem Namen »Web Applications 1.0« begonnen, und es wird derzeit erwartet, dass die Sprache im Jahr 2022 die offizielle W3C-Empfehlung erhalten wird. Zum Vergleich: Die derzeit gültige HTML5-Spezifikation wurde Mitte der 90er-Jahre begonnen.

Nun sollten Sie auf keinen Fall bis 2022 warten, sondern die Möglichkeiten von HTML5 und den oft in einem Atemzug genannten Technologien JavaScript und CSS schon heute nutzen. HTML5 ist bereits da, wird aber in der Zukunft noch wesentlich umfangreicher und von mehr Plattformen und Browsern unterstützt werden, als es heute der Fall ist. Die hier im Buch vorgestellten HTML5-Methoden sind bereits größtenteils von den führenden mobilen Plattformen implementiert, ganz vorn dabei natürlich iOS und Android, aber auch BlackBerry OS 6.

⊡ Lesezeichen

http://dev.w3.org/html5/spec/

Offizielle HTML5-Spezifikation: In diesem Buch wird die offizielle HTML5-Spezifikation immer in einem mobilen Umfeld betrachtet. Falls Sie sich für die komplette Spezifikation interessieren, finden Sie sie hier.

Webanwendung vs. native Anwendung

Bereits weiter oben wurde häufig von einer Webanwendung und einer »richtigen« nativen Anwendung gesprochen. Auch wenn HTML5 die Grenzen und Möglichkeiten zwischen einem richtig programmierten Programm und einer Webseite verschwimmen lässt, so gibt es doch einige nennenswerte Unterschiede, die zu bedenken sind.

Native Anwendung

Unter einer nativen Anwendung versteht man ein Programm, das nicht im Browser des Telefons (oder eines Computers) läuft, sondern als eigenständiges Programm auf das Gerät geladen wurde und ein richtiges Computerprogramm ist. Native Anwendungen haben in der Regel einen besseren Zugriff auf Systemfunktionen, wie beispielsweise die Kamera eines Handys oder Bewegungssensoren, können Rechenoperationen viel schneller ausführen und die Grafikmöglichkeiten eines Systems wesentlich besser ausnutzen. Sämtliche Spiele, die auf schnelle Grafikroutinen angewiesen sind, werden als native Anwendungen erstellt.

Diese Anwendungen haben gegenüber Webanwendungen nicht nur auf der technischen Seite Vorzüge, sondern bieten einen riesigen Vorteil: Man kann sie in die App-Stores der verschiedenen Plattformanbieter einstellen. Distribution ist das Zauberwort!

Eine native Anwendung kann beispielsweise im iPhone App Store eingestellt, beworben und vertrieben werden. Benutzer müssen nicht umständlich im Internet nach ihrer mobilen Webseite suchen, sondern finden ein fertiges Programm zum Installieren in iTunes oder im App Store des iPhones. Die Wahrscheinlichkeit, dass ein Kunde Ihren Dienst in einem App Store entdeckt, ist wesentlich höher, als wenn er bei Google oder Yahoo! danach sucht. Ein zusätzlicher Bonus: Nach dem Hinzufügen ist das Programm direkt auf dem Home-Bildschirm des iPhones oder Android-Handys verewigt.

Dies alles klingt gut, und Sie werden sich jetzt vielleicht fragen, warum Sie sich kein Buch zur iPhone-Programmierung mit Objective-C gekauft haben, doch leider ist nicht alles so einfach:

- Der Aufwand, eine native Anwendung zu erstellen, ist pro Plattform immens hoch. Zwar kann es Ihnen gelingen, auf einer Plattform eine hohe Reichweite Ihrer Anwendung und Ihres Angebots zu erreichen, allerdings bedienen Sie nur eine von vielen Plattformen.
- Bevor Ihre Anwendung in einem der bekannten App-Stores erscheint, muss sie einen Genehmigungsprozess durchlaufen. Dies gilt nicht nur, wenn Ihre Anwendung erstmalig eingestellt wird, sondern der Prozess muss bei jeder Aktualisierung wiederholt werden.

Der Genehmigungsprozess, insbesondere um in den iPhone App Store zu kommen, wurde in der Presse schon oft heiß diskutiert. Selbst Internetgrößen wie Google wurde der Eintritt in den Store beispielsweise mit der Google Voice-Anwendung verwehrt. Interessanterweise hat sich Google damit beholfen, eine Webanwendung mit HTML5 – *www.youtube.com/watch?v=neiOa38DuqI* – zu erstellen, die fast identische Funktionen wie die eigentliche native Anwendung auf das iPhone bringt.

Viele Entwickler warten bislang oft mehrere Wochen, bis ihre Anwendung im App Store erscheint oder genehmigt wird. Warum eine App bei Apple abgelehnt wurde, wird in der Regel nicht sehr detailliert begründet. Wie bereits gesagt, ist eine Überprüfung sogar bei jeder neuen Programmversion erforderlich. Wenn Ihre Anwendung beispielsweise einen Fehler enthält, kann er nicht einfach wie bei einer Webseite sofort behoben werden. Stattdessen müssen Sie ein neues Anwendungspaket bauen, dieses Paket auf verschiedenen Telefonen testen, es an Apple oder einen anderen Betreiber des App Store übermitteln, auf das Testergebnis warten, gegebenenfalls nachbessern und dann hoffen, dass die Anwendung jetzt zum Verkauf bzw. Download freigegeben wird.

Eine weitere Einschränkung, insbesondere wenn Sie eine native iPhone-Anwendung erstellen möchten: Sie müssen einen Apple-Rechner besitzen, Objective-C lernen und Apple für die Aufnahme als Entwickler bezahlen.

Webanwendung

Im Gegensatz zu einer nativen Anwendung braucht eine Webanwendung ein anderes Programm zur Ausführung: den Browser. Daher ist die Webanwendung auch nur so gut wie der Browser, in dem sie läuft. Funktionen, die der Browser nicht weitergeben kann, beispielsweise den Zugriff auf die Kamera eines Handys, kann demzufolge eine reine Webanwendung auch (noch) nicht nutzen.

Den Vorteil, dass eine Webanwendung über einen App-Store verbreitet und sichtbar gemacht werden kann, besitzt leider auch die native Anwendung. Insbesondere wenn Sie einen kostenpflichtigen Dienst bzw. eine Anwendung anbieten möchten, müssen Sie bei einer Webanwendung eine für Sie und den zukünftigen Kunden recht umständliche Zahlungsabwicklung implementieren, während die Kreditkarte des Kunden bei iTunes und einer nativen Anwendung quasi einen Klick entfernt ist.

Dennoch: Mit immer mehr Browsern, unabhängig vom Betriebssystem, die die neuesten Standards wie beispielsweise HTML5 und CSS 3 unterstützen, kann eine Webanwendung auf allen mobilen Plattformen laufen und Nutzer erreichen, egal ob diese ein iPhone, ein Google-Handy, einen Palm Pre oder ein Nokia-Telefon benutzen. Webseiten sind in der Regel viel einfacher und günstiger zu programmieren als native Programme. Viele Firmen beschäftigen bereits Heerscharen von Webentwicklern.

Wie jede andere Webseite im Internet muss eine Webanwendung auch keine Genehmigungsprozedur durchlaufen, bevor sie auf einem mobilen Gerät angewendet werden kann. Der Webseitencode wird in das Internet gestellt und ist sofort weltweit auf jedem Gerät mit geeignetem Browser verfügbar. Sie müssen keine Zertifizierungsgebühr bezahlen, keine zusätzliche Software kaufen, keine komplette Entwicklungsumgebung erlernen und können Aktualisierungen Ihrer Webanwendung in Sekundenschnelle weltweit veröffentlichen.

Warum nicht beides?

Dieses Buch zeigt Ihnen auf, wie Sie eine moderne mobile Webanwendung mit HTML5, CSS und JavaScript erstellen, die sich in vielen Punkten bereits im Browser wie eine richtige »native« Anwendung verhalten kann, und zeigt Ihnen gleichzeitig, wie Sie aus dieser Web App eine richtige »native« Anwendung für den App-Store und den Android Market bauen können.

Das gibt Ihnen die Freiheit, Dienste zu erstellen, die von jedermann erreichbar sind. Ihnen ist aber die Chance nicht verbaut, mit der gleichen Anwendung auch in einen der App-Stores zu gelangen und dort bekannt zu werden. Idealerweise können Sie eine umfassende Webanwendung bauen, die Sie ständig (und ohne umständliche Genehmigungen) aktualisieren und ausbauen und von der Sie einen Teil der Webanwendung als native App in die verschiedenen Stores stellen können.

Wer sollte das Buch lesen?

Das Buch richtet sich an alle, die sich für neuartige Webanwendungen für den mobilen Markt interessieren oder sich dort bereits bewegen und aus mobilen Webseiten leistungsfähige native Anwendungen erstellen möchten. Sie finden im Buch allerhand Codebeispiele, doch richtet es sich nicht nur an Entwickler, sondern auch an Produktmanager, Designer, Vordenker, Entrepreneurs und alle anderen, die sich mit dem neuesten Trend im mobilen Bereich auseinandersetzen möchten. Entwickler werden an dem Buch auch ihre Freude haben, da es zum einen Verständnis für den mobilen Markt außerhalb der Technik vermittelt und zum anderen geballte HTML5-Beispiele auflistet, die helfen, mobile Seiten mit einer Funktionsfülle auszustatten, die bisher nur von »richtigen« Anwendungen bekannt war.

Besonders tiefe Vorkenntnisse im Bereich der Programmierung setzt dieses Buch nicht voraus. Natürlich sollten Sie wissen, was ein Browser ist, wie man sich im Internet bewegt, wie man einfache Webseiten baut und woraus diese bestehen. Begriffe wie HTML, CSS und JavaScript sollten Ihnen geläufig sein – Sie müssen sie aber nicht bis ins letzte Detail verstehen, um dieses Buch zu lesen.

Das Ziel dieses Buchs ist, Ihnen die verschiedenen Möglichkeiten aufzuzeigen, die heutzutage mit HTML5 auf mobilen Geräten machbar sind und, falls Sie das auch noch interessiert, wie Sie es machen können. Es vermittelt Ihnen einen Überblick über den mobilen Markt mit seinen Schwierigkeiten und Vorzügen und zeigt Ihnen auf interessante Art und Weise, wie Sie möglichst viele mobile Plattformen mit wenig Aufwand abdecken und dazu noch in den wichtigen App-Stores wie dem iPhone App Store und dem Android Market vertreten sein können.

Wenn Sie noch weiter in die Welt von HTML, CSS und JavaScript vordringen oder sich tiefer mit der nativen App-Entwicklung für mobile Plattformen oder mit mobilen Konzepten beschäftigen wollen, können wir Ihnen die Lektüre der folgenden Bücher, Blogs und Podcasts empfehlen.

⊡ Lesezeichen

http://bit.ly/d8467s

Webseitenlayout mit CSS: Das neue und sehr aktuelle CSS-Buch aus dem Franzis Verlag führt den Leser an CSS heran und berücksichtigt die neueste CSS 3-Spezifikation.

⊡ Lesezeichen

http://bit.ly/doTlti

HTML-Handbuch: Ganze Generationen von Webentwicklern sind mit der fabelhaften Webseite selfhtml.org aufgewachsen. Die Standardreferenz im Internet ist auch als Buch erhältlich.

⊡ Lesezeichen

http://css-tricks.com/

Das englischsprachige Blog berichtet in regelmäßigen Abständen über CSS-Tipps und zeigt diese anhand von Live-Beispielen und Videos. Ein großer Teil der dort vorgestellten Themen behandelt den Einsatz von CSS in der mobilen Webentwicklung.

⊡ Lesezeichen

http://bit.ly/a5S2Ky

JavaScript und Ajax: Wenn Sie tiefer in die JavaScript-Materie eintauchen wollen, sei Ihnen dieses Buch empfohlen. Anhand konkreter Programmierbeispiele erlernen Sie die Grundlagen bis hin zu den fortgeschrittenen Techniken von JavaScript und AJAX.

Danksagung

Bevor es jetzt richtig losgeht, möchten wir uns als Autoren bei einigen Menschen bedanken, ohne die dieses Buch nicht entstanden wäre: zuallererst bei unseren Familien – insbesondere unseren Frauen Steffi und Sandra –, die von uns während der Monate vor dem Rechner nicht viel hatten und uns dennoch unterstützten und uns mit viel Espresso wach hielten. Des Weiteren geht unser Dank an:

Ross Harmes, der das wunderbare Flickr-Geo-Fotobeispiel zum Buch beigesteuert hat, unserem Verleger für die aufgebrachte Geduld, die netten Kollegen vom Yahoo! Editorial Board, die ebenfalls an der Entstehung des Buchs mitarbeiteten, an Matteo Spinelli von *cubiq.org*, mit dem wir an einigen CSS-/JavaScript-Beispielen im Buch gearbeitet haben, und an jeden, der uns geholfen hat, die Beispiele im Buch zu testen, und die zeitlichen Möglichkeiten gegeben hat, dieses Buch zu schreiben.

Was wird benötigt?

Außer dem Buch, das Sie glücklicherweise schon in Ihren Händen halten, brauchen Sie nicht viel: etwas Zeit, einen Texteditor und einen modernen Browser. Das ist wirklich die Mindestausstattung. Obwohl das ein Buch über mobile Technologien ist, brauchen Sie theoretisch nicht unbedingt ein internetfähiges Telefon. Wie gesagt, »theoretisch« – denn eigentlich würden wir Ihnen entweder ein iPhone, iPad, iPod touch oder ein Smartphone mit Android-Betriebssystem zum Testen Ihrer mobilen Webseiten oder ein anderes Mobiltelefon mit einem halbwegs guten Browser empfehlen.

Wie gesagt: Alle Schritte lassen sich auch auf einem normalen Computer abbilden – ein Telefon ist nicht vonnöten! Wenn Sie aber wissen wollen, wie sich Ihre Webseite und dann auch Ihre mobile Anwendung wirklich »anfühlt«, kommen Sie um ein Handy zum Testen nicht herum. Den größten Unterschied werden Sie wahrscheinlich in der Geschwindigkeit merken: Ist auf dem Desktop-Rechner dank Hochgeschwindigkeitsinternet und Rechenpower alles superschnell, muss man beim Design und der Programmierung von mobilen Seiten sehr auf die Datengrößen achten, ansonsten kann es auf einem Telefon oft zu langwierigen und für den Benutzer sehr unschönen Ladezeiten kommen.

Eine mobile Webseite können Sie zwar auf Ihrem Computer erstellen und dort auch testen. Möchten Sie jedoch die Seite auf einem Telefon testen bzw. Ihre Webseite gegebenenfalls auch ins Internet stellen, kommen Sie um etwas Speicherplatz im Internet, sogenannten Webspace, nicht herum. Eine Suche bei Google oder Yahoo! nach »Webspace-Anbieter« wird Ihnen eine ganze Liste offenbaren. Wir haben viel mit dem Anbieter 1&1 (*www.1und1.de*) in Deutschland gearbeitet und können ihn auch sehr empfehlen.

Für eine iPhone-, iPod touch- oder iPad-Anwendung

Wenn Sie aus Ihren mobilen Webseiten eine iOS-Anwendung für den App Store erstellen möchten, benötigen Sie einen Mac, also einen Rechner von Apple mit dem Mac OS X-Betriebssystem. Das ist nicht der Sonderbehandlung geschuldet, dass wir aus einer Webseite eine Anwendung generieren – selbst wenn Sie über den herkömmlichen Weg unter Benutzung von Xcode und dem iPhone SDK eine iPhone-Anwendung programmieren möchten, müssen Sie das immer auf einem Mac tun.

Für eine Android-Anwendung

Eine Android-Anwendung für Telefone mit diesem Betriebssystem können Sie auch auf einem PC mit Windows Betriebssystem erstellen. Alternativ können Sie einen Mac oder einen Rechner mit Linux-Betriebssystem benutzen.

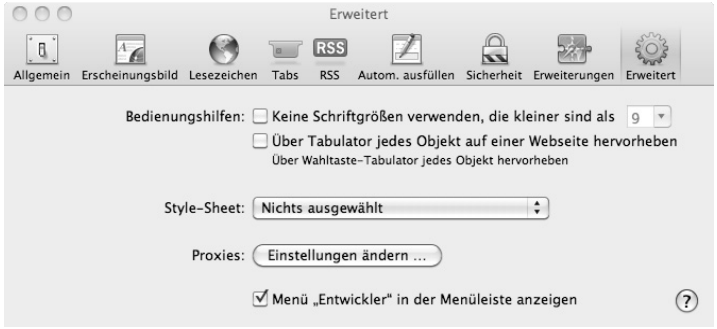
Welcher Browser?

Nicht nur der mobile Browsermarkt, sondern auch das Desktop-Browserangebot ist sehr reichhaltig. Da HTML5 zum Erscheinungszeitpunkt dieses Buchs wirklich eine brandneue Technologie ist, haben einige Browser noch nicht alle Bestandteile des neuen HTML-Standards integriert.

Safari

Sehr empfehlenswert für das Testen von mobilen Seiten ist die Installation von Safari – dem Browser aus dem Hause Apple. Apple hat in Bezug auf HTML5-Kompatibilität Standards gesetzt und auch aktiv an HTML5 und CSS mitgearbeitet.

Das iPhone nutzt exakt die gleiche Safari-Technologie auf dem Telefon, wie sie auch in der Desktopversion des Browsers verwendet wird. Es kommt aber noch besser: In Safari kann man über die Programmeinstellungen ein Extramenü für Entwickler freischalten und sogar den User-Agent (quasi den Erkennungscode des Browsers) auf den eines iPhones oder iPods umschalten.



Die versteckte, aber nützliche *Entwickler*-Option in Safari unter *Einstellungen/Erweitert*.

Gehen Sie dazu in das Safari-Menü, rufen Sie dort das Dialogfenster *Einstellungen* auf und klicken Sie auf *Erweitert* und das Zahnradsymbol. Im darauffolgenden Fenster sehen Sie die Option *Menü »Entwickler« in der Menüleiste anzeigen*, die Sie einfach markieren. Im Menü *Entwickler* finden Sie dann die entsprechenden Optionen unter

dem Punkt *User Agent*. Aber auch andere hilfreiche Funktionen, wie beispielsweise der Safari Web Inspector (*Webinformationen einblenden*), auf die wir später noch zurückkommen, sind dort versteckt.

Firefox

Wesentlich verbreiteter als Safari ist Firefox, ein ebenfalls kostenloser Browser, der der Open-Source-Gemeinde entstammt und nach dem Internet Explorer der meistgenutzte Browser in Deutschland ist. Wie Safari ist Firefox führend in der Aktualität, wenn es um die neuen HTML5-Funktionen geht. Wenn Sie sich ausschließlich um das iPhone sorgen, sind Sie mit Safari besser bedient – falls es Ihnen aber auch um andere mobile Betriebssysteme geht, sollten Sie auf jeden Fall Firefox installieren und zum Testen benutzen.

Ein weiterer Vorteil von Firefox ist die Menge an erhältlichen Erweiterungen. Was Sie im Safari-Browser über das *Entwickler*-Menü bewerkstelligen können, ist bei Firefox über die sogenannten Add-ons – *https://addons.mozilla.org/de/firefox/* – möglich. Zwei Erweiterungen, die ich zumindest schon nennen, später aber noch näher vorstellen möchte, sind »User Agent Switcher« und »Firebug«. Safari bietet in der neuesten Version 5 sogenannte Extensions und ist somit ähnlich erweiterbar wie Firefox, doch das Angebot der Safari-Extensions ist gegenüber Firefox noch erheblich eingeschränkt.

Google Chrome

Die meisten Dinge, die wir über Safari und Firefox gesagt haben, gelten auch für Google Chrome: Der Browser ist in Sachen HTML5-Unterstützung up to date und besitzt wie Safari zahlreiche Funktionen, die das Entwicklerleben einfacher machen.

Internet Explorer

Ein Browser, der schon fast traditionell den neuesten Entwicklungen hinterherhängt, ist der Internet Explorer. Falls Sie noch die alte Version 6 des Internet Explorers besitzen, sollten Sie schnellstmöglich zur Microsoft-Seite eilen und sich die neueste Version besorgen. Viele Firmen, unter anderem Google, haben bereits die offizielle Unterstützung für diese Browserversion beendet. Für den Kontext dieses Buchs und derzeit für mobile Webentwicklung im Allgemeinen möchten wir vom Internet Explorer, auch in der Version 8, abraten. Doch auch Microsoft kann sich der Webinnovation nicht weiter verschließen: Der Internet Explorer 9 wird bereits Elemente von HTML5 verstehen und CSS 3 unterstützen.

Welcher Texteditor?

Neben einem guten Browser ist ein Texteditor das wichtigste Werkzeug in der Webentwicklung. Vor dem Texteditor werden Sie die meiste Zeit verbringen, und ein guter Editor kann Ihnen helfen, den Überblick über Ihren Code zu behalten. Allerdings heißt es hier wie so oft: Weniger ist mehr. Texteditoren, die es Ihnen erlauben, den Text schön zu formatieren, wie beispielsweise Microsoft Word, erzeugen Dateien, die Sie nicht für Webseiten nutzen können.

Für Windows-User gibt es eine ganze Schar an guten und auch kostenlosen Programmen, die sich hervorragend zur Webentwicklung eignen. Ich möchte den auch in Deutsch erhältlichen Editor ConTEXT (*www.contexteditor.org*) hervorheben, der HTML, CSS und JavaScript versteht, Sie auf Syntaxfehler aufmerksam macht und den Code übersichtlich darstellt.

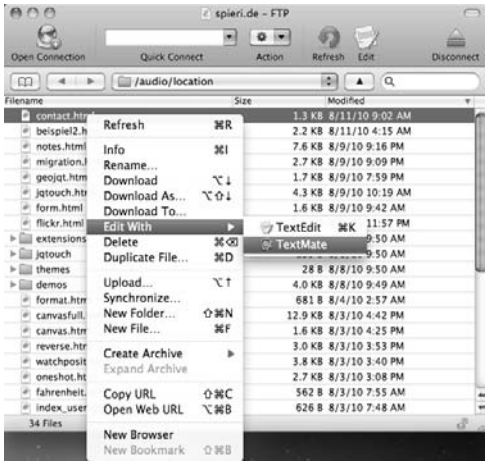
Für Mac-Benutzer empfiehlt sich der kostenlose Editor TextWrangler (*www.barebones.com/products/textwrangler*). Das Programm ist sehr stabil, hilft bei der Codeformatierung und kann Änderungen am Webseitencode auch gleich auf Ihren Webserver speichern.

Mein Lieblingseditor und auch der fast aller Webentwicklerkollegen ist TextMate auf dem Mac. Das Programm ist simpel gehalten, hilft bei der Darstellung der verschiedenen HTML-Tags und lässt sich sogar in die Apple-Entwicklungsumgebung Xcode integrieren. TextMate (*www.macromates.com*) kann 30 Tage kostenlos getestet werden, kostet dann allerdings knapp 40 Euro. TextMate arbeitet auch vorzüglich mit dem FTP-Programm Cyberduck zusammen. Dateien können direkt aus Cyberduck heraus mit TextMate bearbeitet und direkt auf dem Webserver wieder gespeichert werden.

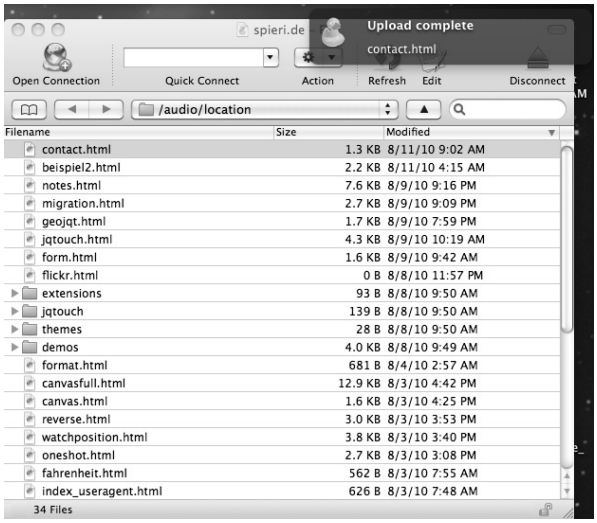
Welches FTP-Programm?

Falls Sie Ihre Webseiten auf einem Telefon testen oder auch veröffentlichen möchten, muss Ihr frisch geschriebener Code von Ihrem Computer in das Internet, sprich auf Ihren Webserver oder Webspace gelangen. Das gelingt Ihnen in der Regel mit einem sogenannten FTP-Programm, einer Software, die Dateien von Ihrem lokalen Rechner in das Internet über das File Transfer Protocol verschieben kann. In der Regel sind FTP-Programme so aufgebaut, dass Sie auf der einen Seite ein Fenster mit dem Inhalt Ihrer Festplatte und auf der anderen Seite den Inhalt Ihres Webservers ebenfalls als Dateistruktur sehen. Dateien können dann einfach hin- und hergeschoben werden.

Für Windows-Benutzer empfiehlt sich die kostenlose Software WinSCP – *www.winscp.net/eng/docs/lang:de*. Das Programm ähnelt sehr der Oberfläche des Windows Explorers und dürfte für die meisten Windows-Benutzer daher sehr vertraut wirken. Mac-Anwender, die TextWrangler benutzen, brauchen sich um keine zusätzliche Software zu sorgen, da die FTP-Funktion bereits im Programm eingebaut ist. Wer einen anderen Texteditor auf dem Mac benutzt, dem sei das Sharewareprogramm Cyberduck – *www.cyberduck.de* – empfohlen.



Aus Cyberduck heraus können HTML-Dateien direkt mit TextMate bearbeitet und unmittelbar auf den Server zurückgespeichert werden.



Upload complete – mit dem kostenlosen Programm Growl – *www.growl.info* – sehen Sie über eine kleine Bildschirmmeldung auf einen Blick, dass die Dateien auf den Server hochgeladen sind und Sie mit dem Testen beginnen können.

Download der Beispiele

Wir haben zahlreiche Beispiele für Sie erstellt – von der einfachen HTML-Seite bis hin zu komplexeren Web Apps – und den Code im Buch veröffentlicht. Doch keine Sorge: Das Abtippen des Codes können Sie sich sparen, da alle Beispiele auf der unten aufgeführten Website zum Download zur Verfügung stehen.

▣ Download

http://www.buch.cd

Hier finden Sie den Code aller im Buch besprochenen Beispiele. Von hier aus lassen sich die mobilen Web Apps auch auf Ihrem iPhone, iPod, Android-Telefon oder WebOS-Palm ausprobieren.

Inhaltsverzeichnis

1 Mobile Designprinzipien

Bevor wir uns an das Programmieren einer mobilen Webseite machen, möchten wir auf einige Designgrundlagen für das mobile Anwendungs- und Webdesign aufmerksam machen. Das Design einer mobilen Seite oder Anwendung kann in vielen Punkten als komplexer als das Webdesign einer Desktopseite angesehen werden. Auf folgende Herausforderungen werden Sie stoßen, wenn Sie mobile Programme erstellen möchten:

- Der Bildschirm ist wesentlich kleiner.
- Der Benutzer bedient die Anwendung mit dem Finger, nicht mit der Maus.
- Die Texteingabe ist selbst bei High-End-Telefonen immer noch keine Freude.
- Der Benutzer hat unterwegs andere Anforderungen an eine Anwendung oder Webseite als zu Hause oder bei der Arbeit am PC.
- Trotz UMTS und anderer Technologien ist das mobile Internet nicht beständig schnell, ein Nutzer kann sogar in Gebiete ohne Empfang kommen.

Um diese Probleme so gut wie möglich zu lösen, sollten Sie bei dem Anwendungsdesign einer mobilen Anwendung auf folgende Dinge achten:

- Machen Sie sich am Anfang Ihres Konzepts klar, was Ihr zukünftiger Benutzer für Vorteile hat, wenn er Ihre Anwendung unterwegs benutzt. Was könnte die Hauptnutzung, der hauptsächliche sogenannte »Use Case«, Ihrer Anwendung sein?
- Eine mobile Anwendung wird wahrscheinlich nicht erfolgreich sein, wenn sie unterwegs keinen Sinn ergibt oder die Funktionen, die für die Benutzung unterwegs wenig sinnvoll sind, gegenüber anderen Funktionen heraushebt.
- Nicht alle Inhalte eignen sich für die mobile Nutzung: Große Grafiken oder extrem lange Texte sind ungeeignet, verursachen lange Ladezeiten oder passen nicht in das Design für kleine Bildschirme. Versuchen Sie, angepasste Inhalte zu verwenden.
- Wenn nicht der falsche Use Case oder Designansatz den Erfolg einer mobilen Anwendung »killt«, dann ist es die fehlende Geschwindigkeit. Der mobile Benutzer ist in der Regel viel ungeduldiger und erwartet – insbesondere von einer Seite oder Anwendung, die für die mobile Nutzung optimiert ist – auch eine optimierte Geschwindigkeit.
- Versuchen Sie, die Texteingabe so weit wie möglich zu minimieren. An jeder Stelle, an der der Benutzer Ihrer mobilen Anwendung längere Informationen eingeben muss, ist Scheitern programmiert. Zwar bietet insbesondere das iPhone eine für mobile Geräte recht komfortable Eingabemethode, dennoch wird ein Großteil Ihrer Benutzer Eingabefelder so weit wie möglich meiden.
- Falls Sie nicht um bestimmte Texteingabefelder herumkommen, versuchen Sie nach Möglichkeit, diese schon vorher sinnvoll auszufüllen. In diesem Fall hat der Benutzer

immer noch die Chance, den vorgeschlagenen Inhalt zu verändern oder zu löschen, kann ihn aber auch einfach ohne Texteingabe akzeptieren, wenn er mit dem Vorschlag einverstanden ist.

- Vermeiden Sie komplexe Formulare mit Texteingabe. Die meisten Touchscreen-Telefone blenden eine Bildschirmtastatur ein, die einen Großteil des Bildschirms verdeckt. Unter Umständen sehen Sie gar nicht mehr, was Sie eingeben sollen, da sich die Bildschirmtastatur über die Felder gelegt und der Fokus sich verschoben hat.
- Mittels HTML5 können Sie glücklicherweise wesentlich flexiblere Texteingabefelder definieren. Mehr dazu lesen Sie im Abschnitt 2.2.2 »Neue Input-Typen, Auto-Korrektur und Auto-Großschreibung«.
- Lassen Sie sich nicht von der Traumvorstellung des mobilen Nutzers verführen! Er ist ein ganz normaler Anwender, nicht unbedingt technisch versiert, wenig interessiert am »Entdecken« von Funktionen und Bereichen einer mobilen Anwendung, und er kommt meistens auch ohne Superfinger mit Stylus-Durchmesser daher. Gestalten Sie daher Ihre Schaltflächen auch für etwas dickere Finger.

Auch wenn sich die gerade aufgeführten Punkte nach einer Reihe von Einschränkungen anhören und den Spaß am Anwendungsdesign verderben, so bieten mobile Anwendungen auch einzigartige Möglichkeiten:

- Der Benutzer kann Ihre mobile Anwendung oder Webseite immer bei sich tragen. Dazu ist Ihre Kreation auf dem Medium und dem Gerät da, mit dem der Benutzer einen Großteil seiner Kommunikation abwickelt. Bieten Sie in Ihrer Anwendung oder Webseite Schnittstellen zu den Telefonfunktionen an, wenn es Sinn ergibt. Aus einer Webseite und natürlich auch aus einer Anwendung heraus können Sie Telefonanrufe initiieren, Inhalte per E-Mail verschicken oder sie im Internet direkt vom Handy heraus verbreiten.
- Insbesondere HTML5 bringt mit den Location-Funktionen die aktuelle Position des Benutzers zu Ihnen. Wenn Sie wissen, wo auf der Welt sich Ihr Nutzer befindet, können Sie passgenaue Dienste anbieten. Allein durch die Verfügbarkeit dieser Information lassen sich Tausende mobilspezifische Anwendungen bauen.
- Mit den Offlinekapazitäten von HTML5 haben Sie die Chance, mobile Anwendungen zu konzipieren, die auch ohne Internetverbindung auskommen oder kurze Verbindungsausfälle, wie sie mit einem Telefon schon mal vorkommen, ganz geschickt zu überspielen.
- JavaScript und HTML5 bieten Ihnen unglaubliche Möglichkeiten, nicht immer eine ganze Seite, sondern nur bestimmte Teile der Seite neu zu laden. Machen Sie davon Gebrauch – wenn es sinnvoll ist!

1.1 Mobiles Web- und Anwendungsdesign

Da das iPhone ohne Frage die mobile Welt verändert und Standards im mobilen Webdesign gesetzt hat, möchten wir im nächsten Abschnitt auf bestimmte Charakteristiken im iPhone-Webdesign eingehen und sie vorstellen. Für alle Beispiele im Buch werden

wir ein iPhone-typisches Standarddesign verwenden. Sie werden lernen, wie Sie dieses Design durch Editieren der Cascading Style Sheets verändern und wie Sie mit nur wenigen Zeilen Code Ihrer Anwendung oder Webseite ein iPhone-Look geben können.

1.1.1 Application Prototyping mit Photoshop

Bevor Sie mit dem Programmieren einer iPhone-Webanwendung beginnen, ist es ratsam, sich die verschiedenen Bildschirme, die ein Benutzer sehen und durch die er durchnavigieren soll, zu überlegen und aufzuzeichnen. Am Anfang landen diese Flow-Überlegungen meistens auf Skizzenpapier, einer Serviette oder auf dem Whiteboard. Wenn der grobe Flow allerdings steht, empfiehlt es sich gerade bei iPhone- oder iPod touch-Anwendungen – in denen es zu einem überdurchschnittlich großen Teil um die User-Experience geht –, die einzelnen Bildschirme detaillierter zu gestalten und aufzuzeichnen. Diese Art von »Application Prototyping« ist ein wichtiger Schritt in der Software- und Webentwicklung und hilft beim Entdecken von Problemen im Anwendungsdesign.

Falls Sie Photoshop besitzen und sich mit dem Programm auskennen, können Sie vorgefertigte Bibliotheken mit den verschiedenen iPhone-typischen Elementen wie Titelleisten, Buttons, Eingabefeldern, Tastatur etc. laden und in Photoshop zu der idealen Oberfläche Ihrer Anwendung anordnen.

⊡ Lesezeichen

http://bit.ly/Bvn5o

Besitzen Sie Photoshop und kennen das Programm bereits, können Sie vorgefertigte Bibliotheken mit den verschiedenen iPhone-typischen Elementen laden und wunschgemäß anordnen.

⊡ Lesezeichen

http://bit.ly/89DYC4

Wenn Sie sich speziell für eine iPad-Anwendung interessieren, gibt es unter dieser Adresse eine 24 MByte große Photoshop-Datei mit allen relevanten iPad-Elementen.

Auch für diejenigen, die sich speziell für Android-Webanwendungen oder mittels PhoneGap auch für Android Apps interessieren, gibt es zahlreiche Photoshop-Vorlagen im Internet.

⊡ Lesezeichen

http://bit.ly/cO0aLH

Eine der umfassendsten Vorlagen hat Vladimir Grishin auf seiner Webseite zum kostenlosen Download zur Verfügung gestellt.

1.1.2 Interface Builder und andere Programme

Falls Sie einen Mac Ihr Eigen nennen, können Sie auch das offizielle iPhone SDK herunterladen (mehr dazu am Ende des Buchs) und die Software Interface Builder benutzen.

Besitzen Sie weder Photoshop noch einen Mac, gibt es immer noch zahlreiche Möglichkeiten, Flows visuell zu erstellen. Diese werden keine so detaillierte Darstellung der Benutzeroberfläche bieten, können aber immer noch jede Menge Probleme beim Userflow aufzeigen. Persönlich verwende ich auf dem Mac die Apple-Software Keynote sehr gern, auf dem PC greife ich auf PowerPoint oder Microsoft Visio zurück.

Mit jedem iOS SDK kommt auch das Programm Dashcode, das sogar aus der gestalteten Oberfläche HTML, CSS und JavaScript generiert. Obwohl sich mit Dashcode komfortabel UI-Elemente platzieren und ausprobieren lassen, haben wir auf eine nähere Vorstellung verzichtet, da wir glauben, dass der beste Code immer noch der ist, den man selbst geschrieben hat. Auch nutzt Dashcode viele der mit HTML5 gegebenen Möglichkeiten nicht aus und generiert Code, der zu Problemen mit den erhältlichen mobilen Frameworks führen kann. Ist das der Fall, lassen sich daraus keine nativen Apps erstellen. Trotzdem sollten Sie sich das Programm, wenn Sie einen Mac haben, einmal ansehen. Im hinteren Teil des Buchs beschreiben wir die Installation des iOS SDK.

1.2 iPhone-typisches Design in der Praxis

Um das Design einer Web App wirklich zu verstehen, werden wir in diesem Abschnitt eine normale Webseite, die für die Darstellung auf einem Desktopbildschirm optimiert ist, in eine mobile Webseite umwandeln. Ziel ist es, die HTML-Struktur der Seite unangetastet zu lassen, da, wie bereits gelernt, der HTML-Code idealerweise den Inhalt einer Seite, aber nicht deren Aussehen bestimmt. Das Look & Feel einer Seite wird durch CSS definiert, und dieses kann sich in der Darstellung auf einem PC drastisch von der auf einem mobilen Gerät unterscheiden.

Im folgenden Beispiel wird eine Seite für einen Autohändler erstellt. Im Kopfbereich des HTML-Codes wird auf drei verschiedene CSS-Dateien verwiesen. Die erste CSS-Datei *computer.css* bestimmt die Darstellung auf allen Geräten, die eine Bildschirmgröße von mehr als 961 Pixeln aufweisen. Die zweite Datei ist die für unsere Umgestaltung relevante Datei und nennt sich *mobile.css*. Die dort befindlichen CSS-Definitionen werden bei allen Geräten verwendet, die nur aus einem Bildschirm bestehen und eine maximale Bildschirmbreite von 960 Bildpunkten besitzen. Der dritte CSS-Verweis ist für die Druckausgabe gedacht, und der darunterliegende Eintrag ist die bereits vorher im Buch beschriebene Sonderlösung für Microsofts Internet Explorer.

Warum 960 Bildpunkte?
Während man früher nur eine maximale Bildschirmgröße von 480 Bildpunkten angeben musste, hat sich das mit den neuen Android-Geräten und auch dem iPhone 4 geändert. Würden Sie im unteren Beispiel nur den Wert 480 angeben, erhielten Sie theoretisch eine mobile Seite auf einem älteren iPhone 3G und die volle Desktopseite auf einem iPhone 4. Eine User-Experience, die Sie mit Sicherheit nicht möchten.

Praktisch funktioniert die Angabe von 480 Bildpunkten allerdings noch mit dem iPhone 4, da der Retina-Bildschirm intern die doppelte Pixelanzahl berechnet. Sie könnten daher ein iPhone 4 sogar mit folgender Zeile erkennen und spezielle CSS-Definitionen zuweisen:

```
<link rel="stylesheet" media="only screen and (-webkit-min-device-pixel-
ratio: 2)" type="text/css" href="iphone4mobile.css" />
```

Sie werden etwas weiter unten im Buch noch bessere Möglichkeiten kennenlernen, um ein iOS- oder Android-Gerät von normalen Computern zu unterscheiden. Sie sollten wissen, dass wir pro Ausgabemedium unterschiedliches CSS zuweisen können, allerdings gibt es in der Praxis wesentlich verlässlichere Wege dazu.

Die Seite untergliedert sich in vier Hauptbereiche: den Kopfbereich mit dem Namen des Autohändlers, die Navigation, den eigentlichen Inhalt und die Kontaktadresse, die im Fußbereich der Webseite steht. Alle Bereiche sind deutlich als einzelne `div`-Bereiche gekennzeichnet. Die Stylesheets in der Datei *computer.css* bezeichnen nicht nur die Schriftgröße oder die Hintergrundfarbe, sondern bestimmen auch die Anordnung der einzelnen Hauptteile. Was früher oft von unsichtbaren Tabellen übernommen wurde, lässt sich einfach und wesentlich präziser mit `div`-Bereichen und CSS übernehmen.

```
<html>
<head>
<title>Oldtimer Spiering</title>
<link rel="stylesheet" type="text/css"
      media="screen and (min-width: 961px)" href="computer.css">
<link rel="stylesheet" type="text/css"
      media="only screen and (max-width: 960px)" href="mobile.css">
<link rel="stylesheet" type="text/css"
      media="print" href="drucker.css">
<!-- [if IE]>
 <link rel="stylesheet" type="text/css" media="all" href="ie.css">
<![endif]-->
</head>
<body>

<div id="kopfbereich">
   <h1>Oldtimer Spiering</h1>
   <h2>Spitzen-Oldtimer von VW und Porsche</h2>
   <p>Wir f&uuml;hren und reparieren Old- und Youngtimer der Marken VW und
Porsche.</p>
</div>

<div id="navigation">
   <ul>
     <li><a href="angebote.html">Aktuelle Angebote</a></li>
     <li><a href="volskwagen.html">Unsere VWs</a></li>
     <li><a href="porsche.html">Unsere Porsche</a></li>
     <li><a href="preisrechner.html">Preisrechner</a></li>
     <li><a href="kontakt.html">Reparaturen</a></li>
   </ul>
</div>
```

```
<div id="inhalt">
   <p>Wir importieren auch aus Kalifornien!</p>
   <img src="beispiel_2-0-car.png" alt="Karmann Ghia Auto">
   <p>Neu im Angebot: Volkswagen Karmann Ghia</p>
</div>

<div id="kontakt">
   <p><b>Oldtimer Spiering</b><br>Beispielgasse 1, 12345 Bad
Beispielstadt</p>
</div>

</body>
</html>
```

Mit den entsprechenden CSS-Definitionen für die Desktop-Ausgabe (*computer.css*) wird der Beispielcode im Safari-Browser auf dem Mac (und auf dem PC) wie folgt dargestellt:

Bild 1.1: Darstellung des Beispiels als Desktopwebseite.

Alle neuen Smartphones, allen voran natürlich das iPhone, Handys mit dem Android-Betriebssystem oder Handys von Palm mit dem WebOS-System besitzen bereits fantastische Browser, die die PC-Webseite ohne Probleme auch ohne handyspezifisches CSS wiedergeben können. Nehmen wir also an, dass wir die gleichen CSS-Definitionen wie in *computer.css* auch in *mobile.css* abgespeichert haben, dann sieht unsere Seite auf einem iPhone wie folgt aus:

Bild 1.2: Darstellung des Beispiels auf einem iPhone ohne mobilspezifisches CSS.

Nicht schlecht, oder? Die Seite ist identisch mit der Seite auf dem PC, bringt aber auch einige Probleme mit sich. Die Links in der Navigation sind kaum sichtbar und, da das iPhone und viele andere moderne Handys mit dem Finger bedient werden, kaum verlässlich selektierbar. Weiterhin ist die kleine Schrift schlecht zu lesen, insbesondere wenn die gesamte Seite dargestellt wird. Man kann natürlich den Zoom benutzen, doch dann muss man nicht nur vertikal, sondern auch horizontal scrollen, was den Spaß am Lesen der Seite verdirbt.

Um diese Desktop-Seite nun in eine mobile Webseite zu verwandeln, bedarf sie nur ein bisschen CSS und einer HTML-Änderung. Zwar hatten wir versprochen, dass der HTML-Code nicht angefasst werden muss, doch wird der folgende Befehl von allen nicht mobilen Browsern ignoriert und spielt nur für mobile Browser eine entscheidende Rolle:

```
<meta name="viewport" content="width=device-width; initial-scale=1.0;
maximum-scale=1.0; user-scalable=0;" />
```

Der Viewport eines Smartphones ist der Bereich, in dem eine Webanwendung sichtbar laufen kann. Der Viewport einer PC-Webseite ist in der Regel die Größe des Browserfensters, das auch vom Benutzer vergrößert oder verkleinert werden kann. Beim iPhone ist der Viewport wesentlich kleiner und auf die Bildschirmgröße des iPhones minus des Bereichs für die Zeit- und Batterieanzeige, für die URL-Leiste und die untere Funktionsleiste reduziert.

Den Viewport kann man am besten mit einem Stück Papier vergleichen, das über einem Buch liegt und es abdeckt. Wenn Sie aus diesem Stück Papier ein Viereck ausschneiden und es über das Buch schieben, ist dies ein ähnlicher Mechanismus wie der Viewport eines mobilen Geräts.

Wenn Sie eine Webseite auf einem mobilen Gerät scrollen, wollen Sie etwas unbedingt vermeiden: einen horizontalen Scroll. Bricht Ihre Webseite beim Hoch- und Runterscrollen jedes Mal links oder rechts aus und verdeckt dabei Inhalte, ist sie schwer zu bedienen und wird nur wenig Liebhaber finden. Die Definition des Viewports mit dem oben genannten `meta`-Tag bewirkt, dass der Viewport exakt auf die Bildschirmbreite beschränkt wird.

Sie sehen hier, dass wir mit `content="width=device-width..."` keinen exakten Zahlenwert angeben, da sich die Breite des Viewports beispielsweise ändert, wenn Sie das iPhone drehen und der Webseiteninhalt im Querformat angezeigt wird. Die weiteren Parameter beschreiben, dass die Webseite in der Originalgröße angezeigt wird und der Benutzer die Darstellung selbst nicht skalieren kann: `user-scalable=0`.

Allein durch diese eine Zeile verbessert sich die Lesbarkeit der Webseite enorm, da Safari nicht versucht, die Seite in der kompletten Breite darzustellen, und eine besonders kleine Zoomstufe für die Seite wählt.

Bild 1.3: Webseite nach Verwendung des Viewport-`meta`-Tags.

Die Seite berücksichtigt immer noch die in der CSS-Definition festgelegten Seitenabstände, wird aber nicht mehr automatisch »klein gezoomt« und lässt sich nicht mehr seitlich, sondern nur noch nach oben und nach unten verschieben. Dennoch ist das Design noch meilenweit von dem einer iPhone-Webanwendung entfernt. Aus diesem Grund geht es jetzt an das CSS in der Datei *mobile.css*.

In fast allen iPhone-Webanwendungen wird die Titelleiste von den mitgelieferten Anwendungen nachgebaut. Beispielsweise nutzt die iPod-Anwendung auf dem iPhone die Titelleiste zur Hauptnavigation.

Navigation im Kopfbereich

Der Nachbau der Kopfleiste als Navigationsleiste ist vor allem sinnvoll, da Safari nach dem Laden einer Webseite oder unter bestimmten Umständen (dazu später mehr) die URL-Eingabeleiste ausblendet und nur noch die Kopfleiste als erstes Element sichtbar bleibt. Dieses Verhalten finden Sie übrigens nicht nur im mobilen Safari-Browser, sondern auch bei seinen Kollegen auf WebOS-Geräten von Palm, den Android-Handys und anderen Browserimplementierungen.

Mit dem folgenden CSS wird noch keine Schaltfläche in den Kopfbereich implementiert, allerdings wird die Titelleiste nach der CSS-Behandlung schon wesentlich mehr nach dem iPhone aussehen.

Zuerst wird der gesamte Seitenhintergrund auf einen Grauton gesetzt. Als Schriftart nehmen wir die Standardschriftart des iPhones, Helvetica (`font-family: Helvetica`) in einem sehr dunklen Grauton (`color: #222;`).

```
body {
   background-color: #ddd;
   color: #222;
   font-family: Helvetica;
   font-size: 14px;
   margin: 0; padding: 0;
   }
```

Als Titelzeile soll nicht der gesamte Kopfbereich, wie er in `div id="kopfbereich"` definiert wurde, sondern nur die erste Zeile mit dem HTML-Tag `h1` verwendet werden. Das CSS, das das Aussehen der Titelzeile definiert, wird demnach nur auf das `div #kopfbereich` und das HTML-Tag `h1` beschränkt:

```
#kopfbereich h1 {
   background-color: #fcc97d;
   border-bottom: 1px solid #666;
   color: #222;
   display: block;
   font-size: 20px;
   font-weight: bold;
   padding: 10px 0;
   text-align: center;
   text-decoration: none;
text-shadow: 0px 1px 0px #fff;
}
```

Mit der Farbdefinition `background-color: #fcc97d;` wird ein orangefarbener Hintergrund gesetzt, der sich auch auf der PC-Seite fand. Falls Ihnen mehr nach dem Apple-Grau der iPod-Anwendung ist, setzen Sie den Wert auf `#98a8bf`. Die CSS-Definition `text-shadow: 0px 1px 0px #fff;` versieht die dunkelgraue Schrift in der Kopfzeile mit einem kleinen, weißen Schatten.

Damit auch die anderen Inhalte im Kopfbereich-`div` ordentlich dargestellt werden, auch wenn sie nicht zur Titelzeile gehören, fügen wir noch folgendes CSS ein:

```
#kopfbereich h2, p {
  margin-left: 12px;
  margin-right: 10px;
}
```

Die Titelzeile erscheint jetzt schon als eigenständiger Block, allerdings fehlt noch der für das iPhone typische Farbverlauf. Dieser kann entweder als Bild eingebunden oder viel besser automatisch vom Browser über die folgende Anweisung generiert werden:

```
#kopfbereich h1 {
background-image: -webkit-gradient(linear, left top, left bottom,
from(#fcc97d), to(#fcaf3b));
}
```

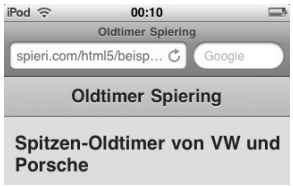


Bild 1.4: Fertig ist eine iPhone-typische Kopfzeile in Orange.

Die Farbangaben bestimmen einen Verlauf von einem hellen Orange in ein dunkles Orange. Wollen Sie die iPhone-Grautöne verwenden, ändern Sie in der CSS-Definition für `#kopfbereich h1` die folgenden Zeilen:

```
background-image: -webkit-gradient(linear, left top, left bottom,
from(#b2bcc8), to(#76859a));
text-shadow: 0px -1px 0px #222;
color: #fff;
```

Diese Angaben versehen die Kopfleiste mit einem grauen Farbverlauf, einer weißen Schrift und einem umgekehrten dunklen Schatten.

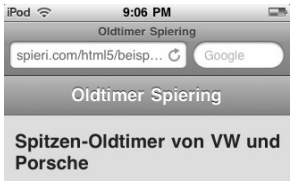


Bild 1.5: Eine Kopfzeile im iPhone-typischen Grau mit weißer Schrift.

Ein wirklicher Problembereich war die Navigationsspalte, als die Webseite mit der PC-Formatierung auf dem iPhone angezeigt wurde. Die Links waren zu klein und somit schwer anzutippen und sahen auch überhaupt nicht nach den gewohnten Linkflächen einer iPhone-Webanwendung aus. Mit den folgenden CSS-Definitionen werden die Links, die als `li`-Elemente aufgelistet waren, in große, schöne Schaltflächen umgewandelt. Dabei wird eine weiße Hintergrundfarbe für die Schaltflächen benutzt, die mit einem 1 Pixel starken Rahmen umgeben sind:

```
div#navigation ul {
   list-style: none;
   margin: 10px; padding: 0;
}

div#navigation ul li a {
   background-color: #FFFFFF;
   border: 1px solid #d9d9d9;
   color: #222222;
   display: block;
   font-size: 17px;
   font-weight: bold;
   margin-bottom: -1px;
   padding: 12px 10px;
   text-decoration: none;
}
```

Was für die Titelzeile der Farbverlauf ist, sind für die Schaltflächen die runden Ecken. Die folgenden CSS-Zeilen fügen genau diese Rundungen an den jeweiligen Ecken mit einem Radius von `10px` der Liste hinzu:

```
#navigation ul li:first-child a{
   -webkit-border-top-right-radius: 10px;
   -webkit-border-top-left-radius: 10px;
}

#navigation ul li:last-child a{
   -webkit-border-bottom-right-radius: 10px;
   -webkit-border-bottom-left-radius: 10px;
}
```

Das Ergebnis kann sich nach dem Hinzufügen dieser wenigen CSS-Zeilen bereits sehen lassen und dürfte kaum noch an eine normale PC-Seite erinnern, sondern vielmehr an eine schicke iPhone-Webanwendung:

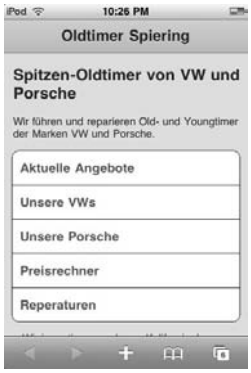


Bild 1.6: Webseite mit iPhone-typischer Kopfzeile und Navigationselementen.

Jetzt ist nur noch der restliche Inhalt in die passende Form zu bringen, was bedeutet, dass sich der Inhalt an den rechten und linken Abständen orientieren muss:

```
#inhalt {
  margin-left: 12px;
  margin-right: 10px;
}
```

Beim angezeigten Bild haben wir etwas geschummelt und das Bild bereits auf eine Breite von 298 Bildpunkten zugeschnitten, sodass es perfekt in die Seite passt. Warum aber 298 Bildpunkte?

320 Bildpunkte ist die Breite des iPhone-Bildschirms. Wenn Sie 12 Punkte am linken Rand (`margin-left: 12px;`) und 10 Punkte am rechten Rand (`margin-right: 10px;`) abziehen, ergibt sich die Bildbreite von 298 Pixeln.

Was man beim Bild noch hinzufügen kann, sind ähnlich wie bei den Navigationselementen die Abrundungen an den Ecken. Man sieht diese Behandlung von Bildern beispielsweise im Adressbuch des iPhones, wo ein dünner Rahmen um das Kontaktfoto gezeichnet ist und leichte Abrundungen an den Ecken vorgenommen wurden. Die folgende CSS-Definition verändert genau dies am angezeigten Bild:

```
#inhalt img {
   border: 1px solid #222222;
   -webkit-border-radius: 5px;
}
```

Die Fußzeile, also das Adressfeld des Autohändlers, soll ähnlich wie der Kopfbereich aussehen, jedoch eine Linie am oberen Rand erhalten, die Schrift soll kleiner und ohne Schatten sowie zentriert ausgerichtet sein. Demzufolge muss der Datei *mobile.css* noch folgende CSS-Definition hinzugefügt werden:

```
#kontakt {
   background-color: #fcc97d;
   border-top: 1px solid #666;
   color: #222;
   display: block;
   font-size: 10px;
   font-weight: bold;
   padding: 5px 0;
   text-align: center;
   text-decoration: center;
}
```

Nach diesen eigentlich wenigen Zeilen CSS haben Sie eine Weboberfläche, die für das iPhone und andere Touchscreen-Handys optimiert ist. Dabei musste, bis auf eine zusätzliche Zeile, der HTML-Code nicht geändert werden, und die zwei komplett unterschiedlichen Designs wurden einfach nur von einigen Zeilen CSS definiert.

Bild 1.7: Fertig – die Webseite sieht jetzt nach einer iPhone-Webanwendung aus.

Sie wissen nun, dass das Aussehen einer Webseite getrennt vom HTML-Code definiert werden kann, und haben erfahren, wie man mit ein paar CSS-Kenntnissen eine iPhone-Seite gestalten kann. Natürlich sind wir in diesem Buch nicht die Ersten, die mit einer Reihe von CSS-Definitionen versucht haben, Webseiten für das iPhone und andere mobile Plattformen zu optimieren. Es gibt bereits eine ganze Reihe von vorgefertigten CSS- und JavaScript-Frameworks, die Sie einfach herunterladen und in Ihren Projekten verwenden können.

1.3 Erkennung von mobilen Browsern und Desktop-Browsern

Sie haben es vielleicht schon geahnt, aber die Unterscheidung anhand der folgenden Codezeilen, ob ein mobiles Gerät oder ein Desktop-Browser die Seite aufruft, ist relativ unzuverlässig:

```
<link rel="stylesheet" type="text/css"
      media="screen and (min-width: 961px)" href="computer.css">
<link rel="stylesheet" type="text/css"
      media="only screen and (max-width: 960px)" href="mobile.css">
```

Diese Art der Unterscheidung war früher relevant, als Mobiltelefone noch nicht die Bildschirmauflösungen von PC-Monitoren oder Laptops hatten, doch jetzt verschwindet diese Grenze immer mehr. Wir haben aber mit dem Beispiel noch etwas anderes bezweckt: Wir wollten Ihnen zeigen, dass Inhalt und Aussehen einer Seite komplett unabhängig voneinander gestaltet werden können und dass Sie die Möglichkeit besitzen, einfach über CSS eine iPhone-Version Ihrer PC-Seite zu erstellen, ohne dass Sie auch nur eine Zeile der HTML-Befehle ändern müssen.

In der Praxis würden wir von dieser Methode abraten, es sei denn, Sie haben nur eine kleine, gut kontrollierbare Desktop-Seite. Alle größeren Webanbieter haben einen separaten Seitencode für Desktop-Webseite und Mobilseite. Dies sehen Sie allein schon daran, dass mobile Webseiten oft sogar unter einer eigenen Domain (beispielsweise *m.spiegel.de*) laufen.

Die meisten Webseitenanbieter verweisen automatisch auf die mobile Seite, wenn ein iPhone-Benutzer die Desktop-Seite ansteuert. Auch wenn Sie in Ihrem iPhone *www.spiegel.de* eingeben, landen Sie wenige Augenblicke später auf *m.spiegel.de*.

Um eine derartige Logik in der eigenen Webseite zu implementieren, bedarf es der Kenntnis des User-Agent-Headers. Neben den auf einer Webseite sichtbaren Daten wird noch eine Reihe unsichtbarer Daten, die sogenannten HTTP-Header, ausgetauscht. Dies passiert immer dann, wenn Sie einen Webserver, beispielsweise durch die Eingabe einer Webadresse, von Ihrem Browser aufrufen.

Die Webseite *www.xhaus.com/headers* zeigt Ihnen, welche Informationen im HTTP-Header übertragen werden. Testen Sie es einmal von Ihrem Telefon oder Ihrem Desktop-Browser aus.

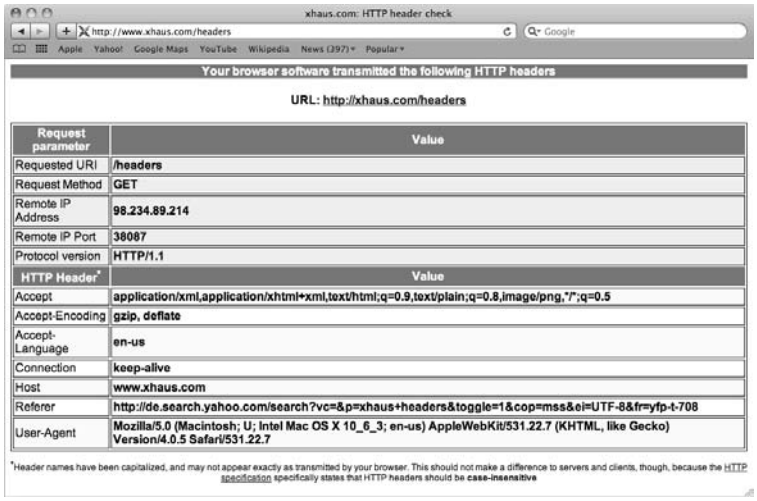


Bild 1.8: Anzeige der übertragenen Informationen im HTTP-Header.

Diese HTTP-Header helfen dem Webserver, die richtigen Daten auszuliefern, da sie wichtige Informationen wie …

- … die Spracheinstellung des Browsers mitteilen: *Accept-Language*.
- … die URL übertragen, von der aus der Benutzer Ihre Seite erreicht hat: *Referer*.
- … den Browser der Benutzer übermittelt: *User-Agent*.

In diesen sogenannten *User-Agents* wird beispielsweise für den von mir gerade verwendeten Browser folgende Zeichenfolge angezeigt:

```
Mozilla/5.0 (Macintosh; U; Intel Mac OS X 10_6_3; en-us)
AppleWebKit/531.22.7 (KHTML, like Gecko) Version/4.0.5 Safari/531.22.7
```

Neben der Information, dass ein Apple-Computer mit Mac OS X 10.6.3 die Seite aufgerufen hat, wird auch deutlich, dass der Safari-Browser hinter dem Aufruf steckt.

User-Agent-Zeichenfolgen für mobile Browser	
iPhone 4	Mozilla/5.0 (iPhone; U; CPU iPhone OS 4_0 like Mac OS X; de-de) AppleWebKit/532.9 (KHTML, like Gecko) Version/4.05 Mobile/8A293 Safari/6531.22.7
iPod touch	Mozilla/5.0 (iPod; U; CPU iPhone OS 4_0 like Mac OS X; de-de) AppleWebKit/532.9 (KHTML, like Gecko) Version/4.0.5 Mobile/8A293 Safari/6531.22.7
Google Nexus One	Mozilla/5.0 (Linux; U; Android 2.2; de-de; Nexus One Build/FRF91) AppleWebKit/533.1 (KHTML, like Gecko) Version/4.0 Mobile Safari/533.1
iPad	Mozilla/5.0 (iPad; U; CPU OS 3_2 like Mac OS X; de-de) AppleWebKit/531.21.10 (KHTML, like Gecko) Version/4.0.4 Mobile/7B334b Safari/531.21.10
Palm Pre	Mozilla/5.0 (webOS/1.0; U; de-DE) AppleWebKit/525.27.1 (KHTML, like Gecko) Version/1.0 Safari/525.27.1 Pre/1.0

In jedem dieser User-Agents sehen Sie, welches Gerät, welcher Browser und welches Betriebssystem vom Benutzer verwendet wird. Anhand dieser Informationen ist es einfach, die richtige Version der Webseite (mobil oder nicht mobil) auszuliefern. Dazu gibt es verschiedene Möglichkeiten:

1.3.1 User-Agent mit JavaScript abfragen

Die User-Agent-Zeichenfolge ist im JavaScript-Objekt `navigator.userAgent` enthalten. Die folgende kleine Webseite demonstriert, wie einfach es ist, den User-Agent eines Benutzers zu erhalten und anzuzeigen:

```
<html>
    <head>
        <title>User-Agent-Bestimmung</title>
      <meta name="layout" content="webkit" />
      <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
      <meta name="layout" content="webkit" />
      <link href="css/style.css" rel="stylesheet" media="screen"
type="text/css" />
    </head>
    <body>

      <div id="topbar">
         <div id="title">Ihr User-Agent</div>
      </div>
      <div id="content">
         <ul class="pageitem">
            <li class="textbox">
               <script>document.write(navigator.userAgent)</script>
            </li>
         </ul>
      </div>
    </body>
</html>
```

Das Ergebnis ist eine iPhone-Seite, die immer den aktuellen User-Agent des Browsers ausgibt:

Bild 1.9: iPhone-Seite mit aktuellem User-Agent

Um eine Umleitung einzurichten, wäre es aber aufgrund der Vielzahl der möglichen User-Agent-Zeichenfolgen – der User-Agent ändert sich bei jeder neuen Spracheinstellung, beim Aktualisieren des Systems etc. – falsch, die komplette Zeichenfolge zu vergleichen. Stattdessen werden wir nur im User-Agent schauen, ob die Begriffe »iPhone«, »iPod« oder »Android« vorkommen. In JavaScript kann man das mit folgendem Code erreichen:

```
if((navigator.userAgent.match(/iPhone/i)) ||
(navigator.userAgent.match(/iPod/i)) ||
(navigator.userAgent.match(/Android/i))) {
    if (document.cookie.indexOf("iphone_redirect=false") == -1)
window.location = "http://m.ihreseite.de";
}
```

Dieser Code schaut also nach den oben genannten Begriffen und leitet, wenn sie im User-Agent vorkommen, automatisch auf die Seite *m.ihreseite.de* weiter. Beachten Sie, dass Sie bei der Zieladresse unbedingt das *http://* voranstellen. Folgerichtig muss dieser Code auf der Desktop-Seite eingebaut werden:

```
<html>
 <head>
  <title>Browser-Erkennung</title>
  <script>
     if((navigator.userAgent.match(/iPhone/i)) || (navigator.userAgent.
match(/iPod/i)) || (navigator.userAgent.match(/Android/i)))
       {
    if (document.cookie.indexOf("iphone_redirect=false") == -1)
window.location = "http://m.ihreseite.de";
       }
  </script>
 </head>
 <body>
  <h1>Ich bin eine PC-Seite!</h1>
 </body>
</html>
```

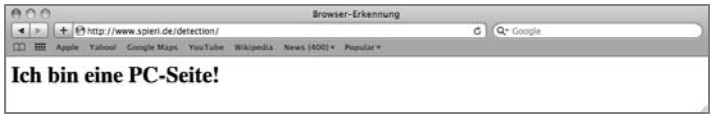


Bild 1.10: Wird der oben stehende Code von einem Desktop-Browser aufgerufen, bleibt der Benutzer auf der aufgerufenen Seite. Es wird nicht auf die mobile Seite umgeschaltet.

Bild 1.11: Ruft der Anwender den oben stehenden Code auf einem iPhone, einem iPod oder einem Android-Gerät auf, wird er auf die im Code angegebene Adresse weitergeleitet.

1.3.2 User-Agent mittels der .htaccess-Datei abfragen

Anstatt mit JavaScript zu arbeiten, können Sie auch in der *.htaccess*-Datei eines Verzeichnisses auf Ihrem Webserver eine ähnliche Weiterleitung realisieren.

Eine *.htaccess*-Datei ist eine einfache Textdatei, die Sie mit jedem Texteditor, mit dem Sie auch HTML-Code schreiben, erstellen und bearbeiten können. Falls Sie sich bereits mit Webentwicklung auskennen, haben Sie bestimmt schon von dieser Datei gehört, die in der Regel immer dazu benutzt wird, bestimmte Verzeichnisse auf einem Webserver mit einem Passwort zu schützen. Die Angaben in dieser Datei können aber noch viel mehr – siehe hierzu auch das Kapitel 2.4 »Audio und Video«.

Eine solche Datei zu erstellen, ist nicht immer einfach, da der volle Dateiname *.htaccess* lautet. Streng genommen handelt es sich um eine Datei, die keinen Namen besitzt, sondern nur die Endung *.htaccess*. Eine ganze Reihe von Texteditoren kann diesen Namen nicht abspeichern, da er mit dem Symbol einer Dateiendung beginnt. In diesem Fall müssen Sie einfach einen Buchstaben vergeben, die Datei abspeichern und sie dann auf dem Webserver umbenennen.

1.3.3 Geltungsbereiche der .htaccess-Datei auf dem Webserver

Da wir die *.htaccess*-Datei für eine Weiterleitung verwenden möchten, ist es absolut notwendig, uns kurz mit dem Geltungsbereich einer solcher Datei zu beschäftigen. Die *.htaccess*-Datei kann das Verhalten des Browsers in vielen Bereichen bestimmen. Dabei ist eine *.htaccess*-Datei für das Verzeichnis gültig, in dem sie sich befindet, aber auch für alle darin befindlichen Unterverzeichnisse, solange sich in denen keine neue *.htaccess*-Datei befindet.

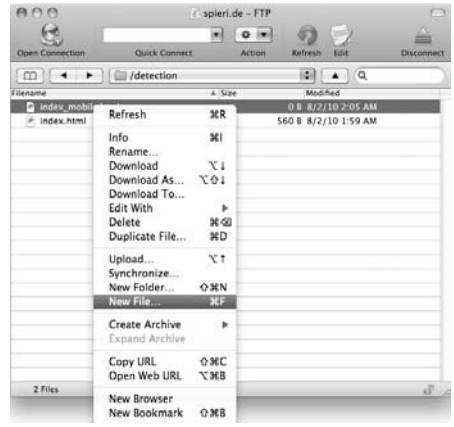
Richten Sie also eine *.htaccess*-Datei für Ihr Verzeichnis *www.ihreseite.de/web/* ein, dann sind die in der Datei enthaltenen Regeln auch gültig für *www.ihreseite.de/web/beispiel* oder *www.ihreseite.de/web/adresse*.

Tragisch wird es, wenn Sie die Weiterleitungsregel in der *.htaccess*-Datei für das Verzeichnis *www.ihreseite.de/web/* schreiben und mobile Benutzer zu Ihrer mobilen Adresse *www.ihreseite.de/web/mobil* schicken. In diesem Fall kommt ein mobiler Benutzer auf Ihre Seite, wird zu der mobilen Adresse geschickt, für die wiederum die Weiterleitungsregel gilt, und befindet sich ab sofort in einer Schleife, die auf dem iPhone mit dem Fehler *Too many redirects* (zu viele Weiterleitungen) endet.

Unser Tipp daher: Verwenden Sie eine andere Webadresse oder eine neue *.htaccess*-Datei im Zielverzeichnis ohne den Weiterleitungscode.

Am einfachsten ist es, wenn Sie die *.htaccess*-Datei mit einem FTP-Programm auf dem Webserver anlegen. Wir zeigen Ihnen das am Beispiel von Cyberduck:

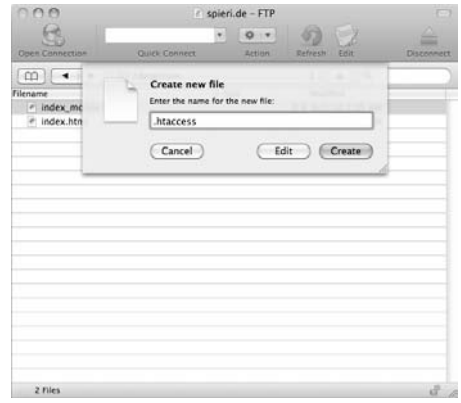
1. Gehen Sie in das Verzeichnis, in dem Sie Ihre Desktop-Webseite bereits haben oder anlegen wollen. Über die rechte Maustaste können Sie aus dem Kontextmenü die Option *Neue Datei* bzw. *New File* auswählen.



2. Vor dem Erstellen der Datei fragt Cyberduck Sie nach dem Dateinamen. An dieser Stelle geben Sie einfach *.htaccess* ein und klicken auf *Create*.

3. Anschließend wird die Datei als leeres File (0 Byte) auf dem Webserver erstellt. Mit der rechten Maustaste können Sie nun aus dem Kontextmenü die Funktion *Bearbeiten mit* bzw. *Edit with* und den entsprechenden Texteditor wählen.

   Falls Sie TextMate verwenden, wird dieser automatisch dort in das Menü eingebunden, und jegliche Änderungen werden von TextMate über die *Sichern*-Funktion auch direkt zurück auf den Webserver gespeichert.



Ist die Datei erstellt, versehen Sie die *.htaccess*-Datei mit dem folgenden Code:

```
RewriteEngine on
RewriteCond %{HTTP_USER_AGENT} ^.*iPhone.*$
RewriteRule ^(.*)$ http://m.ihreseite.de [R=301]
RewriteCond %{HTTP_USER_AGENT} ^.*iPod.*$
RewriteRule ^(.*)$ http://m.ihreseite.de [R=301]
RewriteCond %{HTTP_USER_AGENT} ^.*Android.*$
RewriteRule ^(.*)$ http://m.ihreseite.de [R=301]
```

Beide vorgestellten Methoden funktionieren sehr gut, und es ist fast ein wenig Geschmackssache, auf welche Lösung Sie vertrauen wollen. Die *.htaccess*-Methode hat den Vorteil, dass der Benutzer etwas Zeit spart, da die Weiterleitungsregeln bereits beim Auslesen des Verzeichnisses verwendet werden und nicht erst, nachdem die erste Webseite geladen und das darin enthaltene JavaScript ausgeführt ist.

Bei vielen Billig-Webservern wird Ihnen die Wahl automatisch abgenommen, da sie keine Veränderung oder Erstellung einer *.htaccess*-Datei zulassen. Informieren Sie sich daher bei Ihrem Hosting-Anbieter, ob Sie Zugriff auf die *.htaccess*-Dateien innerhalb Ihres Webspace haben.

Exkurs: Die PHP-Variante
Obwohl wir in diesem Buch PHP nicht weiter vorstellen oder verwenden, möchten wir Sie auch noch auf diese Variante aufmerksam machen. Sie hat ebenfalls den Vorteil, dass nicht erst der Inhalt einer Seite geladen werden muss und die Entscheidung, ob die mobile Seite oder die normale Seite aufgerufen wird, bereits auf dem Webserver fällt. Nur für PHP-Kenner – hier ist der passende PHP-Code:

```
if(strstr($_SERVER['HTTP_USER_AGENT'],'iPhone') || strstr($_SERVER['HTTP_
USER_AGENT'],'iPod') || strstr($_SERVER['HTTP_USER_AGENT'],'Android'))
{
        header('Location: http://m.ihreseite.de');
        exit();
}
```

⊡ Lesezeichen

http://bit.ly/bgvJGS

Sollten Sie sich für PHP interessieren, bietet Franzis mit »Das Franzis-Lernpaket PHP/MySQL« einen soliden und leicht verständlichen Einstieg an.

1.3.4 Einen falschen User-Agent vorgeben

Wenn Sie viel auf Ihrem Desktop-Computer arbeiten und dort auch Ihre Webanwendungen testen möchten, ist es ratsam, dass sich Ihr Desktop-Browser nicht als PC, sondern als Mobiltelefon ausgibt. Insbesondere wenn Sie beispielsweise eine Android-Erkennung einbauen oder unterschiedliche Webseiten für iPhone-, Android- und BlackBerry-Plattformen erstellen, ist es oft einfacher, auf dem PC statt auf vielen verschiedenen Mobilgeräten zu testen. Einige gängige User-Agent-Strings haben wir bereits abgedruckt – den passenden String zu nahezu jedem Telefon auf der Welt können Sie im Internet finden.

Am einfachsten lassen sich die User-Agents mit Safari umstellen. Ist das *Entwickler*-Menü aktiviert, können Sie aus einer ganzen Reihe verschiedener User-Agent-Strings wählen – angefangen bei Safari für Mac und Windows über Safari für iPhone, iPad und iPod touch bis hin zu Firefox und Opera. Alternativ können Sie über *Anderer* bzw. *Other* einen eigenen User-Agent-String eingeben, mit dem sich der Browser nun ausgeben soll.

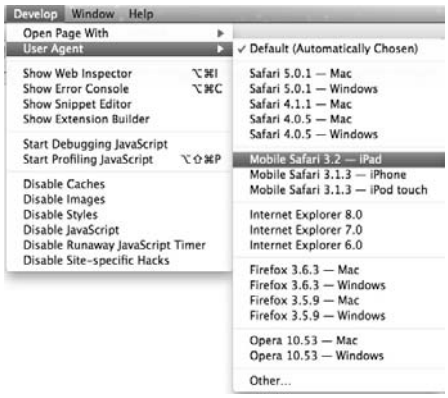


Bild 1.12: Verschiedene User-Agents können bei Safari einfach aus dem *Entwickler*-Menü ausgewählt werden.

Firefox hat keine eingebaute Funktion zum Wechseln des User-Agents, man kann diese über die kostenlose Erweiterung *User Agent Switcher* aber nachrüsten. Die Erweiterung finden Sie auf der Webseite unter *https://addons.mozilla.org/en-US/firefox/addon/59* und können sie von dort direkt in Firefox installieren. Nach dem erforderlichen Neustart des Browsers finden Sie im *Tools*-Menü den nun aktuell benutzten User-Agent angezeigt. *User Agent Switcher* bietet als einzigen mobilen User-Agent den iPhone-String an, lässt sich aber leicht um alle möglichen Strings erweitern. Dazu wählen Sie einfach *Edit User Agents* aus dem *Tools*-Menüeintrag aus und legen einen neuen User-Agent über *New* an.

Glücklicherweise haben viele andere das schon gemacht und XML-Dateien mit vorgefertigten User-Agents im Internet veröffentlicht. Eine solche Liste finden Sie beispielsweise unter *http://techpatterns.com/downloads/firefox/useragentswitcher.xml*. Diese XML-

Datei können Sie per Importfunktion über *Edit User Agents* laden und somit Ihr virtuelles Geräteportfolio um einige Dutzend neuer User-Agent-Strings erweitern.

Für den Google Chrome-Browser gibt es bislang noch keine einfache Möglichkeit, den User-Agent zu modifizieren. Da Chrome aber, wie Firefox und Safari, auch mit Plug-ins, sogenannten Extensions, erweitert werden kann, lohnt es sich, bei Bedarf von Zeit zu Zeit unter *https://chrome.google.com/extensions* nach einer solchen Erweiterung zu suchen.

1.4 CSS- und JavaScript-Frameworks

Diese sogenannten Frameworks nehmen Ihnen in der Regel viel Arbeit ab. Sie brauchen nicht wie im vorangegangenen Beispiel jede Zeile CSS neu zu erfinden, sondern verwenden einfach die CSS-Dateien und Bilddateien, die mit dem Framework geliefert werden. Sie brauchen dann nur noch darauf zu achten, dass Sie im HTML-Code auf die richtigen Klassen verweisen. Das Aussehen einer Titelzeile, eines iPhone-Buttons oder der iPhone-Schieberegler wird komplett vom Framework bestimmt. Zwar brauchen Sie nur noch einen Bruchteil an Zeit (und Können), um eine mobile iPhone-Seite so zu gestalten, dass sie professionell aussieht, allerdings gibt es auch zwei Nachteile:

- Ein Framework enthält in der Regel immer mehr Definitionen und Code, als Sie für Ihr Projekt und Ihre Seite brauchen. Beispielsweise sind in den Frameworks oft verschiedene Farben für die Titelzeile einer iPhone-Webanwendung definiert, die Sie – außer Sie ändern per Hand den vordefinierten CSS-Code – alle installieren, obwohl Sie sehr wahrscheinlich nur eine Farbdefinition nutzen werden. Die Verwendung eines Frameworks bringt es zwangsläufig mit sich, dass ein Besucher Ihrer Seite mehr Daten auf sein mobiles Gerät laden muss als unbedingt notwendig. Falls Sie sich entscheiden, aus HTML-Code, CSS und JavaScript eine native Anwendung zu bauen, müssen die CSS-Framework-Daten nicht aus dem Netz heruntergeladen, aber mit der Datei gespeichert werden, was eine größere Datei nach sich zieht.
- Die Namen der Selektoren und CSS-Klassen sind vorbestimmt, und Sie müssen Ihr HTML an diese Klassen anpassen bzw. von Anfang an Ihren HTML-Code so konzipieren. Den gleichen HTML-Code für eine normale Webseite und für eine mobile Seite zu benutzen, wird dadurch wesentlich schwieriger.

Zwei der Frameworks, iWebKit 5 und jQTouch, stellen wir Ihnen jetzt vor. Bei den Beispielen im Buch haben wir uns auf iWebKit 5 beschränkt, da dieses Framework einfacher zu verstehen ist und einen schnelleren Einstieg ermöglicht. Im Gegensatz zu iWebKit ist jQTouch dynamischer und bietet Ihnen mehr Effekte und Animationen, als Sie bei iWebKit finden werden.

1.4.1 iWebKit 5

iWebKit 5 ist ein sehr beliebtes und kompaktes CSS-Framework und kann unter *http://iwebkit.net* kostenlos heruntergeladen werden. Falls Sie Gefallen an dem Framework finden, sollten Sie einen Betrag per PayPal spenden.

Das nächste Beispiel zeigt Ihnen einen ähnlichen HTML-Code wie den in unserem letzten Beispiel, in dem wir alle iPhone-CSS-Definitionen noch selbst gebaut hatten, allerdings werden `div`-Bereiche definiert. Anhand dieser `div`-Zuweisungen gestalten Sie die mobile Seite, da jeder `div`-Name für ein bestimmtes Element aus dem Framework steht. Um mit dem Framework zu arbeiten, müssen Sie es herunterladen, die ZIP-Datei entpacken und die Dateien und Ordner aus dem Verzeichnis *Framework* in das Verzeichnis Ihrer Webseite kopieren. Die CSS-Definitionen des iWebKit-Frameworks sind in den beiden Dateien *style.css* und *developer-style.css* definiert. Beide Dateien haben den gleichen Inhalt, allerdings wurden in der Datei *style.css* alle Leerzeichen und Zeilenumbrüche entfernt und somit einige KByte an Daten eingespart. Durch die fehlende Formatierung kann diese Datei aber nur von einem Browser verlässlich gelesen werden. Falls Sie wissen möchten, welche CSS-Selektoren definiert wurden, sollten Sie die Datei *developer-style.css* verwenden.

Da wir bei der aktuellen Seite nur von einem mobilen Design ausgehen, wird auch nur auf die eine CSS-Datei mit `link rel="stylesheet" type="text/css" href="style.css"` verwiesen. Zwar enthält iWebKit die meisten iPhone-typischen Elemente, allerdings keine vordefinierte CSS-Klasse, mit der man die Abrundungen an Bildecken hervorrufen kann. Diese Funktion wird im HTML-Code mithilfe des `style`-Elements und der darin befindlichen CSS-Definition bestimmt: Jedes `img`-Tag im `div`-Bereich `textimg` erhält runde Ecken mit einem Radius von fünf Bildpunkten und einen dunkelgrauen Rand.

Schon bei der Gestaltung der Titelzeile bemerken Sie den Komfort eines CSS-Frameworks. Sie müssen im Code einfach `div id="topbar"` und innerhalb dieser `div`-Definition den gewünschten Titel mit `div id="title"` einfügen, und schon haben Sie die Standardtitelzeile eines iPhones. In diesem Beispielcode haben wir die Kopfzeile noch weiter angepasst:

Die Zeile `div id="topbar" class="transparent"` bewirkt eine leicht dunkle, transparente Titelzeile. Alternativ können Sie mit `div id="topbar" class="black"` auch eine komplett schwarze Titelzeile erstellen. Da beim iPhone diese Titelzeile auch zur Navigation verwendet wird, lassen sich mit iWebKit ganz einfach Navigationselemente in den Titel einbauen. Das Beispiel besitzt dazu einen blauen Button, den man oft in iPhone-Anwendungen als Anmeldebutton findet.

Für unsere Beispielseite wollen wir dieses Navigationselement verwenden, um auf unsere Kontaktseite zu verweisen. Der blaue Button wird ebenfalls innerhalb des `div id="topbar"`-Bereichs mit folgender Zeile eingesetzt:

```
<div id="bluerightbutton"><a href="kontakt.html">Kontakt</a></div>.
```

Insbesondere auf Unterseiten einer iPhone-Anwendung befinden sich auf der linken Seite der Titelzeile ein oder mehrere Pfeile, die es dem Benutzer erlauben, auf den vorangegangenen Bildschirm zurückzugehen. Diese Navigationspfeile lassen sich innerhalb der Titelzeilendefinition mit

```
<div id="leftnav"><a href="seite.html">Zurück</a></div>
```

für den Pfeil nach links und mit

```
<div id="rightnav"><a href="seite.html">Weiter</a></div>
```

für den Pfeil nach rechts definieren.

Falls Sie anstatt eines Texts lieber ein Home-Symbol für den rechten Pfeil verwenden möchten, müssen Sie folgende HTML-Zeile einbauen:

```
<div id="leftnav"><a href="seite.html"><img alt="home" src="images/home.png"
/></a></div>
```

Probieren Sie einfach die verschiedenen Variationen aus. Sie werden erstaunt sein, wie einfach sich eine professionell aussehende iPhone-Titelzeile erstellen lässt.

Das nächste Element hatten wir im vorherigen Beispiel nicht verwendet, wir hätten dazu wahrscheinlich seitenweise CSS-Code abdrucken und erfinden müssen. Was Sie ebenfalls in vielen iPhone-Anwendungen finden, ist ein Suchfeld, das sich direkt unterhalb der Titelzeile befindet und in der Regel in grauer Schrift mitteilt, was man in diesem Feld suchen kann.

Mittels iWebKit ist die Einbindung eines solchen Suchfelds ein Kinderspiel und erfordert in der Tat nur wenige Zeilen HTML-Code:

```
<div class="searchbox">
 <form action="http://..." method="get">
  <fieldset>
   <input id="suche" placeholder="Oldtimer-Datenbank durchsuchen"
type="text" />
   <input id="submit" type="hidden" />
  </fieldset>
 </form>
</div>
```

Mit `div class="searchbox"` legen Sie fest, dass die folgenden Zeilen HTML für den Inhalt und die Funktionsweise der Suchbox reserviert werden. Das `form`-Tag ist herkömmliches HTML und beschreibt eine Formulareingabe und was mit den Formulardaten geschehen soll. Interessant sind hier das Tag `fieldset`, das dem Browser mitteilt, dass alle folgenden Formularelemente in einer Gruppe zusammengefasst sind, und natürlich die beiden `input`-Tags für das Texteingabefeld sowie das unsichtbare `submit`-Tag, das ein Absenden des Formulars ermöglicht.

Der eigentliche Inhalt einer iWebKit-Anwendung wird durch das Tag `div id="content"` definiert. Das erste Element ist eine graue Überschrift, die nicht über ein `div`, sondern einfach über eine Klasse des `span`-Tags definiert wird:

```
<span class="graytitle">Spitzen-Oldtimer von VW &amp; Porsche</span>
```

Im Gegensatz zum letzten Beispiel mit dem hausgemachten CSS haben wir die Überschrift etwas gekürzt, da die grauen Überschriften in der Regel nur einzeilig auftreten.

Eines der wichtigsten Gestaltungselemente beim Arbeiten mit iWebKit ist die Definition der Seitenelemente (Page Items). Zusammengehörende Seitenelemente werden durch `ul class="pageitem"` definiert. Im nächsten Beispiel ist das erkennbar an den `pageitem`-Definitionen für die Textbox *Wir führen und reparieren ...* und die nachfolgende Menüstruktur.

Beide Seitenelemente können weitere Unterelemente beinhalten (das Menü hat beispielsweise mehrere Menüpunkte), aber alle zu einem Seitenelement gehörenden

Inhalte und Definitionen werden innerhalb von `ul class="pageitem"` zusammengefasst. Text stellt man am besten, wie im Beispiel gezeigt, durch die Definition einer Textbox dar:

```
<li class="textbox"></li>
```

Damit ist kein Texteingabefeld gemeint, sondern ein sich vom grauen Untergrund abhebender Bereich für die Textanzeige.

Besonders einfach ist auch der Einbau von Touchfeldern für Links. Innerhalb einer Seitenelementdefinition können folgende Bereiche definiert werden:

- die Zielseite mittels des Tags `a href=""`,
- ein Icon,
- die Bezeichnung des Links sowie
- ein Pfeil nach rechts, der kenntlich macht, dass es sich bei dem Feld um einen Verweis auf eine neue Seite handelt.

Im HTML-Code sieht das wie folgt aus:

```
<li class="menu">
   <a href="preisrechner.html">
    <img alt="Rechensymbol" src="../thumbs/calculator.png" />
   <span class="name">Preisrechner</span>
   <span class="arrow"></span>
   </a>
</li>
```

Die Klasse `menu` ruft das Design eines Touch-Menüfelds auf, das Tag `a href=""` verweist auf die Zielseite *preisrechner.html*. Piktogramme und Icons können ohne Klassenzuweisung im `img`-Tag eingefügt werden, während für die Linkbezeichnung die Klasse `span class="name"` und für den Pfeil die Klasse `span class="arrow">` angegeben werden muss.

Zum Abschluss zeigt unser Beispiel noch einmal die Verwendung eines Seitenelements, das neben einer Textbox auch ein Bild enthält. Im Kopfbereich der HTML-Datei hatten wir definiert, dass dieses Bild, da es innerhalb des `textimg-div`-Bereichs liegt, die für das iPhone typischen Abrundungen erhält. Optisch abgesetzt ist der Fußbereich, der einfach über `div id="footer"` definiert wird und sich insbesondere in der Schriftgröße vom Rest des Inhalts absetzt.

Jetzt folgt der Beispielcode einer iPhone-Seite mit iWebKit-CSS-Selektoren:

```
!--
   Einfaches HTML-Beispiel mit iWebKit-CSS-Selektoren
   Markus Spiering
-->
<!DOCTYPE>
<html>
<head>
 <title>Oldtimer Spiering</title>
 <meta name="viewport" content="width=device-width; initial-scale=1.0;
maximum-scale=1.0; user-scalable=0;" />
 <link rel="stylesheet" type="text/css" href="style.css">
 <style>
 <!--
  #textimg img {
   border: 1px solid #222222;
   -webkit-border-radius: 5px;
   }
 -->
 </style>
</head>
<body>
<div id="topbar" class="transparent" >
 <div id="title">Oldtimer Spiering</div>
 <div id="bluerightbutton"><a href="kontakt.html">Kontakt</a></div>
</div>

<div class="searchbox">
 <form action="http://..." method="get">
  <fieldset>
   <input id="suche" placeholder="Oldtimer-Datenbank durchsuchen"
type="text" />
   <input id="submit" type="hidden" />
  </fieldset>
 </form>
</div>

<div id="content">

 <span class="graytitle">Spitzen-Oldtimer von VW &amp;amp; Porsche</span>
  <ul class="pageitem">
   <li class="textbox">
    <p>Wir f&uuml;hren und reparieren Old- und Youngtimer der Marken VW und
Porsche.</p>
   </li>
  </ul>

 <ul class="pageitem">
  <li class="menu"><a href="angebote.html"><span class="name">Aktuelle
Angebote</span><span class="arrow"></span></a></li>
  <li class="menu"><a href="volkswagen.html"><span class="name">Unsere
VWs</span><span class="arrow"></span></a></li>
```

```
  <li class="menu"><a href="porsche.html"><span class="name">Unsere
Porsche</span><span class="arrow"></span></a></li>
  <li class="menu"><a href="kontakt.html"><span
class="name">Reparaturen</span><span class="arrow"></span></a></li>
  <li class="menu">
   <a href="preisrechner.html">
    <img alt="Rechensymbol" src="thumbs/calculator.png" />
   <span class="name">Preisrechner</span>
   <span class="arrow"></span>
   </a>
  </li>
 </ul>

 <span class="graytitle">Wir importieren aus Kalifornien!</span>

 <ul class="pageitem">
  <li class="textbox">
   <div id="textimg"><p align="center"><img src="images/beispiel_2-0-
car.png" alt="Karmann Ghia Auto"></p></div>
   <p>Neu im Angebot: Volkswagen Karmann Ghia</p>
  </li>
</ul>

</div>

<div id="footer">
   <p><b>Oldtimer Spiering</b><br>Beispielgasse 1, 12345 Bad
Beispielstadt</p>
</div>

</body>
</html>
```

1.4.2 Verwendete Ressourcen speichern

Speichern Sie alle bislang verwendeten Ressourcen und Dateien in einem Verzeichnis namens *iWebKit_Beispiel*. Wir werden auf dieses Beispiel am Ende des Buchs zurückkommen, wenn wir aus dieser einfachen mobilen Webseite eine native iPhone-Anwendung bauen. Der Inhalt des Verzeichnisses sollte bislang wie folgt aussehen:

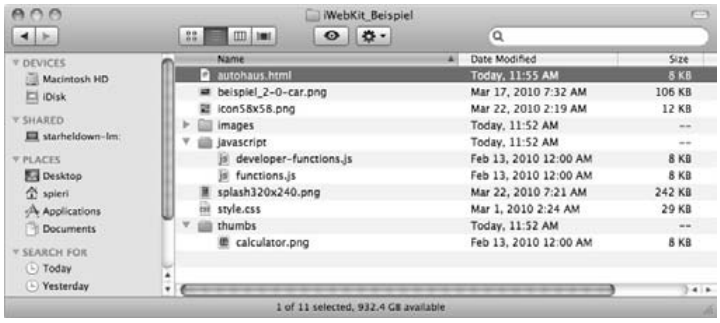


Bild 1.13: Verwendete Ressourcen speichern.

Es gibt eine Reihe weiterer `div`-Identifikatoren, die das Aussehen einer Webanwendung beeinflussen, auch wenn sie bislang noch nicht genannt wurden. Einige davon werden Sie im weiteren Verlauf dieses Buchs kennenlernen, da wir iWebKit für die meisten Beispiele verwenden. Auf einige weitere Elemente des Frameworks möchten wir allerdings noch in dem nächsten Beispiel eingehen.

Safari Web Inspector, Google Chrome Developer Tools und Firebug

Ohne eines dieser Tools für den Desktop-Browser kommt kaum ein Webentwickler aus. Alle drei sind relativ ähnlich und werden schnell unabdingbar, wenn Sie sie einmal benutzt haben.

Safari Web Inspector

Als wir ganz am Anfang des Buchs über die verschiedenen Webbrowser sprachen, wurde schon erwähnt, dass das *Entwickler*-Menü in Safari nicht nur die Möglichkeit anbietet, den User-Agent eines iPhones zu simulieren, sondern auch noch weitere nützliche Funktionen enthält. Eine besonders hilfreiche Funktion ist der *Web Inspector* – im deutschen Safari zu finden unter *Entwickler/Webinformationen einblenden*.
Um den Web Inspector nutzen zu können, müssen Sie das *Entwickler*-Menü in den Safari-Einstellungen aktiviert haben. Wie das genau funktioniert, können Sie in der Einführung oben im Abschnitt über Safari unter »Welcher Browser?« nachlesen.

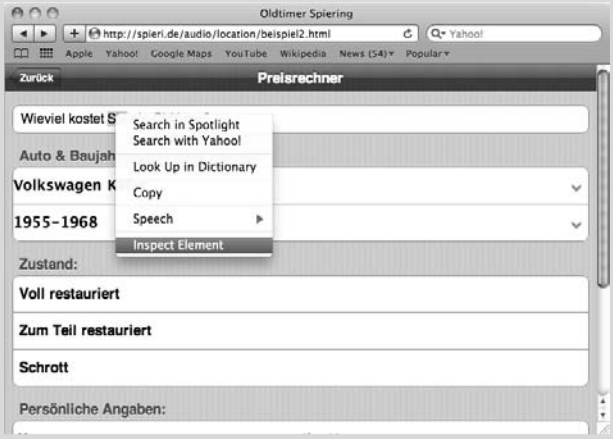


Bild 1.14: Den Web Inspector kann man über *Inspect Element* aus dem Kontextmenü oder über *Show Web Inspector* aus dem Menü *Develop* aufrufen. In der deutschen Safari-Version wählen Sie *Entwickler/Webinformationen einblenden*.

Sobald Sie in Safari das *Develop*-Menü aktiviert haben, rufen Sie einfach eine unserer Beispielseiten auf, klicken mit der rechten Maustaste auf ein Element auf der Seite und wählen *Inspect Element* aus.

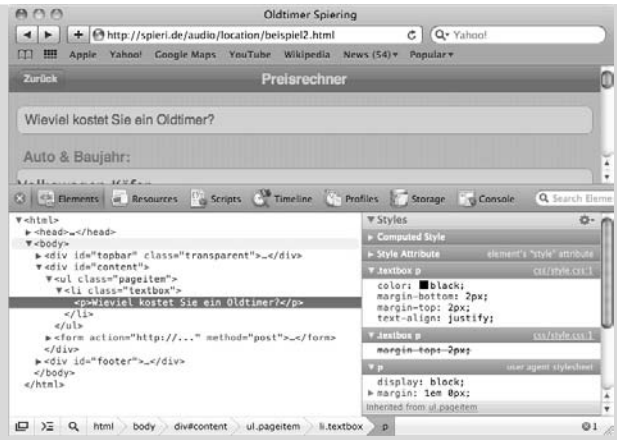


Bild 1.15: HTML-Tags, Klassen und CSS-Definitionen auf einem Blick.

Daraufhin öffnet sich der Web Inspector und zeigt Ihnen nicht nur den HTML-Code mit all seinen definierten Klassen an, sondern bietet Ihnen auch eine Ansicht der benutzten CSS-Definitionen für das jeweilige Element an. Sie können sich mit den kleinen Pfeilen am linken Rand durch den ganzen Seitencode klicken und somit Element für Element der Seite analysieren. Insbesondere bei der Fehlersuche ist der Web Inspector unwahrscheinlich hilfreich.

Am unteren Rand sehen Sie, in welcher Ebene sich das aktuell ausgewählte Element befindet. In unserem Beispiel befindet sich das Element

```
<p>Wie viel kostet Sie ein Oldtimer</p>
```

im HTML-Dokument innerhalb des `body`-Bereichs unter dem `div`-Container `content`, dem `ul`-Tag mit der Klasse `pageitem` und dem `li`-Tag mit der Klasse `textbox`.

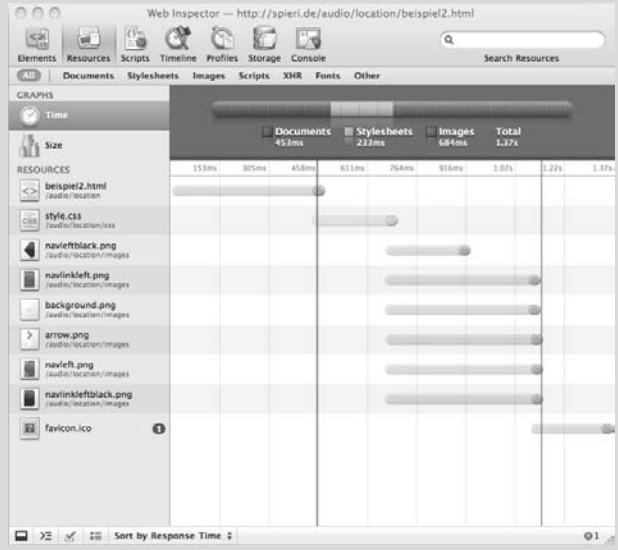


Bild 1.16: Die *Resources*-Ansicht erlaubt eine genaue Analyse der Ladezeiten für eine Seite.

Insbesondere wenn Sie umfangreichere Webseiten verfassen, lohnt sich auch die *Resources*-Ansicht innerhalb des Web Inspectors, um die Ladezeiten der einzelnen Elemente zu analysieren und festzustellen, ob wirklich alle Dateien geladen werden konnten.

Google Chrome
Den Safari Web Inspector finden Sie in Chrome ebenfalls, nur umbenannt in *Developer Tools*. Ansonsten sind die Funktionen und selbst das Aussehen des nützlichen Werkzeugs gleich.

Firefox Firebug
Firefox ist leider (noch) nicht von Haus aus ausgestattet mit einer derartigen Funktion, selbst wenn Firefox der erste Browser war, der mit einem derartigen Tool benutzt werden konnte. Das kommt daher, dass das Firefox-Plug-in *Firebug* zu den beliebtesten Webentwicklungs-Tools gehört. Um Firebug zu benutzen, müssen Sie das Plug-in von *www.getfirebug.com* in Firefox herunterladen und installieren. Die Funktionsweise ist nahezu identisch mit dem Safari Web Inspector.

1.4.3 Form-Elemente mit iWebKit 5

Im folgenden Abschnitt erfahren Sie, wie Sie Form-Elemente mit dem iWebKit-Framework erstellen können. Dazu ein einfaches Beispiel:

```
<!--
   Einfaches HTML-Beispiel mit iWebKit-CSS-Selektoren
   Markus Spiering
-->
<!DOCTYPE>
<html>
<head>
 <title>Oldtimer Spiering</title>
 <meta name="viewport" content="width=device-width; initial-scale=1.0;
maximum-scale=1.0; user-scalable=0;" />
 <meta name="apple-mobile-web-app-status-bar-style" content="black"/>
 <link rel="stylesheet" type="text/css" href="style.css">
</head>
<body>

<div id="topbar" class="transparent" >
 <div id="leftnav"><a href="iwebkit-autohaus.html">Zur&uuml;ck</a></div>
 <div id="title">Preisrechner</div>
</div>

<div id="content">
  <ul class="pageitem">
   <li class="textbox">
    <p>Wie viel kostet Sie ein Oldtimer?</p>
   </li>
  </ul>

 <form action="http://..." method="post">
 <fieldset>
   <span class="graytitle">Auto &amp; Baujahr:</span>
```

```
   <ul class="pageitem">
    <li class="select">
      <select name="markemodell">
         <option value="1">Volkswagen K&auml;fer</option>
         <option value="2">Volkswagen Bus</option>
            <option value="3">Volkswagen Karmann Ghia</option>
      </select>
      <span class="arrow"></span>
   </li>
    <li class="select">
      <select name="baujahr">
         <option value="1955">1955-1968</option>
         <option value="1969">1969-1978</option>
            <option value="1979">1979-1985</option>
            <option value="1980">1986 und &auml;lter</option>
      </select>
      <span class="arrow"></span>
   </li>
   </ul>
   <span class="graytitle">Zustand:</span>
   <ul class="pageitem">
      <li class="radiobutton"><span class="name">Voll
restauriert</span><input name="zustand" type="radio" value="super" /></li>
      <li class="radiobutton"><span class="name">Zum Teil
restauriert</span><input name="zustand" type="radio" value="halb" /></li>
      <li class="radiobutton"><span class="name">Schrott</span><input
name="zustand" type="radio" value="schrott" /></li>
   </ul>
   <span class="graytitle">Pers&ouml;nliche Angaben:</span>
   <ul class="pageitem">
      <li class="smallfield"><span class="name">Name</span><input
placeholder="Ihr Name" type="text" /></li>
        <li class="smallfield"><span class="name">Telefon</span><input
placeholder="(030) 12121212" type="tel" /></li>
   </ul>

   <ul class="pageitem">
    <li class="button">
     <input name="Submit input" type="submit" value="Berechnen" />
    </li>
   </ul>
  </fieldset>
 </form>
</div>

<div id="footer">
   <p><b>Oldtimer Spiering</b><br>Beispielgasse 1, 12345 Bad
Beispielstadt</p>
</div>

</body>
</html>
```

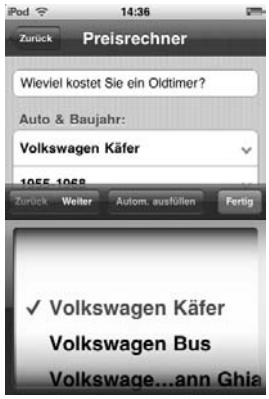


Bild 1.17: Beispielseite mit Titelzeilennavigation und geöffneter Auswahlliste.

Unser Beispiel zeigt eine Unterseite der Oldtimer-Seite und verwendet somit in der Titelleiste einen Linkspfeil, um wieder auf die Hauptseite zurücknavigieren zu können. Pfeil und Link werden einfach über die Zeile

```
<div id="leftnav"><a href="index.html">Zurück</a></div>
```

definiert. Sie sehen in dem Beispiel, dass wir die Umlaute mit einem HTML-Code umschrieben haben. Umlaute, andere Sonderzeichen oder auch HTML-eigene Zeichen wie <" or "> sollten immer mit einem HTML-Code umschrieben werden. Eine komplette Tabelle der HTML-Sonderzeichen und -Codes finden Sie hier: *http://www.asciizeichen.de/HTML.html.*

Die ersten Auswahloptionen auf der Beispielseite sind sogenannte Auswahlboxen. Im PC-Browser werden sie meistens als Drop-down- bzw. Pop-up-Menü angezeigt, auf dem iPhone erscheinen sie als scrollbare Auswahlliste am unteren Bildschirmrand, und bei Android-Telefonen werden diese Auswahlboxen als Menü über die Webseite angezeigt. Ein Auswahlmenü besteht aus einem `li class="select"`-Eintrag mit mehreren definierten Optionen, beispielsweise:

```
<option value="2">Volkswagen Bus</option>
```

Der ausgewählte Wert wird nach dem Abschicken des Formulars im Feld `value` übertragen.

Bild 1.18: Das Formular mit Radiobuttons.

Die vom PC-Browser gewohnten Radiobuttons werden bei iWebKit und in vielen iPhone-Anwendungen nicht als normale Radiobuttons angezeigt, sondern als einzeilige Einträge mit einer Markierung am rechten Bildschirmrand. Radiobuttons werden verwendet, wenn der Benutzer aus einer Reihe von Optionen nur eine Option auswählen kann. Eine Liste dieser Schaltflächen wird mit `li class="radiobutton"` für jeden Eintrag definiert. Im `li`-Tag wird über die `span`-Zuweisung mit der Klasse `name` der Titel definiert und mit dem `input`-Tag die eigentliche Funktionsweise des Radiobuttons bestimmt.

Bild 1.19: Formular mit Texteingabefeldern des Typs `smallfield`.

Am unteren Ende des Beispiels befinden sich zwei weitere Texteingabefelder. iWebKit bietet drei verschiedene Arten von Texteingabefeldern. Der im Beispiel gezeigte Typ (`smallfield`) erlaubt neben dem eigentlichen Texteingabefeld noch die Angabe einer Bezeichnung, beispielsweise *Name:* oder *Telefon:*. Die Definition des Placeholders (`Ihr Name`) ist optional, kann aber den Benutzer auf das gewünschte Eingabeformat hinweisen. Sobald der Benutzer das Texteingabefeld auswählt, verschwindet der `placeholder`-Text.

Eine andere Variante eines Texteingabefelds ist `bigfield`. Diese Art wird nicht im Beispiel gezeigt, allerdings haben Sie sicherlich diese Art von Texteingabefeldern bereits auf einem iPhone oder einem anderen High-End-Telefon gesehen. Das Texteingabefeld erstreckt sich über nahezu die gesamte Bildschirmbreite und hat keine nebenstehende Bezeichnung. Als Kennzeichnung und als Hinweis darauf, was der Benutzer eingeben muss, wird auch hier die `placeholder`-Definition innerhalb des `input`-Tags verwendet. Beispiele von `bigfield`-Eingabefeldern finden Sie sehr oft auf Anmeldebildschirmen, auf denen Sie Ihren Benutzernamen und Ihr Passwort eingeben müssen.

Um mehr als eine Zeile oder einige Worte Text abzufragen, müssen Sie ein normales Texteingabefeld verwenden. Dies wurde zwar nicht im letzten Beispielcode gezeigt, kann aber sehr einfach mit den folgenden Zeilen definiert werden:

```
<ul class="pageitem">
   <li class="textbox">
     <textarea name="TextArea" rows="3"></textarea>
    </li>
</ul>
```

Die Angabe von `rows` bezeichnet die maximale Anzahl der dargestellten und erlaubten Zeilen. Als letztes Form-Element im Beispiel wird mit `li class="button"` noch eine Schaltfläche zum Absenden der Daten definiert.

Sie werden vielleicht bemerkt haben, dass ein wichtiges Form-Element fehlt: die An-/Aus-Schieberegler, die das iPhone aus dem `checkbox`-Element generiert. Natürlich unterstützen iWebKit und andere CSS-Frameworks auch dieses Element, allerdings werden diese Schieberegler nicht aus CSS-Rechtecken gezeichnet, sondern es werden einfach Bilder der Schaltflächen geladen. Diese Bilder reagieren nicht auf die Sprachauswahl der Seite und werden – egal ob Sie Ihre Webseite in Deutsch, Englisch oder einer anderen Sprache verfassen – immer als englisches Bild geladen und dargestellt.

Wenn das kein Problem ist, binden Sie für eine sogenannte Checkbox folgenden Code in Ihr `<form>`-Element und untergeordnete `ul class="pageitem"`-Elemente ein:

```
<li class="checkbox">
   <span class="name">Neuwertig</span>
   <input name="zustand" type="checkbox"/>
</li>
```

Falls Sie Schaltflächen in Deutsch benötigen, kommen Sie nicht darum herum, sie selbst mit einer Bildbearbeitungssoftware umzugestalten und die folgende Datei im iWebKit-Framework auszutauschen: *Framework/images/checkbox.png*. Achten Sie darauf, nicht die Position oder die Größe der Schaltflächen zu ändern.

1.4.4 jQuery und jQTouch

Wir möchten Ihnen jetzt die wirklich fantastischen Möglichkeiten der JavaScript-Frameworks jQuery und jQTouch vorstellen. Mithilfe dieser Frameworks lassen sich ebenfalls iPhone-Seiten erstellen, die nicht nur das vom iPhone bekannte Aussehen haben, sondern auch von iPhone-Anwendungen bekannte Animationen mit nur wenigen Zeilen Code ermöglichen. Selbst wenn wir jetzt detailliert auf jQTouch eingehen, bevorzugen wir bei unseren Beispielen im Buch, möglichst konsistent die eben vorgestellte CSS-Bibliothek iWebKit 5 zu verwenden.

Der Grund dafür ist einfach: Um Ihnen die Möglichkeiten mit HTML5 näher zu bringen, brauchen wir keine Effekte in der Benutzeroberfläche, allerdings wollen wir Ihnen eine gut aufgeräumte und klare UI in den Beispielen bieten. Daher fällt unsere Wahl im Buch auf iWebKit 5. Falls Ihnen die Möglichkeiten mit jQTouch attraktiver erscheinen, werden Sie sehen, dass sich alle Beispiele im Buch ganz einfach auf jQTouch umschreiben lassen.

Das Framework jQuery ist mittlerweile die weltweit am meisten verwendete JavaScript-Bibliothek. Sie erlaubt es den Programmierern, mit wenig Aufwand umfangreiche JavaScript-Funktionen beispielsweise zur Navigation in die eigenen Projekte zu integrieren. Während jQuery keine mobilspezifischen Funktionen zur Verfügung stellt, ist jQTouch komplett auf iPhone und iPod touch ausgerichtet. Das jQTouch-Framework ist ein Plug-in für jQuery und benötigt diese Bibliothek.

Ein paar Worte zur Kompatibilität: jQTouch ist großartig, wenn Sie eine Webanwendung für das iPhone, das iPad oder den iPod touch erstellen möchten, aber wir sind oft in Probleme mit anderen Plattformen und WebKit-Browsern geraten. Während alle Beispiele im Buch mindestens auf dem iPhone oder unter Android laufen sollten, beschränken sich die jQTouch-Beispiele zu diesem Zeitpunkt auf iOS. Manche Animationen (Slide-ups) sind auf einigen Android-Telefonen so langsam, dass sie kaum benutzbar sind. Volle Unterstützung für Android dürfte aber in Kürze zu erwarten sein.

Die Frameworks in den Code einbinden

Um jQTouch zu benutzen, muss auch jQuery in den Code mit eingebunden werden. Sie können jQuery zum einen von *http://jquery.com* herunterladen und die Datei *jquery-1.4.2.min.js* (bzw. eine neuere Version) auf Ihrem Server speichern oder über die folgenden Zeilen im HTML-Header dynamisch von den Google-Servern laden:

```
<script type="text/javascript" src="http://www.google.com/jsapi"></script>
<script type="text/javascript"> google.load("jquery", "1.4.2"); </script>
```

Die verschiedenen jQTouch-Dateien und -Verzeichnisse sollten Sie auf Ihrem Webserver speichern. Um die aktuelle jQTouch-ZIP-Datei herunterzuladen, gehen Sie einfach auf *http://www.jqtouch.com* und klicken dort auf *Download*.

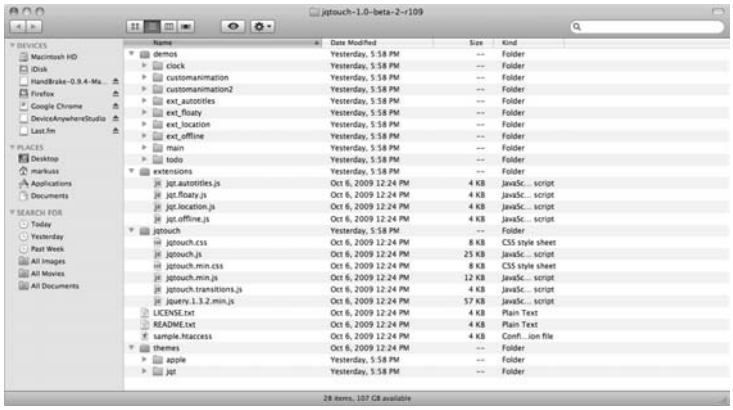


Bild 1.20: Inhalt und Struktur des jQTouch-ZIP-Archivs.

Entpacken Sie die ZIP-Datei und kopieren Sie die Verzeichnisse *jqtouch*, *themes* und *extensions* auf Ihren Webserver. Fügen Sie dann im Header des HTML-Dokuments unterhalb der jQuery-Zeilen folgenden Code ein:

```
<script type="text/javascript" src="jqtouch/jqtouch.js"></script>
<link type="text/css" rel="stylesheet" media="screen"
href="jqtouch/jqtouch.css">
<link type="text/css" rel="stylesheet" media="screen"
href="themes/jqt/theme.css">
```

In der ersten Zeile wird die jQuery-JavaScript-Bibliothek eingebunden, in den beiden folgenden Zeilen folgen die von jQuery verwendeten CSS-Definitionen, die zur Oberflächengestaltung benutzt werden. Wie Sie vielleicht schon bemerkt haben, bietet

jQTouch zwei verschiedene Themes, also zwei unterschiedliche Motive. Diese Themes bestimmen das Aussehen der gesamten Anwendung:

Bild 1.21: Das *jqt*-Theme gibt den Webanwendungen eine sehr stylische Oberfläche, die an native iPhone-Apps erinnert.

Bild 1.22: Das *apple*-Theme ist konservativer und bietet, wie auch in iWebKit 5, eine Oberfläche, die sehr an die Standard-Apps des iPhones erinnert. Das Einzige, was wir am Code für diese beiden Screenshots verändert haben, ist jeweils der Verweis auf das *jqt*- und das *apple*-Theme.

Bevor wir zu dem eigentlichen Inhalt der Webanwendung kommen, muss jQTouch noch initialisiert werden. Über die folgende Initialisierung werden alle jQTouch-Funktionen im Code bereitgestellt und können verwendet werden. Falls Sie die folgenden Zeilen im `head`-Bereich vergessen, erhalten Sie, egal wie viel HTML-Code Sie schreiben, immer nur einen schwarzen Browserbildschirm.

```
<script type="text/javascript">
   var jQT = $.jQTouch({ });
</script>
```

Seiten innerhalb der Webanwendung anlegen

Im Gegensatz zu einem normalen Webauftritt werden alle Seiten einer jQTouch-Webanwendung innerhalb einer HTML-Datei angelegt. Anstatt also mehrere einzelne HTML-Seiten, wie beispielsweise

- *index.html*
- *angebote.html*
- *kontakt.html*

zu generieren, erstellen wir mit jQTouch nur eine *index.html* mit verschiedenen `div`- oder `form`-Tags jeweils mit einer eigenen ID im Code:

```
<body>
   <div class="current" id="home">
   </div>
   <div id="kontakt">
   </div>
 </body>
```

Mit diesem Beispiel haben wir zwei leere Seiten innerhalb unserer jQTouch-Webanwendung erstellt. Der `div class="current"`-Parameter ist optional und bezeichnet die Seite, die als Startseite für die Webanwendung genutzt werden soll. Wird diese Klassenzuweisung weggelassen, wird die erste ID nach dem `body`-Tag als Startseite benutzt.

Innerhalb der einzelnen `div`-Container können Sie die Seiten mit eigenem HTML und CSS gestalten. jQTouch bietet Ihnen aber auch, ähnlich wie iWebKit 5, vorgefertigte Elemente, die meistens nicht nur besser aussehen, sondern teilweise auch mit einem bestimmten Verhalten versehen sind.

Als erstes Element werden wir auch in diesem Beispiel die Kopfzeile, also die Toolbar, wieder in iPhone-gerechtes Styling bringen. Dazu geben wir folgenden Code in den ersten `div`-Container ein:

```
<div class="toolbar">
  <h1>Oldtimer</h1>
  <a href="#kontakt" class="button flip">Kontakt</a>
</div>
```

Bild 1.23: Stylische Kopfzeile mit Titel und Button.

Der Titel einer Kopfzeile wird innerhalb eines `h1`-Tags beschrieben. Weiterhin haben Sie die Möglichkeit, zwei Schaltflächen innerhalb der Kopfzeile zu definieren. Sie werden einfach innerhalb des `div class="toolbar"`-Containers als Links definiert. Da sich alle Seiten einer jQTouch-Anwendung innerhalb einer Datei befinden, dürfen Sie keine externen Links, sondern nur Links zu den anderen `div`-Containern erstellen, die eine Seite der Webanwendung beinhalten.

Auch wenn wir noch keine Inhalte für die beiden Seiten haben, lassen Sie uns dennoch das Beispiel mit einer Kopfzeile für den zweiten Container vervollständigen. Der Code der Beispielanwendung sollte jetzt so aussehen:

```
<!--
   Einfaches HTML-Beispiel mit jQTouch-Einbindung
   Markus Spiering
-->
<!DOCTYPE>
<html>
 <head>
   <meta name="layout" content="webkit" />
```

```
   <title>Oldtimer Spiering</title>

   <!-- jQuery direkt von Google Code einbinden -->
   <script type="text/javascript"
src="http://www.google.com/jsapi"></script>
   <script type="text/javascript"> google.load("jquery", "1.3.2"); </script>

   <!-- jQTouch einbinden -->
   <script type="text/javascript" src="jqtouch/jqtouch.js"></script>
   <link type="text/css" rel="stylesheet" media="screen"
href="jqtouch/jqtouch.css">
   <link type="text/css" rel="stylesheet" media="screen"
href="themes/jqt/theme.css">

    <!-- jQTouch initialisieren -->
   <script type="text/javascript">
      var jQT = $.jQTouch({ });
   </script>

</head>
 <body>
   <div id="home">
      <div class="toolbar">
       <h1>Oldtimer</h1>
      <a href="#kontakt" class="button flip">Kontakt</a>
      </div>
   </div>
   <div id="kontakt">
      <div class="toolbar">
       <h1>Kontakt</h1>
      <a href="#home" class="button back">Zur&uuml;ck</a>
      </div>
   </div>
 </body>
</html>
```

Wenn Sie dieses Beispiel ausführen, sollten Sie folgendes Verhalten bereits sehen können:

Die Webanwendung besitzt eine Kopfzeile mit einem Button, der im `div`-Container der Klasse `toolbar` durch einen einfachen `a href=""`-Verweis definiert wurde. Um den Button auch als Button erscheinen zu lassen, muss dem `a`-Tag die Klasse `button` zugewiesen werden.

So etwas kannten Sie wahrscheinlich bisher nur von nativen iPhone-Anwendungen. Nach Antippen von *Kontakt* dreht sich der Bildschirm zur Kontaktseite. Die Animation wird durch die Angabe von `flip` innerhalb der Klassendefinition des `a`-Tags erreicht.

Bild 1.24: Kopfzeile mit einem Button.

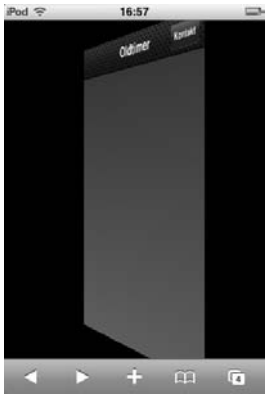


Bild 1.25: Nach dem Antippen von *Kontakt* dreht sich der Bildschirm zur Kontaktseite.

Der Inhalt des Containers `div id="kontakt"` wird jetzt als einzelne Seite dargestellt. Dabei wird für die Zurück-Funktion die gleiche Animation verwendet. Der Stil des Zurück-Buttons kommt durch die Spezifikation von `back` innerhalb der Klassendefinition des `a`-Tags. Im Gegensatz zu nativen iPhone-Anwendungen müssen Sie in jQTouch (wie auch in iWebKit 5) den Zurück-Button mit dem Namen *Zurück* versehen.

Bild 1.26: Der Inhalt des Containers wird als einzelne Seite dargestellt.

Listen und Animationen

Nachdem wir bereits eine Anwendung mit einer Kopfzeile haben, wollen wir ein weiteres iPhone-typisches Element hinzufügen: Linklisten. Im iWebKit 5-Beispiel haben wir auf der Startseite eine Reihe von Links angeboten, und das lässt sich mit jQTouch ähnlich bewerkstelligen. Fügen Sie einfach unterhalb der Kopfzeilendefinition im `home`-Container den folgenden Code hinzu:

```
<ul>
  <li class="forward"><a href="#angebote">Aktuelle Angebote</a></li>
  <li class="arrow"><a href="#repair">Reparaturen</a></li>
  <li class="arrow"><a href="#preisrechner">Preisrechner</a></li>
  <li class="arrow"><a href="#kontakt">Kontakt</a></li>
</ul>
```

Damit diese Links auch funktionieren, müssen im Code `div`-Container für weitere Unterseiten mit den IDs `angebote`, `repair` und `preisrechner` angelegt werden.

Listen werden wie in iWebKit 5 über die Tags `ul` und `li` angelegt. Dabei kann das `li`-Tag folgende Klassen haben:

- `arrow` – der für iPhone-Links typische Linkpfeil wird verwendet.
- `forward` – ein buttonähnlicher Pfeil wird statt des normalen Linkpfeils verwendet.

Bild 1.27: Die Startseite unseres Beispiels sieht nun so aus.

Beim Antippen der Links wird ebenfalls eine leichte Animation verwendet: Der aktuelle Bildschirm rollt zur linken Seite, und die Zielseite wird von der rechten Seite hereingescrollt. Sie können das durch die Angabe der folgenden Klassen im `a`-Tag beeinflussen:

Klasse	*Beschreibung*
‹a href="#..." class="slideup"›	Die neue Seite wird vom unteren Rand in den Bildschirm gerollt.
‹a href="#..." class="dissolve"›	Die alte Seite wird langsamer ausgeblendet.
‹a href="#..." class="fade"›	Die Seiten werden langsamer aus- und eingeblendet.

Klasse	*Beschreibung*
‹a href="#..." class="flip"›	Diese Animation haben wir schon kennengelernt. Der Bildschirm dreht sich um eine vertikale Achse, um die neue Seite anzuzeigen.
‹a href="#..." class="pop"›	Die neue Seite wird aus der Mitte des Bildschirms hereingezoomt.
‹a href="#..." class="swap"›	Eine recht aufwendige Animation. Die alte Seite wird verkleinert und verschwindet gleichzeitig im Hintergrund, während die neue Seite aus dem Hintergrund hereingezoomt wird.
‹a href="#..." class="cube"›	Hier sind die beiden Seiten wie auf Würfelseiten projiziert, und der Würfel dreht sich einmal um die eigene Achse, um die Seiten auszutauschen.

Die einzelnen Animationen lassen sich schwer mit Worten beschreiben. Ausprobieren geht hier definitiv über Studieren. Um wieder auf die vorhergehende Seite zurückzukehren, können wir die Klasse `back` in der Linkdefinition verwenden, allerdings können auch die Klassen `cancel` und `goback` verwendet werden.

Sollten Sie keinerlei Animationen in Ihrer Webanwendung verwenden wollen, können Sie das im Header-Bereich des Codes, in dem jQTouch initialisiert wird, global festlegen. Ändern Sie einfach die Initialisierungszeile wie folgt:

```
var jQT = $.jQTouch({useAnimations: false});
```

Ab sofort können Sie noch so viele Animationen in Ihren Linkklassen definieren – die Seiten werden ohne Schnickschnack einfach geladen und angezeigt.

Wie auch iWebKit 5 kann jQTouch verschiedene Arten von Listen darstellen: Welche Art von Liste angezeigt werden soll, wird durch eine Klassenzuweisung im `ul`-Tag definiert. Fügen Sie beispielsweise einfach die Klasse `rounded` dem `ul`-Tag im Beispielcode hinzu:

```
<ul class="rounded">
```

Wie der Name der Klasse schon erahnen lässt, sind die Kanten der Linkumrandungen nicht mehr eckig, sondern abgerundet:

Das Ganze geht natürlich auch wesentlich radikaler. Falls Sie es noch nicht getan haben, erstellen Sie jetzt für das Beispiel einen `div`-Container mit der ID `angebote`. Der folgende Code erstellt wiederum eine Listenansicht:

```
<div id="angebote">
   <div class="toolbar">
    <h1>Angebote</h1>
   <a href="#home" class="button back">Zur&uuml;
ck</a>
   </div>
   <ul class="edgetoedge">
    <li class="sep">Unter 1000 Euro</li>
     <li><a href="">Lenkrad</a></li>
```

```
    <li><a href="">Bremse</a></li>
   <li class="sep">Unter 2000 Euro</li>
    <li><a href="">Motorhaube</a></li>
    <li><a href="">Achse</a></li>
  </ul>
</div>
```

Die Klasse `edgetoedge` definiert Linkfelder von der rechten Bildschirmseite bis zur linken Seite – es gibt also keine unklickbare Fläche links und rechts wie bei `rounded`. Mit der Klasse `sep` können Überschriften und Separatoren im `li`-Tag definiert werden:

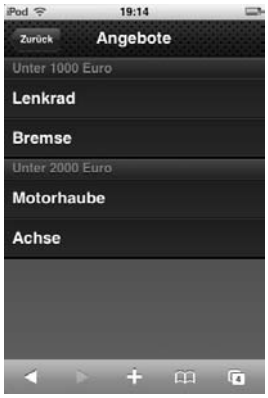


Bild 1.28: Mit `edgetoedge` definierte Linkfelder.

Ein weiterer Stil ist `plastic`, der im Gegensatz zu `edgetoedge` die Klassen `arrow` und `forward` für das `li`-Tag unterstützt:

```
<ul class="plastic">
 <li>Unter 1000 Euro</li>
  <li class="forward"><a href="">Lenkrad</a></li>
  <li class="forward"><a href="">Bremse</a></li>
 <li>Unter 2000 Euro</li>
  <li class="arrow"><a href="">Motorhaube</a></li>
  <li class="arrow"><a href="">Achse</a></li>
</ul>
```

Bei `plastic` erstrecken sich die Linkfelder ebenfalls von Bildschirmrand zu Bildschirmrand, und die Hintergrundfarbe alterniert pro Link.

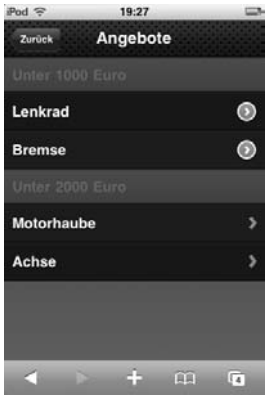
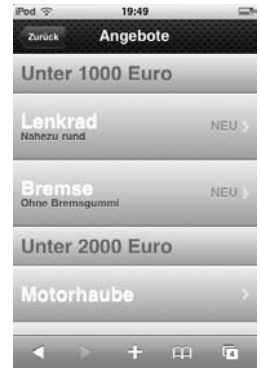


Bild 1.29: Mit `plastic` definierte Linkfelder.

Als letzten Listenstil möchten wir Ihnen das auffällige `metal` vorstellen.

```
<ul class="metal">
 <li>Unter 1000 Euro</li>
  <li class="arrow"><a href=""><small>NEU</small>Lenkrad<em>Nahezu
rund</em></a></li>
  <li class="arrow"><a href=""><small>NEU</small>Bremse<em>Ohne
Bremsgummi</em></a></li>
 <li>Unter 2000 Euro</li>
  <li class="arrow"><a href="">Motorhaube</a></li>
  <li class="arrow"><a href="">Achse</a></li>
</ul>
```

Der Stil unterstützt nicht die `forward`-Klassifizierung für das `li`-Tag, beherrscht aber zusätzliche Beschreibungen, die mit `small` und `em`, wie im Beispiel gezeigt, in das Linkfeld mit eingearbeitet werden können:



Auch in den anderen Listenformen kann das Tag `small` definiert werden. Hier ein Beispiel mit der Standardlistenansicht:

```
<li class="arrow">
<a href="#preisrechner">Preisrechner<small>
NEU</small></a>
</li>
```


```
<li class="arrow">
 <a href="#emails">Ungelesene E-Mails</a>
 <small class=counter>112</small>
</li>
```

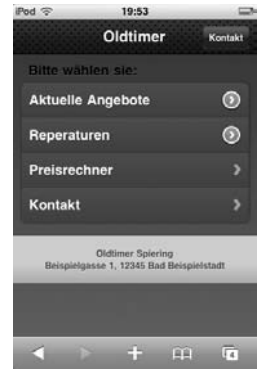
Lassen Sie uns jetzt noch die Startseite der Anwendung etwas aufhübschen. Fügen Sie unter dem `toolbar`-Container die folgende Zeile hinzu:


```
<h2>Bitte wählen Sie:</h2>
```

Des Weiteren setzen Sie unterhalb der auf der Startseite definierten Liste noch folgenden `div`-Container ein:

```
<div class="info">
   <p><b>Oldtimer Spiering</b><br>Beispielgasse 1, 12345 Bad
Beispielstadt</p>
</div>
```

Und schon sieht unsere Startseite wie eine richtige iPhone-Anwendung aus:



Letztendlich ist der Gestaltungsprozess ähnlich wie der unter iWebKit 5: Sie schreiben größtenteils HTML-Code und weisen den HTML-Elementen vordefinierte CSS-Klassen zu, die nicht nur das Aussehen verändern, sondern teilweise auch Animationen hinzufügen.

1.4.5 Formulare mit jQTouch

Als Nächstes werden wir uns die Formulardefinitionen mit jQTouch ansehen. In unserem Beispiel werden wir die *Kontakt*-Seite als Formular umschreiben. Dazu entfernen Sie einfach den `div`-Container mit der ID `kontakt` und kopieren folgenden Code in das Beispiel:

```
<form action=""method="" id="kontakt">
   <div class="toolbar">
    <h1>Kontakt</h1>
   <a href="#home" class="button back">Zur&uuml;ck</a>
   </div>
   <ul class="rounded">
      <li><input name="nachname" value="" type="text" placeholder="Ihr
Name"></li>
      <li><input name="telefon" value="" type="text" placeholder="Ihre
Telefonnummer"></li>
   </ul>
   <ul class="rounded">
      <li>Newsletter<span class="toggle">
         <input name="newsletter" type="checkbox">
       </span>
      </li>
   </ul>
   <a class="whiteButton submit">Absenden</a>
   <a class="grayButton goback">Abbrechen</a>
</form>
```

Zuerst wird ein Formular mit dem `form`-Tag initiiert und erst einmal kein Zielort angegeben, an den die Daten geschickt werden sollen. Wichtig im `form`-Tag ist die Angabe der ID, da der `div`-Container mit der `id=kontakt` gelöscht wurde. An seine Stelle tritt nun der `form`-Bereich. Sie erkennen also, dass eine einzelne jQTouch-Seite nicht nur aus einem `div`-Container bestehen kann, sondern auch aus einem Formular.

Anschließend haben wir wieder eine Kopfleiste mit einer *Zurück*-Option eingefügt und beginnen dann mit den eigentlichen Formularfeldern. Das erste Eingabefeld würde einen Wert in der Variablen `nachname` abspeichern. Vor der Texteingabe wird direkt auf dem Textfeld *Ihr Name* angezeigt, wodurch der Benutzer sehen kann, was er dort eingeben soll. Das Gleiche machen wir für das Feld mit der Telefonnummer.

Da wir den An- und Ausschalter optisch von der Eingabe des Namens und der Telefonnummer getrennt haben möchten, wurde für dieses Element ein eigenes `ul`-Tag aufgemacht und mit der Klasse `rounded` versehen. Der Schieberegler wird durch `span class="toggle"` und dem nachfolgenden `input`-Tag definiert.

Bild 1.30: Das erste Formular mit jQTouch.

Die beiden Buttons im Formular werden ganz einfach über die beiden letzten `a`-Tags im Formular definiert. Dabei soll der Button zum Absenden des Formulars in weißer Farbe erscheinen, was durch die Klasse `whiteButton` erreicht wird. Dass dies auch der *Submit*-Button ist, wird durch die Klasse `submit` definiert. Alternativ dazu der *Abbrechen*-Button: Dieser soll in dunkelgrauer Farbe angezeigt werden (Klasse `grayButton`) und soll, ohne die Daten abzusenden, einfach wieder zurück auf die letzte Seite springen. Auch das kann einfach über die Angabe der Klasse `goback` erreicht werden.

Texteingabefelder für Passwörter werden ebenfalls mit dem `input`-Tag erstellt, allerdings wird hier der Typ mit `type=password` auf ein Passwortfeld gesetzt:

```
<h2>Passworteingabe:</h2>
<ul class="rounded">
  <li><input name="passwort" value="" type="password"
placeholder="Passwort"></li>
</ul>
```

Auswahllisten, die auf dem iPhone mit dem bekannten Rollauswahlfenster dargestellt werden, sind über den folgenden Code möglich:

```
<h2>Marke wählen:</h2>
<ul class="rounded">
   <li>
      <select id="auto">
         <optgroup label="Volkswagen">
```

```
            <option value="bug">Käfer</option>
            <option value="karmann">Karmann Ghia</option>
            <option value="bus">Hippie-Bus</option>
         </optgroup>
         <optgroup label="Porsche">
            <option value="911">911</option>
            <option value="365">365</option>
         </optgroup>
      </select>
   </li>
</ul>
```

Beachten Sie, wie man über das Tag `optgroup` unanklickbare Kategorienfelder innerhalb der Auswahlliste definieren kann.

Bild 1.31: Passwortfeld und Auswahlliste in einem jQTouch-Formular.

Was jetzt noch fehlt, sind Checkboxen und Radiobuttons im Formular. Diese sind mit den jeweiligen HTML-Zeilen schnell im Formular eingefügt:

```
<!-- Checkboxen -->
<h2>Checkboxen zur Multi-Auswahl:</h2>
<ul class="rounded">
   <li><input type="checkbox" name="wert1" checked="checked" title="Mich
antippen"></li>
   <li><input type="checkbox" name="wert2" title="Mich aber auch!"></li>
</ul>

<!-- Radiobuttons -->
<h2>Entweder-oder:</h2>
<ul class="edgetoedge">
   <li><input type="radio" name="wert3" checked="checked" value="1"
title="Nimm mich!"></li>
   <li><input type="radio" name="wert3" value="2" title="Nein mich!"></li>3
</ul>
```

Jeweils die erste Option wurde über `checked="checked"` vorausgewählt.

Bild 1.32: Checkboxen und Radiobuttons funktionieren auch im Querformat.

1.4.6 jQTouch-Fehler für Radiobuttons

Auch die besten Frameworks und Bibliotheken sind nicht frei von Fehlern. So auch jQTouch nicht, zumindest zu dem Zeitpunkt, da wir dieses Buch im Sommer 2010 geschrieben haben. Wenn Sie den oben stehenden Code für die Radiobuttons verwendet haben, haben Sie alles richtig gemacht, und trotzdem werden die Radiobuttons nicht wie gewünscht auf einem Gerät funktionieren. Sie arbeiten im Desktop-Webbrowser, aber nicht auf dem iPhone oder einem iPod touch.

Glücklicherweise können Sie den jQTouch-Fehler im Handumdrehen selbst beheben. Rufen Sie die jQTouch-JavaScript-Datei *jqtouch.js* aus dem Verzeichnis *jqtouch* in Ihrem Texteditor auf. Dort wird in Zeile 159 die JavaScript-Funktion `body.click` definiert:

```
$body.click(function(e){
    var $el = $(e.target);
    if ($el.attr('target') == '_blank' || $el.attr('rel') == 'external' ||
$el.is('input[type="checkbox"]'))
    {
        return true;
    } else {
        return false;
    }
});
```

Tauschen Sie die Zeile mit der `if`-Bedingung durch die folgenden Zeilen aus:

```
if ($el.attr('target') == '_blank' || $el.attr('rel') == 'external' ||
$el.is('input[type="checkbox"]') || $el.is('input[type="radio"]'))
```

Speichern Sie anschließend die Datei wieder auf dem Server ab, und das Problem ist behoben.

1.4.7 Dynamische Inhalte

Nun würden wir eine ziemlich langweilige Anwendung programmieren, die nur auf statische Daten innerhalb der einen HTML-Datei zugreifen könnte. jQTouch kann durch

die Verwendung der AJAX-Technologie externe Dateien und Daten aus dem Internet holen, ohne die aktuelle Seite neu laden oder eine andere Seite laden zu müssen.

Um das zu verdeutlichen, kopieren Sie einfach den Code, den wir für das Kontaktformular verwendet haben, in eine neue Datei und speichern ihn unter dem Dateinamen *form.html* auf dem Server ab:

```
<form action="" method="" id="kontakt">
   <div class="toolbar">
    <h1>Kontakt</h1>
   <a href="#home" class="button back">Zurück</a>
   </div>
   ...
</form>
```

Für diese HTML-Datei brauchen Sie keinen `<body>`- oder `<head>`-Bereich zu definieren, sondern Sie verwenden wirklich ganz genau den Code der Haupt-HTML-Seite unseres Beispiels für die Formulare.

Jetzt müssen Sie die Links zu dem Kontaktformular ändern. Das Kontaktformular war bei uns auf der Hauptseite in der Kopfzeile im Button verlinkt. Ändern Sie diese Zeile wie folgt:

```
<a href="form.html" class="button flip">Kontakt</a>
```

Das Gleiche gilt für die Verlinkung in der Linkliste der Hauptseite:

```
<li class="arrow"><a href="form.html">Kontakt</a></li>
```

Nachdem Sie die Änderungen vorgenommen, auf dem Server gespeichert und die Webanwendung auf dem iPhone oder iPod touch neu geladen haben, sehen Sie folgendes Verhalten: Nach dem Antippen von *Kontakt* lädt das Gerät die Informationen, die in der Datei *form.html* enthalten sind.

Normalerweise würde der Browser bei einem derartigen Link eine neue Webseite aufrufen. Mit jQTouch bleibt aber der alte Bildschirm so lange bestehen, bis die Informationen aus *form.html* geladen sind. Sobald das geschehen ist, wird mit der eingestellten Animation zum neuen Bildschirm gewechselt.

jQTouch hat es Ihnen damit ermöglicht – auch ohne eine einzige Zeile JavaScript zu programmieren –, AJAX-Verhalten in Ihre Webanwendung zu integrieren. Wenn Sie wieder zurück auf den Hauptbildschirm gehen und nochmals auf *Kontakt* tippen, werden Sie bemerken, dass jQTouch die Daten im Cache gespeichert hat und sie daher nicht nochmals geladen werden müssen.

1.5 HTML und Telefonfunktionen

Von HTML aus können Sie einfach und effektiv einige Telefonfunktionen ansprechen. Manche, wie beispielsweise die Implementierung einer Landkarte, lassen sich eleganter mit JavaScript und der Google Maps-API realisieren, allerdings ist das auch wesentlich aufwendiger. Viele Telefonfunktionen wie beispielsweise die Kamera, das Adressbuch

oder auch der Kalender, lassen sich noch nicht vom Browser aus ansprechen. Zum Zeitpunkt der Entstehung dieses Buchs hat Google zwar angekündigt, den Kamerazugriff vom Browser aus in einer zukünftigen Version von Android zu ermöglichen, allerdings ohne eine Versionsnummer oder ein Datum zu nennen. Was der Browser noch nicht kann, lässt sich jedoch mit nativen Anwendungen realisieren. Dazu aber mehr am Ende des Buchs, jetzt schauen wir erst mal, was wir alles im Webbrowser machen können.

1.5.1 Telefonnummer benutzen

Sie können von einer Webseite aus dem Benutzer direkt anbieten, eine Telefonnummer zu wählen. Das geschieht einfach über das `tel`-Protokoll und einen ganz normalen HTML-Link:

```
<a href="tel:0351123456789">0351-123456789</a>
```

Nach dem Antippen eines solchen Links passiert Folgendes:

Bevor der Anruf ausgeführt wird, sieht der Benutzer einen kleinen Dialog und kann einem Anruf zustimmen oder ihn ablehnen. Auf einem iPod touch oder einem iPad kann der Benutzer zwar keinen Anruf tätigen, dafür aber die Telefonnummer in seinen Kontakten speichern.

Bild 1.33: Bevor der Anruf ausgeführt wird ...

Auf einem Android-Telefon wird der Telefonbildschirm eingeblendet. Entscheidet sich der Benutzer, den Anruf doch nicht durchführen zu wollen, kann er über die *Zurück*-Taste eines Android-Telefons zurück in die Webanwendung kommen.

Das iPhone versteht übrigens nicht nur Nummern: Teile des RFC 2086-Protokolls sind im mobilen Safari implementiert. Somit können beispielsweise über den Buchstaben »p« einsekündige Pausen in die Telefonnummer mit eingebaut werden. Weiterhin ist die Angabe von Buchstaben zugelassen. Unsere Tests haben allerdings gezeigt, dass diese Funktionalität – die ohnehin nicht häufig genutzt wird – nur auf dem iPhone, nicht aber auf Android-Geräten funktioniert.

1.5.2 SMS-Anwendung aufrufen

Falls Sie von Ihrer Webseite aus den Versand einer Textnachricht veranlassen möchten, können Sie den folgenden HTML-Link verwenden:

```
<a href="sms:0351123456789">Schick mir eine SMS</a>
```

Auf Android-Geräten wird der Benutzer gefragt, ob er eine Textnachricht verfassen will, auf den Apple-Geräten zeigt sich folgendes Verhalten: Auf einem iPhone wird sofort von der Webseite in die SMS-Anwendung gewechselt und die Telefonnummer als Empfängernummer eingestellt. Die Meldung im rechten Bild auf einem iPod touch bzw. einem iPad, die natürlich keine SMS versenden können, ist leider suboptimal.

Bild 1.34: Links auf dem iPhone, rechts auf iPod touch oder iPad.

Noch zwei Tipps: Bei der Angabe einer SMS-Nummer dürfen Sie im Gegensatz zum `tel`-Protokoll nur Ziffern, Bindestriche und Punkte verwenden. Eine eigentliche Textnachricht ist leider nicht möglich.

Falls Sie unterdrücken möchten, dass Safari die Ziffernfolgen, die nach Telefonnummern aussehen, automatisch als solche klassifiziert und klickbar macht, können Sie das mit folgender Zeile im `head`-Bereich Ihrer Webseite unterdrücken:

```
<meta name="format-detection" content="telephone=no">
```

Das `meta`-Tag ist Apple-spezifisch und funktioniert bislang nicht auf anderen Browsern.

1.5.3 E-Mails versenden

Insbesondere in den frühen Tagen des Internets fand man E-Mail-Links häufig auf normalen Webseiten. Die Links mit dem `mailto:`-Protokoll haben dann Outlook Express oder das jeweilige Standard-E-Mail-Programm auf dem Rechner geändert. Mittlerweile, da ein großer Teil der E-Mail-Kommunikation über Webseiten abläuft, hat das `mailto:`-Protokoll auf Desktop-Webseiten seinen Glanz verloren, ist jedoch bei mobilen Webseiten durchaus sinnvoll, da viele Benutzer beispielsweise auf dem iPhone die eingebaute Mailanwendung verwenden, die mit dem Protokoll tadellos funktioniert.

Um die E-Mail-Anwendung von einem Telefon aus aufzurufen, müssen Sie nur folgenden Link angeben:

```
<a href="mailto:spieri@ymail.com">E-Mail an mich</a>
```

Sobald der Benutzer auf den Link tippt, schiebt sich auf einem iPhone oder einem iPod touch die Mailanwendung von unten in den Bildschirm.

Bild 1.35: Aufgerufener E-Mail-Bildschirm.

Anstatt nur eine E-Mail-Adresse anzugeben, an die die fertige E-Mail verschickt werden soll, lassen sich innerhalb des `mailto:`-Protokolls noch weitere optionale Angaben machen:

Optionale Angaben im `mailto:`*-Protokoll*	
`subject=Text`	Mit dem `subject`-Attribut können Sie bereits im Link eine Betreffzeile festlegen.
`body=Text`	Über das `body`-Attribut kann der Inhalt einer E-Mail definiert werden. Dabei ist zu beachten, dass nur das iPhone derzeit HTML-Tags innerhalb des `body`-Attributs versteht und korrekt umsetzen kann. Um auch unter Android zu funktionieren, empfiehlt es sich, für Zeilenumbrüche den Code `%0A%0A` zu verwenden. Das iPhone hatte früher Probleme damit, kann aber mittlerweile auch diesen Code umsetzen.
`cc=name@email.de`	Legt fest, wer die E-Mail als Kopie (CC) empfangen soll.
`bcc=name@email.de`	Legt fest, wer die E-Mail als Blindkopie (BCC) erhalten soll.
`name1@email.de, name2@email.de`	Mehrere Empfängeradressen werden durch Kommata getrennt angegeben.

Das folgende Beispiel enthält alle in der Tabelle vorgestellten Parameter. Der erste Parameter wird, wie in HTML üblich, mit einem Fragezeichen eingeleitet, und zusätzliche Parameter werden mit dem `&`-Zeichen angebunden:

```
 <a href="mailto:spieri@ymail.com?subject=Anfrage&body=Hallo
Markus,%0A%0Ahier ist eine
<b>Anfrage</b>:&cc=anfragen@spieri.de&bcc=geheim@spieri.de">E-Mail an
mich</a>
```

Bild 1.36: Die Ansicht im E-Mail-Programm auf dem iPod touch.

1.5.4 Google Maps aufrufen

Weiter unten im Buch, wenn die Geolocation-API für HTML5 untersucht wird, gehen wir detaillierter auf Google Maps und die Google Maps-API ein. Zunächst aber kommen wir zu einer sehr einfachen Möglichkeit, die Dienste des Google Maps-Programms zu nutzen. Auch hier müssen Sie kein JavaScript benutzen, sondern nur einen speziellen Link setzen. Den Rest machen die Telefone von allein.

Als Beispiel verwenden wir wieder einen ganz einfachen Link. Es ist genau der gleiche Link, den Sie auch dazu benutzen können, Google Maps von jedem Webbrowser der Welt aus aufzurufen:

```
<a href="http://maps.google.com/maps?q=1120+union,+san+francisco">Hier wohne
ich</a>
```

Wie Sie sehen, benutzt der Link das ganz normale HTTP-Protokoll für Webseiten. Da Google Maps als native Anwendung unter iOS aber schneller und besser zu bedienen ist als die Webseite, leitet ein iPhone, iPod touch oder iPad gleich auf die Anwendung weiter. Der Link wird wie jeder andere Weblink dargestellt, allerdings wird beim Anklicken der Browser in den Hintergrund gelegt und die Google Maps-Anwendung geöffnet.

Sobald die Anwendung aufgerufen ist, wird der Ort gesucht, der mittels des `q`-Attributs in der URL angegeben wurde. Sie können dabei eine ganz normale Adresse angeben, müssen aber darauf achten, dass die Leerzeichen durch Pluszeichen ersetzt werden.

Bild 1.37: Über den Link wird der Browser in den Hintergrund gelegt und die Google Maps-Anwendung geöffnet.

Ähnlich ist es auch bei Android-Geräten. Hier wird der Benutzer allerdings vor dem Aufrufen der Google Maps-Anwendung vor die Frage gestellt, ob er die Karte nicht doch im Browser anzeigen lassen möchte.

Wie im normalen Browser unterstützt der Aufruf der nativen Google Maps-Anwendung auch weitere (wenn auch bei Weitem nicht alle) Attribute:

Attribute beim Aufruf von Google Maps	
`+(Name+des+Labels)`	Am Ende der angegebenen Adresse kann die Bezeichnung definiert werden, die anstelle der Adresse auf der Karte angezeigt wird. Das unten stehende Beispiel bestimmt, dass anstatt *1120 Union* die Bezeichnung *Meine Bude* verwendet wird.
`t=k`	Die Karte wird nicht im normalen Kartenmodus, sondern mit Satellitenbildern angezeigt.
`t=h`	Die Karte wird nicht im normalen Kartenmodus, sondern im Hybridmodus (Satellitenbilder mit Straßennamen etc.) angezeigt.

Leider werden weitere Layer wie beispielsweise Verkehrsmeldungen oder auch die zu verwendende Zoomstufe nicht mit an die Anwendung übertragen, sondern funktionieren bislang nur im Desktop-Browser nach einem Linkaufruf.

Mit dem nächsten Aufruf erhalten Sie die folgende Kartenansicht:

```
<a href="http://maps.google.com/maps?q=1120+union,+san+francisco+
(Meine+Bude)&layer=t&t=h">Hier wohne ich</a>
```

Bild 1.38: Anstatt *1120 Union* wird die Bezeichnung *Meine Bude* verwendet.

Neben einer einzelnen Adresse können Sie über einen etwas modifizierten Link auch den Google Maps-Routenplaner aufrufen. Unter Android wird Ihnen zusätzlich nach dem Aufruf von Google Maps die Option angeboten, Sie mittels Google-Navigation zum Ziel zu führen. Für die Routenplanung werden zwei Attribute verwendet:

Attribute für die Routenplanung	
`saddr=`	Die Startadresse wird mit diesem Attribut formuliert. Wie schon bei der Suchanfrage wird die Adresse ganz normal angegeben. Nur die Leerzeichen werden wieder mit Pluszeichen ersetzt.
`daddr=`	Die Zieladresse wird mittels dieses Attributs angegeben.

Ein Beispiel:

```
<a href="http://maps.google.com/maps?saddr=New+York&daddr=1120+union,
san+francisco">So kommen Sie von New York zu mir</a>
```

Als Startadresse haben wir nur einen Stadtnamen und als Zieladresse einen Straßennamen, eine Hausnummer und einen Stadtnamen angegeben.

Wird nur die Zieladresse angegeben, hat der Benutzer noch die Möglichkeit, seinen eigenen Startpunkt auszuwählen. Unser Beispiel fragt nach der Route zwischen New York und San Francisco. Mit einem Tag und 22 Stunden konstanter Fahrzeit ist Google da recht optimistisch.

Bild 1.39: Zeigt die Entfernung zwischen zwei Orten.

Alle Beispiele haben wir in diesem kleinen Codebeispiel noch einmal zusammengefasst und es mit iWebKit 5 gestaltet:

```
<!--
   Telefonfunktionen (Call, SMS, E-Mail, Google Maps)
   Markus Spiering
-->
<!DOCTYPE>
<html>
 <head>
   <title>Adresse</title>
   <meta name="layout" content="webkit" />
   <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
   <link href="style.css" rel="stylesheet" media="screen" type="text/css" />
 </head>
 <body>
   <div id="topbar">
      <div id="title">Markus Spiering</div>
   </div>
   <div id="content">
   <span class="graytitle">Telefonnummer:</span>
     <ul class="pageitem">
      <li class="textbox">
        <a href="tel:0351123456789">0351-123456789</a>
       </li>
      </ul>
     <ul class="pageitem">
      <li class="textbox">
        <a href="sms:0351123456789">Schick mir eine SMS</a>
       </li>
      </ul>
     <ul class="pageitem">
      <li class="textbox">
        <a href="mailto:spieri@ymail.com?subject=Anfrage&body=Hallo Mar-kus,
%OA%OAhier ist eine Anfra-ge:&cc=anfragen@spieri.de&bcc=geheim@spieri.de">
Email an mich</a>
       </li>
      </ul>
     <ul class="pageitem">
      <li class="textbox">
        <a href="http://maps.google.com/maps?q=1120+union,+san+francisco+
(Meine+Bude)&layer=t&t=h">Hier wohne ich</a>
       </li>
      </ul>
     <ul class="pageitem">
      <li class="textbox">
        <a href="http://maps.google.com/maps?saddr=New+York&daddr=1120+
union,san+francisco">So kommen Sie von New York zu mir</a>
       </li>
      </ul>
    </div>
```

```
 </body>
</html>
```

Das Beispiel zeigt Ihnen alle vorgestellten Linkformate auf einer Seite, sodass Sie die Funktionen leicht auf verschiedenen Plattformen und Geräten testen können:

Bild 1.40: Beispiel der vorgestellten Linkformate.

1.6 iPhone-spezifisch: Web Apps, die den Browser verbergen

Auch wenn die vorgestellten Beispiele auf den meisten neuen Smartphones gut aussehen, geht das iPhone noch einen Schritt weiter und verwischt den Unterschied zwischen einer Webseite, die im Browser läuft, und einer richtigen App. Weiter unten im Buch werden wir noch darauf eingehen, wie man aus einem Webseitencode eine wirklich native Anwendung baut, doch auch ohne diesen Schritt und mit nur etwas zusätzlichem HTML kann der Safari-Browser ausgeblendet werden.

Das iPhone und auch der iPod touch oder das iPad erlauben es, Webseiten als kleine Symbole auf dem Home-Bildschirm zu speichern. Um das zu tun, muss der Benutzer auf dem iPhone auf das kleine +-Symbol in der unteren Werkzeugleiste des Safari-Browsers klicken und *Zum Home-Bildschirm* auswählen. Anschließend erhält er einen Bildschirm angezeigt, in dem er den Namen des Symbols noch verändern kann. Wenn Sie keine weiteren Anpassungen an Ihrer Webseite vornehmen, wird das iPhone einen Screenshot Ihrer Seite aufnehmen, diesen verkleinern und als Symbol verwenden. Der Benutzer kann anschließend einfach auf Ihr Screenshot-Symbol auf dem Home-Bildschirm tippen und Ihre Seite aufrufen. Um wie eine richtige Webanwendung auszusehen, müsste in diesem Fall

- ein richtiges Symbol und kein Screenshot verwendet werden,
- der Safari-Browser nicht sichtbar sein,
- die Webanwendung sofort nach dem Antippen komplett auf dem Bildschirm erscheinen und nicht erst Element für Element geladen werden.

Die gute Nachricht: Das alles geht beim iPhone! Fangen wir mit einem richtigen Symbol an. Als Icon für Ihre Webanwendung können Sie jede Grafik nehmen, die 114 x 114 Bildpunkte groß und im PNG-Format gesichert ist (beispielsweise unter dem Namen

icon114x114.png). Diese Grafik speichern Sie im Hauptverzeichnis Ihrer Webseite auf dem Server und fügen im `head`-Bereich Ihres HTML-Codes die folgende Zeile hinzu:

```
<link rel="apple-touch-icon" href="icon114x114.png"/>
```

Sie können optional auch ein Piktogramm mit den Maßen 57 x 57 erstellen, was einmal die Originalgröße des iPhones war. Durch den hochauflösenden Bildschirm des iPhones 4 sieht die Variante mit 114 Bildpunkten besser aus und wird auf den älteren iPhones gut heruntergerechnet. Wenn Sie nun die Webseite auf Ihrem iPhone aufrufen und das Pluszeichen antippen, sehen Sie bereits im nächsten Bildschirm, dass Ihre neue Grafik als Symbol verwendet wird.

Bild 1.41: Eine Webanwendung wird dem Home-Bildschirm hinzugefügt. Auf dem Home-Bildschirm wird die Anwendung wie eine iPhone App dargestellt.

Um jetzt beim Start den Safari-Browser auszublenden, brauchen Sie nur folgende Zeile in den `head`-Bereich der Webseite einzubinden:

```
<meta name="apple-mobile-web-app-capable" content="yes" />
```

Dieses `meta`-Tag wird ebenfalls nur von Safari für das iPhone verstanden und von anderen Browsern ignoriert, bewirkt aber, dass beim Antippen des gespeicherten Symbols nur die Webanwendung aufgerufen wird – nicht aber die Safari-Bedienflächen. Sobald der Benutzer eine Webanwendung, die mit diesem `meta`-Tag ausgestattet ist, auf seinem Home-Bildschirm ablegt und wieder startet, bekommt er also nicht mehr die URL-Eingabeleiste oder die Werkzeugleiste am unteren Bildschirmende des Safari-Browsers zu sehen – genau wie bei einer richtigen iPhone-Anwendung.

Bild 1.42: Die Safari-Bedienflächen sind verschwunden. Deutlich wird das, wenn man, wie hier dargestellt, die gesamte Seite nach unten zieht.

Tipp: Es kann nun sogar mit einem bisschen JavaScript und dem `standalone`-Attribut des `navigator`-Objekts überprüft werden, ob die Webseite im Vollbildmodus angezeigt wird oder nicht:

```
<script>
if (window.navigator.standalone)
  alert('Wir sind im Vollbildmodus');
<script>
```

Sie können sogar noch etwas weitergehen und Ihren Benutzern einen kleinen Hinweis darauf geben, dass sie Ihre Web App auf den Home-Bildschirm legen können, um leichter wieder zur Seite zurückzufinden und diese dann auch im Vollbildmodus zu sehen. Das folgende Codebeispiel zeigt jeweils eine Meldungsbox, wenn die Anwendung im Vollbildmodus läuft bzw. wenn nicht:

```
<script>
if (window.navigator.standalone) {
   alert ('Wir sind im Vollbildmodus!')
} else {
   alert('Tipp: Legen Sie diese Seite auf Ihren Home-Bildschirm. Tippen Sie
auf das Pluszeichen im Safari-Browser.')
}
</script>
```

Natürlich sollten Sie in der Praxis die Meldungsbox für den Vollbildmodus weglassen und beispielsweise ein CSS-Overlay für die Meldung verwenden, wenn die Webanwendung nicht im Vollbildmodus läuft.

Die Webanwendung fühlt sich bereits viel mehr nach Anwendung an, allerdings gibt es noch einen großen Unterschied: Nach dem Antippen und Starten einer nativen Anwendung wartet man nicht ein paar Sekunden, bis alle Bildschirmelemente geladen sind. Typisch für iPhone-Anwendungen ist der sogenannte Startup-Bildschirm, der meistens so lange eine schön designte Grafik zeigt, bis alle Daten initialisiert und geladen sind. Mit nur einem weiteren HTML-Tag im `head`-Bereich der Seite lässt sich das genau so bei einer Webanwendung integrieren. Sie benötigen dazu wieder eine Grafik bzw. Bilddatei im PNG-Format, die das Format 320 x 460 oder 640 x 960 Bildpunkte (iPhone 4) besitzen muss. Wie bei der eben beschriebenen Symboldatei kann das PNG auch in 24-Bit-Farbtiefe vorliegen. Nennen Sie die Datei beispielsweise *splash320x240.png,* und fügen Sie folgendes HTML-Tag in den Code ein:

```
<link rel="apple-touch-startup-image" href="splash320x240.png" />
```

Ab sofort wird nach dem Antippen des Symbols auf dem Home-Bildschirm genau diese Grafik angezeigt und erst nach wenigen Augenblicken entfernt, wenn die komplette Webseite geladen ist und dargestellt wird.

Bild 1.43: Dieser Startup-Bildschirm wird so lange angezeigt, bis alle Seitenelemente vollständig geladen sind.

Die Startzeit kürzer erscheinen lassen

Ein Trick, um die Startzeit Ihrer Anwendung kürzer erscheinen zu lassen, ist, ein identisches »Bild« der Anwendung als Startup-Bildschirm zu benutzen. Wenn Sie einmal genau in den App Store oder sogar auf Ihrem iPhone in die YouTube-Anwendung schauen, sehen Sie, was wir meinen. Bei der Auswahl oder Modifikation des Screenshots müssen Sie darauf achten, dass sich keine Linien oder Elemente verschieben und – wesentlich wichtiger – keine Elemente antippbar erscheinen.

Hier zwei Beispiele

Die Flickr-iPhone-Anwendung benutzt einen Startbildschirm, in dem die Kopfleiste mit dem Flickr-Logo bereits sichtbar ist, der eigentliche Inhalt in der Mitte aber noch fehlt. Auch auf die Buttons am unteren Rand wurde verzichtet, da der Benutzer sie sonst versehentlich auf dem Bild benutzen könnte. Sobald die Anwendung vollständig geladen ist, wird der Inhalt in der Mitte geladen, und die Navigationselemente werden eingeblendet.

Bild 1.44: Startbildschirm der Flickr-iPhone-Anwendung.

Ähnlich sieht es bei Facebook aus. Die Aufteilung der Anwendungsoberfläche wird bereits nach dem Start über ein statisches Bild sichtbar, das auch schon das Logo verwendet. Ist die Anwendung geladen, werden die zusätzlichen und antippbaren Elemente angezeigt. Bei der deutschen Version verschiebt sich das Logo, was in der englischen Originalversion nicht der Fall ist. Statt des langen Worts *Live-Meldungen* wird einfach *Live Feed* verwendet, was den Button nicht zu groß werden lässt.

Bild 1.45: Startbildschirm der Facebook-Anwendung.

Standardmäßig besitzt jede iPhone-Anwendung eine Kopfzeile. Diese hat einen grauen Hintergrund und ist der Bereich, in dem beispielsweise der aktuelle Netzbetreiber, die Uhrzeit und der Akkuladestand angezeigt werden. Insbesondere für Webanwendungen, die nach richtigen Anwendungen aussehen sollten, lässt sich das konfigurieren und passt damit eher beispielsweise zu einem schwarzen, modernen Seitendesign. Um die Kopfzeile zu ändern, geben Sie eine der folgenden Zeilen im `head`-Bereich der Seite ein:

Für eine schwarze Kopfzeile:

```
<meta name="apple-mobile-web-app-status-bar-style" content="black"/>
```

Für eine dunkle, aber transparente Kopfzeile:

```
<meta name="apple-mobile-web-app-status-bar-style" content="black-
translucent"/>
```

Für die standardmäßig graue Kopfzeile:

```
<meta name="apple-mobile-web-app-status-bar-style" content="default"/>
```

1.6.1 Startup-Bildschirm und App-Icon mit jQTouch

Falls Sie ohnehin mit der vor einigen Seiten vorgestellten JavaScript- und CSS-Bibliothek jQTouch arbeiten, können Sie den Startup-Bildschirm und das App-Icon gleich dort mitdefinieren, wo Sie auch die jQTouch-Funktionen initialisieren. Mit diesen drei Extrazeilen wird genau das erreicht, was wir eben mittels der CSS-Methoden beschrieben haben:

```
<script type="text/javascript">
var jQT = $.jQTouch({

 icon: 'apple-touch-icon.png',
 statusBar: 'black-translucent',
 startupScreen: 'startup-bildschirm.png'

});
</script>
```

- `icon`: Legt ein Piktogramm für die Webanwendung fest, wenn der Benutzer sie auf dem Home-Bildschirm speichert. Als Icon wird die hinter `icon:` stehende PNG-Datei verwendet.
- `statusBar`: Die Statuszeile wird in diesem Fall schwarz transparent gesetzt. Alternativ ist hier noch `black` möglich. Wird dieser Parameter nicht spezifiziert, wird die Standardstatuszeile in Grau verwendet.
- `startupScreen`: Die hinter dem Parameter stehende Bilddatei wird als Startup-Bildschirm verwendet und so lange angezeigt, bis die Webanwendung vollständig geladen ist.

1.6.2 CSS-Gestaltung für Hoch- und Querformat

Bei vielen iPhone-Anwendungen und auch bei einigen wenigen mobilen Webseiten ist es üblich, dass sie gezielt ihr Aussehen nach der Ausrichtung des Telefons ändern, je nachdem ob der Benutzer es im Hoch- oder im Querformat hält. Im `head`-Bereich einer HTML-Datei können Sie mit den folgenden Anweisungen unterschiedliche CSS-Dateien für die jeweilige Bildschirmausrichtung zuweisen:

```
<link rel="stylesheet" media="all and (orientation:portrait)"
href="portrait-style.css" />
 <link rel="stylesheet" media="all and (orientation:landscape)"
href="landscape-style.css" />
```

Das CSS-`orientation`-Attribut funktioniert auf der iOS-Plattform ab iOS 4.0 für den iPod touch und das iPhone sowie für das iPad ab Version 3.2. Auch Telefone mit Android-Betriebssystem verstehen es.

Am besten lässt sich diese Funktionalität wieder anhand eines kleinen Beispiels verdeutlichen. Wir erstellen eine einfache mobile Seite, die im Hochformat ein kleineres Bild auf weißem Hintergrund anzeigt und im Querformat das gleiche Bild lädt, es aber auf einem schwarzen Hintergrund größer darstellen soll. Zuerst erstellen Sie die CSS-Zuweisungen für die Hochformatansicht in der Datei *portrait-style.css*:

```
body  { color: black;
        background-color: #ffffff;
        font-family: Arial;
        font-size: 12px;
}
#imageContainer {
    background: url('stairs.jpg') no-repeat top center;
    height: 240px;
    width: 320px;
}
```

Setzen Sie am Anfang den ganzen `body`-Bereich in einen weißen Hintergrund, und nehmen Sie die üblichen Schriftzuweisungen vor. Weiterhin legen Sie ein ID-Attribut `#imageContainer` an, das nicht nur die Bilddatei enthält, sondern auch die Höhe und Breite des anzuzeigenden Bilds definiert. Das Gleiche machen Sie noch einmal in der Datei `landscape-style.css`, allerdings mit etwas geänderten Werten:

```
body  { color: white;
        background-color: #000000;
        font-family: Arial;
        font-size: 12px;
}
#imageContainer {
    background: url('stairs.jpg') no-repeat top center;
    height: 267px;
    width: 400px;
}
```

In der Datei *landscape-style.css* setzen Sie die Hintergrundfarbe auf Schwarz und sorgen auch mittels der `height`- und `width`-Attribute für eine größere Darstellung des Bilds. Der HTML-Code für das kleine Beispiel:

```
<html>
<head>
 <title>Rotationsbeispiel</title>
 <meta name="viewport" content="width=device-width; initial-scale=1.0;
maximum-scale=1.0; user-scalable=0;" />
 <link rel="stylesheet" media="all and (orientation:portrait)"
href="portrait-style.css" />
 <link rel="stylesheet" media="all and (orientation:landscape)"
href="landscape-style.css" />
</head>
<body>
   <div id="imageContainer">
   </div>
</body>
</html>
```

Das Ergebnis ist: Hält der Benutzer sein Telefon hochkant, wird nur ein kleines Bild geladen und auf weißem Hintergrund dargestellt. Sobald der Benutzer sein Telefon in das Querformat dreht, wechselt sofort die Hintergrundfarbe, und das Bild wird wesentlich größer dargestellt.

Bild 1.46: Die Ansicht im Hoch- und im Querformat.

Alternativ: doppelte CSS-Verweise im HTML-Code

Es gibt einen Trick, mit dem es sich am Anfang besser arbeiten lässt, vor allem dann, wenn Sie nicht zwei komplett unterschiedliche CSS-Varianten pflegen und dazu noch unterschiedliche Dateien in den jeweiligen Ansichten verwenden wollen.

Auch bei diesem Alternativvorschlag verwenden wir die oben genannten Verweise auf die beiden CSS-Dateien. Allerdings sieht deren Inhalt nun anders aus. Für die Datei *portrait-style.css* verwenden Sie jetzt folgende Definitionen:

```
body  { color: black;
        background-color: #ffffff;
        font-family: Arial;
        font-size: 12px;
}
#landscape {display:none}
```

Führen Sie hier die ID-Zuweisung `#landscape` ein, und definieren Sie im CSS, dass, wenn diese ID im HTML-Code verwendet wird, dieser HTML-Code nicht angezeigt werden soll. Genau das Gleiche, aber umgekehrt, machen Sie für die Datei *landscape-style.css*:

```
body  { color: white;
        background-color: #000000;
        font-family: Arial;
        font-size: 12px;
}
#portrait {display:none}
```

Dreht der Benutzer sein Telefon in die horizontale Richtung und wird die Datei *landscape-style.css* für die Darstellung benutzt, werden alle mit der ID `portrait` versehenen Elemente im HTML-Code nicht angezeigt. Auch der HTML-Code muss etwas angepasst werden:

```
<html>
<head>
 <title>Rotationsbeispiel</title>
 ...
</head>
<body>
   <img id="portrait" src="stairs_small.jpg" height="240" width="320"
alt="Hochformatbild" />
   <img id="landscape" src="stairs_normal.jpg" height="267" width="400"
alt="Querformatbild" />
</body>
</html>
```

Verwenden Sie einfach zwei `img`-Tags auf einer Seite. Ein Tag wird mit Sicherheit ignoriert, da Sie das ja in den CSS-Definitionen festgeschrieben haben. Der Vorteil bei dieser Variante liegt darin, dass Sie genau steuern können, welche HTML-Elemente in der jeweiligen Ansicht sichtbar sein sollen und welche nicht.

2 HTML5 in der mobilen Webentwicklung

In diesem Kapitel erwartet Sie die technische Vorstellung der wichtigsten Neuerungen und Änderungen rund um HTML5. Wie Sie bereits wissen, war der Umfang der HTML5-Spezifikation häufigen Änderungen unterworfen, und somit hat sich in den Medien ein etwas konfuses Bild des Umfangs der HTML5-Spezifikation gebildet. Da APIs wie beispielsweise die Geolocation-API, genau genommen nicht in den Umfang der HTML5-Spezifikation gehören, die darin enthaltenen Features rund um die Ortung von Geräten jedoch gerade für die mobile Webapplikationsentwicklung immens wichtig sind, wird das ebenfalls in diesem Kapitel behandelt.

Weil der Umfang dieses Buchs jedoch beschränkt ist und sich viele hier besprochene Spezifikationen leider noch nicht im finalen Stadium befinden, konzentrieren wir uns auf die für die mobile Webentwicklung relevanten und derzeit implementierten Features.

Um die Beispiele in diesem Kapitel verstehen zu können, müssen die Grundlagen der Webentwicklung und die damit verknüpften Technologien wie HTML, CSS und JavaScript verstanden werden. Der folgende Abschnitt erklärt diese Technologien nicht von Grund auf, sondern stellt die neuen, relevanten Funktionen und Möglichkeiten vor.

▣ Download

http://www.buch.cd

Um die Codebeispiele in diesem Kapitel strukturiert überprüfen zu können, wurde eine kleine mobile Webanwendung erstellt, die Sie über die Seite zum Buch abrufen können.

2.1 Von HTML 4 nach HTML5

HTML5 definiert zwei Schreibweisen, die sogenannte HTML-Schreibweise und eine XML-konforme Schreibweise.

2.1.1 Syntaktische Unterschiede

Diese beiden Schreibweisen erläutern wir am besten anhand von kurzen Beispielen. Betrachten wir zunächst die HTML-Schreibweise:

```
<!doctype html>
<html>
  <head>
    <meta charset="UTF-8">
    <title>Beispiel HTML5-Schreibweise</title>
```

```
  </head>
  <body>
    <p>Ein Paragraf</p>
  </body>
</html>
```

Auffallend neu ist zunächst die erste Zeile, die ungewohnt kurz erscheint. Auf den HTML5-`doctype`, der nur bei der HTML-Schreibweise vorgeschrieben ist, werden wir im nächsten Abschnitt noch näher eingehen. Kurz gefasst, teilt er dem Browser mit, dass die standardkonforme Darstellung der HTML-Seite benutzt werden soll. Mit dem HTML 4-`doctype`-Hell ist damit ein für alle Mal Schluss.

Der HTTP-Content-Type, der bei der HTML-Schreibweise benutzt werden sollte, ist `doctype/html`.

Mit Ausnahme des neuen `doctype` ist bei dem obigen Beispiel noch anzumerken, dass – im Unterschied zur XML-Schreibweise – manche Tags nicht zwingend geschlossen werden müssen. Die folgende Zeile aus dem HTML-Beispiel ist kein Fehler, sondern korrektes HTML5 nach der HTML-Schreibweise:

```
<meta charset="UTF-8">
```

Beachten Sie, dass das Tag frei steht und nicht geschlossen ist. Es ist nicht XML-konform, da kein geschlossenes Tag vorhanden ist, jedoch korrektes HTML5.

Die zweite Syntax, die HTML5 definiert, ist die XML-Schreibweise. Das HTML-Dokument muss hierbei XML-konform sein, also beispielsweise muss jedes öffnende Element ein entsprechendes schließendes Element haben:

```
<?xml version="1.0" encoding="UTF-8"?>
<html xmlns="http://www.w3.org/1999/xhtml">
  <head>
    <title>Beispieldokument</title>
  </head>
  <body>
    <p>Beispielabsatz</p>
  </body>
</html>
```

Das Encoding wird hierbei auch durch die XML-Deklaration in der ersten Zeile angegeben und nicht per `meta`-Tag wie bei der HTML-Schreibweise. Ebenso wichtig ist, dass der XHTML-Namespace durch das `xmls`-Attribut des HTML-Tags angegeben werden sollte. Der Content-Type, der für die XML-konforme Schreibweise benutzt werden sollte, ist `application/xhtml+xml` oder `application/xml`.

Beachten Sie auch, dass der HTML5-`doctype` hier nicht mehr benötigt wird, da die standardkonforme Darstellung der HTML-Seite die XML-Schreibweise impliziert.

In der Realität müssen wir feststellen, dass viele HTML-Autoren die XML-Schreibweise vorziehen, diese HTML-Dokumente jedoch selten mit `application/xml`-Content-Type vom Webserver ausgeliefert werden. Ebenso ist der `doctype` bei Benutzung der XML-Schreibweise nicht notwendig, er wird aber dennoch von vielen Autoren benutzt.

Glücklicherweise scheinen die aktuellen Browser, die zunehmend HTML5-Features beherrschen, damit überaus gnädig umzugehen.

2.1.2 Character Encoding

Bei Benutzung der HTML-Syntax haben die Autoren mehrere Möglichkeiten, das Character-Encoding anzugeben:

- Auf der Transportebene durch den HTTP-Content-Type-Header.
- Durch die Benutzung eines `meta`-Elements und dessen `charset`-Attribut.

Für die XML-Syntax gelten die üblichen XML-Regeln, also beispielsweise die Deklaration des Encodings in der XML-Deklaration, wie im XML-Beispiel demonstriert.

2.1.3 Der !doctype html

Der `!doctype html` ist lediglich bei der HTML-Schreibweise vorgeschrieben und hat den einzigen Zweck, dem Browser mitzuteilen, die standardkonforme Darstellung der HTML-Seite zu benutzen. Bei der XML-Schreibweise ist er komplett überflüssig, da die Benutzung dieser Schreibweise bereits die standardkonforme Darstellung impliziert. Die noch unter HTML 4 umständliche und lange Schreibweise des `doctype` ist nicht mehr erforderlich. Verwenden Sie daher für Seiten, die HTML5 nutzen, als erste Zeile im Code immer:

```
<!doctype html>
```

2.2 Sprachliche Unterschiede

Die sprachlichen Unterschiede zu HTML 4 werden in den folgenden Absätzen erläutert. HTML5 definiert zahlreiche neue Elemente und neue Attribute für bestehende Elemente, verändert die Bedeutung bestehender Elemente und verwirft Attribute oder Elemente. An dieser Stelle soll nur ein kurzer Überblick über die wichtigsten Neuerungen gegeben werden. Diese Ausführungen können also eine HTML5-Referenz nicht ersetzen.

Allen voran stehen neue Elemente in HTML5, die HTML eine bessere Struktur geben. Dazu gehören Elemente wie `section`, `article`, `aside` sowie die `header`- und `footer`-Elemente. Oftmals lösen sie `div`-Elemente ab, die per `id`-Attribut eindeutig in der HTML-Seite referenziert werden konnten. Der Vorteil dieser neuen Elemente liegt darin, dass sie tatsächlich über den Inhalt eine Aussage machen. Damit wird eine maschinelle Auswertung des Inhalts überhaupt erst möglich. Betrachten wir einige neue Elemente:

- `article`
- `aside`
- `hgroup`
- `header`
- `footer`
- `nav`

- `figure/figcaption`

Weitere neue Elemente, die nicht sprachlicher Natur sind, sondern konkret neue Features in HTML5 ermöglichen, sind:

- `audio` und `video` (inklusive `source`-Element)
- `canvas`
- `time`
- `details/summary`
- `output`

Für das `input`-Element werden zahlreiche neue Werte des `type`-Attributs definiert. Dadurch hat der Browser die Möglichkeit, die Darstellung dieser Elemente je nach Typ zu optimieren. Zu den derzeit im iPhone unterstützten Typen gehören beispielsweise `tel`, `url`, `email` und `number`. Im Fall eines

```
<input type='number'/>
```

wird dem Benutzer keine komplette Tastatur präsentiert, sondern nur ein Ziffernblock. Die Eingabe der Daten durch den Nutzer wird dadurch natürlich deutlich vereinfacht. Ebenso wurden bestehenden HTML-Elementen zahlreiche neue Attribute hinzugefügt:

- Das `a`-Element hat nun ein `ping`-Attribut, das die URL zu einem Tracking-Pixel enthalten kann. Tracking-Pixel sind üblicherweise GIF-Bilder, die 1 x 1 Pixel groß und transparent sind. Diese Pixel werden dazu verwendet, Werbeeinblendungen oder Aktionen des Benutzers zu verfolgen. Der Browser fordert das definierte Pixel an, bevor der eigentliche Link aufgerufen wird.
- Ein äußerst praktisches neues Merkmal des `input`- und `textarea`-Elements ist das Attribut `placeholder`. Sicherlich haben Sie auch schon JavaScript geschrieben, um einen Platzhaltertext innerhalb eines `input`-Elements zu platzieren, wenn dieses Element keinen Inhalt hat. Das `placeholder`-Attribut macht diesen JavaScript-Code nun überflüssig.
- Das `input`-Element hat jetzt ebenfalls zahlreiche neue Attribute, um weitere Einschränkungen zu definieren. Dazu gehören die Attribute `min`, `max`, `pattern` und `step`.
- Ganz neu in HTML5 sind die sogenannten Offline Web Applications. Es wird später noch genauer darauf eingegangen. Kurz gefasst, kann eine gewöhnliche HTML-Seite damit offline, also komplett ohne Internetverbindung, weitergenutzt werden, da sämtliche Ressourcen dieser Seite auf dem Client gespeichert werden.
- Das `iframe`-Element hat drei neue Attribute: `sandbox`, `seamless` und `srcdoc`. Dadurch lässt sich definieren, welchen Zugriff das enthaltene HTML-Dokument auf die Hostseite erhält – praktisch für Blog-Kommentare. Um hier zu verhindern, dass ein eingeschleustes Stück JavaScript mit der Hostseite Unfug treibt, kann das entsprechend restriktiv belegt werden. Die `iframe`-HTML-Seite hat damit keinen Zugriff auf die umgebende HTML-Seite.

Des Weiteren sind nun bekannte Attribute aus HTML 4 zu globalen Attributen gemacht worden. Die Attribute `class`, `dir`, `id`, `lang`, `style` und `tabindex` sind jetzt in jedem Element vorhanden.

Es wurden auch neue globale Attribute definiert. Dazu gehören unter anderem:

- `contenteditable`
- `draggable`
- `hidden`
- `spellcheck`

Zu den nicht mehr empfohlenen Attributen zählen beispielsweise das `border`-Attribut des `img`-Elements und das `language`-Attribut des `script`-Elements. Der Rahmen eines Bilds kann besser über CSS definiert werden, und der Wert des `language`-Attributs des `script`-Tags muss nun, wenn vorhanden, JavaScript sein.

Zu den gänzlich entfernten Elementen gehören `basefont`, `center` und `font` sowie alle `frame`-Elemente. Diese können entweder besser über CSS-Styles ausgedrückt werden oder haben mehr Verwirrung als Nutzen gestiftet. Zur letzteren Kategorie gehört auch das `applet`-Tag, durch das kleine Java-Programme eingebunden werden konnten. Das `object`-Tag macht dessen Benutzung überflüssig.

2.2.1 Neue APIs

HTML5 definiert auch zahlreiche neue APIs, die über JavaScript zugänglich sind. Nachfolgend werden ein paar der APIs kurz erläutert:

- Eine API für die `audio`- und `video`-Tags ermöglicht die Kontrolle der Medien.
- Die ApplicationCache-API ermöglicht es, offline Webapplikationen programmatisch zu kontrollieren. Beispielsweise kann festgestellt werden, ob eine Applikation gecacht wurde oder ob die Ressourcen derzeit noch gespeichert werden.
- Es wird auch eine API definiert, durch die Webapplikationen für bestimmte Protokolle und Medientypen registriert werden können. Eine webbasierte E-Mail-Applikation kann dadurch als Standardprogramm für E-Mail verwendet werden.
- In Verbindung mit dem `contenteditable`-Attribut wird eine API definiert, durch die das Editieren des Inhalts von Elementen kontrolliert werden kann.
- Die HTML5-Drag & Drop-API definiert, wie die beliebten Drag & Drop-Operationen auf Webapplikationen konform umgesetzt werden können. Dadurch wird auch die Drag & Drop-Interaktion mit browserexternen Programmen definiert.
- Schließlich wird auch eine API definiert, durch die die History des Browsers per JavaScript zugänglich gemacht wird. Dadurch wird es AJAX-Applikationen ermöglicht, die Probleme mit dem Zurück-Button zu lösen.

2.2.2 Neue Input-Typen, Auto-Korrektur und Auto-Großschreibung

HTML5 definiert für das `input`-Element zahlreiche neue Werte des `type`-Attributs, darunter beispielsweise `url`, `email`, `number` und `tel`. Der Großteil der komplexeren Typen wie `datetime` oder `range` werden leider noch nicht von aktuellen Browsern umgesetzt, auf iPhone OS sind jedoch immerhin folgende Typen verfügbar und werden von der UI entsprechend unterstützt:

- `url`
- `email`
- `tel`
- `number`

Die folgende Tabelle zeigt, wie auf iPhone OS derzeit die Tastaturen dieser Typen definiert sind. Generell wird die Tastatur so angepasst, dass der Nutzer die einzugebenden Daten leichter als mit der Standardbelegung der Tastatur eintippen kann.

iPhone-Tastaturbelegung nach input-Type			
			
‹input placeholder= "number " type= "number" /›	‹input placeholder= "email" type="email" /›	‹input placeholder= "url" type="url" /›	‹input placeholder= "tel" type="tel" /›

Sowohl von Android als auch von iPhone OS wird das `placeholder`-Attribut unterstützt, wodurch ein Platzhaltertext für ein leeres `input`-Element angegeben werden kann. Sobald der Benutzer in das Feld klickt, verschwindet der Platzhalter, und die Eingabe kann stattfinden.

Die automatische Korrektur der Eingaben und die automatische Großschreibung nach Punkt, Doppelpunkt, Fragezeichen und Ausrufezeichen sind leider wiederum nur auf dem iPhone OS implementiert.

Bild 2.1: Automatische Korrektur beim iPhone.

Um die automatische Korrektur einzuschalten oder explizit auszuschalten, kann das `autocorrect`-Attribut des `input`-Elements benutzt werden:

```
<input autocorrect="on"/>
<input autocorrect="off"/>
```

Um die Großschreibung zu steuern, wird das Attribut `autocapitalize` benutzt. Ein normales `input`-Element vom Typ `text` hat die automatische Großschreibung standardmäßig aktiviert, um sie also explizit auszuschalten, schreiben Sie:

```
<input type="text" autocapitalize="off"/>
```

Neben diesen neuen Typen und größtenteils iPhone-spezifischen Attributen ist leider recht wenig der HTML5-Spezifikation in diesem Bereich implementiert. Komplexere Typen wie `datetime`, die vom Browser in Datumseingabefelder umgesetzt werden könnten, sind beispielsweise weder auf dem iPhone noch im WebKit-Browser auf Android implementiert.

Auch viele der neuen Attribute, durch die beispielsweise die Validierung der Eingaben vorgenommen werden können (über das `pattern`-Attribut sollte ein regulärer Ausdruck angegeben werden können etc.), werden nicht unterstützt. Sicherlich wird sich die Lage mit der Zeit bessern, derzeit verzichten wir jedoch auf eine weitere Vorstellung dieser Features und nutzen den Platz in diesem Buch für diejenigen, die tatsächlich funktionieren.

Das folgende Beispiel ruft alle derzeit verfügbaren `input`-Types auf:

```
<!-- HTML5-Input-Typen / Eingabefelder
     Markus Spiering / Sven Haiges
-->
<!DOCTYPE html>
<html>
    <head>
        <title>HTML5-Eingabefelder</title>
       <meta name="layout" content="webkit" />
      <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
      <link href="style.css" rel="stylesheet" media="screen" type="text/css" />

      <script>
      (function my() {
         var $i = {};

         $i.init = function()
         {
            var detectionNode, msg = '', formats, pos, audio;
         };
         document.addEventListener("DOMContentLoaded", $i.init, false);
      }).call();
      </script>
    </head>
    <body>

    <div id="topbar">
      <div id="title">Neue Input-Typen</div>
    </div>
     <div id="content">
```

```
         <ul class="pageitem">
            <li class="bigfield">
               <input placeholder="text" type="text" autofocus="true"/>
            </li>
            <li class="bigfield">
               <input placeholder="text autocorrect" type="text"
autocorrect="on"/>
            </li>
            <li class="bigfield">
               <input placeholder="text autocapitalize off" type="text"
autocapitalize="off"/>
            </li>
            <li class="bigfield">
               <input placeholder="tel" type="tel" />
            </li>
            <li class="bigfield">
               <input placeholder="url" type="url" />
            </li>
            <li class="bigfield">
               <input placeholder="email" type="email" />
            </li>
            <li class="bigfield">
               <input placeholder="password" type="password" />
            </li>
            <li class="bigfield">
               <input placeholder="number" type="number" />
            </li>
            <li class="bigfield">
               <input placeholder="text pattern" type="text" pattern="[0-
9]*"/>
            </li>
            <li class="checkbox"><span class="name">Checkbox </span>
               <input type="checkbox" />
            </li>
            <li class="radiobutton">
               <span class="name">A</span><input name="GroupName"
type="radio" value="A" />
            </li>
            <li class="radiobutton">
               <span class="name">B</span><input name="GroupName"
type="radio" value="B" />
            </li>
            <li class="bigfield">
               <input type="range" value="0" min="0" max="10" step="2"/>
            </li>
            <li class="bigfield">
               <input placeholder="datetime*" type="datetime" />
            </li>
            <li class="bigfield">
               <input placeholder="date*" type="date" />
            </li>
            <li class="bigfield">
```

```
            <input placeholder="month*" type="month" />
          </li>
          <li class="bigfield">
            <input placeholder="week*" type="week" />
          </li>
          <li class="bigfield">
            <input placeholder="time*" type="time" />
          </li>
          <li class="bigfield">
            <input placeholder="datetime-local*" type="datetime-local" />
          </li>
          <li class="bigfield">
            <input placeholder="color*" type="color" />
          </li>
        </ul>
      </div>
      <div id="footer">
        <a class="noeffect" href="http://iwebkit.net">Powered by iWebKit</a>
      </div>
    </body>
</html>
```

2.3 Das canvas-Element

Das HTML5-`canvas`-Element erlaubt das programmatische Erstellen von Bitmap-basierten Grafiken. Nachdem ein `canvas`-Tag in die HTML-Seite eingefügt worden ist, kann es per JavaScript angesprochen werden, und es kann darauf gezeichnet werden. Das `canvas`-Element und die damit on-the-fly erstellbaren Grafiken können für eine Vielzahl von Anwendungsfällen nützlich sein:

- Eine Finanzapplikation kann nahezu in Echtzeit die aktualisierten Chartdaten per AJAX vom Server abrufen. Mit den erhaltenen Daten wird ein Aktienchart aktualisiert. Anstatt die Bilddaten auf dem Server zu rendern und die Bilder abzurufen, müssen nur die Daten übertragen werden, das Rendern erledigen die HTML5-Clients (die Browser). Der Server wird dadurch entlastet, weniger Bandbreite wird verbraucht, und die Aktualisierung der Daten geht schneller.
- Ein Jump & Run-Spiel rendert den kompletten Spielscreen per HTML5-`canvas`-Element. Dieser Anwendungsfall wird durch das `canvas`-Element überhaupt erst ermöglicht oder war mit bisherigen Mitteln nur schwer zu erreichen.
- Eine Zeichenapplikation nimmt vom Benutzer Maus- und Tastaturkommandos entgegen und setzt sie auf einem Canvas um. Das erstellte Bild kann im Browser dank SQL Storage gespeichert werden, die Applikation kann offline voll genutzt werden.

Die Möglichkeiten sind (fast) unbegrenzt. Aus mobiler Sicht ist das `canvas`-Element doppelt interessant, da wir im mobilen Bereich zum einen nicht immer auf eine beständige Internetverbindung setzen können. Des Weiteren sind die Verbindungen immer noch langsam, zudem wird oft nutzungsabhängig abgerechnet. Um den Nutzer der

Applikation nicht mit der nächsten Rechnung böse zu überraschen, sollte eine mobile Webapplikation intelligenter mit den vorhandenen Ressourcen umgehen.

In diesem Unterkapitel werden die Grundlagen und Features des `canvas`-Elements anhand vieler kleiner Beispiele vorgestellt. Die Beispiele (inklusive der vielen Grafiken) sollen es Ihnen ermöglichen, einen raschen Überblick zu bekommen, um selbst abschätzen zu können, inwiefern Sie das `canvas`-Element für eigene Projekte benutzen können.

Genug der vielen Worte, jetzt wird gezeichnet! Das folgende Beispiel zeigt, wie einfach ein `canvas`-Element in eine HTML5-Seite eingebunden wird:

```
<canvas>Ihr Browser unterstützt kein HTML5-Canvas!</canvas>
```

Einfach, oder? `canvas` hat nur zwei Attribute, `width` und `height`. Standardmäßig sind diese mit 300 (Breite) und 150 (Höhe) gesetzt. Der Inhalt des `canvas`-Elements ist der sogenannte Fallback-Content. Er wird nur angezeigt, wenn der darstellende Browser kein `canvas`-Element unterstützt. Ein so erstelltes `canvas`-Element sieht im Browser dann so aus:

Bild vergessen? Nein. Ein Canvas ist zunächst leer, d. h. vollständig transparent. Um die Fläche des Canvas sichtbar zu machen, kann er jedoch wie jedes andere HTML-Element per CSS gestylt werden.

```
<canvas style="border: 1px dotted black;" id="myCanvas"></canvas>
```

Im obigen Beispiel wurde auch gleich ein `id`-Attribut hinzugefügt, damit wir das Element später aus JavaScript ansprechen können. Der Canvas sieht nun so aus:

Bild 2.2: `canvas` mit `id`-Attribut.

Jetzt können wir per JavaScript auf den Canvas zugreifen und darauf zeichnen. Das folgende Beispiel, das wir gleich Zeile für Zeile durchgehen, zeichnet zehn Quadrate mit unterschiedlichen Füllungen auf den Canvas:

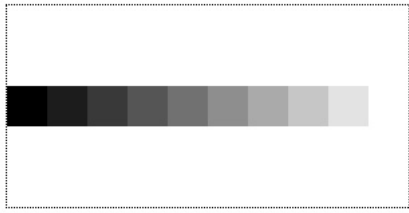


Bild 2.3: Auf dem Canvas zeichnen.

```
<!doctype html>
<html>
<head><title>Canvas-Demos</title>
<script>
   function run()
   {
      var canvas = document.getElementById('myCanvas');
      var ctx = canvas.getContext('2d');

      for (var i = 0; i < 10; i++)
      {
         var color = Math.round((i/9) * 255);
         ctx.fillStyle = "rgb("+color+","+color+","+color+")";
         ctx.fillRect (i*30, 60, 30, 30);
      }
   }
   document.addEventListener("DOMContentLoaded", run, false);
</script>
</head>
<body>
   <canvas style="border: 1px dotted black;" id="myCanvas">
      Ihr Browser unterstützt kein HTML5-Canvas!
   </canvas>
</body>
</html>
```

Das `canvas`-Element im Body der Seite ist uns schon bekannt. Da wir `width` und `height` nicht weiter angegeben haben, wird der Canvas standardmäßig 300 x 150 Pixel groß sein. Wirklich spannend wird es im JavaScript-Teil dieser Seite.

Zunächst wird ein Event-Listener für das `DOMContentLoaded`-Event registriert. Dadurch wird unsere Funktion `run` aufgerufen, sobald der DOM-Tree (Document Object Model) der Seite vollständig verfügbar ist. Die ersten beiden Zeilen der Funktion greifen dann auf das `canvas`-Element und dessen Kontext zu:

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
```

Der Context, genau gesagt ein `CanvasRenderingContext2D`, ist das Objekt, das zum Zeichnen innerhalb des Canvas benutzt wird. Die HTML5-Spezifikation schreibt lediglich die Implementierung eines 2-D-Inhalts vor.

Um auf dem nun referenzierten Inhalt zu zeichnen, setzen wir Eigenschaften des referenzierten Inhalts und rufen dessen Methoden auf – im Beispiel legen Sie die aktuell

benutzte Füllfarbe dynamisch innerhalb der `for`-Schleife fest und erzeugen so pro Durchgang einen unterschiedlichen Grauwert. Im nächsten Schritt wird ein Quadrat mit den Maßen 30 x 30 an dynamischer x-Position (je nach Durchgang also 0, 30, 60, 90 etc.) und an statischer y-Position 60 gezeichnet. Der Ursprung für alle Operationen ist dabei immer die obere linke Ecke des Canvas.

```
ctx.fillStyle = "rgb("+color+","+color+","+color+")";
ctx.fillRect (i*30, 60, 30, 30);
```

Da wir mit der Füllfarbe `rgb(0,0,0)` beginnen und bei `rgb(255,255,255)` enden, werden durch die zehn Durchgänge der Schleife zehn Quadrate in unterschiedlichen Graustufen aneinandergereiht. Unter Zuhilfenahme des CSS-Frameworks iWebKit können Sie mit leicht modifiziertem Code das Beispiel wie folgt für ein iPhone oder Android-Telefon umsetzen:

```
<!--
   Einfache Canvas-Demo mit iWebKit 5
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
<html>
  <head>
    <title>Mobile Canvas</title>
   <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
   <meta name="layout" content="webkit" />
   <link href="style.css" rel="stylesheet" media="screen" type="text/css" />

         <script>
         function init()
         {
            var canvas = document.getElementById('myCanvas');
            var ctx = canvas.getContext('2d');

            for (var i = 0; i < 10; i++)
            {
               var color = Math.round((i/9) * 255);
               ctx.fillStyle = "rgb("+color+","+color+","+color+")";
               ctx.fillRect (i*28, 61, 28, 28);
            }

            // Funktioniert auf dem iPhone ab 3.1
            var dataURL = canvas.toDataURL();
            var img = document.createElement('img');
            img.setAttribute('src', dataURL);
            document.getElementById('box').appendChild(img);

            // Funktioniert in FF, bislang aber noch nicht auf
iPhone/Android 2.2
            var imageData = ctx.getImageData(0,0,280,150);
            document.getElementById('message').innerHTML = data.width + '/'
+ data.height;
         }
         document.addEventListener("DOMContentLoaded", init, false);
```

```
        </script>
      </head>
      <body>
        <div id="topbar">
           <div id="title">Einfacher Canvas</div>
        </div>
        <div id="content">
           <ul class="pageitem">
              <li class="textbox" id="box">
                 <canvas style="border: 1px dotted black;" width="280"
height="150" id="myCanvas">
                    Ihr Browser unterstützt kein HTML5-Canvas!
                 </canvas>
                 <p id='message'>
                 Falls Sie eine zweite Grafik unten sehen, dann unterstüzzt
ihr Browser canvas.toDataURL()</p>
              </li>

           </ul>

        </div>
      </body>
</html>
```

Bild 2.4: Ein einfacher Canvas unter iOS 4.

Die folgenden Abschnitte stellen Ihnen Schritt für Schritt die Methoden und Konzepte des HTML5-Canvas vor, sind aber keineswegs als vollständige und umfassende Einführung gedacht, denn das würde ein eigenes Buch füllen. Dennoch werden die meisten und wichtigsten Features des HTML5-Canvas vorgestellt, um Ihnen einen guten Überblick zu geben. Der Canvas aller Beispiele hat eine Breite von 280 Pixeln und eine Höhe von 300 Pixeln. Die Variable `ctx` referenziert in allen Beispielen den 2-D-Kontext des Canvas:

```
var canvas = document.getElementById('myCanvas');
var ctx = canvas.getContext('2d');
... Beispielcode ...
```

2.3.1 Formen und Pfade

Das HTML5-`canvas`-Element unterstützt nur eine primitive Form: das Rechteck. Alle anderen Formen wie beispielsweise Kreise oder zusammengesetzte Formen werden durch Pfade gezeichnet und dann gefüllt. Die Canvas-API stellt uns drei Methoden speziell für Rechtecke zur Verfügung:

```
void clearRect(in float x, in float y, in float w, in float h);
void fillRect(in float x, in float y, in float w, in float h);
void strokeRect(in float x, in float y, in float w, in float h);
```

Die Parameter dieser drei Methoden haben die gleiche Bedeutung: `x` und `y` geben die Position relativ zur oberen linken Ecke des Canvas an. Die Parameter `w` und `h` geben die Breite und Höhe der rechteckigen Fläche an.

Um den Canvas zu löschen – entweder komplett oder teilweise –, bietet sich die Methode `clearRect()` an. Nehmen wir einen Canvas mit den Maßen 100 x 100 an, würde der folgende Aufruf somit die Fläche des Canvas komplett löschen.

```
clearRect(0,0,100,100);
```

Was zurückbleibt, ist eine transparente Fläche. Wenn also der Hintergrund des Canvas – beispielsweise durch eine zugewiesene Hintergrundfarbe im `<body>` der HTML-Seite – eingefärbt ist, wird die Farbe des Hintergrunds durchgereicht.

Um ein nicht gefülltes Rechteck zu zeichnen, also ein Rechteck, das nur durch die äußeren Randlinien erkennbar ist, wird die Methode `strokeRect()` benutzt; um dieses Rechteck gefüllt auf den Canvas zu zeichnen, die Methode `fillRect()`.

Das nächste Codebeispiel demonstriert diese drei Methoden, das Resultat ist in der folgenden Abbildung zu sehen:

```
ctx.fillStyle="rgb(255,0,0)";
ctx.fillRect(10, 10, 260, 30);

ctx.strokeStyle="rgb(0,255,0)";
ctx.strokeRect(10, 50, 260, 30);
ctx.clearRect(20, 20, 100, 10);
```

Um Formen durch Pfadfunktionen zu zeichnen, gibt die Canvas-API folgende Methoden vor:

```
void beginPath();
void closePath();
void fill();
void stroke();
void clip();
```

Auf die `clip()`-Methode wird als Letztes in diesem Abschnitt eingegangen, zunächst ist es wichtig, die vier anderen Methoden zu verstehen. Um einen Pfad zu zeichnen, muss zunächst die Methode `beginPath()` aufgerufen werden. Der Canvas, oder besser der 2-D-Kontext des Canvas, ist dann bereit, weitere Zeichenfunktionen aufzunehmen. Die `closePath()`-Methode verbindet den Anfangspunkt des Pfads mit dem aktuellen End-

punkt durch eine gerade Linie. Der Aufruf ist optional, da der Pfad bereits durch die angewendeten Zeichenmethoden geschlossen sein kann. Zuletzt werden die Methoden `stroke()` und `fill()` aufgerufen. `stroke()` zeichnet dabei den Umriss der gezeichneten Form, während `fill()` die Form ausfüllt.

Eine sehr nützliche Methode ist `moveTo()`. Vereinfacht ausgedrückt, können damit Pfade gezeichnet werden, die Lücken haben. Betrachten Sie die beiden Linien des folgenden Beispiels, die mit einem einzigen Pfad gezeichnet wurden:

```
//2 Linien, 5px Abstand
ctx.beginPath();
ctx.strokeStyle="rgb(0,0,255)";
ctx.moveTo(10, 10);
ctx.lineTo(260, 10);
ctx.moveTo(10, 15);
ctx.lineTo(260, 15);
ctx.stroke();
```

Zunächst wird `beginPath()` aufgerufen, um einen neuen Pfad zu zeichnen. Da der Ursprung der ersten Linie der Punkt `(10,10)` sein soll, wird `moveTo(10,10)` aufgerufen. Die Methode `lineTo()` zeichnet dann die erste Linie bis zum Punkt `(260, 10)`. Ein weiteres Mal wird `moveTo()` aufgerufen, um den Ursprung der zweiten Linie 5 Pixel unterhalb des ersten Ursprungs zu positionieren. Die zweite Linie wird gezeichnet, und letztlich wird mittels `stroke()` gezeichnet. Es sind nun zwei nicht verbundene Linien entstanden.

Weitere Formen, wie beispielsweise ein Dreieck, müssen ebenfalls per Pfad-API gezeichnet werden. Um ein Dreieck zu zeichnen, wird per `moveTo()` zunächst der Ursprung der ersten Linie angegeben. Es folgen zwei konkret per `lineTo()` gezeichnete Linien. Die dritte Kante des Dreiecks muss nicht gezeichnet werden, da `fill()` automatisch den Pfad schließt, also `closePath()` aufruft, dadurch den aktuellen Endpunkt mit dem Beginn des Pfads verbindet und so das Dreieck schließt.

```
ctx.beginPath();
ctx.fillStyle="rgb(0,255,0)";
ctx.moveTo(10, 50);
ctx.lineTo(25, 20);
ctx.lineTo(40, 50);
ctx.fill();
```

Um kreisförmige Pfade zu zeichnen, stehen die Methoden `arc()` und `arcTo()` zur Verfügung, die wie folgt deklariert werden:

```
void arc(in float x, in float y, in float radius, in float startAngle,
   in float endAngle, in boolean anticlockwise);
void arcTo(in float x1, in float y1, in float x2, in float y2, in float
radius);
```

Schauen wir uns zunächst die Methode `arc()` an. Die ersten beiden Parameter, hier `x` und `y`, geben den Mittelpunkt an. Der dritte Parameter, `radius`, gibt den Kreisradius an. `startAngle` und `endAngle` bestimmen den Start- und Endwinkel des zu zeichnenden Kreises im Bogenmaß. Durch `anticlockwise` kann die Zeichenrichtung umgedreht werden.

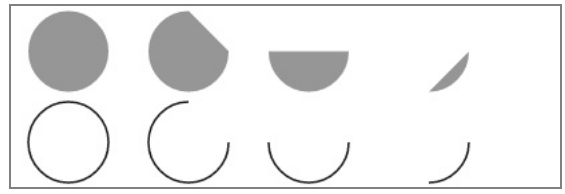
Da wir im Umgang mit Kreisen meistens in Grad denken, erstellen wir zunächst eine Funktion, die uns die Gradzahl ins Bogenmaß umrechnet:

```
function getRadians(deg)
{
   return (Math.PI/180)*deg;
}
```

Durch diese kleine Hilfsfunktion können wir dann recht einfach einen Kreis zeichnen. Das folgende Beispiel zeichnet vier kreisförmige Formen. Nur die erste stellt einen vollständigen Kreis dar. Wir ziehen pro Durchgang der Schleife 90 Grad ab und erhalten auf diese Weise unterschiedliche Formen:

```
for (var i = 0; i < 4; i++)
{
   ctx.beginPath();
   var x = 30 + (60*i);
   ctx.arc(x, 125, 20, getRadians(0), getRadians(360-(90*i)), false);
   ctx.fill();
}

for (var i = 0; i < 4; i++)
{
   ctx.beginPath();
   var x = 30 + (60*i);
   ctx.arc(x, 170, 20, getRadians(0), getRadians(360-(90*i)), false);
   ctx.stroke();
}
```

Das folgende Bild zeigt das Resultat sowohl für den Aufruf von `stroke()`, um den Umriss der Form zu zeichnen, als auch für den Aufruf von `fill()`, um die gezeichnete Form gefüllt darzustellen:



Um einen Dreiviertelkreis zu zeichnen, reicht also die `arc()`-Methode allein nicht aus. Das Problem ist, dass der Start- und Endpunkt des gezeichneten Pfads durch eine gerade Linie verbunden wird. Um den Dreiviertelkreis zu zeichnen, bedarf es zwei weiterer Linien:

```
ctx.beginPath();
ctx.arc(30, 215, 20, getRadians(0), getRadians(270), false);
ctx.lineTo(30, 215);
```

```
ctx.lineTo(50, 215);
ctx.stroke();

ctx.beginPath();
ctx.arc(90, 215, 20, getRadians(0), getRadians(270), false);
ctx.lineTo(90, 215);
ctx.lineTo(110, 215);
ctx.fill();
```

Das Ergebnis kann sich sehen lassen:

Die Methode `arcTo()` zeichnet einen kreisförmigen Abschnitt. Schauen wir uns die Deklaration der Methode zunächst nochmals an:

```
void arcTo(in float x1, in float y1, in float x2, in float y2, in float
radius);
```

Zum Punkt `(x1,y1)` zeichnet `arcTo()` zunächst eine gerade Linie, ausgehend vom aktuellen Ursprung. `(x2,y2)` gibt nun den Endpunkt des kreisförmigen Pfadabschnitts an und `radius` den zu benutzenden Radius (vom Mittelpunkt, also `x1,y1`). Ein Beispiel veranschaulicht das:

```
ctx.beginPath();
ctx.moveTo(100, 100);
ctx.lineTo(160, 100);
ctx.arcTo(180, 100, 180, 120, 20);
ctx.lineTo(180, 180);
ctx.arcTo(180, 200, 120, 200, 20);
ctx.lineTo(120, 200);
ctx.arcTo(80, 200, 80, 180, 20);
ctx.lineTo(80, 120);
ctx.arcTo(80, 100, 100, 100, 20);
ctx.fill();
```

Die Benutzung der `arcTo()`-Methode braucht sicherlich etwas Übung. Sie sollten nicht vergessen, dass die ersten beiden Parameter einen Punkt angeben, zu dem zunächst eine gerade Linie gezeichnet wird und der später als Mittelpunkt des Kreises fungiert. Die nächsten beiden Parameter geben den Endpunkt an, also den Punkt, zu dem ein Kreisabschnitt gezeichnet werden soll. Der letzte Parameter gibt den Radius des Kreises an.

Die Methoden `quadraticCurveTo()` und `bezierCurveTo()` werden meist für organische Formen verwendet. Sie sind wie folgt deklariert:

```
void quadraticCurveTo(in float cpx, in float cpy, in float x, in float y);

void bezierCurveTo(in float cp1x, in float cp1y, in float cp2x, in float
cp2y, in float x, in float y);
```

Der wichtigste Unterschied ist, dass `quadraticCurveTo` lediglich einen Kontrollpunkt benötigt, da die Funktion quadratisch und damit symmetrisch ist. `bezierCurveTo()` braucht zwei Kontrollpunkte und ist daher flexibler. Folgende Darstellung zeigt je ein Beispiel.

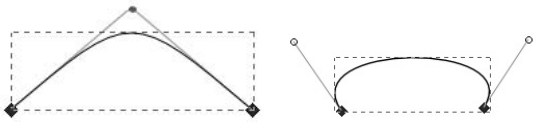
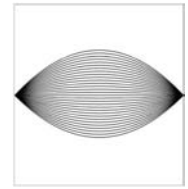


Bild 2.5: Links `quadraticCurveTo` und rechts `bezierCurveTo`.

Bei beiden Methoden ist zu beachten, dass außer den Kontrollpunkten lediglich der Endpunkt angegeben wird. Der Startpunkt ist der Endpunkt der letzten Zeichenoperation oder der Punkt, der mittels `moveTo()` angegeben wurde. Betrachten wir nun die `quadraticCurveTo()`-Methode:

```
for (var i=0; i<300; i+=10)
{
   ctx.beginPath();
   ctx.moveTo(0, 150);
   ctx.quadraticCurveTo(140, i, 280, 150);
   ctx.stroke();
}
```

Die `quadraticCurveTo()`-Methode wird 30 Mal aufgerufen. Durch `moveTo(0, 150)` wird bei jedem Durchgang der Startpunkt gleich gesetzt, ebenso ist der Endpunkt mit `(280, 150)` jeweils gleich angegeben. Einzig der Kontrollpunkt verändert sich, die `y`-Position verändert sich um je 10 Pixel. Das Resultat sieht wie folgt aus:



Die Methode `bezierCurveTo()`ist flexibler, da zwei Kontrollpunkte angegeben werden müssen. Im Methodenaufruf werden zunächst die beiden Kontrollpunkte (jeweils `x` und `y`) übergeben, dann der Endpunkt der Bézierkurve.

```
var i = 0;
var draw = function() {
   ctx.beginPath();
   ctx.moveTo(0, 150);
   ctx.bezierCurveTo(70, i, 210, 300-i, 280, 150);
   ctx.stroke();
   i+= 10;
   if (i < 300)
      setTimeout(draw, 500);
}
setTimeout(draw, 500);
```

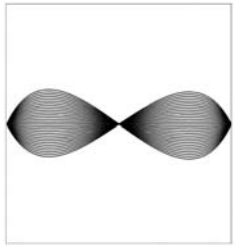


Bild 2.6: Die fertige Bézierkurve.

Das obige Codebeispiel demonstriert bereits eine einfache Animation. Die Funktion `draw()` wird zunächst durch `setTimeout()` nach 500 ms aufgerufen. Pro Durchgang wird die Variable `i` um 10 erhöht, wodurch die beiden Kontrollpunkte von unten nach oben und umgekehrt laufen.

An dieser Stelle sei erwähnt, dass sich das `canvas`-Element nicht wirklich zur Erstellung von Animationen eignet, zumindest nicht, wenn die API aus einem mobilen Browser heraus benutzt wird. Es gibt viele beeindruckende Beispiele für Desktop-Browser, allerdings lässt die Performance auf mobilen Geräten immer noch zu wünschen übrig. Es stellt zwar kein Problem dar, den Canvas alle 100 ms neu zu zeichnen, allerdings kommt dadurch keine flüssige Animation zustande.

Durch die Methode `clip()` kann ein zuvor gezeichneter Pfad in eine sogenannte Clipping Area, zu Deutsch Guckloch, umgewandelt werden. Die standardmäßige Zeichenfläche ist rechteckig und hat exakt die Maße des erstellten Canvas. Durch eine Clipping Area kann eine Form angegeben werden, die als Maske für alle nachfolgenden Zeichenoperationen dient. Alle Punkte, die sich außerhalb der Clipping Area befinden, werden nicht auf den Canvas gezeichnet. Das folgende Beispiel demonstriert das, indem zunächst durch die Methode `translate()` alle Zeichenoperationen in den Mittelpunkt des Canvas verschoben werden. (`translate()` wird später noch genauer vorgestellt.) Als Nächstes wird eine kreisförmige Clipping Area erstellt und im Anschluss ein Rechteck, das dank der Clipping Area leicht kreisförmige Seiten hat:

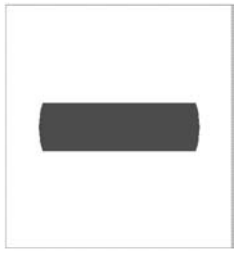


Bild 2.7: Rechteck mit abgerundeten Seiten.

```
//center of canvas
ctx.translate(140, 150);

//create clipping
ctx.beginPath();
ctx.arc(0, 0, 100, getRadians(0), getRadians(360), false);
ctx.clip();

//draw to see clipping
ctx.translate(-140,-150);
ctx.fillStyle="rgb(255,0,0)";
ctx.fillRect(10, 120, 260, 60);
```

2.3.2 Bilder

Mittels der `drawImage()`-Methode können vorhandene Bilder oder sogar andere `canvas`-Elemente in einen Canvas übernommen werden. In den folgenden Abschnitten

werden nur die Methoden vorgestellt, die mit vorhandenen Bildern arbeiten, sprich mit HTML-`img`-Elementen oder JavaScript-Image-Objekten.

Bevor wir Bilder in einen Canvas einfügen können, müssen wir sie zunächst laden. Da wir mittels JavaScript-API auf den Canvas zugreifen, bietet es sich an, ein JavaScript-Image-Objekt zu erstellen und dessen `load`-Event abzuwarten, da dadurch sichergestellt ist, dass das referenzierte Bild vollständig geladen ist.

```
var image = new Image();
image.addEventListener('load', function(evt) {
   ctx.drawImage(image, 10, 10);
});
image.src = 'http://flavor.de/html5/thumbs/calendar.png';
```

Bild 2.8: Das Kalender-Icon im Canvas.

Zunächst erstellen wir ein neues JavaScript-Image-Objekt. Im zweiten Schritt wird ein Event-Handler für das `load`-Event registriert. Sobald das Bild vollständig geladen ist, wird damit die Funktion `drawImage()` aufgerufen. Um nun das Laden des Bilds auszulösen, wird die `src`-Eigenschaft des Image-Objekts gesetzt, wodurch der Browser versucht, das per URL referenzierte Bild sofort zu laden.

Selbstverständlich können Sie auch alle bereits vorhandenen Bilder auf der aktuellen HTML-Seite referenzieren und innerhalb des Canvas benutzen. Die Methode `document.getElementsByTagName('img')` liefert Ihnen beispielsweise alle HTML-Image-Elemente der aktuellen Seite. Wenn die ID eines Bilds bekannt ist, können Sie das Bild auch per `document.getElementbById('meineId')` referenzieren.

Betrachten wir nun die `drawImage()`-Methoden des HTML5-Canvas genauer:

```
void drawImage(in HTMLImageElement image, in float dx, in float dy);
void drawImage(in HTMLImageElement image, in float dx, in float dy,
   in float dw, in float dh);
void drawImage(in HTMLImageElement image, in float sx, in float sy, in float
sw,
   in float sh, in float dx, in float dy, in float dw, in float dh);
```

Die einfachste Form der `drawImage()`-Methode benötigt lediglich drei Parameter, und zwar einmal eine Referenz auf das zu zeichnende Bild, also ein `HTMLImageElement` oder ein HTML-`canvas`-Element. Mit den Parametern `dx` und `dy` wird die Position im Canvas angegeben, ab der das Bild gezeichnet werden soll. Dabei wird die obere linke Kante des Bilds angegeben. Das Bild wird in diesem Fall mit der Originalhöhe und -breite ab dieser Position in den Ziel-Canvas übernommen. Das obige Beispiel demonstrierte die Anwendung bereits.

Die zweite Variante der `drawImage()`-Methode erlaubt die Skalierung des Bilds. Durch die zusätzlichen zwei Parameter `dw` und `dh` (steht für »destination width« und »destination height«) wird die Höhe und Breite angegeben, auf die das Bild skaliert werden soll.

Zur Demonstration zeichnen Sie innerhalb einer Schleife das gleiche `image`-Objekt viermal mit unterschiedlicher Breite und Höhe in den Canvas. Pro Durchgang wird das Bild an veränderter y-Position gezeichnet, sodass sich die vier Bilder nicht überlappen:

```
var image = new Image();
image.addEventListener('load', function(evt) {
   for (var i = 0; i<4; i++)
   {
      var len = 10+10*i;
      ctx.drawImage(image, (10 + len*i), 50, len, len); //scaling
   }
});
image.src = 'http://flavor.de/html5/thumbs/calendar.png';
```

Zu guter Letzt besprechen wir noch die dritte Variante der `drawImage()`-Methode, die insgesamt stolze neun Parameter erwartet. Durch diese Methode kann das Bild nicht nur skaliert werden, zusätzlich kann aus dem Ursprungsbild ein rechteckiger Teil sozusagen herausgeschnitten werden, der dann in den Ziel-Canvas gesetzt wird.

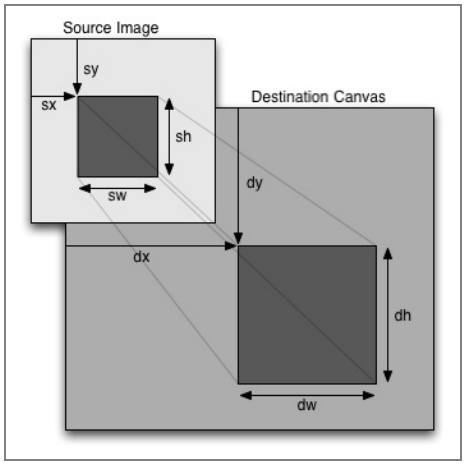


Bild 2.9: Illustration der `drawImage()`-Methode: *http://dev.w3.org/html5/2dcontext/#images*.

Der `drawImage()`-Aufruf für folgendes Beispiel schneidet aus dem Ursprungsbild ab Position `(0,0)`, also ab der oberen linken Ecke des Ursprungsbilds, ein 10 x 10 Pixel großes Quadrat aus. Dieses Quadrat wird an die Position `(10, 80)` in den Ziel-Canvas gesetzt, jedoch nicht als 10 x 10, sondern als 40 x 40 großes Quadrat.

```
... gleiches Bild ...
ctx.drawImage(image, 0, 0, 10, 10,   10, 80, 40, 40);
```

Bild 2.10: Ausgeschnittenes Quadrat aus dem Ursprungsbild.

Die praktischen Anwendungen der `drawImage()`-Methode sind vielfältig. So lassen sich beispielsweise Bilder innerhalb von zuvor platzierten Rahmen einbinden, oder mosaikähnliche Grafiken können mit vielen Einzelbildern auf einen Canvas gemalt werden.

2.3.3 Text

Die Canvas-API stellt ebenfalls je drei Methoden und Attribute zur Verfügung, um Text auf einen Canvas zu zeichnen. Durch die Attribute `font` (Schriftart) , `textAlign` (Textausrichtung) und `textBaseline` (Textgrundlinie) kann der zu zeichnende Text näher spezifiziert werden. Die Methoden `fillText()` und `strokeText()` arbeiten analog zu den bekannten Methoden `fill()` und `stroke()`, die Sie von der Pfad-API kennen. `fillText()` füllt den zu zeichnenden Text komplett aus, während `strokeText()` nur die Umrisse des Texts zeichnet. Die folgende Deklaration der Attribute und Methoden zeigt die Standardwerte fett an.

```
attribute DOMString font; // "10px sans-serif"
attribute DOMString textAlign; // "start", "end", "left", "right", "center"
attribute DOMString textBaseline; // "top", "hanging", "middle",
"alphabetic", "ideographic", "bottom"

void fillText(in DOMString text, in float x, in float y,
   in optional float maxWidth);
void strokeText(in DOMString text, in float x, in float y,
   in optional float maxWidth);
TextMetrics measureText(in DOMString text);
```

Die Methode `measureText()` gibt für den per Parameter übergebenen Text die Breite in Pixeln zurück. Zwar wird ein `TextMetrics`-Objekt zurückgegeben, jedoch hat das nur ein einziges Attribut, nämlich `width`.

Das Attribut `font` des 2-D-Kontext kann auf jeden beliebigen gültigen CSS-Font gesetzt werden. Um die Attribute `textAlign` und `textBaseline` mit all ihren Varianten zu verstehen, betrachten Sie am besten das folgende Beispiel. Der Reihe nach werden alle Variationen von `textBaseLine` und `textAlign` durchgespielt.

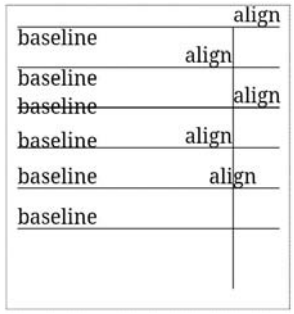


Bild 2.11: Beispiel zur Textausrichtung.

Das Wort *baseline* wird mittels `fillText()` auf den Canvas gezeichnet. Je Durchgang wird die y-Position verändert, ebenso wird eine horizontale Linie ab der Position gezeichnet, die auch für die `fillText()`-Methode verwendet wird.

Von oben nach unten:

- `top`
- `hanging`
- `middle`
- `alphabetic` (Standard)
- `ideographic`
- `bottom`

Das Wort *align* wird zunächst per `measureText()` abgemessen. Dann wird ab der Wortmitte eine vertikale Linie zum Ende des Canvas gezeichnet. Das verdeutlicht die Wirkung des `textAlign`-Attributs.

Von oben nach unten:

- `start` (Standard)
- `end`
- `left`
- `right`
- `center`

```
function text(){
   var textBaseline = ["top", "hanging", "middle", "alphabetic",
"ideographic", "bottom"];
   var y;
   ctx.font = "20px Georgia";
   for (pos in textBaseline)
   {
      ctx.textBaseline = textBaseline[pos];
      ctx.beginPath();
      y = (20 + (40*pos));
      ctx.lineStyle= "rgb(255,0,0)";
      ctx.moveTo(10, y);
      ctx.lineTo(270, y);
      ctx.stroke();
      ctx.fillText('baseline', 10, y);
```

```
    }

    var metrics = ctx.measureText('align');
    ctx.beginPath();
    var centerTextX = 200+(metrics.width/2)
    ctx.moveTo(centerTextX, 20);
    ctx.lineTo(centerTextX, 280)
    ctx.stroke();

    var textAlign = ["start", "end", "left", "right", "center"];
    for (pos in textAlign)
    {
       ctx.textAlign = textAlign[pos];
       y = (20 + (40*pos));
       ctx.fillText('align', centerTextX, y);
    }
}
```

2.3.4 Füllstil, Strichstil, Linienstil

Wir haben die Attribute `fillStyle`, `strokeStyle` und sämtliche Attribute zur Definition der Linien bereits in vorherigen Beispielen mehr oder weniger bewusst eingesetzt, da jedes Attribute einen Standardwert besitzt. `fillStyle` gibt an, wie eine zu füllende Fläche gefüllt werden soll. Soll es beispielsweise ein CSS-Farbwert sein (mit oder ohne Transparenz), ein Farbverlauf oder ein Muster? Ähnlich wie `fillStyle` gibt `strokeStyle` die Füllung der zu zeichnenden Linie an. Das ist sicherlich etwas ungewöhnlich, aber die Art der Linie selbst kontrollieren die Attribute `lineWidth`, `lineCap`, `lineJoin` und `miterLimit`. Auf all diese Attribute wird in diesem Abschnitt noch eingegangen.

Den Attributen `fillStyle` und `strokeStyle` kann entweder ein gültiger CSS-Farbwert (z. B. `#FF0000` oder `rgb(255,0,0)`), ein Verlauf oder ein Muster zugewiesen werden. Auf Verläufe und Muster gehen wir im nächsten Abschnitt ein, die Beispiele hier verwenden lediglich CSS-Farbwerte.

Fangen wir doch gleich mit `fillStyle` an. Das folgende Beispiel zeichnet insgesamt zehn Quadrate auf einen Canvas, jeweils mit unterschiedlichem CSS-Farbwert und an anderer Position:

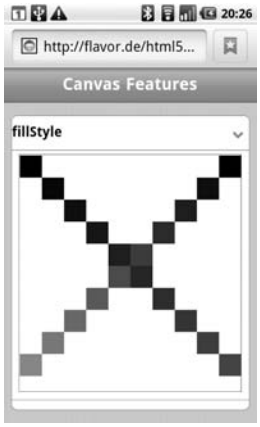


Bild 2.12: Auf einen Canvas werden zehn Quadrate gezeichnet.

```
//fillStyle can be CSS color value, a gradient object, or a pattern object
for (var i = 0; i < 10; i++){
   ctx.fillStyle = "rgb("+(Math.floor((255/10)*i))+",0, 0)";
   ctx.fillRect(i*28, i*28, 28, 28);

   ctx.fillStyle = "rgb(0,"+(Math.floor((255/10)*i))+", 0)";
   ctx.fillRect(252-(i*28), i*28, 28, 28);
}
```

`fillStyle` wird im obigen Beispiel je ein CSS-Farbwert zugewiesen. Seit CSS 3 gibt es die Möglichkeit, die Transparenz der Farbe durch `rgba()` anzugeben. Im Folgenden wird die Farbe Grün mit 50 % Transparenz definiert und `fillStyle` zugewiesen:

```
ctx.fillStyle = 'rgba(0,255,0,0.5)';
```

Zwar kann die Transparenz auch über das `globalAlpha`-Attribut zugewiesen werden, allerdings erreicht die Benutzung von `rgba` eine kompaktere Schreibweise.

Analog zu `fillStyle` wird auch `strokeStyle` angegeben Folgendes Beispiel zeichnet 150 horizontale Linien auf den Canvas mit je 1 Pixel Abstand. Mit jedem Durchlauf wird der Rotwert verändert, sodass die Linien langsam von Schwarz nach Rot übergehen. Zum Schluss werden noch zwei Rechtecke über diese Linien gezeichnet, die aus Quadraten unterschiedlicher Transparenz und Farbe bestehen.

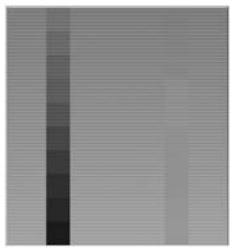


Bild 2.13: Auf einen Canvas werden 150 Linien mit je 1 Pixel Abstand gezeichnet.

```
for (var i = 0; i <= 300; i+=2){
   ctx.strokeStyle = "rgb("+Math.floor((255/300)*i)+",0, 0)";
   ctx.beginPath();
```

```
   ctx.moveTo(0, i);
   ctx.lineTo(280, i);
   ctx.stroke();
}

//draw partially transparent rectangles over this
for (var i = 0; i<10; i++)
{
   ctx.globalAlpha = 0.1 + 0.1 * i;
   ctx.fillStyle = "rgb(0,0,255)";
   ctx.fillRect(50, 300/10*i, 30, 30);
}

//use rgba instead of globalAlpha
ctx.globalAlpha = 1;
for (var i = 0; i<10; i++)
{
   ctx.fillStyle = "rgba(0,255,0,"+(0.1 + 0.1*i)+")";
   ctx.fillRect(200, 300/10*i, 30, 30);
}
```

Zur Kontrolle der Linien stellt die Canvas-API gleich mehrere Attribute zur Verfügung. Diese sind `lineWidth` (Breite der Linie), `lineCap` (definiert das Aussehen der Linienenden), `lineJoin` (definiert, wie zwei Linien verbunden werden) und `miterLimit` (siehe Beispiel).

`lineWidth` ist einfach zu verstehen, da sicherlich jeder schon einmal mit unterschiedlichen Linienbreiten in Malprogrammen experimentiert hat:

Bild 2.14: Unterschiedliche Linienbreiten mit `lineWidth`.

```
//lineWidth
for (var i = 0; i<10; i++) {
   ctx.lineWidth = i;
   ctx.beginPath();
   ctx.moveTo(280/10*i, 0);
   ctx.lineTo(280/10*i, 20);
   ctx.stroke();
}
```

Dem Attribut `lineCap` können drei Werte zugewiesen werden, `butt`, `round` und `square`. Unser Beispiel demonstriert die Auswirkungen auf die Linienenden. Zunächst wird ein Quadrat gezeichnet, dann werden drei Linien gleicher Länge innerhalb des Quadrats gezeichnet:

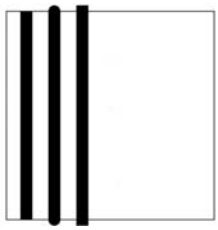


Bild 2.15: Auswirkung der zugewiesenen Werte auf die Linien.

```
//lineCap
var lineCap = ['butt','round','square'];
var lineStyle = "rgb(0,0,0)";
ctx.lineWidth = 1;
ctx.strokeRect(10, 30, 260, 260);

ctx.lineWidth = 15;
var pos;
for (pos in lineCap)
{
   ctx.lineCap = lineCap[pos];
   ctx.beginPath();
   ctx.moveTo((35 + 35*pos), 30);
   ctx.lineTo((35 + 35*pos), 290);
   ctx.stroke();

}
```

Das Attribut `lineJoin` definiert, wie zwei aneinandergrenzende Linien verbunden werden sollen. Es gibt die Varianten `round`, `bevel` und `miter`.

Bild 2.16: Definition angrenzender Linien mit `lineJoin`.

```
var lineJoin = ['round','bevel','miter'];
var startY;
for (i = 0; i<3; i++)
{
   ctx.beginPath();
   ctx.lineJoin = lineJoin[i];
   startY = 90 + (i*30);
   ctx.moveTo(150, startY);
   ctx.lineTo(170, startY-20);
   ctx.lineTo(190, startY);
   ctx.stroke();
}
```

Zuletzt demonstrieren wir die Auswirkung eines veränderten `miterLimit`. Wir zeichnen Zickzacklinien und setzen pro Durchgang das `miterLimit`. Während der ersten drei Durchgänge ist `miterLimit` nicht hoch genug gesetzt, wodurch die aufeinander zulaufenden Linien hart abgebrochen werden. Die letzten beiden Verbindungen sind jedoch spitz, da hier `miterLimit` auf einen Wert gesetzt wurde, der diese weit über die Linien hinausgehende Verbindung zulässt.

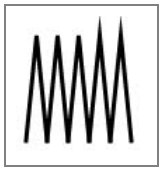


Bild 2.17: Zickzacklinien mit `miterLimit`.

```
//zigzag for miterLimit
ctx.lineWidth = 2;
ctx.lineJoin = 'miter';
var startX;
for (i = 0; i<5; i++)
{
   ctx.beginPath();
   ctx.miterLimit = i*5;
   startX = (150+(i*10));
   ctx.moveTo(startX, 250);
   ctx.lineTo(startX+5, 200);
   ctx.lineTo(startX+10, 250);
   ctx.stroke();
}
```

Die folgende Tabelle fasst die wichtigsten Fakten zu den Attributen rund um Füllstil, Strichstil und die diversen Linienstile zusammen.

Attribut	*Standard*	*Mögliche Werte*
`fillStyle`	`#000000`	Ein CSS-Farbwert, Verlauf (`CanvasGradient`) oder Muster (`CanvasPattern`).
`strokeStyle`	`#000000`	Ein CSS-Farbwert, Verlauf (`CanvasGradient`) oder Muster (`CanvasPattern`).
`lineWidth`	`1`	Positiver Zahlenwert.
`lineCap`	`butt`	`butt`, `round` und `square`.
`lineJoin`	`miter`	`bevel`, `round` und `miter`.
`miterLimit`	`10`	Positiver Zahlenwert.

2.3.5 Verläufe, Muster, Schatten

Die Canvas-API erlaubt die Benutzung von Verläufen und Mustern für die nun bekannten Attribute `fillStyle` und `strokeStyle`. Um ein Verlauf oder ein Muster zu

verwenden, muss zunächst ein Objekt vom Typ `canvasGradient` oder `canvasPattern` erstellt und bearbeitet werden. Um einen neuen Verlauf zu erzeugen, können wir folgende Methoden nehmen:

```
canvasGradient createLinearGradient(in float x0, in float y0, in float x1,
   in float y1);
canvasGradient createRadialGradient(in float x0, in float y0, in float r0,
   in float x1, in float y1, in float r1);
```

`createLinearGradient` erwartet je zwei x-/y-Koordinaten, die als Start-und Endpunkt des zu erstellenden Verlaufs verstanden werden können. Stellen Sie sich vor, wie die beiden Punkte eine Linie im aktuellen Canvas definieren. An dieser Stelle wurden noch keinerlei Farbwerte für den Verlauf zugewiesen.

Um nun den Farbverlauf zu definieren, wird die Methode `addColorStop()` benutzt. Der erste Parameter kann ein Wert von 0 bis 1 sein und definiert prozentual, ab welcher Stelle im Verlauf der zweite Parameter, ein beliebiger CSS-Farbwert, zur Wirkung kommt.

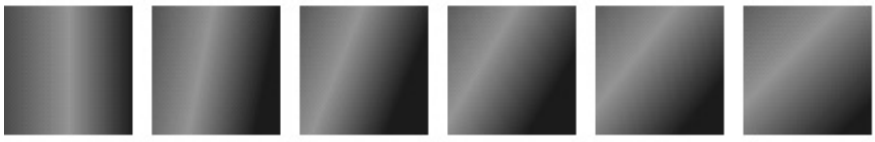


Bild 2.18: Definition eines Farbverlaufs mit `addColorStop()`.

```
//linearGradient
var linearGradient;
for (i = 0; i<6; i++)
{
   linearGradient = ctx.createLinearGradient(i*46, 0, (i*46)+40,
Math.floor((40/5)*i)); //absolute to canvas!!
   linearGradient.addColorStop(0, 'rgb(255,0,0)');
   linearGradient.addColorStop(0.5, 'rgb(0,255,0)');
   linearGradient.addColorStop(1, 'rgb(0,0,255)');
   ctx.fillStyle = linearGradient;
   ctx.fillRect(i*46,10,40,40);
}
```

Recht ähnlich wird die Methode `createRadialGradient()` benutzt, um kreisförmige Verläufe zu erstellen. Die Parameter der Methode geben wie bei `createLinearGradient()` nicht die zu verwendenden Farben an, sondern definieren zwei Kreise, die wiederum als Start- und Endkreis bezeichnet werden können. Betrachten wir einen solchen Methodenaufruf genauer:

```
var radialGradient = ctx.createRadialGradient(140,150,0,140,150,50);
```

Die drei Parameter `140,150,0` geben dabei den Startkreis des Verlaufs an. Am Punkt `(140, 150)` wird der Startkreis mit 0 Pixeln Radius definiert. Es folgt die Definition des

Endkreises mit gleichem Mittelpunkt `(140, 150)`, jedoch 50 Pixeln Radius. Die eigentlichen Farbwerte werden nun wieder per `addColorStop()` zugewiesen:

```
radialGradient.addColorStop(0, 'rgb(255,0,0)');
radialGradient.addColorStop(0.5, 'rgb(0,255,0)');
radialGradient.addColorStop(1, 'rgb(0,0,255)');
```

Das folgende Beispiel erzeugt einen kreisförmigen Verlauf mit 50 Pixeln Radius um den Punkt `(140, 150)`:

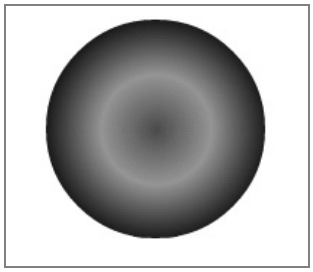


Bild 2.19: Beispiel eines kreisförmigen Verlaufs.

```
//radialGradient
var radialGradient;
radialGradient = ctx.createRadialGradient(140,150,0,140,150,50);
radialGradient.addColorStop(0, 'rgb(255,0,0)');
radialGradient.addColorStop(0.5, 'rgb(0,255,0)');
radialGradient.addColorStop(1, 'rgb(0,0,255)');

ctx.fillStyle = radialGradient;
ctx.beginPath();
ctx.arc(140, 150, 50, getRadians(0), getRadians(360), false);
ctx.fill();
```

Um Füllmuster zu erstellen, steht uns die Methode `createPattern()` zur Verfügung.

```
CanvasPattern createPattern(in HTMLImageElement image,
   in optional DOMString repetition);
```

Wie auch die `drawImage()`-Methoden kann diese Methode sowohl mit HTML-Image-Elementen als auch mit HTML-`canvas`-Elementen aufgerufen werden. Der zweite Parameter, `repetition`, kann die Werte `repeat`, `repeat-x`, `repeat-y` und `no-repeat` haben. Der Standardwert ist `repeat` und besagt, dass sich das angegebene Bild beliebig horizontal und vertikal wiederholen kann, um die Fläche mit dem Muster zu füllen.

Das folgende Beispiel füllt unseren 280 x 300 Pixel großen Canvas mit dem Kalender-Icon:

Bild 2.20: Beispiel auf der Android-Plattform.

```
var img = new Image();
img.src = 'http://flavor.de/html5/thumbs/calendar.png';
img.onload = function() {
   var pat = ctx.createPattern(img,'repeat');
   ctx.fillStyle = pat;
   ctx.fillRect(0,0,280,300);
}
```

Sämtliche Elemente, die auf den Canvas gezeichnet werden, können automatisch einen durch die Attribute `shadowOffsetX`, `shadowOffsetY`, `shadowBlur` und `shadowColor` definierten Schatten erhalten.

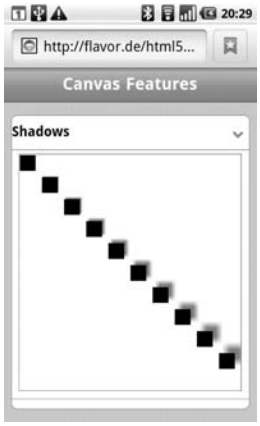


Bild 2.21: Beispiel definierter Schatten.

```
for (i = 0; i<10; i++)
{
   ctx.shadowOffsetX = i;
   ctx.shadowOffsetY = i;
   ctx.shadowBlur = i;
   ctx.shadowColor = "rgba(0, 0, 0, 0.5)";
   var x, y;
```

```
    x = y = (280/10)*i;
    ctx.fillRect(x, y, 20, 20);
}
```

2.3.6 Transformationen

Zu den von der Canvas-API unterstützten Transformationen zählen Skalierung, Rotation und Translation. Bevor wir diese Transformationen jedoch anhand kurzer Beispiele vorstellen, sollten Sie unbedingt mit den Methoden `save()` und `restore()` vertraut sein.

`canvas.save()` speichert den aktuellen Zustand des Canvas auf den Stack, `canvas.restore()` stellt den zuletzt gesicherten Zustand wieder her. Zum Zustand des Canvas gehören dabei:

- sämtliche Transformationen, die aktuell angewendet werden,
- sämtliche Attribute des Canvas, also `strokeStyle`, `fillStyle`, `lineWidth` etc., sowie
- der aktuelle Clipping Path.

`save()` und `restore()` erlauben es Ihnen, den aktuellen Zustand zu speichern, eine Transformation und Zeichenoperationen anzuwenden und dann wieder zu dem alten Zustand zurückzukehren. Je komplexer Ihre Zeichnungen werden, desto mehr werden Sie `save()` und `restore()` schätzen, da diese Methoden Ihnen ermöglichen, den Überblick zu behalten. Sie werden die Funktion der Methoden neben dieser theoretischen Vorstellung anhand von Beispielen auch praktisch kennenlernen.

Neben den Transformationen `scale`, `rotate` und `translate` erlaubt die Methode `transform()` die direkte Veränderung der Transformationsmatrix. Diese Transformationen werden über die gleichnamigen Methoden angesprochen:

```
void scale(in float x, in float y);
void rotate(in float angle);
void translate(in float x, in float y);
```

Die `translate()`-Methode wird dazu verwendet, den Ursprung des kompletten Canvas an eine andere Position zu verschieben. Wenn sich der Punkt `(0, 0)` normalerweise an der oberen linken Ecke des Canvas befindet, befindet er sich nach angewandter `translate(100, 100)` an Position `(100, 100)`. Alle nachfolgenden Zeichenoperationen werden also automatisch von diesem neuen Ursprung aus gezeichnet. Das folgende Beispiel demonstriert das:

```
//translate the drawing context
ctx.save();
ctx.translate(100, 100);
ctx.fillText('translated 100,100', 0, 0);
ctx.restore();
```

Zunächst wird mittels `save()` der aktuelle Zustand gespeichert. Für eine einzelne Transformation mit Zeichenoperation wäre das freilich nicht notwendig, es dient hier jedoch der Demonstration. Im nächsten Schritt wird dann der Ursprung auf den Punkt `(100, 100)` gelegt. Der nun mittels `fillText()` gezeichnete Text wird zwar an Posi-

tion `(0, 0)` gezeichnet, erscheint jedoch auf dem transformierten Canvas ab Position `(100, 100)`.

Stellen Sie sich vor, Sie haben eine JavaScript-Funktion geschrieben, die eine relativ komplexe Zeichenoperation beinhaltet. Um exakt gleiche Zeichenoperationen an zwei Stellen im Canvas anzuwenden, ist es sinnvoll, vor jedem Aufruf dieser Funktion den aktuellen Zustand zu speichern, die gewünschte Transformation (`translate`) anzuwenden, um den Ursprung zu verschieben, die eigentliche Zeichenoperation dann anzuwenden, den Zustand wiederherzustellen und dann die nächste Transformation und Zeichenoperation anzuwenden.

Bild 2.22: Beispiel zu den Canvas-Transformationen.

Die `rotate`-Transformation dreht den kompletten Canvas und wird mit Werten im Bogenmaß aufgerufen. Aus diesem Grund zeigt folgendes Beispiel, wie zunächst unter Anwendung der Hilfsfunktion der Wert ins Bogenmaß umgewandelt wird:

```
//rotate
ctx.save();
ctx.translate(140, 150);
ctx.rotate(getRadians(90));
ctx.fillText('rotated 90deg', 0, 0);
ctx.restore();
```

Es werden dann zwei Transformationen angewandt: Die erste Transformation, `translate()`, bewegt den Ursprung an die Position `(140, 150)`. Im nächsten Schritt wird der Canvas dann per `rotate`-Transformation um 90 Grad gedreht. Anhand der Grafik können Sie erkennen, dass die Zeichenoperation an Position `(0, 0)` von `translate()` und `rotate()` transformiert wird.

Zuletzt betrachten wir die Transformation `scale`, die den Canvas skaliert. Die x-/y-Skalierung kann mittels getrennter Parameter angegeben werden, somit können auch nicht proportionale Skalierungen erreicht werden. Die Zahl 1 steht in diesem Fall für 100 %, also die aktuelle Skalierung des Canvas. Ein Wert unter 1, also beispielsweise 0,5, würde nachträglich gezeichnete Objekte um 50 % verkleinert darstellen, ein Wert über 1 vergrößert alle Zeichenoperationen. Folgendes Beispiel skaliert zunächst auf 200 % und wendet dann eine Textzeichenoperation an, die vergrößert wird.

```
//scale
ctx.save();
ctx.scale(2,2); //1 is 100%
ctx.fillText('scaled 2x', 75,100);
ctx.restore();
```

2.3.7 Anordnung

Bislang wurden neue Zeichenoperationen immer auf den Canvas gezeichnet, d. h., die neue Zeichenoperation zeichnet über die bestehenden. Die bisher verwendete `globalCompositeOperation` war `source-over`, also die Standardeinstellung. Insgesamt gibt es jedoch zwölf mögliche Werte für die `globalCompositeOperation`, die in folgender Tabelle kurz vorgestellt werden.

Der aktuelle Stand der Dinge
Ein Test auf aktuellen mobilen Browsern zeigt, dass viele der zwölf Möglichkeiten entweder noch nicht oder scheinbar fehlerhaft implementiert sind. Dennoch seien diese Möglichkeiten hier kurz vorgestellt, da das Erreichen einer vollständigen und korrekten Implementierung im WebKit in den nächsten Monaten als wahrscheinlich gilt.

Folgende Grafiken wurden in OpenOffice gezeichnet, da die Implementierung derzeit zu lückenhaft ist. Erstes Zeichenobjekt in allen Grafiken ist das blaue Quadrat, die zweite Zeichenoperation zeichnet den roten Kreis.

Zeichenoperationen	
	`source-over` (Standard) Neue Formen werden über den aktuellen Inhalt des Canvas gezeichnet.
	`destination-over` Neue Formen werden hinter den bestehenden Formen gezeichnet. Neue Formen innerhalb der Fläche bestehender Formen sind daher nicht oder nur teilweise sichtbar.
	`source-in` Die neue Form wird nur dort gezeichnet, wo sich neue und alte Form überlappen. Die restlichen Bereiche werden komplett transparent.
	`destination-in` Der bestehende Inhalt des Canvas wird für alle Teile übernommen, die sich mit der neuen Form überlappen. Die restlichen Bereiche werden komplett transparent.
	`source-out` Die neue Form wird dort gezeichnet, wo sie sich mit dem bestehenden Inhalt des Canvas nicht überlappt.

Zeichenoperationen	
	`destination-out` Der bestehende Inhalt bleibt dort erhalten, wo die neue Form nicht gezeichnet wird.
	`source-atop` Die neue Form wird nur dort gezeichnet, wo bestehender Inhalt bereits vorhanden ist.
	`destination-atop` Der bestehende Inhalt wird dort erhalten, wo er sich mit der neuen Form überlappt. Die neue Form wird hinter den bestehenden Inhalt gezeichnet.
	`lighter` Die überlappenden Bereiche zwischen altem Inhalt und neuer Form werden heller gezeichnet. Der genaue Farbwert wird durch Addition der überlappenden Pixelfarbwerte bestimmt.
	`darker` Die überlappenden Bereiche zwischen altem Inhalt und neuer Form werden dunkler gezeichnet. Der genaue Farbwert wird durch Subtraktion der überlappenden Pixelfarbwerte bestimmt.
	`xor` Überlappende Bereiche werden transparent.
	`copy` Zeichnet nur die neue Form und löscht den restlichen Inhalt.

2.3.8 Diagramme mit Flot

Sie haben in den letzten Kapiteln die Canvas-API grundlegend kennengelernt. Sicherlich sind Sie vom Umfang der API erschlagen, obwohl sich damit fast jede zeichnerische Meisterleistung vollbringen lässt. Allerdings müssen Sie das Rad nicht immer neu erfinden. Es gibt bereits zahlreiche JavaScript-Toolkits, die auf der Canvas-API aufsetzen, um beispielsweise Diagramme zu zeichnen. Während Sie durch die Benutzung einer höheren API natürlich automatisch an Freiheit verlieren, bietet sich deren Nutzung oftmals für spezifische Gebrauchsfälle an. Die Zeitersparnis kann enorm sein, da Sie lediglich die Funktionalität in der erwarteten Form angeben und die API die passenden Canvas-Operationen ausführt.

Dieser Teil stellt die Flot JavaScript Plotting Libary (*http://code.google.com/p/flot/*) vor, mit der sich beeindruckende Diagramme in wenigen Zeilen JavaScript-Code erstellen lassen. Das folgende Beispiel macht dabei nicht nur vom Flot-Toolkit Gebrauch, sondern auch von jQuery, einem beliebten JavaScript-Framework. Wir benutzen jQuery als Basis und spezifischer das jQuery-Flot-Plug-in, um folgendes Diagramm zu zeichnen:

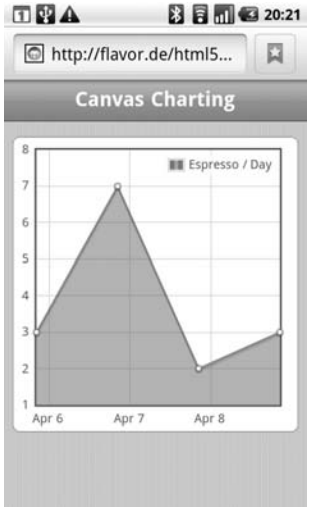


Bild 2.23: Espresso-Verbrauch pro Tag: einfaches Diagramm, mit Flot erstellt.

Zunächst müssen Sie sowohl jQuery als auch das jQuery-Flot-Plug-in in Form von externen JavaScript-Dateien in Ihre HTML-Seite einbauen. Nach dem Download (Sie finden jQuery unter *http://jquery.com* und das jQuery-Flot-Plug-in unter *http://plugins.jquery.com/project/flot*) müssen Sie beide Dateien wie folgt in den `head`-Bereich der HTML-Seite einbauen:

```
<html>
    <head>
        <title>Flot-Demo</title>
      <script language="javascript" src="<PFAD_JQUERY>/jquery-
1.4.2.min.js"></script>
      <script language="javascript"
src="<PFAD_FLOT_PLUGIN>/jquery.flot.js"></script>
      ...
```

Im `body` der HTML-Seite muss nun noch ein `div`-Element erstellt werden, das später als Container für das Diagramm dient. Beachten Sie auch, dass die CSS-Größenangaben dieses Elements die spätere Größe des Diagramms vorgeben:

```
<div id="chart" style="width:280px;height:300px"></div>
```

In einem weiteren Inline-JavaScript wird nun das eigentliche Diagramm erstellt. Schauen Sie sich das Beispiel kurz durch, bevor wir es in den folgenden Absätzen besprechen:

```
<!--
  Canvas-Diagramm mit iWebKit 5 und jQuery
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
<html>
<head>
        <title>Canvas-Diagramm</title>
```

```
      <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
      <meta name="layout" content="webkit" />
      <link href="style.css" rel="stylesheet" media="screen" type="text/css"
/>

        <!-- jQuery und jQueryFlot lokal einbinden -->
        <script language="javascript" src="js/jquery-1.4.2.js"
type="text/javascript" ></script>
        <script language="javascript" src="js/jquery.flot.js"
type="text/javascript" ></script>

      <script language="javascript">
      function init()
      {
         var now = new Date().getTime();
         var oneDay = 1000*60*60*24;

         //UTC offset at the time of writing (based in Sunnyvale, CA, April
2010): -7 hrs.
         //this should be normalized on the server side
         now -= 1000*60*60*7;

         var data = [
            { label: "Espresso / Day", data: [[now, 3], [now+oneDay, 7],
[now+(oneDay*2), 2], [now+(oneDay*3), 3] ] }
         ];

         var options = {
            series: {
                lines: {
                  show: true, fill: true
                },
               points: { show:true }
            },
            colors: ["rgba(255,0,0,0.5)"],
            xaxis: {
               mode: "time",
               minTickSize: [1, "day"]
            }
           };

         $.plot($("#chart"), data, options );

      }
      document.addEventListener("DOMContentLoaded", init, false);
      </script>
    </head>
    <body>
      <div id="topbar">
```

```
        <div id="title">Canvas Charting</div>
      </div>
      <div id="content">
        <ul class="pageitem">
          <li class="textbox">
            <div id="chart" style="width:280px;height:300px">
            </div>
          </li>
        </ul>
      </div>
    </body>
</html>
```

Zuerst warten Sie mittels `DOMContentLoaded`-Event, bis sämtliche Elemente der HTML-Seite ansprechbar sind, und starten dann die Funktion `init()`. Der grundlegende Aufruf von Flot geschieht in einer einzelnen Zeile.

```
$.plot($("#chart"), data, options );
```

Per jQuery wird zunächst das im `<body>` der HTML-Seite befindliche `div`-Element (mit der ID `chart`) referenziert und zusammen mit einem `data`- und einem `options`-Objekt an die Methode `plot()` des Flot-Plug-ins übergeben. Der Code über diesem Aufruf erstellt lediglich die Beispieldaten und Optionen.

Unsere Beispieldaten bestehen aus einer zeitbasierten Datenserie. Wir erstellen beispielhaft eine Datenserie für den Espresso-Konsum pro Tag. Das Array `data` enthält die Datenserien in Form von JavaScript-Objekten. Laut Flot-Konvention kann das Label (für die Legende) mit dem Attribut `label` angegeben werden. Die Daten der Serie werden im Attribut `data` als Array von Arrays angeführt. Pro Element werden die x- und y-Koordinaten des späteren Diagramms angegeben. Da wir ein zeitbasiertes Diagramm wünschen, müssen die Daten in Form von Timestamps angegeben werden.

```
var data = [
   { label: "Espresso / Day",
      data: [
            [now, 3],
            [now+oneDay, 7],
            [now+(oneDay*2), 2],
            [now+(oneDay*3), 3]
         ]
   }
];
```

Die Optionen werden ebenfalls als JavaScript-Objekte definiert und als Parameter der Plot-Methode übergeben. Auf Details soll hier nicht eingegangen werden, die Bedeutung der einzelnen Optionen ist doch recht selbsterklärend. Die vollständige Beschreibung der Flot-API finden Sie unter *http://bit.ly/bEXQYn*.

```
var options = {
      series: {
          lines: {
```

```
            show: true, fill: true
          },
          points: { show:true }
      },
      colors: ["rgba(255,0,0,0.5)"],
      xaxis: {
          mode: "time",
          minTickSize: [1, "day"]
      }
    };
```

Die Beispielseite bietet einen Überblick über alle hier im Kapitel kennengelernten Canvas-Funktionen. Die gepunkteten Linien definieren den Canvas-Bereich, in den mithilfe der verschiedenen Funktionen gezeichnet wird. Die Auswahl geschieht über ein Drop-down-Menü, das wie gewohnt unter iOS 4 als scrollbare Liste angezeigt wird.

Download

http://www.buch.cd

Hier finden Sie ein umfassendes Beispiel, in dem alle hier gezeigten canvas-Funktionen in einer mobilen Web-App gezeigt werden.

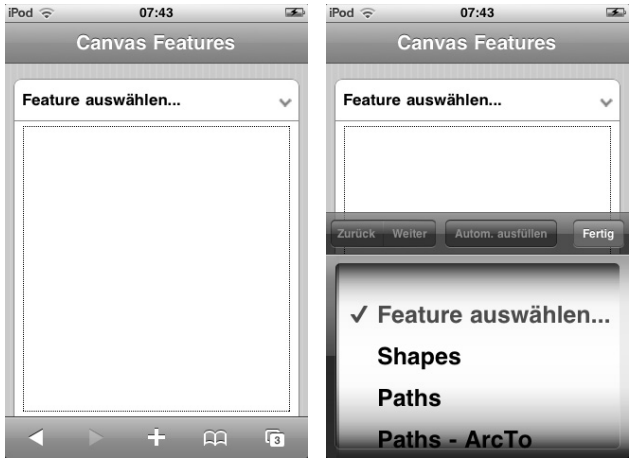


Bild 2.24: Die gepunkteten Linien definieren den Canvas-Bereich. Die Auswahl erfolgt über ein Drop-down-Menü.

Das Beispiel lässt sich besonders gut zum Testen auf verschiedenen Systemen und mobilen Browsern einsetzen. Während diese Funktion auf vielen Handsets und Plattformen funktioniert, sind die Composite-Operationen nicht überall so umfassend wie unter iOS 4 implementiert, obwohl auch hier einige Optionen noch nicht vollständig unterstützt werden.

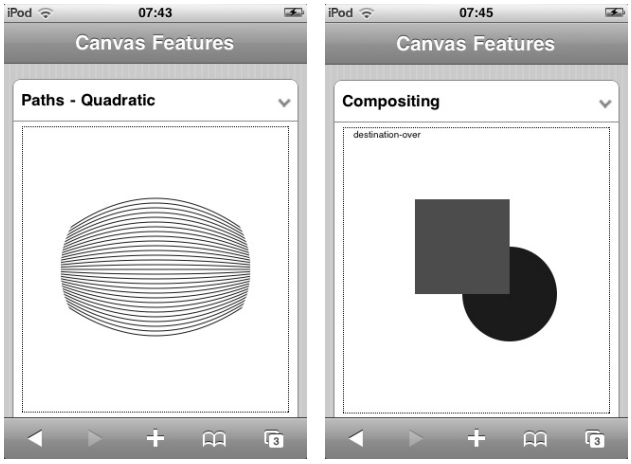


Bild 2.25: Nicht überall sind die Composite-Operationen so umfassend implementiert wie unter iOS 4.

2.4 Audio und Video

Erinnern Sie sich an die Zeiten, als es noch kein YouTube gab? Wann immer man im Internet ein Video sehen wollte, klickte man auf einen Link, und ein externes Programm, beispielsweise der Windows Media Player, wurde geöffnet, um das Video oder auch die Audiodatei wiederzugeben. Flash hieß Anfang dieses Jahrtausends das Zauberwort und hat die Internetvideolandschaft grundlegend verändert. Mit HTML5 steht die nächste Videorevolution bereit, die bereits damit begonnen hat, die Abhängigkeit von Flash zu beseitigen und dem Browser das Abspielen von Videos ganz zu überlassen.

Auch wenn wir uns in diesem Buch meistens mit HTML5-Techniken beschäftigen, die mobilspezifisch, also für das mobile Internet und einen kleinen Bildschirm relevant sind, so ist es hier sinnvoll, auch über das iPad zu sprechen. Ein iPad-Benutzer wird auf seinem Tablet-Computer keine mobile Webseite erwarten, allerdings unterstützt das iPad kein Flash und kann somit »normale« Flash-Videos nicht wiedergeben. Der Medienrummel um das iPad und die fehlende Flash-Unterstützung hat Anfang 2010 viele der großen Webseiten dazu veranlasst, HTML5 für Video zu implementieren.

HTML5 wird derzeit nur dann verwendet, wenn der Benutzer die Webseite auch mit einem kompatiblen Gerät aufruft – aber man geht davon aus, dass ohne iPad die Verwendung von HTML5-Video noch nicht so weit wäre. Reihenweise purzelten im März 2010 – einen Monat vor dem iPad-Start in den USA – die Pressemitteilungen ins Haus, dass YouTube, Flickr, Vimeo, die New York Times und viele andere auf die neue HTML5-Technologie umgestellt hätten.

Flash-Video wird sicher nicht von heute auf morgen verschwinden, und mit der Android-Version 2.2 (Froyo) hat auch Flash auf mobilen Geräten wieder Einzug gehalten. Auch Microsofts mobiles Betriebssystem Windows Phone 7 soll laut Unternehmensangaben Flash unterstützen. Aber irgendwie passt es für kleine Bildschirme und mobile Geräte nicht: Wer auf den Android-Geräten die Wiedergabe von Flash-Videoinhalten ausprobiert hat, wird schnell feststellen, dass eine HTML5-Einbindung nicht nur stabiler, sondern einfach auch benutzerfreundlicher ist.

Der große Vorteil von Flash gegenüber HTML5 ist heutzutage noch, dass die verschiedenen Browser sich nicht auf ein einheitliches Video- und Audioformat einigen konnten. Flash ist im Prinzip ein Format: Flash. Das gibt es höchstens in verschiedenen Versionen, aber es ist ein relativ einheitliches Format. Im Gegensatz dazu versteht der Safari-Browser beispielsweise nur Videoformate, die im Opera-Browser nicht funktionieren. Anbieter müssen also ihre Audio- und Videoinhalte, sofern sie sie mittels HTML5 anbieten möchten, in verschiedenen Formaten bereithalten.

2.4.1 Audiowiedergabe

Da das `video`-Tag komplexer als das Tag zur Audiowiedergabe ist, fangen wir erst einmal mit der Tonwiedergabe `audio` an. Eigentlich ist die `audio`-Syntax ganz einfach:

```
<audio src="beispiel.oga" controls="controls">
 <!-- Das folgende HTML wird nur dann dargestellt, wenn das Audio-Tag nicht
unterstützt wird. -->
 <p>Ogg Vorbis Audio wird leider nicht unterstützt. Direkter Link zur Datei:
<a href="beispiel.oga">beispiel.oga</a>.</p>
</audio>
```

Im `audio`-Element wird auf eine Datei verwiesen, die dann im Browser oder dem Standard-Medienplayer des Telefons wiedergegeben wird. Auf die Audiodatei wird mittels des `src`-Attributs verwiesen. Innerhalb des `audio`-Tags können Sie weiteres HTML einfügen, das aber nur dann angezeigt wird, falls ein Browser (beispielsweise der Internet Explorer auf einem Windows Mobile-Gerät) kein HTML5-`audio`-Tag versteht.

Eine Audiodatei wird entweder direkt im mobilen Browser auf der Webseite oder im Standard-Medienplayer wiedergegeben, der auch zum Betrachten von Videos benutzt wird. Seit iOS 4 wird die Audiodatei direkt im Browserfenster abgespielt. Ein kleines Steuerelement wird dazu auf der Seite angezeigt. Bei älteren iPhone OS-Versionen wurde nach dem Antippen eines Buttons der Videoplayer gestartet und die Audiodatei abgespielt. Das `control`-Attribut, das die Einblendung der Standardsteuerelemente festlegt, wird auf einem iPhone oder einem iPod ignoriert, bei einem iPad beispielsweise aber beachtet.

Wie schon oben in der Einleitung angekündigt, gibt es leider keinen Standard, dessen Dateiformat von allen Browsern unterstützt wird. Hier ist eine Übersicht:

Browser	*Unterstützte Audioformate*				
iOS Safari (iPhone, iPod touch und iPad)	AAC (M4A)	MP3		WAV (unkomprimiert)	
Android	AAC (M4A)	MP3	MP4		OGG Vorbis
Safari 4+	AAC (M4A)	MP3	MP4	WAV	

Browser	*Unterstützte Audioformate*				
Chrome 3+		MP3	MP4		OGG Vorbis
Firefox 3.5+				WAV	OGG Vorbis
Opera 10.5+				WAV	OGG Vorbis
Internet Explorer 9	AAC (M4A)	MP3			

Sie müssen also, und das gilt auch für das später vorgestellte `video`-Element, mehrere Formate anbieten, um wirklich sicherzustellen, dass der Audioinhalt auch auf allen Geräten und Browsern wiedergegeben werden kann. Dazu ist das `source`-Attribut genau richtig. Mittels dieses Attributs können Sie mehrere Formate innerhalb eines `audio`- oder `video`-Tags anbieten. Dabei ist es natürlich erforderlich, dass Sie Ihre Audio- oder Videoquelle auch auf Ihrem Server in verschiedenen Formaten anbieten.

Im HTML-Markup sieht das wie folgt aus:

```
<html>
   <head>
      <title>Einfacher Audio-Player</title>
   </head>
   <body>
      <audio controls="controls">
            <source src="audio/musik.oga">
            <source src="audio/musik.mp3">
      </audio>
  </body>
</html>
```

Wenn Sie diesen HTML-Code auf Ihren Server laden, sieht das Ergebnis unter iOS 4 so aus:

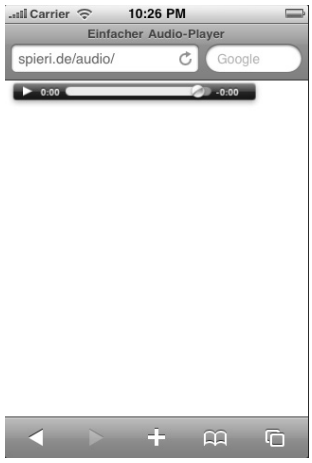


Bild 2.26: So wird das `audio`-Element auf einem iPhone angezeigt.

Über das kleine Playsymbol wird die Audiowiedergabe gestartet. Sollten Sie diesen kurzen Beispielcode zusammen mit einer AAC- oder MP3-Datei auf Ihren Server laden,

kann es passieren, dass Sie in einigen Browsern dennoch nichts hören, obwohl Sie augenscheinlich alles richtig gemacht haben. Audio- und Videodateien müssen vom Server mit dem richtigen MIME-Type ausgeliefert werden.

Der – und nur der – MIME-Type übermittelt dem Browser die Information darüber, welche Art von Daten geliefert werden. Falls Sie sich bereits mit der Konfiguration auskennen, können Sie den folgenden Abschnitt überspringen, falls nicht, empfehlen wir Ihnen diese Lektüre unbedingt.

2.4.2 MIME-Types

Wann immer ein Browser eine Webseite anfragt, erhält er noch weit vor den eigentlichen HTML-Daten einige Metainformationen zur Seite, die sogenannten Header. Die Informationen in diesen Headern sind in der Regel nicht sichtbar, helfen aber dem Browser, wichtige Entscheidungen zu treffen, beispielsweise wie später der empfangene Inhalt darzustellen ist. Der Header, der dem Browser übermittelt, ob es sich bei den empfangenen Daten um eine Textdatei (also eine HTML-Datei), um Video-, Bild- oder Audiodaten handelt, nennt sich Content-Type-Header und besitzt meist folgende Werte:

Content-Type	*Inhalt*
`text/html`	Normale HTML-Webseite und Textdokumente.
`image/jpeg`	Bilder im JPEG-Format.
`image/png`	Bilder und Grafiken im PNG-Format.
`audio/ogg`	Audiodateien im OGG Vorbis-Format.
`video/mp4`	Videodateien im MP4-Format.

Die Konfiguration der MIME-Types ist je nach verwendetem Webserversystem unterschiedlich. Sind Sie sich nicht sicher, wie die MIME-Types auf Ihrem Server gesetzt werden, hilft es meistens schon, die Audio- und Videodateien in separate Verzeichnisse zu speichern und diese Verzeichnisse mit einer sogenannten *.htaccess*-Datei zu versehen.

Die *.htaccess*-Datei ist eine Datei ohne Endung und wird in einem Verzeichnis mit den Mediendaten gespeichert. Schreiben Sie den folgenden Inhalt in die *.htaccess*-Datei, um die unterstützten Medienformate festzulegen und den Webserver zu veranlassen, bei Anfrage den richtigen Content-Header zu schicken:

```
AddType audio/mpeg .mp3
AddType audio/ogg .oga
AddType video/ogg .ogv
AddType video/mp4 .mp4
AddType video/webm .webm
```

Mit diesen fünf Zeilen sagen Sie dem Webserver, dass er, beispielsweise wenn eine OGG Vorbis-Datei mit der Endung *.oga* angefragt wird, den Content-Type `audio/ogg` mit zu versenden hat. Anhand dieser Information kann dann der Browser gleich sehen, ob er die Musikdatei unterstützt, und muss sie nicht erst selbst herunterladen, um das festzustellen.

2.4.3 Aufbau einer Videodatei

Wie schon bei den unterstützten Audioformaten gibt es auch bei den in den jeweiligen Browsern unterstützten Videoformaten keine Übereinstimmung. Man hatte sogar versucht, sich darauf zu einigen, dass mindestens das offene Videoformat OGG Theora unterstützt wird – leider ohne Erfolg.

Um Video bestmöglich und effektiv in HTML5 und speziell auf mobilen Geräten einsetzen zu können, müssen wir uns erst einmal mit dem Aufbau einer Videodatei beschäftigen. Wie schon bei einer Audiodatei wird in HTML5 einfach über ein simples Tag auf die Videodatei verwiesen:

```
<video src="film.ogv"></video>
```

Die Datei *film.ogv* ist aber, wie jede Videodatei, keine einzelne Videodatei, sondern eine Art ZIP-Archiv, das weitere Dateien enthält. Man nennt diese Dateien Videocontainer, da sie wie eine große Box eine Vielzahl von weiteren Daten in unterschiedlichen Formaten enthalten können.

Videocontainerformate

In der Regel befinden sich in einem Videocontainer neben der Sammlung von Metadaten zum Video (Titel, Herstellername) mehrere Spuren (Tracks): eine Bildspur und mindestens eine Tonspur. Tonspuren bestehen meistens aus einem der im `audio`-Abschnitt vorgestellten Formate.

Für HTML5 sind die folgenden Videocontainerformate interessant:

Format	*Dateiendung*	*Beschreibung*
OGG	*.ogv*	Offenes und ziemlich verbreitetes Videoformat, das mit dem OGG Vorbis-Audioformat verwandt ist. Das Videocontainerformat wird auch »OGG Theora« genannt. OGG Theora wird von einer Vielzahl von Browsern unterstützt, beispielsweise auch von Android.
MPEG-4	*.mp4* oder *.m4v*	Ein sehr bekanntes und vor allem von Apple verwendetes Medienformat. MPEG-4-Container können neben den Video- und Audiospuren auch weitere Informationen, wie beispielsweise Untertitel oder Standbilder, enthalten. Die enge Verbindung zu Apple kommt nicht von ungefähr: Das Format basiert auf dem alten QuickTime-Containerformat der Firma. MPEG-4 kann im Gegensatz zu OGG Theora auf den Apple-Geräten verwendet werden.
AVI	*.avi*	Dieses Containerformat dürfte vor allen Dingen Windows-Benutzern etwas sagen. Schließlich wurde es bereits 1992 von Microsoft als Teil der »Videos für Windows«-Technologie vorgestellt. Obwohl das Videocontainerformat alt ist und wenig weiterentwickelt wurde, findet es immer noch eine große Verbreitung und wird auch von diversen HTML5-kompatiblen Browsern unterstützt.

Format	*Dateiendung*	*Beschreibung*
WebM	*.webm*	Nachdem wir schon Dateiformate von Microsoft und Apple kennengelernt haben, darf natürlich eine Firma nicht fehlen: Google. Das WebM-Format wurde erst 2010 auf Googles Entwicklerkonferenz vorgestellt und soll insbesondere für HTML5 ein lizenzfreies und hoch qualitatives Videoformat anbieten. Ein WebM-Container enthält in der Regel eine Videospur im VP8-Videoformat und OGG Vorbis-Audiospuren. Als erster mobiler Browser wird Android das neue offene Videoformat ab Ende 2010 unterstützen.

2.4.4 Videocodecs

Sicher waren Sie auch schon einmal in der Situation, dass Sie eine Videodatei – oder wie wir jetzt wissen, eine Videocontainerdatei – hatten, die einfach kein Bild, sondern nur Ton lieferte. QuickTime auf dem Mac zeigt dann den wohlbekannten Dialog: *Zusätzliche Software ist zur Wiedergabe erforderlich.* Was dem Videoabspielprogramm in solchen Momenten fehlt, ist der richtige Videocodec.

Ein Videocodec ist ein Algorithmus, der dem Abspielprogramm sagt, wie die Videodaten im Videocontainer decodiert und dargestellt werden sollen. Würden Sie im Internet die Bilder wie in einem klassischen Film darstellen – also lauter abgespeicherte Einzelbilder, die mit 25 Bildern pro Sekunde angezeigt werden –, hätten Sie es mit riesigen Dateien und, gerade auf Mobiltelefonen, wahrscheinlich mit stundenlangen Ladezeiten zu tun.

Moderne Videocodecs codieren Bildabfolgen viel intelligenter. Anstatt jedes Bild einzeln abzuspeichern, werden nur die Unterschiede zwischen den Bildern über einen bestimmten Algorithmus beim Codieren gesichert. Damit das Abspielprogramm diesen Algorithmus versteht, muss es den Codec wissen, mit dem die Videodaten codiert wurden.

Ein Video besteht daher immer aus mindestens drei Teilen:

1. der in einem bestimmten Codec codierten Videodatei,
2. einer Audiodatei sowie
3. dem entsprechenden Videocodec.

Folgende Videocodecs sind für HTML5 relevant:

Für HTML5 relevante Videocodecs	
Theora	Theora ist wie auch das OGG-Containerformat lizenzfrei. Während die Unterstützung von Theora auf Desktop-Browsern relativ weit verbreitet ist (Firefox, Chrome, Opera), wird der Codec derzeit nicht von iOS und bislang auch noch nicht von Android unterstützt. Ein weiterer Nachteil des Codecs ist, dass er seit geraumer Zeit kaum noch aktiv weiterentwickelt wird.

Für HTML5 relevante Videocodecs	
VP8	Die Chancen stehen gut, dass sich VP8 als aktiv weiterentwickelter Nachfolger von Theora etabliert. Der Codec wurde von der gleichen Firma entwickelt, die auch den Theora-Vorgänger programmierte, und wurde schließlich 2010 von Google gekauft. Google hat die auf dem Codec beruhenden Patente freigegeben, und heutzutage ist VP8 als lizenzfreier Codec erhältlich und wird im Zuge des WebM-Containerformats von immer mehr Browserherstellern unterstützt. Im mobilen Bereich wird der Codec, da auch kräftig durch Google gefördert, als Erstes von Android wahrscheinlich Ende des Jahres 2010 unterstützt werden.
H.264	Die meisten großen Videowebnamen unterstützen heutzutage den H.264-Codec: YouTube, Flickr, Vimeo – und einer der Gründe, warum das so ist, nennt sich iPhone. Apple unterstützt auf all seinen mobilen Geräten (iPhone, iPod touch, iPad) den Codec, und auch sämtliche Android-Telefone verstehen den Algorithmus. H.264 wurde von der MPEG-Gruppe entwickelt und ist daher auch als MPEG-4 bekannt. H.264 ist prädestiniert für die Benutzung auf mobilen Geräten, da der Codec darauf zielt, einen einzigen Codec für Geräte mit geringer Rechenleistung und langsamen Datenverbindungen anzubieten (beispielsweise Mobiltelefone), gleichzeitig aber auch für Computer, die hoch qualitative Videos mit hoher Rechenleistung und hoher Datenbandbreite abspielen. Um diesen Spagat zu bewältigen, verwendet H.264 verschiedene Profile, mit denen festgelegt wird, welche Funktionen auf welchen Geräten zur Verfügung gestellt werden. Hoch qualitative Blu-ray-Videos werden mit einem High-Profile encodiert, bieten daher auch eine sehr gute Bildqualität, benötigen aber auch recht viel Rechenpower, um die Daten zu decodieren. Diese Rechenpower hat ein Mobiltelefon bislang noch nicht, deshalb unterstützt Apple für das iPhone und den iPod touch nur ein niedrigeres Profil, das sogenannte »Baseline-Profil«. Obwohl sich mit H.264 Videos zuverlässig auf mobilen Geräten und vielen Desktop-Webbrowsern wiedergeben lassen, ist der Codec nicht lizenzfrei. Anbieter von En- und Decodern für den H.264-Codec müssen ab 1. Januar 2011 Lizenzgebühren an den Lizenzverwalter MPEG LA zahlen. Eine Ausnahmeregelung, die kostenfreies Webvideo von Lizenzzahlungen ausnimmt, gilt noch bis 31. Dezember 2016. Bitte beachten Sie, dass diese Ausnahmeregelung nur für im Internet angebotenes H.264-Video gilt, das der Benutzer kostenfrei ansehen kann. Andere Webvertriebsformen, beispielsweise kostenpflichtige Abodienste, sind möglicherweise lizenzpflichtig. Nähere Informationen finden Sie direkt beim Lizenzverwalter: *www.mpegla.com* (»Request a License«).

Nach so vielen Informationen zu den verschiedenen Codecs ist es sicherlich hilfreich, alles einmal in eine Tabellenform zu bringen, um zu sehen, welcher Browser welches Videocontainerformat und welchen Codec unterstützt.

Browser	*Unterstützte Videocontainerformate und -codecs*		
iOS Safari (iPhone, iPod touch und iPad)	MPEG-4 und QuickTime mit H.264 (Baseline-Profil-encoded)		
Android	MPEG-4 mit H.264 (Baseline-Profil-encoded)	ab Ende 2010: WebM mit VP8 Codec	
BlackBerry OS 6	MPEG-4 mit H.264 (Baseline-Profil-encoded)		
Safari 3+	MPEG-4 mit H.264		
Chrome	MPEG-4 mit H.264	ab Chrome 6: WebM mit VP8 Codec	OGG Theora
Firefox		ab Firefox 4: WebM mit VP8 Codec	OGG Theora
Opera		ab Opera 11: WebM mit VP8 Codec	OGG Theora
Internet Explorer 9		WebM mit VP8 Codec; Codec muss zusätzlich installiert sein	

Mit der Vielzahl von unterschiedlichen Videocontainerformaten und -codecs ist es auch wie beim `audio`-Tag erforderlich, dass Sie mehrere Formate zur Verfügung stellen. Eine Anleitung, mit welchen Tools Sie Video- und Audiodateien am besten in die gewünschten Formate konvertieren, finden Sie weiter unten in diesem Kapitel.

Unterschiedliche Quellen können Sie wie beim Einbinden von Audiodateien auch über den `source`-Parameter steuern:

```
<video controls="controls">

 <!-- option 1: H.264 -->
 <source src="film.mp4" type='"video/mp4; codecs="mp4a.40.2"'/>

 <!-- Option 2: OGG Theora Video und Vorbis Audio -->
 <source scr="film.ogv" type='"video/ogg; codecs="theora, vorbis"' />

 <!-- HTML für nicht-unterstützte Player -->
<p>HTML5-Video kann nicht abgespielt werden. Besorg dir einen neuen
Browser!</p>

</video>
```

Wie auch beim `audio`-Tag versucht der Browser jetzt, die angebotenen Formate von oben nach unten durchzugehen, um zu sehen, welches Videocontainerformat und welcher Codec wiedergegeben werden kann. Es empfiehlt sich, MPEG-4-/H.264-Videos als erste Option anzubieten, da diese Kombination von den meisten (mobilen) Browsern unterstützt wird und beispielsweise das iPad nur den ersten `source`-Eintrag liest. Ein Fehler, der hoffentlich bald behoben wird.

Mittels des im Beispielcode verwendeten `type`-Attributs können Sie dem Browser gleich von Anfang an die Information geben, welches Containerformat und welcher Codec für das Video gewählt wurde und vom Browser gewählt werden soll. Das hat den großen Vorteil, dass der Browser nicht erst das Video laden muss, um das Format zu erkennen, sondern gleich im HTML-Code sehen kann, ob er das Videocontainerformat und den Codec unterstützt oder nicht.

Das `video`-Tag kann mit einer Reihe von Attributen kommen, die wir jetzt im Detail vorstellen möchten.

autoplay

Das `autoplay`-Attribut haben wir bereits beim `audio`-Tag kennengelernt. Wie schon dort würde dieses Tag veranlassen, dass das Video gleich nach dem Laden der Seite automatisch gestartet wird. Da mobile Benutzer möglicherweise viel Geld für die Datenübertragung – gerade bei datenintensiven Inhalten wie Videos – bezahlen müssten, funktioniert die `autoplay`-Option nicht auf dem iPhone und auch nicht auf dem iPad.

Auf einem nicht mobilen Gerät braucht der Benutzer jedoch keinen Play-Button anzutippen. Die `autoplay`-Option wird wie folgt eingebunden:

```
<video src="beispiel.ogv" autoplay="autoplay" />
```

oder so:

```
<video src="beispiel.ogv" autoplay />
```

Da das `autoplay`-Attribut nur ein Boole'scher Ausdruck ist, reicht die Angabe von `autoplay` auch ohne Parameter.

controls

Falls verwendet, werden im Browser die Standardsteuerelemente zur Videowiedergabe mit angezeigt. Man könnte mit JavaScript eigene Steuerelemente bauen, allerdings ist das für die mobile Verwendung wenig sinnvoll. Die Bildschirme sind ohnehin klein, und der Nutzer hat sich bereits an die Steuerelemente beispielsweise auf einem iPhone gewöhnt. Aus diesem Grund wird das Attribut auch auf einem iPhone oder einem iPod touch ignoriert, auf einem iPad allerdings berücksichtigt. Die Standardelemente beinhalten:

- Play/Abspielen
- Pause
- Volume
- Vor- und Zurückspulen

Manche Browser, insbesondere Desktop-Browser, bieten noch zusätzliche Control-Elemente. Auch das Aussehen der Standardelemente können Sie ohne die Verwendung von JavaScript nicht beeinflussen. Wie schon das `autoplay`-Attribut kann auch `controls` ohne Parameter in das `video`- oder `audio`-Tag mit eingebunden werden:

```
<video src="beispiel.ogv" controls />
```

loop

Falls vorhanden, wird der Audio- oder Videoinhalt nach dem Abspielen automatisch wiederholt.

```
<video src="beispiel.ogv" loop />
```

poster="url" (nur für Video-Tag)

Falls vorhanden, gibt die im Attribut angegebene URL den Pfad zu einem Bild an, das das Video repräsentieren soll. Es handelt sich dabei um etwas Ähnliches wie das Startup-Bild beim iPhone, das nach dem Antippen des Icons so lange dargestellt wird, bis die Anwendung geladen ist. Bei einem Video kann es zum Beispiel der erste Videoframe sein, der angezeigt wird, bis das Video geladen ist.

Das Posterbild wird also so lange angezeigt, bis der erste Frame des Videos verfügbar ist. Danach wird es gegen den ersten Videoframe ausgetauscht. Auf einem iPhone oder einem iPod touch wird seit iOS 4 der erste Videoframe ebenfalls bereits im Browser angezeigt, obwohl das Video im Standardmedienprogramm wiedergegeben wird. Das folgende einfache Codebeispiel verwendet kein `poster`-Attribut und wird auf einem iPhone oder iPod mit einem Standardabspielelement dargestellt.

```
<!DOCTYPE html>
<html>
   <head>
      <title>Einfacher Videoplayer</title>
   </head>
   <body>
      <video src="film.mp4" controls />
  </body>
</html>
```

Auf einem iPod oder iPhone sieht der Ablauf dann wie folgt aus. Ein Standardelement wird nach dem Laden der Seite angezeigt. Der Benutzer kann jetzt schon den Playbutton antippen.

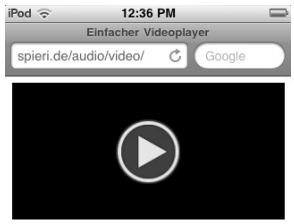


Bild 2.27: Anzeige eines Standardelements.

Nach wenigen Sekunden wird der erste Videoframe geladen und angezeigt. Das Antippen des Play-Buttons ruft den Standardmedienplayer auf.

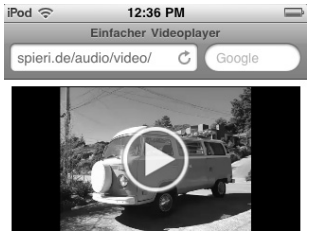


Bild 2.28: Anzeige des ersten Videoframes.

Im Standardplayer wird das Video wiedergegeben. Als Programmierer muss man sich nicht um die Steuerelemente kümmern, da sie vom System angeboten werden.

Bild 2.29: Wiedergabe des Videos im Standardplayer.

Wird der erste Videoframe als Bild abgespeichert und im `poster`-Attribut verwendet, ändern sich Code und Darstellung wie folgt:

```
<!DOCTYPE html>
<html>
   <head>
      <title>Einfacher Videoplayer</title>
   </head>
   <body>
      <video src="film.mp4"
```

```
          controls
          poster="poster.jpg"
      />
  </body>
</html>
```

Gleich nach dem Laden der Seite wird das Posterbild anstelle der schwarzen Box angezeigt. Für den Benutzer sieht es so aus, als sei das Video gleich verfügbar und müsste nicht erst noch geladen werden. Nach dem Antippen wird wieder der Standardplayer aufgerufen. In der Zeit zwischen dem Laden des Posterbilds auf der Webseite und dem Aufrufen des Players sollte schon so viel Zeit vergangen sein, dass hier auf jeden Fall bereits der erste Frame angezeigt werden kann.

Bild 2.30: Anzeige des Posterbilds. Beim Antippen wird der Standardplayer geladen.

- `width="` – Anzahl der Bildpunkte oder Prozentwert"
- `height="` – Anzahl der Bildpunkte oder Prozentwert" (nur für `video`-Tag)

Mit diesen Werten kann die Wiedergabegröße des Videos bestimmt werden, zumindest wenn das Video nicht auf einem kleinen Bildschirm wiedergegeben wird. Wir erinnern uns: Bei vielen mobilen Geräten werden Videos nicht im Browser, sondern im Standardmedienplayer wiedergegeben. Werden die `width`- und `height`-Attribute nicht verwendet, wird, wie in den Abbildungen oben dargestellt, eine schwarze Box von mindestens 300 x 150 Bildpunkten auf iPhones oder iPod touch-Geräten angezeigt.

Wichtig ist auch zu bemerken, dass die Angabe der Breite (`width`) und Höhe (`height`) im `video`-Tag nicht das Verhältnis ändert, mit dem das Video aufgezeichnet oder konvertiert wurde. Sie können nicht aus einem 4:3-Video mittels dieses Attributs ein schickes 16:9-Video zaubern.

```
<!DOCTYPE html>
<html>
   <head>
      <title>Einfacher Videoplayer</title>
  </head>
```

```
   <body>
      <video src="film.mp4"
          controls
          width="320"
          height="240"
          poster="poster.jpg"
      />
  </body>
</html>
```

Hier sehen Sie ein eingebundenes Video auf einem iPod ohne `width`- und `height`-Parameter und daneben ein eingebundenes Video auf einem iPod mit `width`- und `height`-Parametern. Das Videos wird weiterhin im Standardmedienplayer wiedergegeben.

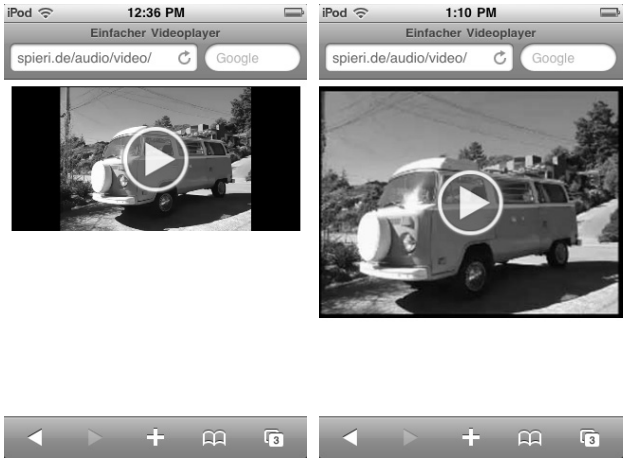


Bild 2.31: Video auf einem iPod ohne und mit `width`- und `height`-Parameter.

Und dies ist ein eingebundenes Video auf dem iPad, ebenfalls mit `width` und `height`. Im Gegensatz zu iPhone und iPod findet die Videowiedergabe direkt auf der Webseite und in den angegebenen `width`- und `height`-Abmessungen statt.

Bild 2.32: Eingebundenes Video auf dem iPad.

Falls Sie Videos mittels HTML5 in viele Bereiche Ihrer Webpräsenz einbauen wollen und alle über eine konstante Breite und Höhe verfügen sollen, empfiehlt es sich, das `video`-Tag gleich mit CSS am Anfang des Codes zu bearbeiten und die Standardhöhen dem Tag generell zuzuweisen:

```
video { width: 320px; height: 240px; }
```

preload="none", preload="metadata", preload="auto"

Mit dem `preload`-Attribut kann bestimmt werden, dass ein Video teilweise oder vollständig geladen wird, bevor der Benutzer die Wiedergabe starten kann. Sie können dem Desktop-Browser mit diesem Attribut also mitteilen, ab wann Ihrer Meinung nach die Wiedergabe eines Videos (oder einer Audiodatei) sinnvoll gestartet werden kann. Manche Browser ignorieren das `preload`-Attribut, wenn beispielsweise genug Bandbreite zum einwandfreien Laden und gleichzeitigem Wiedergeben des Videos vorhanden ist. Auch alle bislang bekannten mobilen Browser ignorieren dieses Attribut, da es eventuell ungewollten und mitunter für den Benutzer teuren Netzwerk-Traffic verursachen könnte. Folgende Werte kann das `preload`-Attribut haben:

- `none` – Ein Vorausladen der Audio- oder Videodatei ist nicht notwendig. Das sollten Sie insbesondere dann verwenden, wenn das jeweilige Video kein Hauptbestandteil der Seite ist und die Wahrscheinlichkeit nicht sehr groß ist, dass der Benutzer das Video ansehen möchte.
- `metadata` – Hat die gleiche Auswirkung wie `none`, nur dass hier die sogenannten Metadaten des Videos, also Dauer, Dateigröße etc., geladen werden, jedoch nicht die wirklichen Audio- und Videodaten.
- `auto` – Sagt dem Browser, dass entweder einige Teile oder das komplette Video idealerweise bereits vorgeladen werden sollten.

Safari kocht derzeit noch sein eigenes Süppchen und ignoriert das Attribut komplett, hört aber auf ein Attribut namens `autobuffer` – das ebenfalls veranlasst, dass ein Video nach Möglichkeit sofort in den Speicher geladen wird. Auch `autobuffer` ist in der mobilen Safari-Version wirkungslos, da ein mobiler Nutzer über entstehenden Datenverkehr immer explizit selbst entscheiden muss.

2.4.5 Fehlerbehandlung

Sie haben sicherlich schon gemerkt, dass die Wiedergabe von Videos (und in bestimmtem Umfang auch die Wiedergabe von Audiodateien) eine recht komplexe Sache mit vielen Fehlerquellen ist. Mithilfe der JavaScript-API kann man dem Benutzer zumindest einige Fehlerquellen kommunizieren.

```
<!DOCTYPE html>
<html>
  <head>
   <title>Videoplayer mit Fehler-Handling</title>
   <meta name="layout" content="webkit" />

   <style type="text/css" media="screen">
```

```
  </style>
  <script>
  function fehler(e) {
     // Fehler bei Videowiedergabe - Warum?
   switch (e.target.error.code) {
    case e.target.error.MEDIA_ERR_ABORTED:
      alert('Abbruch durch Benutzer.');
      break;
    case e.target.error.MEDIA_ERR_NETWORK:
      alert('Durch fehlende Netzwerkverbindung konnte das Video ganz oder
      teilweise nicht geladen werden.');
      break;
    case e.target.error.MEDIA_ERR_DECODE:
      alert('Video-Datei ist fehlerhaft.');
      break;
    case e.target.error.MEDIA_ERR_SRC_NOT_SUPPORTED:
      alert('Video-Format wird nicht unterstützt.');
      break;
    default:
      alert('Unbekannter Fehler ist aufgetreten.');
      break;
    }
   }
  </script>
  </head>
  <body>
   <video src="film.mp4" controls onerror="fehler(event)"></video>
  </body>
</html>
```

Falls beispielsweise im `video`-Tag auf ein Video verwiesen wird, das der Browser nicht unterstützt, sieht der Benutzer die folgende Fehlermeldung auf einem iPod touch oder iPhone. In der Abbildung sehen Sie auch, dass im iOS bei einem fehlerhaften Video der Play-Button nicht richtig dargestellt wird. Mit dem gerade vorgestellten Error-Handling können Sie Ihren Benutzern schnell mitteilen, warum das Video nicht abgespielt werden kann.

Bild 2.33: iOS meldet ein fehlerhaftes Videoformat.

Die möglichen Fehlercodes noch einmal in Tabellenform:

Mögliche Fehlercodes	
MEDIA_ERR_ABORTED	Der Ladeprozess bzw. Prozess zum Abspielen des Videos wurde vom Benutzer abgebrochen.
MEDIA_ERR_NETWORK	Das angeforderte Video konnte teilweise oder ganz nicht geladen werden. Diese Fehlermeldung kann des Öfteren auf mobilen Geräten auftreten, wenn beispielsweise kein oder nur sehr geringer Datenempfang vorherrscht.
MEDIA_ERR_DECODE	Der Videoplayer unterstützt das Format, kann aber das Video nicht decodieren – ein Anzeichen dafür, dass die Videodatei beschädigt sein könnte oder Probleme mit dem Decoder vorliegen.
MEDIA_ERR_SRC_NOT_SUPPORTED	Der Videoplayer kann das verwendete Videoformat nicht erkennen. Dabei kann es sich um das Videocontainerformat handeln, aber auch um den jeweiligen Codec. Sie erhalten diese Fehlermeldung auch, wenn Sie beispielsweise eine vom iPhone unterstützte MP4-Datei mit einem H.264-Codec aufrufen wollen, der mit einem High-Profile codiert wurde. Wir erinnern uns: Safari für iPhone und iPod touch unterstützt, wie die meisten anderen mobilen Browser auch, nur das Baseline-Profile in H.264-Codierungen.

2.4.6 Videoimplementierung mittels JavaScript-API

Mit der JavaScript-API erhalten Sie weitaus umfangreichere Möglichkeiten, Videoinhalte mit HTML5 zu steuern und einzubauen. Das nächste Beispiel funktioniert auf den meisten Desktop-Browsern, insbesondere bei Chrome und Safari. Die nahezu volle Unterstützung der JavaScript-API für Video konnten wir bislang nur unter iOS 4 finden, deshalb funktionierte dieser Code auch zum Zeitpunkt der Entstehung dieses Buchs nur auf einem iPhone 4 oder einem anderen Apple-Gerät mit iOS 4 fehlerfrei.

Das Beispiel zeigt am Anfang, ob der verwendete Browser das HTML5-`video`-Tag überhaupt unterstützt. Falls HTML5-Video unterstützt wird, überprüft der Code, ob die beiden Videocodecs H.264 und Theora vom Browser verstanden werden. Auf einem iPod mit OS 4.0 ergibt sich folgendes Bild:

Bild 2.34: Richtig erkannt: Safari für iOS 4.0 unterstützt H.264, aber nicht Theora.

Über den Button *Video buffern* wird die JavaScript-Funktion `movie.load()` aufgerufen, die das Video in den Speicher lädt. Ohne die Seite komplett neu zu laden, wird der aktuelle Ladestatus oberhalb des Videos in Prozenten angezeigt. Die Funktion `movie.play()` ruft bei iPod touch und iPhone den Standardmedienplayer auf und beginnt damit, das Video abzuspielen. In einem Desktop-Browser wie beispielsweise Chrome wird das Video in der angegebenen Größe direkt im Browserfenster abgespielt. Dort ergibt dann auch die Verwendung von `movie.pause()` mehr Sinn, da damit das Abspielen des Films im Browserfenster unterbrochen werden kann.

```
<!DOCTYPE html>
<html>
    <head>
        <title>HTML5-Videoplayer</title>
      <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
      <meta name="layout" content="webkit" />
      <link href="css/style.css" rel="stylesheet" media="screen"
type="text/css" />
      <script src="javascript/functions.js" type="text/javascript"></script>

      <script>

      (function my() {

         var $i = {};

         $i.init = function()
         {
            var detectionNode, msg = '', formats, pos, audio;
            detectionNode = document.getElementById('detection');

            if (!$i.hasVideo())
               msg += 'Video-Tag wurde NICHT erkannt<br/>';
            else
            {
               msg += 'Video-Tag wurde erkannt<br/>';
```

```
               formats = {
                  h264 : 'video/mp4; codecs="avc1.42E01E, mp4a.40.2"',
                  theora : 'video/ogg; codecs="theora, vorbis"',
               };
                  for (pos in formats)
                  {
                     video = document.createElement('video');
                     if(!!(video.canPlayType &&
video.canPlayType(formats[pos]).replace(/no/, '')))
                        msg += pos + ' wird unterstützt<br/>';
                     else
                        msg += pos + ' wird nicht unterstützt<br/>';
                  }
            }

            detectionNode.innerHTML = msg;

            if (!$i.hasVideo())
               return;

            var movie = document.getElementById('movie');

            document.getElementById('load').addEventListener('click',
function(e) {
               movie.load();
            }, false);

            document.getElementById('play').addEventListener('click',
function(e) {
               movie.play();
            }, false);

            document.getElementById('pause').addEventListener('click',
function(e) {
               movie.pause();
            }, false);

            movie.addEventListener('progress',function(e){
               var soFar = parseInt(((movie.buffered.end(0) /
movie.duration) * 100));
               document.getElementById("status").innerHTML =  soFar + '%';
            },false);

            movie.addEventListener('canplaythrough',function(e){
               movie.play();
            },false);

            movie.addEventListener('loadedmetadata',function(e){
               alert('size: ' + movie.videoWidth + '/' + movie.videoHeight);
            },false);


         };

         $i.hasVideo = function()
```

```
        {
           return !!document.createElement('video').canPlayType;
        };

        document.addEventListener("DOMContentLoaded", $i.init, false);
     }).call();

     </script>
   </head>
   <body>

     <div id="topbar">
        <div id="title">Erkennung</div>
     </div>
     <div id="content">
        <ul class="pageitem">
           <li class="textbox">
              <p id="detection"></p>
           </li>
           <li class="textbox">
              <p id="status"></p>
           </li>
           <li class="textbox">
              <video id="movie" controls poster="poster.jpg"
src="film_iphone.mp4" width="280">Video-Tag not recognized</video>
           </li>
           <li class="button">
              <input id="load" type="submit" value="Video buffern"/>
           </li>
           <li class="button">
              <input id="play" type="submit" value="Video abspielen"/>
           </li>
           <li class="button">
              <input id="pause" type="submit" value="Pause"/>
           </li>
        </ul>
     </div>
   </body>
</html>
```

2.4.7 2-pass Encoding – was ist das?

Wenn Sie als Qualitätskriterium entweder die Zielgröße der Filmdatei oder die mittlere Bitrate verwenden, haben Sie auch noch die Möglichkeit, über eine kleine Checkbox das sogenannte »2-pass Encoding« zu aktivieren. In diesem Fall erhöht sich zwar die Zeit, die das quelloffene Encoding-Programm HandBrake zum Codieren des Videos benötigt, aber das Codieren wird dann in zwei Schritten durchgeführt.

Im ersten Schritt wird das Video genau nach den Gesichtspunkten analysiert, die für den zweiten Schritt – das eigentliche Codieren – hilfreich sind. Auch wenn das 2-

pass Encoding länger dauert, sichert es in der Regel eine bessere Videoqualität zu. Beachten Sie, dass bei den iPhone-Voreinstellungen automatisch die *Constant Quality*-Qualitätsoption ausgewählt wird, die kein 2-pass Encoding zulässt.

- Die Framerate in Frames pro Sekunde (FPS) brauchen Sie ebenfalls nicht zu verändern. In der Standardeinstellung wird die gleiche Framerate wie in der Originaldatei verwendet, was für den Großteil der Videocodierungen passt.
- Die Option *Web optimized* ist leider standardmäßig nicht aktiviert, sollte aber für jegliche Videocodierung für die Internetwiedergabe ausgewählt sein. Ist diese Option gesetzt, werden die Videodaten so codiert, dass das Video schon mit wenigen geladenen Daten am Anfang wiedergegeben werden kann und im Hintergrund noch weitere Daten nachlädt. Die Einstellung hat keine Auswirkungen auf die Videoqualität oder den Codec, resultiert aber in einer besseren Benutzbarkeit für den Anwender.
- Vergewissern Sie sich noch einmal, ob in der Auswahl des Videocodecs H.264 gewählt wurde. Der alternativ angebotene Codec, FFmgpeg, funktioniert auf den meisten mobilen Geräten nicht.

Werfen Sie zum Abschluss einen Blick auf die Audioeinstellungen. Sofern Sie in den Voreinstellungen *Apple iPhone & iPod touch* ausgewählt haben, brauchen Sie wahrscheinlich auch hier keine Änderungen mehr vorzunehmen. Beachten Sie nur, dass für die gewünschte Tonspur auch wirklich der AAC-Audiocodec ausgewählt ist.

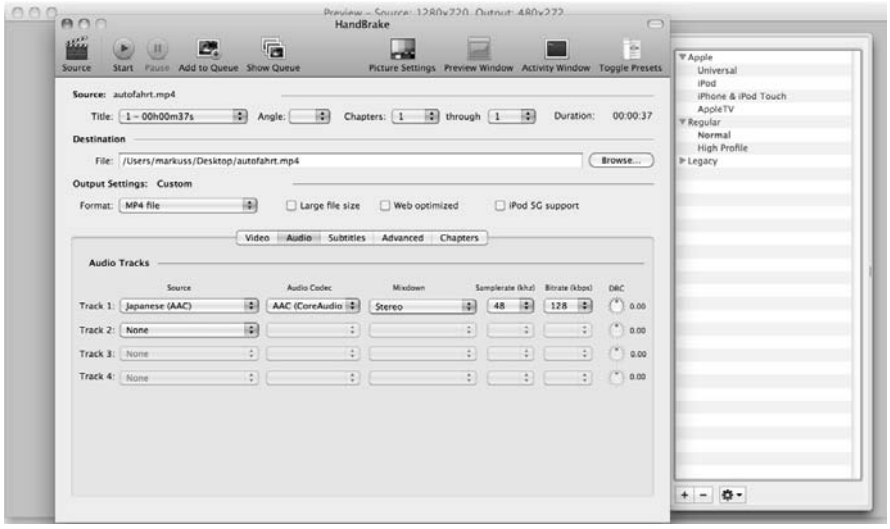


Bild 2.35: Einstellung auf AAC-Audiocodec?

Sie haben alles überprüft? Dann klicken Sie im oberen Fensterbereich auf *Start*, und HandBrake wird damit beginnen, Ihr Video mit den gewünschten Einstellungen zu codieren und in das Zielverzeichnis zu speichern. Ein Statusbalken zeigt den Fortschritt im unteren Fensterbereich an.

Ob die Codierung mit HandBrake wirklich erfolgreich war, können Sie am besten mit einem kleinen HTML5-Video feststellen. Hier der Sourcecode:

```
<!DOCTYPE html>
<html>
   <head>
      <title>HandBrake-Test</title>
   </head>
   <body>
      <video src="autofahrt.mp4" controlswidth="480"height="272"/>
   </body>
</html>
```

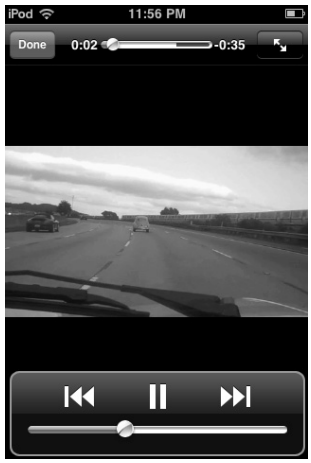


Bild 2.36: Die HandBrake-Codierung war erfolgreich.

OGG Theora/OGG Vorbis mit Firefogg

Das nächste Programm, das wir Ihnen zur Videokonvertierung vorstellen, ist Firefogg. Firefogg ist eigentlich kein richtiges Programm, sondern eine Erweiterung für den Firefox-Browser und läuft daher auf allen Betriebssystemen, auf denen auch Firefox läuft: Windows, Mac OS X und Linux. Und Sie benötigen natürlich den Firefox-Browser ab Version 3.5.

Das Herzstück von Firefogg ist ein Programm namens *ffmpeg2theora*. Das Konvertierungsprogramm wird zwar auf der OGG Theora-Seite empfohlen, lässt sich aber nur über die Kommandozeile bedienen. Firefogg bietet zu *ffmpeg2theora* die grafische Benutzeroberfläche.

Um Firefogg zu installieren, starten Sie einfach den Firefox-Browser und gehen zu folgender Webadresse: *www.firefogg.org*. Auf der Webseite sehen Sie einen roten Button, über den Sie Firefogg auf Ihrem Rechner als Firefox-Erweiterung installieren können.

Bild 2.37: Klicken Sie auf *Allow* oder *Erlauben*, um Firefogg zu installieren.

Nach dem Anklicken von *Install Firefogg* erscheint im oberen Browserfenster eine gelbe Leiste mit einer Sicherheitsabfrage. Diese können Sie ruhigen Gewissens mit *Allow* oder *Erlauben* beantworten. Anschließend wird Firefogg heruntergeladen und installiert. Um die Installation abzuschließen, muss Firefox neu gestartet werden; ein entsprechendes Dialogfenster bietet das nach der Installation auch gleich an.

Nach dem Neustart des Browsers und einer erfolgreichen Installation von Firefogg sehen Sie beim Aufrufen von *www.firefogg.org* nun ein grünes Rechteck mit der Nachricht *Firefogg installed.* Um ein erstes Video zu konvertieren, klicken Sie auf den Link *Make Web Video.*

Bild 2.38: Firefogg versteht viele Sprachen, darunter auch Deutsch.

Falls Sie Firefogg noch nicht in Deutsch sehen, können Sie es jetzt oder auch eine der vielen anderen Sprachen auswählen. Wie bei HandBrake startet auch hier der Konvertierungsprozess mit einer Quelldatei.

Wir verwenden in unserem Beispiel wieder die gleiche HD-Videodatei der Flip-Kamera, die wir bereits im H.264-Beispiel in ein mobiles Format konvertiert haben. Klicken Sie dazu auf *Wähle Datei* und wählen Sie dann eine Videodatei von Ihrer Festplatte aus.

Sobald das Video geladen ist, sehen Sie im Firefogg-Fenster viele Einstellungsmöglichkeiten. Wir gehen alle sechs Menüpunkte durch:

Voreinstellungen

Wie HandBrake bietet auch Firefogg eine Reihe von Voreinstellungen an. Für das Theora-Videoformat gibt es drei vordefinierte Auswahlmöglichkeiten, die sehr unterschiedliche Ergebnisse liefern. Um Videos auf mobilen Geräten anzuzeigen, dürfte die Standardoption *Internetvideo (400 kBits und 400 Pixel maximale Größe)* ausreichen, jedoch werden wir gleich noch sehen, wie wir Bildgröße und Videoqualität anders beeinflussen können.

Das Original-HD-Video hat in unserem Beispiel eine Größe von knapp 49 MByte bei einer Laufzeit von weniger als 40 Sekunden. Die verschiedenen Theora-Voreinstellungen liefern folgende Ergebnisse:

Bildbeispiel	*Bezeichnung der Voreinstellungen*	*Dateigröße*
	Niedrige Bandbreite Theora, 164 KBit/s und 200 Pixel maximale Größe	940 KBit
	Internetvideo Theora, 400 KBit/s und 400 Pixel maximale Größe	3 MByte
	Hohe Qualität Theora, 1.080 Pixel maximale Größe	11 MByte

Codierungsbereich

Über diesen Menüpunkt legen Sie fest, welcher zeitliche Bereich des Videos codiert werden soll. Das ist insbesondere dann hilfreich, wenn Sie die Qualität der Codierung bei einem längeren Video testen wollen. Gute Codierungen brauchen ihre Zeit und können bei längeren Videos leicht viele Minuten dauern. Mithilfe der Menüoption können Sie beispielsweise nur die ersten oder letzten zehn Sekunden codieren lassen, das Ergebnis evaluieren und gegebenenfalls nach etwas »Feintuning« das gesamte Video codieren.

Qualitäts- und Auflösungseinstellungen

Wie der Name schon sagt: Mit diesen Einstellungen legen Sie die Video- und Audioqualität, aber auch den zu verwendenden Codec fest.

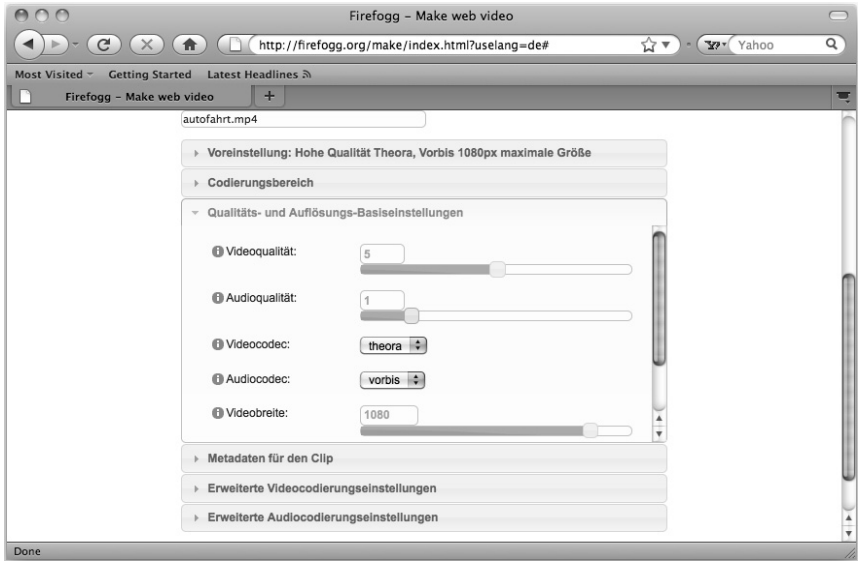


Bild 2.39: Festlegen der Qualitäts- und Auflösungseinstellungen.

Für die Video- und Audioqualität stehen Ihnen zwei selbsterklärende Schieberegler zur Verfügung. Sind sie ganz weit nach rechts geschoben, bedeutet das eine hohe Qualität, scharfe Bilder und klaren Sound, allerdings auch eine immense Dateigröße, weit nach links eingestellt, bedeuten sie das Gegenteil. Je nach Ihren Bedürfnissen müssen Sie diese Einstellungen verändern, bis Sie zufrieden sind.

Unter den oberen Schiebereglern finden Sie die Auswahlmenüs für die zu verwendenden Codecs. Da wir in diesem Beispiel ein OGG Theora-Video mit OGG Vorbis-Audiocodec erstellen möchten, sollten diese beiden Codecs auch ausgewählt sein.

Wie HandBrake lässt Firefogg Änderungen an der Videogröße zu. Für eine mobile Wiedergabe bietet es sich in der Regel an, eine der üblichen Bildschirmgrößen zu benutzen. HandBrake hat für eine iPod touch- und iPhone-Konvertierung eine Videobreite von 480 Bildpunkten vorgeschlagen, was wir hier anhand des *Videobreite*-Schiebereglers auch einstellen können.

Sobald Sie einen der beiden Regler verschieben, sehen Sie, dass sich die Regler für Höhe bzw. Breite ebenfalls auf einen neuen Wert verschieben. Firefogg behält also immer die Proportionen eines Videos bei und erlaubt dort, im Gegensatz zu HandBrake, keine Änderungen. Nur in den erweiterten Einstellungen findet sich eine Option, mit der Sie das Seitenverhältnis des Videos von 4:3 in 16:9 und umgekehrt ändern können.

Metadaten für den Clip

In diesem Bereich können Sie einige Stammdaten zum Video eintragen, beispielsweise den Titel, den Autor, Copyright-Informationen oder auch, an welchem Ort das Video aufgenommen wurde. Manche Wiedergabeprogramme nutzen diese Informationen, in den meisten Abspielprogrammen sind die Daten aber nicht sichtbar.

Erweiterte Videocodierungseinstellungen

Wenn Sie schon etwas erfahrener mit der Videokonvertierung sind, können Sie hier noch ein paar grundlegende Einstellungen vornehmen.

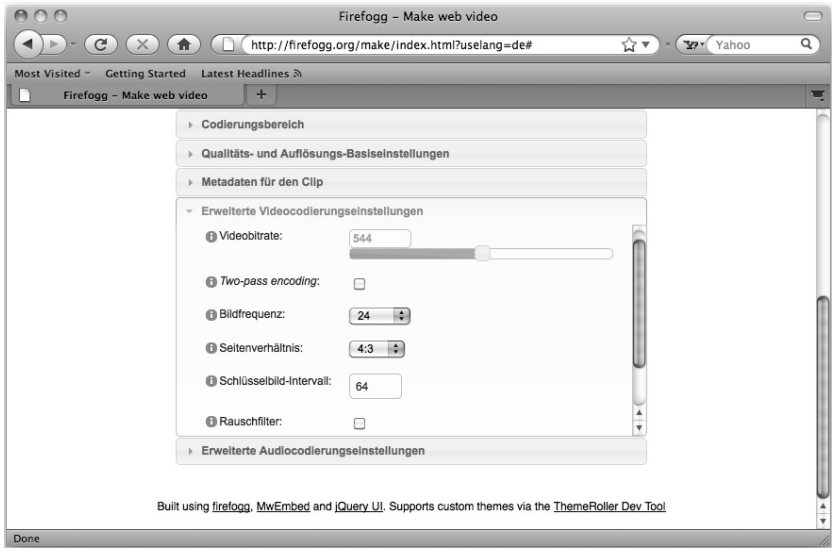


Bild 2.40: Festlegen der erweiterten Videocodierungseinstellungen.

Wie HandBrake erlaubt auch Firefogg eine Änderung der Bitrate. Dieser Parameter hat eine unmittelbare Auswirkung auf die Videoqualität und – zur Erinnerung – bestimmt, wie viele Bits Speicherplatz pro Videosekunde maximal gespeichert werden können.

Auch das *2-pass encoding* kennen Sie schon von HandBrake. Wir empfehlen, diese Option zu aktivieren, falls Sie genug Zeit dafür haben. Der Codierungsvorgang wird ungefähr doppelt so lange dauern, resultiert aber in einem qualitativ höherwertigen Endergebnis. Eine genaue Beschreibung dazu, was diese Funktion macht, finden Sie im Abschnitt über HandBrake.

In der Regel haben Sie keinen Grund, die anderen Optionen zu benutzen. Die Bildfrequenz wird üblicherweise vom Quellvideo übernommen, ebenso das Seitenverhältnis.

Erweiterte Audiocodierungseinstellungen

Hinter diesem langen Menünamen verbirgt sich nur eine Option: Audio im codierten Video auszuschalten. Sind Sie mit allen Ihren Einstellungen zufrieden, können Sie auf *Ogg speichern* klicken und einen Zielpfad sowie einen Dateinamen für das ins OGG Theora-Format konvertierte Video eingeben. Anschließend zeigt Firefogg einen Statusbalken und ein Vorschaufenster.

Jetzt fragen Sie sich womöglich, warum Sie so viel über die Konvertierung von OGG Theora lernen mussten, obwohl es derzeit weder unter iOS noch unter Android verfügbar ist. Dazu gibt es zwei Gründe:

- Theora funktioniert auf einer Reihe von Desktop-Browsern und möglicherweise auch auf zukünftigen Versionen von Firefox Mobile und anderen mobilen Browsern und Betriebssystemen.
- Mit Theora sind Sie lizenzrechtlich auf der sicheren Seite – auch nach 2016, wenn sich die Lizenzbedingungen für H.264 wieder ändern werden.
- Alles, was Sie über die Konvertierung von OGG Theora mittels Firefogg gelernt haben, können Sie auch zur Codierung von WebM/VP8-Video nutzen.

Bild 2.41: Die Codierung des Videos nach OGG läuft.

WebM/VP8 mit Firefogg

Glaubt man den Zusagen der Browserhersteller, steht WebM/VP8 eine große Zukunft bevor. Die meisten Betriebssysteme und Browserhersteller mit Ausnahme von Apple/Safari wollen VP8 als neuen Standard implementieren. Den Ankündigungen von Google auf der Entwicklerkonferenz zufolge wird auch eine der neuen Android-Versionen dieses Format unterstützen.

Auch wenn WebM/VP8 noch eine sehr neue Technologie ist, wird sie dennoch bereits von Firefogg unterstützt. In den Voreinstellungen befinden sich schon zwei Optionen, um WebM-Video zu codieren.

Bildbeispiel	*Bezeichnung der Voreinstellungen*	*Dateigröße*
	WebM Web-Video mit VP8 (600 KBits/s bei 480 Pixeln maximaler Breite)	2,9 MByte
	WebM Video (hochwertig) mit VP8 (1.080 Pixel maximale Breite)	9,9 MByte

Alles Andere können Sie wie bei der OGG Theora-Konvertierung einstellen.

2.5 Geolocation-API

Die Geolocation-API – *http://dev.w3.org/geo/api/spec-source.html* – ermöglicht den Zugriff auf die geografische Position des Geräts via JavaScript. Da mobile Geräte häufig von nur einer Person benutzt werden, ist das in den meisten Fällen die Position des Eigentümers und ständigen Benutzers. Obwohl diese API des W3C aktuell im Status des »Editor Draft« vom 10. Februar 2010 vorliegt, wurde sie bereits umfassend und der Spezifikation entsprechend konform auf mehreren Plattformen implementiert. Dazu zählen iPhone OS 3+ sowie Android 2+.

Zudem ist die Geolocation-API mit Sicherheit eine der faszinierendsten APIs. Der Zugriff auf den aktuellen Ort des Benutzers ermöglicht zahlreiche ortsbasierte Informationsdienste, die ohne den geskripteten Zugriff nur umständlich möglich wären.

Location! Location! Location! Das hört man nicht nur immer vom Immobilienberater, sondern man sieht es auch in einer Vielzahl von mobilen Anwendungen und dank der Geolocation-API immer mehr auf mobilen Webseiten. Dabei ergeben ortsabhängige Dienste gerade mobil den wahrscheinlich größten Sinn: Der Anwender hat sein Mobiltelefon ständig dabei, und ohne umständlich seine Position anzugeben, kann nun sogar eine Webseite leicht herausfinden, wo sich der Benutzer befindet.

Eine Vielzahl von bekannten Webdiensten nutzt die Möglichkeiten der Location-API. Viele Firmen und Dienste gründen ihr gesamtes Konzept auf der Tatsache, dass der aktuelle Ort eines Benutzers in einer mobilen Anwendung oder einer Seite vorhanden ist.

Eines der ersten Mobiltelefone mit integriertem GPS, das eine große Masse erreichte, war das Nokia N95. Zahlreiche Symbian- and Java-Programme wurden um diese neue Funktionalität entwickelt. Für Webseiten blieb die Location-API aber bis Ende 2008 verschlossen. In den USA erschien das erste Android-Telefon, das T-Mobile G1, und bot erstmals Zugriff auf die aktuelle Position des Benutzers für Webseiten an. Den großen Durchbruch gab es aber am 17. Juni 2009: Zusammen mit dem iPhone 3GS wurde iPhone OS 3.0 veröffentlicht, das eben auch diese Funktionalität enthält.

Bild 2.42: Die Geolocation-API wird auch von Flickr benutzt, um Fotos in der Nähe des Benutzers anzuzeigen.

Die Fotoseite Flickr gehörte zu den ersten Webseiten, die gleich am 18. Juni 2009 (*http://blog.flickr.net/de/2009/06/18/*) dieses neue Feature seinen mobilen Nutzern unter *m.flickr.com* anbot. Und auch heute noch ist diese Funktion eine der beliebtesten Links auf Flickrs mobiler Seite. Gehen Sie einfach mit einem iPhone, einem Palm Pre oder einem Android-Telefon auf *m.flickr.com* und tippen Sie auf *In der Nähe*. Sie sehen nach zwei Sicherheitsabfragen (dazu gleich mehr) eine ganze Reihe von Fotos, die in Ihrer Nähe aufgenommen wurden. Weiter unten im Kapitel geben wir Ihnen auch noch einige Tipps dazu, wie Sie georeferenzierte Fotos von Flickr in Ihre Webseiten einbauen können.

Seitdem hat sich die Funktion auf vielen mobilen Webdiensten ausgebreitet: Das reicht von sozialen Netzwerken über standortbasierte Spiele bis hin zu Immobilienseiten, die die passenden Wohnungen und Häuser gleich in der Nähe des Benutzers heraussuchen.

2.5.1 Privatsphäre und Sicherheit

Aus Sicht der Entwickler stellt die Positionsbestimmung des Benutzers einen immensen Wert dar. Allerdings darf nicht vergessen werden, dass diese Information selbstverständlich auch missbraucht werden kann. Die Spezifikation schreibt deshalb vor, dass der Benutzer der Positionsbestimmung ausdrücklich zustimmen muss.

Das geschieht ab iPhone OS 3.0 durch einen modalen Dialog, der den Nutzer zwingt, zuzustimmen oder abzulehnen. Auf gängigen Android-basierten Geräten hingegen erscheint ein nicht modaler Dialog meist am unteren Rand des Bildschirms. Der Nutzer hat hier auch die Möglichkeit, diese Entscheidung für zukünftige Anfragen derselben Website zu speichern. Zwar kann die Entscheidung, der Ortsbestimmung zuzustimmen, zurückgesetzt werden, allerdings befindet sich die entsprechende Funktion drei Klicks tief in den Einstellungen des Browsers auf Android-Geräten und auch unter iOS einigermaßen versteckt in den allgemeinen Einstellungen.

Ebenso interessant wie praktisch für Entwickler, aber auch erschreckend, ist, dass die Zustimmung nicht für eine bestimmte URL gilt, sondern komplett für eine Domain. Sobald Sie *google.com* also die Zustimmung zur Ortung geben und diese Entscheidung speichern, kann jede beliebige Seite unterhalb der *google.com*-Domain automatisch auf Ihre Position zugreifen.

Sicherheitsabfragen unter iOS

Unter iOS 4 gibt es ein interessantes Prinzip bei Sicherheitsabfragen für Webseiten, die die aktuelle Position des Anwenders im System anfragen. Der Nutzer wird insgesamt in drei kleinen, modalen Dialogfenstern um Zustimmung gebeten. Hier ein Beispiel mit *m.flickr.com* auf einem iPod touch, der noch nie eine Webseite mit Geolocation-API-Unterstützung aufgerufen hat:

Falls zum ersten Mal im mobilen Safari-Browser eine Webseite nach der aktuellen Position des Benutzers fragt, muss der Anwender Safari den Zugriff auf diese Daten gestatten. Dieser Schritt ist logisch: Nur wenn der Browser erlaubten Zugriff auf die Geolocation-API und damit die aktuelle Nutzerposition hat, kann auch eine Webseite diese Daten erhalten.

Bild 2.43: Safari den Zugriff gestatten.

Nachdem der Anwender dem Browser den Zugriff auf die Daten erlaubt hat, muss der Benutzer jetzt auch noch der jeweiligen Webseite den Zugriff auf die aktuelle Position gestatten.

Ab iOS 4 und nachdem der Benutzer Safari mit den Zugriffsrechten ausgestattet hat, sehen Sie in der Titelleiste einen blauen Pfeil, der anzeigt, dass Safari jetzt gerade nach Positionsdaten fragt.

Bild 2.44: Flickr den Zugriff gestatten.

Vielleicht fragen Sie sich jetzt: Wieso drei Dialogfenster, dies waren doch nur zwei? Ganz einfach: Sobald der Benutzer auf Ihrer Webseite die Geolocation-Funktion ein zweites Mal aufruft, erscheint der letzte Dialog ebenfalls ein zweites Mal und muss nochmals vom Anwender bestätigt werden. Das ist ein Hilfsmittel, um sicherzugehen, dass die Erlaubnis beim ersten Mal nicht aus Versehen gegeben wurde.

Natürlich kann der Anwender unter iOS 4 diese Erlaubnis auch zurücknehmen. Dabei kann er nicht nur einer bestimmten Webseite die Erlaubnis entziehen, sondern dem Browser insgesamt.

Bild 2.45: Von diesem Bildschirm aus lässt sich bestimmen, welches Programm Geoinformationen verwenden darf und welches nicht.

Zu dem Bildschirm gelangen Sie über *Einstellungen/Allgemein* und schließlich *Ortungsdienste*. Nachdem Sie Safari die Erlaubnis entzogen haben, hat keine der vorher benutzten und autorisierten Webseiten mehr Zugriff auf die Position des Benutzers. Auch wenn Sie die Einstellung für Safari wieder aktivieren, müssen dennoch alle Webseiten erneut die Dialogabfragen durchlaufen.

Sicherheitsabfragen unter Android

Die Abfragen für Geolocation-Anfragen von Webseiten unter Android sind etwas anders. Sie brauchen nicht erst dem Browser die Erlaubnis zu geben, sondern werden gefragt, sobald Sie eine Funktion auf der Webseite aufrufen, die einen Call zu der Geolocation-API initiiert. Dabei erscheint ein kleiner Dialog am unteren Rand der Webseite mit folgendem Text (wir verwenden wieder das Flickr-Beispiel):

m.flickr.com möchte Ihren Standort in Erfahrung bringen.

Sie können diesen Dialog mit *Ablehnen* schließen und den Zugriff nicht gestatten oder auf *Standort freigeben* tippen, um den Zugriff auf die Positionsdaten zu gewähren.

Der Benutzer kann diese Auswahl als Voreinstellung speichern und wird dementsprechend den Dialog kein zweites Mal sehen. Anderenfalls wird der Dialog jedes Mal angezeigt.

Wie unter iOS kann auch unter Android der Zugriff nur allen Webseiten und nicht einzelnen Seiten entzogen werden. Um zu den entsprechenden Einstellungsfenstern zu gelangen, tippen Sie einfach im Android-Browser auf *Menü*, dann auf *Mehr* und schließlich auf *Einstellungen*.

In den Browsereinstellungen können Sie dann festlegen, ob der Browser generell den Standortzugriff für Webseiten ermöglichen soll oder nicht. Weiterhin finden Sie mit *Standortzugriff löschen* eine Funktion, mit der Sie allen bereits autorisierten Webseiten den Zugriff wieder entziehen.

Fair Play gilt auch für Entwickler

Auch dem Nutzer der Ortsinformationen, dem Entwickler, wird vorgeschrieben, wie mit diesen Informationen umzugehen ist. Freilich schützt das den Nutzer nicht vor Missbrauch, jedoch soll damit ein Standard geschaffen werden, der letztendlich Vertrauen schafft und die Nutzung von Ortsinformationen akzeptabel macht.

Zusammenfassend kann gesagt werden, dass der Empfänger der Ortsinformationen die Daten nur für den angegebenen Zweck verwenden darf, sie auf Wunsch des Nutzers löscht und die Daten so vertraulich und sicher wie möglich gespeichert werden. Als Entwickler möchten wir Sie daher noch einmal eindringlich darauf hinweisen, dass aktuelle Positionen von Menschen extrem vertrauliche Daten sind und auch so behandelt werden sollten.

Genug der langen Rede, tauchen wir ab in die technische Welt der Geolocation-API.

2.5.2 Feature Detection: Geolocation-API

Das Vorhandensein der Geolocation-API kann einfach überprüft werden. Das folgende Beispiel verdeutlicht das:

```
var hasGeo = function()
{
   if (navigator.geolocation)
      return true;
   else
      return false;
};
alert (hasGeo()) //true oder false
```

Das Vorhandensein der Eigenschaft `geolocation Objects` kann also ganz einfach im Navigator überprüft werden. Im gesamten HTML-Kontext einer Webseite sieht das wie folgt aus:

```
<!--
   Geolocation-API-Erkennung mit Modernizr und iWebKit 5
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
<html class="no-js">
    <head>
        <title>HTML5-Geolocation</title>
      <meta name="layout" content="webkit" />
      <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
      <link href="style.css" rel="stylesheet" media="screen" type="text/css"
/>

      <script src="js/modernizr-1.5.js" type="text/javascript"></script>
      <script>
```

```
      (function my() {

         var $i = {};

         $i.init = function()
         {
            document.getElementById('message').innerHTML = $i.hasGeo() ?
'Vorhanden' : 'Nicht vorhanden';
         };

         $i.hasGeo = function()
         {
            if (Modernizr.geolocation)
               return true;
            else
               return false;
         };

         document.addEventListener("DOMContentLoaded", $i.init, false);
      }).call();

      </script>
    </head>
    <body>

      <div id="topbar">
         <div id="title">Erkennung mit Modernizr</div>
      </div>
      <div id="content">
         <ul class="pageitem">
            <li class="textbox">
               <p>Geolocation-API: <span id="message"></span></p>
            </li>
         </ul>
      </div>
      <div id="footer">
         <a class="noeffect" href="http://iwebkit.net">Powered by
iWebKit</a>
      </div>
    </body>
</html>
```

Bild 2.46: Auf einem iPhone oder iPod touch sieht das Beispiel so aus.

Sollten Sie weiter HTML5-Features benutzen und sämtlichen Detection-Code nicht wiederholt schreiben wollen, empfehlen wir die Benutzung des Modernizr-Toolkits – *http://www.modernizr.com*.

Modernizr ist eine recht kleine, kostenfreie und einfach gehaltene JavaScript-Bibliothek, die sich insbesondere dann anbietet, wenn Sie mehr als eines der neueren HTML5- und CSS 3-Features einsetzen wollen. Modernizr erstellt ein JavaScript-Objekt und liefert entweder `false` oder `true` für jedes der neuen HTML5-/CSS 3-Features zurück, je nachdem, ob das Feature unterstützt wird oder nicht.

Um Modernizr einzubinden, müssen Sie die JavaScript-Bibliothek von der Modernizr-Webseite herunterladen und auf Ihrem Webserver speichern. Anschließend muss die Bibliothek über die folgende Zeile in Ihren HTML-Code im `head`-Bereich eingebunden werden:

```
<script src="modernizr-1.5.min.js" type="text/javascript"></script>
```

Des Weiteren ist es erforderlich, dass das `html`-Tag das Attribut `class="no-js"` erhält:

```
<html class="no-js">
```

Die Überprüfung sieht nach der Einbindung der Bibliothek wie folgt aus:

```
//Einbindung des Modernizr
var hasGeo = function()
{
   if (Modernizr.geolocation)
      return true;
   else
      return false;
}
alert (hasGeo()) //true oder false
```

Das neue JavaScript-Objekt heißt hier `Modernizr.geolocation` und liefert entweder den Wert `true` (Zugriff auf die Geolocation-API ist vorhanden) oder `false` (Geolocation-API wird vom Browser nicht unterstützt) zurück.

2.5.3 Arten der Positionsbestimmung

Die Geolocation-API unter `navigator.geolocation` ermöglicht sogenannte »One-Shot«-Positionsanfragen sowie kontinuierliche Aktualisierungen. Die Funktionen der API sind:

```
void getCurrentPosition(successCallback,
                 in optional PositionErrorCallback errorCallback,
                 in optional PositionOptions options);

long watchPosition(in PositionCallback successCallback,
                 in optional PositionErrorCallback errorCallback,
                 in optional PositionOptions options);

void clearWatch(in long watchId);
```

Die Funktionen `getCurrentPosition` und `watchPosition` unterscheiden sich nur darin, dass `watchPosition` kontinuierlich bei einer Änderung der Position die Callback-Funktionen aufruft. Wie deutlich die Änderung sein muss, um einen erneuten Aufruf auszulösen, ist hierbei der jeweiligen Implementierung überlassen und somit mit Vorsicht zu genießen (dazu später mehr). Der zurückgegebene `long`-Wert der `watch`-Position kann an die Funktion `clearWatch` übergeben werden, die dann die kontinuierliche Positionsbestimmung abschaltet.

2.5.4 Google Maps-API

Wenn wir an Karten im Internet denken, gibt es neben ein paar kleineren Anbietern eigentlich nur den großen Platzhirsch Google Maps. Google Maps ist der Standardkartenanbieter auf den meisten mobilen Plattformen, insbesondere mit den nativen Implementierungen unter iOS, Android und WebOS. Unter *http://code.google.com/apis/maps/* finden Sie eine Übersicht der verschiedenen APIs für Google Maps. Für unsere Zwecke werden wir uns mit der Maps-JavaScript-API beschäftigen.

Bild 2.47: Die Google Maps-API gibt es für JavaScript, Flash und auch für statische Bilder.

Die JavaScript-API wurde im Mai 2010 in einer neuen Version veröffentlicht. Die dritte Version der Google Maps-API ist nicht nur schneller, sondern auch wesentlich mehr auf mobile Geräte mit dem Fokus auf iPhone und Smartphones mit Android-Betriebssystem ausgelegt.

Die API wird über ein `script`-Tag im `head`-Bereich des HTML-Codes eingebunden:

```
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=true"></script>
```

Im Gegensatz zur Version 2 der API erfordert die neue Version nicht mehr die Angabe eines API-Schlüssels. Neu ist allerdings der `sensor`-Parameter, der auch unbedingt mit angegeben werden muss. Er spezifiziert, ob die Position eines Benutzers über ein Gerät mit GPS-Empfänger bezogen wird oder nicht. Da wir in diesem Buch mit neuen, mobilen Betriebssystemen arbeiten, setzen wir den Parameter auf `true`.

Die Google Maps-API benutzt die vom Benutzer gesetzte Spracheinstellung, um die verschiedenen Elemente, Bezeichnungen, Copyright-Informationen etc. auf der Karte anzuzeigen. In den meisten Fällen bedarf das keiner Änderung. Falls Sie jedoch eine Seite speziell für ein Land entwickeln und sicherstellen wollen, dass alle Elemente in der Google Maps-Ansicht in einer festgelegten Sprache angezeigt werden, können Sie den `script`-Aufruf um einen Sprachenparameter erweitern:

```
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=true&language=de">
```

Alle Sprachelemente in der Google Maps-Ansicht werden nach diesem Aufruf auf Deutsch angezeigt, egal welche Sprache der Benutzer für seinen Browser eingestellt hat.

Wichtiger als die Spracheinstellung (die in der Regel automatisch vom Browser übernommen wird) ist die Festlegung der Region, von der aus die meisten Benutzer die Seite mit der eingebundenen Google-Karte aufrufen. Mit dem folgenden `script`-Aufruf können Sie die Standardregion auf Deutschland setzen:

```
<script type="text/javascript"
src="http://maps.google.com/maps/api/js?sensor=true&region=DE">
```

Dies macht dann einen Unterschied, wenn Sie beispielsweise nicht die Geokoordinaten zu Google Maps schicken, sondern Google Maps mit einem Städtenamen als Suchbegriff aufrufen. Wird die Region nicht angegeben, wird in der Regel angenommen, dass die USA die Standardregion ist. Gibt es einen passenden Städtenamen auch in den USA, beispielsweise »Dresden«, kann es passieren, dass das amerikanische Dresden im Bundesstaat Ohio angezeigt wird und nicht die sächsische Landeshauptstadt.

2.5.5 One-Shot-Positionsanfragen

Eine einfache One-Shot-Anfrage kann so erstellt werden:

```
//Callback-Funktion, wird aufgerufen, sobald Position vorhanden ist
function showMap(position) {
   var lat = position.coords.latitude;
   var lon = position.coords.longitude;
```

```
}

function handleError(error)
{
   ...
}

// One-Shot-Anfrage
var options = {timeout:30000, maximumAge:300000, enableHighAccuracy:false};
navigator.geolocation.getCurrentPosition(showMap, handleError, options);
```

Obwohl der zweite und dritte Parameter – hier `handleError`, `options` – optional sind, sollte man auf die Implementierung nicht verzichten. `showMap` wird im obigen Beispiel aufgerufen, sobald der Benutzer der Ortsbestimmung zugestimmt hat und die Position festgestellt werden konnte. Ob das der Fall war, hängt auch von den Positionsoptionen ab, die als dritter Parameter übergeben werden.

Falls eine Bestimmung nach Vorgabe der `PositionOptions` nicht möglich war, wird `handleError` aufgerufen und ein `error`-Objekt übergeben. Das `PositionOptions`-Objekt ist optional und kann folgende Eigenschaften enthalten:

PositionOptions-Objekteigenschaften	
`long timeout`	Gibt die maximale Dauer der Positionsbestimmung an. Der Wert wird in Millisekunden angegeben, im obigen Beispiel also 30 Sekunden. Sobald diese Dauer überschritten ist, wird die Error-Callback-Funktion aufgerufen. Die hier angegebene Dauer betrifft lediglich den Prozess der Ortsbestimmung. Falls der Nutzer also zunächst um die Zustimmung gefragt werden muss, hat der Bestimmungsprozess ab dem Zeitpunkt der Zustimmung den hier angegebenen Wert.
`long maximumAge`	Gibt an, ob auch eine zuvor bereits ermittelte Position zurückgegeben werden kann. Das bietet sich oftmals an, da so die Dauer der Ortsbestimmung deutlich reduziert werden kann, obgleich die Qualität der Position natürlich je nach Veränderung der realen Position deutlich abnimmt. Dieser Wert wird auch in Millisekunden angegeben, im obigen Beispiel wird also eine Position akzeptiert, die maximal 5 Minuten (300 Sekunden) alt ist.
`boolean enableHighAccuracy`	Da die meisten Geräte, die die Geolocation-API implementieren, mehrere Möglichkeiten zur Ortsbestimmung haben, wird über diese Eigenschaft angegeben, ob man die Methode benutzen möchte, die die größte Genauigkeit bietet. In den allermeisten Fällen wird hierdurch der GPS-Chip des Geräts bemüht, die Position zu bestimmen. Das kann je nach Gerät über 30 Sekunden dauern. Viel schneller, jedoch auch ungenauer, ist die Bestimmung per Wifi-Netzwerk in Reichweite oder IP-Adressen der Wifi-Gateways.

Ein erstes Beispiel einer One-Shot-Positionsanfrage:

```
<!-- HTML5-Geoposition One-Shot-Anfrage
     Markus Spiering / Sven Haiges
-->
<!DOCTYPE html>
  <head>
    <title>One-Shot-Anfrage</title>
    <meta name="layout" content="webkit" />
   <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
   <link href="style.css" rel="stylesheet" media="screen" type="text/css" />

   <!-- Einladen/Aufruf der Google Maps-API-Version 3 -->
   <script type="text/javascript"
       src="http://maps.google.com/maps/api/js?sensor=true">
   </script>

   <script>
     (function my() {

      var $i = {};
     $i.vars = {};

      $i.init = function()
     {
       if ($i.hasGeo)
       {
       navigator.geolocation.getCurrentPosition($i.showLocation,
$i.handleError, {timeout:30000, maximumAge:300000, enableHighAccuracy:true})
        }
       else
        document.getElementById('message').innerHTML = 'Geo-Location-API
nicht verf&uuml;gbar!';
      };

        $i.showLocation = function(position)
      {
      document.getElementById('message').innerHTML = 'latitude: ' +
position.coords.latitude + ' / longitude: ' + position.coords.longitude + '
/ accuracy: ' + position.coords.accuracy + 'meters (acquired '+new
Date(position.timestamp)+')';

      var latlng = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);
      var myOptions = {
      zoom: 13,
      center: latlng,
      mapTypeId: google.maps.MapTypeId.ROADMAP
       };
```

```
      // Kartenaufruf wird auf map-Variable gelegt //
        var map = new google.maps.Map(document.getElementById("map"),
myOptions);
      }

      $i.handleError = function(error)
      {
      var msgNode = document.getElementById('message');

      switch (error.code)
      {
        case error.TIMEOUT:
        msgNode.innerHTML += 'Zeit &uuml;berschritten...';
         navigator.geolocation.getCurrentPosition($i.showLocation,
$i.handleError, {timeout:30000, maximumAge:300000});
          break;
          case error.PERMISSION_DENIED:
          msgNode.innerHTML += 'Benutzer hat Positionierung abgelehnt.';
          break;
          case error.POSITION_UNAVAILABLE:
          msgNode.innerHTML += 'Geoposition nicht verfügbar.';
          break;
         }

         msgNode.innerHTML +=  ' - Fehlermeldung: ' + error.message;
         }

         $i.hasGeo = function()
         {
         if (navigator.geolocation)
            return true;
         else
             return false;
         };

          document.addEventListener("DOMContentLoaded", $i.init, false);

            }).call();

        </script>
   </head>
  <body>

   <div id="topbar">
     <div id="title">One-Shot Anfrage</div>
   </div>
  <div id="content">
  <ul class="pageitem">
  <li class="textbox">
```

```
    <p>Geolocation API: <span id="message"></span></p>
  </li>
 </ul>
 <!-- Bereich für Kartenanzeige, Variable "map" -->
 <div style="width: 100%; height:200px" id="map"></div>
 </div>
   <div id="footer">
      <a class="noeffect" href="http://iwebkit.net">Powered by iWebKit</a>
   </div>
    </body>
</html>
```

Das Beispiel zeigt, wie einfach man Ortsinformationen wie Breitengrad (`latitude`), Längengrad (`longitude`) und Genauigkeit (`accuracy`) durch eine einfache Geolocation-API-Anfrage erhält. Weiterhin wird eine Google Maps-Karte angezeigt, die genau auf die ermittelte Geoposition zentriert wird.

Solange der Benutzer noch keinen Zugriff auf seine Geoposition erlaubt hat, werden weder Daten noch Karten angezeigt. Das Beispiel zeigt die Geopositionsdaten an und zentriert eine Google Maps-Karte auf der ermittelten Position.

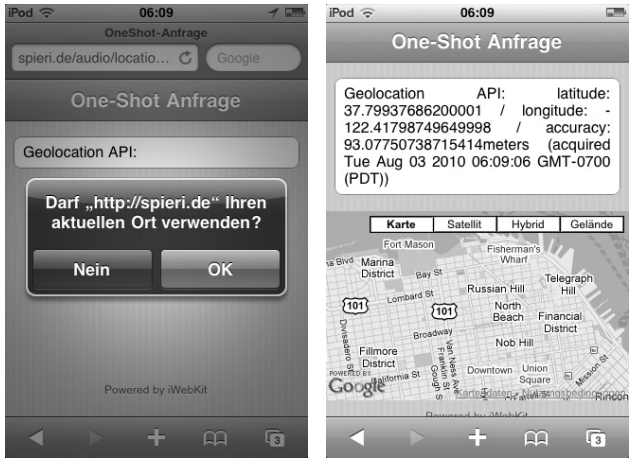


Bild 2.48: Erteilen der Zugriffserlaubnis und Anzeige der Geopositionsdaten.

2.5.6 Ortungsmethoden

Man kann grob zwei Gruppen von Ortungsmethoden unterscheiden: die der netzwerkbasierten Ortung und die der gerätebasierten Ortung. Netzwerkbasierte Ortung bedeutet, dass beispielsweise die mobile Cell-ID der aktuell vom Gerät benutzten Mobilfunkbasisstation benutzt wird, um eine grobe Ortung durchzuführen. Da heutzutage Geräte mit WLAN weit verbreitet sind, kann auch über die aktuell benutzte Gateway-IP-Adresse sowie die Stärke der empfangenen WLAN-Netze eine Ortung durchgeführt werden, die oftmals schon recht genau ist. Mit gerätebasierter Ortung ist praktisch fast ausschließlich die Ortung per GPS-Chips gemeint. Hier empfängt der in das Gerät ein-

gebaute GPS-Chip die Satellitensignale mehrerer GPS-Satelliten und kann dadurch die Position auf wenige Meter genau bestimmen.

Die netzwerkbasierten Ortungsmethoden sind meist schnell, ungenau und verbrauchen wenig Energie. Die gerätebasierten Ortungsmethoden sind langsamer, besonders bei der ersten Ortungsanfrage, jedoch deutlich genauer und verbrauchen leider – der GPS-Chip muss mit Energie versorgt werden – auch deutlich mehr Strom.

Wenn der `PositionOptions`-Parameter nicht angegeben wird, ist `enableHighAccuracy false`, `timout` unbegrenzt (Infinity) und `maximumAge 0`. Auf das `PositionOptions`-Objekt wird später noch genauer eingegangen.

Bei erfolgreicher Bestimmung der Position wird die `showMap`-Funktion unseres Beispiels aufgerufen. An diese Funktion wird ein Objekt mit folgenden Eigenschaften übergeben:

Coordinates coords

Die `coords`-Eigenschaft verweist auf ein weiteres Objekt, das die eigentlichen Positionsdaten laut World Geodic System (WGS84) enthält. Eigenschaften, deren Datentyp in der Tabelle unten mit `?` angegeben ist, sind optional, d. h., die jeweilige Implementierung muss das nicht unterstützen.

Inhalte des coords-Objekts	
`double latitude`	Geografische Koordinaten laut WGS84. (WGS84 ist das am weitesten verbreitete Koordinatensystem und wird beispielsweise durch GPS unterstützt.)
`double longitude`	Geografische Koordinaten laut WGS84.
`double accuracy`	Die Genauigkeit der Koordinaten, angegeben in Meter. Der Wert 75 gibt an, dass die Position, die durch `latitude` und `longitude` angegeben wurde, um bis zu 75 Meter von der realen Position abweichen kann.
`double? altitude`	Die Höhe in Metern oberhalb des WGS84-Ellipsoids. Falls die Implementierung das nicht unterstützt, wird null zurückgegeben.
`double? altitudeAccuracy`	Genauigkeit der Höhenangabe. Eine mögliche Abweichung wird in Metern angegeben. Falls nicht unterstützt, wird null zurückgegeben.
`double? heading`	Die aktuelle Fahrtrichtung, angegeben in Grad relativ zum Norden im Uhrzeigersinn. Null, falls nicht vorhanden.
`double? speed`	Falls vorhanden, die aktuelle Geschwindigkeit in Metern pro Sekunde. ansonsten null.

DOMTimeStamp timestamp

Ein DOMTimeStamp, der per `new Date(timestamp)` am einfachsten in ein leserliches Datum verwandelt wird. Der Timestamp gibt an, wann genau die in `coords` gespeicherten Informationen erhoben wurden.

2.5.7 PositionOptions genauer betrachtet

Als dritter Parameter kann sowohl an `getCurrentPosition` als auch an `watchPosition` ein Objekt übergeben werden, das die vom Entwickler erwartete Position genauer definiert. Die Kombination von `enableHighAccuracy`, `timeout` und `maximumAge` kann unter anderem dazu verwendet werden, eine aktuelle Positionierung zu erzwingen. Schauen wir uns das anhand eines Beispiels an;

```
// One-Shot-Anfrage, schlechte Idee ...
var options = {timeout:0, maximumAge:0, enableHighAccuracy:true};
navigator.geolocation.getCurrentPosition(showMap, handleError, options);
```

Eine so durchgeführte Anfrage an die Geolocation-API wird mit höchster Wahrscheinlichkeit fehlschlagen. `timeout=0` gibt an, dass das Ergebnis der Positionierung sofort verfügbar sein soll. Dies ist jedoch nur dann möglich, wenn kein neuer Positionierungsprozess gestartet wird, denn egal, welche Methode benutzt wird, ein `timeout=0` ist nicht machbar. Da gleichzeitig `maximumAge=0` und `enableHighAccuracy=true` angegeben werden, werden jedoch explizit frühere Ergebnisse abgelehnt, und die genaueste und damit zeitintensivste Methode wird ausgewählt.

Äußerst vielversprechend ist dagegen dieses Beispiel:

```
var options = {timeout:30000, maximumAge:0, enableHighAccuracy:false};
navigator.geolocation.getCurrentPosition(showMap, handleError, options);
```

Das Gerät wird in diesem Fall – sofern unterstützt – die Positionsbestimmung per IP-Adresse oder vorhandenen WiFi-Netzwerken auswählen, da `enableHighAccuracy=false` ist. Durch `maximumAge=0` wird ein neuer Bestimmungsprozess ausgelöst, der mit einem `timeout=30000` und somit 30 Sekunden jedoch üppig bemessen ist.

Folglich sieht ein Beispiel für eine möglichst exakte und aktuelle Position unter Inkaufnahme einer längeren Bestimmungsdauer so aus:

```
var options = {timeout:60000, maximumAge:0, enableHighAccuracy:true};
navigator.geolocation.getCurrentPosition(showMap, handleError, options);
```

Es können natürlich auch mehrere Anfragen zeitlich kombiniert werden. Beispielsweise kann eine Webapplikation zunächst eine Anfrage durchführen, die in aller Regel schnell, jedoch auch ungenau beantwortet wird. Sobald das erste Ergebnis vorliegt, kann eine zweite, genauere Anfrage gestartet werden, die dann im Hintergrund arbeitet. Der Benutzer interagiert dadurch direkt mit der Applikation, seine Position wird nach und nach genauer definiert. Ein weiteres Beispiel demonstriert das:

```
...
var $i = {};
$i.vars = {first:true};

//wird aufgerufen, sobald DOM ready
$i.init = function()
{
   if (Modernizr.geolocation)
   {
```

```
      //erste Anfrage, wir nehmen (fast) alles
      navigator.geolocation.getCurrentPosition($i.showLocation,
$i.handleError, {timeout:30000, maximumAge:600000,
enableHighAccuracy:false});
   }
};

$i.showLocation = function(position)
{
   var msg = 'latitude: ' + position.coords.latitude + ' / longitude: ' +
position.coords.longitude + ' / accuracy: ' + position.coords.accuracy +
'meters (acquired '+new Date(position.timestamp)+')';

   if ($i.vars.first)
   {
      document.getElementById('message').innerHTML = msg;
      $i.vars.first = false;
      //zweite Anfrage, nun mit hoher Genauigkeit und erzwungener Aktualität
      navigator.geolocation.getCurrentPosition($i.showLocation,
$i.handleError, {timeout:120000, maximumAge:0, enableHighAccuracy:true});
   }
   else
      document.getElementById('message').innerHTML += ('<br/><hr/>' + msg);

   var map = new google.maps.Map2(document.getElementById("map"));
   map.setCenter(new google.maps.LatLng(position.coords.latitude,
   position.coords.longitude), 13);


}
...
```

In obigem Beispiel wird also zuerst eine Anfrage mit dem `PositionOptions`-Objekt gestartet, die in der Regel zügig beantwortet wird:

```
{timeout:30000, maximumAge:600000, enableHighAccuracy:false}
```

Sobald das Ergebnis vorliegt, wird eine zweite Anfrage gestartet, diesmal geben wir uns nicht so einfach zufrieden:

```
{timeout:120000, maximumAge:0, enableHighAccuracy:true}
```

Auf der Demoseite sieht das dann folgendermaßen aus:

```
<html>
    <head>
        <title>HTML5</title>
      <meta name="layout" content="webkit" />

      <style type="text/css" media="screen">

      </style>
```

```
      <script src="${resource(dir:'js',file:'modernizr-1.1.min.js')}"
type="text/javascript"></script>
      <script type="text/javascript"
src="http://www.google.com/jsapi?key=ABQIAAAAk6AH6TdjmGE76Sg-
VZ_hzRSYyF8ynozO6JFpZ6CDrLxUILeJ4BRcA2RZA26oyr7iwO13JXdKrPSuCw"></script>


      <script>

      (function my() {

         var $i = {};
         $i.vars = {first:true};

         $i.init = function()
         {
            if (Modernizr.geolocation)
            {
               navigator.geolocation.getCurrentPosition($i.showLocation,
$i.handleError, {timeout:30000, maximumAge:600000,
enableHighAccuracy:false});
            }
            else
               document.getElementById('message').innerHTML = 'Sorry, no
Geolocation API available!';
         };

         $i.showLocation = function(position)
         {
            var msg = 'latitude: ' + position.coords.latitude + ' /
longitude: ' + position.coords.longitude + ' / accuracy: ' +
position.coords.accuracy + 'meters (acquired '+new
Date(position.timestamp)+')';

            if ($i.vars.first)
            {
               document.getElementById('message').innerHTML = msg;
               $i.vars.first = false;
               navigator.geolocation.getCurrentPosition($i.showLocation,
$i.handleError, {timeout:120000, maximumAge:0, enableHighAccuracy:true});
            }
            else
               document.getElementById('message').innerHTML += ('<br/><hr/>'
+ msg);

            var map = new google.maps.Map2(document.getElementById("map"));
             map.setCenter(new google.maps.LatLng(position.coords.latitude,
position.coords.longitude), 13);
```

```
        }

        $i.handleError = function(error)
        {
           var msgNode = document.getElementById('message');

           switch (error.code)
           {
              case error.TIMEOUT:
                 msgNode.innerHTML += 'timed out...';
                 navigator.geolocation.getCurrentPosition($i.showLocation,
$i.handleError, {timeout:30000, maximumAge:300000});
                 break;
              case error.PERMISSION_DENIED:
                 msgNode.innerHTML += 'user declined positioning';
                 break;
              case error.POSITION_UNAVAILABLE:
                 msgNode.innerHTML += 'position not available';
                 break;
           }

           msgNode.innerHTML +=  ' - error message: ' + error.message;
        }

        //document.addEventListener("DOMContentLoaded", $i.init, false);

        google.load("maps", "2.x");
        google.setOnLoadCallback($i.init);
     }).call();


     </script>
   </head>
   <body>

     <div id="topbar">
        <div id="title">Zunehmende Genauigkeit</div>
     </div>
     <div id="content">
        <ul class="pageitem">
           <li class="textbox">
              <p>Geolocation API: <span id="message"></span></p>
           </li>
        </ul>
        <div style="width: 100%; height:200px" id="map"></div>
     </div>
     <div id="footer">
        <a class="noeffect" href="http://iwebkit.net">Powered by
iWebKit</a>
```

```
      </div>
    </body>
</html>
```

2.5.8 Error-Callbacks

Die Implementierung eines Error-Handlers ist zwar optional, allerdings kann, wie Sie ja schon anhand der Beispiele gesehen haben, so einiges schiefgehen. Zunächst einmal braucht der Nutzer der Ortsbestimmung überhaupt nicht zuzustimmen. Und selbst wenn er die Zusage gegeben hat, kann eine Bestimmung der aktuellen Position aus einer Vielzahl technischer Gründe nicht möglich sein. Aus diesem Grund kann sowohl bei `watchPosition` also auch bei `getCurrentPosition` ein Error-Handler angegeben werden. Ein Error-Handler verweist per Referenz auf eine JavaScript-Funktion. An die Funktion wird im Fehlerfall ein `error`-Objekt übergeben. Das folgende Beispiel zeigt eine solche Funktion:

```
function handleError(error)
{
   var msgNode = document.getElementById('message');
   switch (error.code)
   {
      case error.TIMEOUT:
         msgNode.innerHTML += 'timed out...'; //nochmal versuchen?
         break;
      case error.PERMISSION_DENIED:
         msgNode.innerHTML += 'user declined positioning';
         break;
      case error.POSITION_UNAVAILABLE:
         msgNode.innerHTML += 'position not available';
         break;
   }
}
```

Das an die Funktion übergebene `error`-Objekt enthält drei konstante Werte, die Fehlerfälle. Anhand des Property-Codes kann festgestellt werden, welcher dieser Fälle eingetreten ist.

Fehlerfall	*Beschreibung*
TIMEOUT	Die Position konnte nicht festgestellt werden, da der im `PositionOptions`-Objekt angegebene Timeout überschritten worden ist.
PERMISSION_DENIED	Der Benutzer hat der Positionsbestimmung nicht zugestimmt.
POSITION_UNAVAILABLE	Die Position kann derzeit nicht festgestellt werden. Beispielsweise ist der Nutzer nicht online (keine Wifi-Netzwerke in Reichweite, keine Internetverbindung), oder die GPS-Satelliten können nicht erreicht werden.

Das `error`-Objekt enthält zudem das Property `message`, das eine kurze Beschreibung (auf Englisch) des aufgetretenen Fehlers enthält. Dieser Hinweis sollte nur während der Entwicklung in der UI angezeigt werden, da er nicht lokalisiert ist.

Je nach eingetretenem Fall sollte der Entwickler entsprechend mit der Situation umgehen. Falls ein Timeout eintritt, kann beispielsweise ein neuer Ortungsversuch mit höherem Timeout und/oder geringerer Genauigkeit unternommen werden. Falls der Nutzer der Positionsbestimmung nicht zugestimmt hat, kann die Applikation auf eine Standardposition zurückfallen oder dem Nutzer eine Erklärung dafür geben, wieso die angefragte Position zur Nutzung dieser Webseite notwendig ist. Dem Nutzer kann dann auch die Möglichkeit gegeben werden, erneut einer Anfrage zuzustimmen.

2.5.9 WatchPosition – ständige Positionsupdates

Mittels der `watchPosition`-Methode der Geolocation-Schnittstelle kann eine Applikation per JavaScript ständig aktualisierte Positionierungen empfangen. Je nach Implementierung unterschiedlich ist dabei die Häufigkeit, mit der diese Methode den Callback-Handler aufruft. Schauen wir uns zunächst aber ein einfaches Beispiel an, das der One-Shot-Anfrage sehr ähnlich ist:

```
//Callback-Funktion, wird aufgerufen, sobald Position vorhanden ist
function showMap(position) {
   var lat = position.coords.latitude;
   var lon = position.coords.longitude;
}

function handleError(error) {
   ...
}

var options = {timeout:30000, maximumAge:300000, enableHighAccuracy:true};
var watchId = navigator.geolocation.watchPosition(showLocation, handleError,
options);
```

Im Prinzip nichts bahnbrechend Neues, allerdings gibt die `watchPosition`-Methode eine numerische ID zurück, die zur Beendigung der ständigen Ortung benutzt werden kann. Um diese ständige Ortung also zu beenden, wird die `clearWatch`-Methode aufgerufen:

```
navigator.geolocation.clearWatch(watchID);
```

`clearWatch` hat keinen Rückgabewert. Falls die übergebene ID vorhanden ist, wird die ständige Ortung im Zusammenhang mit dieser ID sofort beendet. Es bietet sich also an, die ID, die von `watchPosition` zurückgegeben wird, umgehend in einer Variablen zu speichern. Damit kann dann später die Operation abgebrochen werden, falls die Ortung nicht mehr erwünscht ist.

Leider gibt die Geolocation-API-Spezifikation nicht vor, wie oft die `showMap`-Callback-Methode aus unserem Beispiel aufgerufen werden soll. Somit kann es passieren, dass Ihre Applikation mit Aufrufen überhäuft wird. Es bietet sich daher an, innerhalb der

Callback-Methode einen Check durchzuführen und die eigentliche Logik nur dann auszuführen, wenn beispielsweise einige Sekunden vergangen sind. Dies kann so implementiert werden:

```
var usedCallbacks = 0;
var callbacks = 0;
var lastUpdate = null;

function showLocation(position)
   {
      callbacks++;
      document.getElementById('message1').innerHTML = $i.vars.callbacks;

      if (lastUpdate)
      {
         var differenceInSeconds = (new Date().getTime() - lastUpdate) /
1000;
         //wenn weniger als 15 Sekunden vergangen sind, return
         if (differenceInSeconds < 15)
            return;
      }

      //es sind mehr als 15 sekunden vergangen, update
      lastUpdate = new Date().getTime();
      usedCallbacks++;
      document.getElementById('message2').innerHTML = $i.vars.usedCallbacks;

      ...
}
```

Die folgende kleine Beispielanwendung zeigt uns, wie oft die `showPosition`-Callback-Funktion tatsächlich aufgerufen wird:

```
<!-- HTML5 Geo-Position watchPosition
     Markus Spiering / Sven Haiges
-->
<!DOCTYPE html>
<html>
    <head>
       <title>watchPosition</title>
       <meta name="layout" content="webkit" />
      <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
      <link href="style.css" rel="stylesheet" media="screen" type="text/css"
/>

      <!-- Einladen/Aufruf der Google Maps-API Version 3 -->
      <script type="text/javascript"
          src="http://maps.google.com/maps/api/js?sensor=true">
      </script>
```

```
      <script>
      (function my() {

         var $i = {};
         $i.vars = {};
         $i.vars.callbacks = 0;
         $i.vars.usedCallbacks = 0;
         $i.vars.lastUpdate = null;

         $i.init = function()
         {
            if ($i.hasGeo)
            {
               $i.vars.watchID =
navigator.geolocation.watchPosition($i.showLocation, $i.handleError,
{timeout:30000, maximumAge:300000, enableHighAccuracy:true});
               document.getElementById('stop').addEventListener('click',
function(event) {
                  if ($i.vars.watchID !== null)
                  {
                     navigator.geolocation.clearWatch($i.vars.watchID);
                     alert('Cleared watch with ID:' +$i.vars.watchID);
                     $i.vars.watchID = null;
                     this.value = 'Beobachten starten';
                  }
                  else
                  {
                     $i.vars.watchID =
navigator.geolocation.watchPosition($i.showLocation, $i.handleError,
{timeout:30000, maximumAge:300000, enableHighAccuracy:true});
                     this.value = 'Beobachten stoppen';
                  }
               }, true);
            }
            else
               document.getElementById('message').innerHTML = 'Geo-Location
API nicht verf&uuml;gbar!';
         };

         $i.showLocation = function(position)
         {
            $i.vars.callbacks++;
            document.getElementById('message1').innerHTML =
$i.vars.callbacks;
            if (!$i.vars.map)
            {
                var latlng = new
google.maps.LatLng(position.coords.latitude, position.coords.longitude);
                var myOptions = {
```

```
                zoom: 17,
                center: latlng,
                mapTypeId: google.maps.MapTypeId.ROADMAP
              };
              $i.vars.map = new
google.maps.Map(document.getElementById("map"), myOptions);
           }

           if ($i.vars.lastUpdate)
           {
              var differenceInSeconds = (new Date().getTime() -
$i.vars.lastUpdate) / 1000;
              if (differenceInSeconds < 15)
                 return;
           }

           $i.vars.lastUpdate = new Date().getTime();
           $i.vars.usedCallbacks++;
           document.getElementById('message2').innerHTML =
$i.vars.usedCallbacks;

           // Create our "tiny" marker icon
          var image =
'http://www.google.com/intl/en_us/mapfiles/ms/micons/blue-dot.png';
          var markerLatLng = new
google.maps.LatLng(position.coords.latitude, position.coords.longitude);
          var beachMarker = new google.maps.Marker({
              position: markerLatLng,
              map: $i.vars.map,
              icon: image
          });


        }

        $i.handleError = function(error)
        {
           var msgNode = document.getElementById('message');

           switch (error.code)
           {
              case error.TIMEOUT:
                 msgNode.innerHTML += 'Zeit &uuml;berschritten...';
                 break;
              case error.PERMISSION_DENIED:
                 msgNode.innerHTML += 'Benutzer hat Positionierung
abgelehnt.';
                 break;
              case error.POSITION_UNAVAILABLE:
                 msgNode.innerHTML += 'Geo-Position nicht verf&uuml;gbar.';
```

```
                  break;
            }

            msgNode.innerHTML +=  ' - Fehlermeldung: ' + error.message;
         }

         $i.hasGeo = function()
         {
            if (navigator.geolocation)
               return true;
            else
               return false;
         };

         document.addEventListener("DOMContentLoaded", $i.init, false);

      }).call();


      </script>
    </head>
    <body>

      <div id="topbar">
         <div id="title">watchPosition</div>
      </div>
      <div id="content">
         <ul class="pageitem">
            <li class="textbox">
               <p><input type="button" id="stop" value="Beobachten
stoppen"/> Callbacks: <span id="message1"></span>/<span
id="message2"></span></p>
            </li>
         </ul>
             <!-- Bereich für Kartenanzeige, Variable "map" -->
         <div style="width: 100%; height:350px;" id="map"></div>
      </div>
      <div id="footer">
         <a class="noeffect" href="http://iwebkit.net">Powered by
iWebKit</a>
      </div>
    </body>
</html>
```

Es werden sowohl sämtliche Callbacks wie auch die tatsächlich für die Logik der Applikation benutzten Callbacks hochgezählt. Wird eine Position benutzt, wird auf einer Google Map ein Marker eingetragen. Das folgende Bild zeigt die Wanderung des Autors über den Zeitraum von 8 x 15 Sekunden in der Nähe des Yahoo!-Campus in Sunnyvale, CA.

Im linken Bild sehen Sie eine 120-sekündige Wanderung auf dem Yahoo!-Campus in Sunnyvale, CA. Die Anzeige erfolgt hier mit der Google Maps-API Version 2. Die daneben im Codebeispiel verwendete Google Maps-API Version 3 bietet dem Benutzer auch die Möglichkeit, in verschiedene Ansichten umzuschalten – hier eine Wanderung durch das Wohngebiet in Satellitenbildansicht.

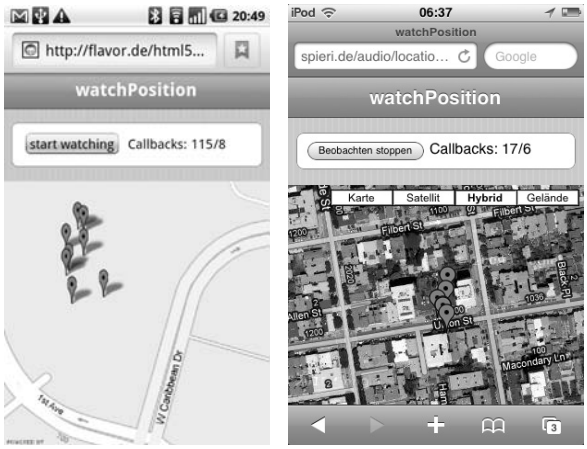


Bild 2.49: Wanderung auf dem Yahoo!-Campus (links) und im Wohngebiet.

Die beiden Zahlen neben den Text-Callbacks stellen die Anzahl sämtlicher Aufrufe der `showPosition`-Funktion und die tatsächlich benutzten Aufrufe dar (je 15 Sekunden wurde eine Position benutzt). Das bedeutet also grob ein Update pro Sekunde. Natürlich kann das je nach Implementierung und benutzter Bestimmungsmethode schwanken.

2.5.10 Reverse Geocoding mit Google Maps-API

Über die Geolocation-API erhalten Sie – vereinfacht ausgedrückt – die `latitude` und `longitude` des Geräts. Das ist häufig die Information, die man benötigt, um beispielsweise den aktuellen Ort via Google Maps anzeigen zu können. Oftmals möchte man jedoch die nächstmögliche mit dieser Position in Verbindung gebrachte Adresse haben. Dazu gibt es sogenannte Reverse Geocoding Services.

Ein prominentes Beispiel ist die Google Maps-Geocoding-API, die sich sowohl zur Umwandlung von Adressen in `latitude`/`longitude` als auch zum Reverse Geocoding, also dem Auflösen eines `latitude`/`longitude`-Paars in eine Adresse, eignet. Die Benutzung der API ist recht einfach:

```
function showAddress = function(response)
{
   if (!response || response.Status.code != 200) {
       alert("Status Code:" + response.Status.code);
    } else {
      var place = response.Placemark[0];
       document.getElementById('message').innerHTML = '<br/><hr/>' +
       '<b>orig latlng:</b>' + response.name + '<br/>' +
```

```
       '<b>latlng:</b>' + place.Point.coordinates[1] + "," +
place.Point.coordinates[0] + '<br>' +
       '<b>Status Code:</b>' + response.Status.code + '<br>' +
       '<b>Status Reqüst:</b>' + response.Status.reqüst + '<br>' +
       '<b>Address:</b>' + place.address + '<br>' +
       '<b>Accuracy:</b>' + place.AddressDetails.Accuracy + '<br>' +
       '<b>Country code:</b> ' +
place.AddressDetails.Country.CountryNameCode;
     }
}

var latlon = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);
new GClientGeocoder().getLocations(latlon, showAddress);
```

Auf einem mobilen Gerät sieht das innerhalb einer kleinen Demoapplikation folgendermaßen aus:

Zuerst ermittelt die Beispielanwendung die Geoposition des Benutzers und zeigt sie wie in den vorangegangenen Beispielen in Breiten- und Längengrad an. Nach dem Aufruf des Geocoders wissen Sie, dass der Breitengrad 37.799 und der Längengrad -122.417 zu dieser Adresse gehören: 1129 Union St, San Francisco, Kalifornien, Postleitzahl 94109, Vereinigte Staaten.

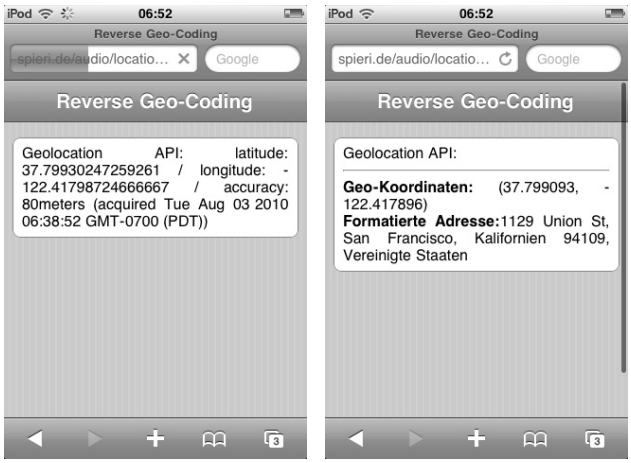


Bild 2.50: Beispiel eines Reverse Geocoding.

Wie auch zuvor finden Sie anschließend wieder den gesamten Code für ein Reverse-Geocoding-Beispiel:

```
<!-- HTML5 Geo-Position Reverse-Geocoding
     Markus Spiering / Sven Haiges
-->
<!DOCTYPE html>
<html>
  <head>
    <title>Reverse Geo-Coding</title>
```

```
    <meta name="layout" content="webkit" />
   <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
   <link href="style.css" rel="stylesheet" media="screen" type="text/css" />

   <!-- Einladen/Aufruf der Google Maps-API Version 3 -->
   <script type="text/javascript"
       src="http://maps.google.com/maps/api/js?sensor=true">
   </script>

   <script>
      (function my() {

      var $i = {};
      $i.vars = {};

      $i.init = function()
      {
         if ($i.hasGeo)
         {
            navigator.geolocation.getCurrentPosition($i.showLocation,
$i.handleError, {timeout:30000, maximumAge:300000, enableHighAccuracy:true})
            }
            else
               document.getElementById('message').innerHTML = 'Geo-Location
API nicht verf&uuml;gbar!';
         };

         $i.showLocation = function(position)
         {
            document.getElementById('message').innerHTML = 'latitude: ' +
position.coords.latitude + ' / longitude: ' + position.coords.longitude + '
/ accuracy: ' + position.coords.accuracy + 'meters (acquired '+new
Date(position.timestamp)+')';

            var latlng = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);
            vargeocoder = new google.maps.Geocoder();

             geocoder.geocode({'latLng': latlng}, function(results, status)
{
               if (status == google.maps.GeocoderStatus.OK) {
                 if (results[0]) {
                    $i.showAddress(results[0]) //0 = most accurate
                 }
               } else {
                  alert("Reverse Geocoding nicht möglich : " + status);
               }
             });

         }
```

```
         $i.showAddress = function(result)
         {
                document.getElementById('message').innerHTML = '<br/><hr/>'
+ '<b>Geo-Koordinaten: </b>' + result.geometry.location + '<br/>' +
'<b>Formatierte Adresse:</b>' + result.formatted_address + '<br>';
            }

         $i.handleError = function(error)
         {
            var msgNode = document.getElementById('message');

            switch (error.code)
            {
               case error.TIMEOUT:
                  msgNode.innerHTML += 'Zeit &uuml;berschritten...';
                  navigator.geolocation.getCurrentPosition($i.showLocation,
$i.handleError, {timeout:30000, maximumAge:300000});
                  break;
               case error.PERMISSION_DENIED:
                  msgNode.innerHTML += 'Benutzer hat Positionierung
abgelehnt.';
                  break;
               case error.POSITION_UNAVAILABLE:
                  msgNode.innerHTML += 'Geo-Position nicht verf&uuml;gbar.';
                  break;
            }

            msgNode.innerHTML +=  ' - Fehlermeldung: ' + error.message;
            }

            $i.hasGeo = function()
            {
               if (navigator.geolocation)
                  return true;
               else
                  return false;
            };

            document.addEventListener("DOMContentLoaded", $i.init, false);

         }).call();

         </script>
       </head>
       <body>

      <div id="topbar">
         <div id="title">Reverse Geo-Coding</div>
      </div>
      <div id="content">
         <ul class="pageitem">
            <li class="textbox">
```

```
            <p>Geolocation API: <span id="message"></span></p>
          </li>
        </ul>
      </div>
      <div id="footer">
        <a class="noeffect" href="http://iwebkit.net">Powered by
iWebKit</a>
      </div>
    </body>
</html>
```

2.5.11 Georeferenzierte Fotos von Flickr einbinden

Eine der reichsten Quellen für georelevante Inhalte ist die Foto-Sharing-Seite Flickr – *www.flickr.com*. Die mittlerweile zu Yahoo! gehörende Seite hat nicht nur circa 5 Milliarden Fotos zu bieten, sondern auch eine offene und einfach zu verwendende Programmierschnittstelle (API), um Daten von Flickr in eigenen Programmen oder Webseiten zu verwenden. Insbesondere für mobile Webseiten und Anwendungen ist das interessant, da Flickr den weltweit größten Pool an georeferenzierten Fotos im Internet bietet.

Das kleine Beispiel, das wir Ihnen jetzt vorstellen, ermittelt auf Abruf des Benutzers die aktuelle Position, ruft dann die Flickr-API auf und fragt nach Fotos, die in einem bestimmten Umkreis dieser Position aufgenommen wurden. Besucher einer fremden Stadt beispielsweise können auf einen Blick erkennen, welche Sehenswürdigkeiten und Plätze um sie herum sind. Das Ganze kann, wie im Beispiel gezeigt, auch mit Google Maps verbunden werden, um die Position der einzelnen Fotos auch auf einer Karte sichtbar zu machen. Neben dieser praktischen Anwendung zeigt Ihnen das Beispiel auch, wie Sie aus JavaScript eine Web-API aufrufen, mit empfangenen Daten dieser API umgehen und sie verwenden können.

Bild 2.51: Mash-up von Flickr-Fotos und Google Maps anhand der aktuellen Position eines Benutzers.

Um Daten von Flickr innerhalb einer Webseite oder Anwendung nutzen zu können, müssen Sie sich bei der Seite anmelden und einen API-Schlüssel beantragen. Sie können die Anmeldung auf *www.flickr.com* mit einer Yahoo!-ID durchführen und einen Flickr-Account erstellen oder eine neue Yahoo!-ID registrieren.

Sobald Sie Ihren eigenen Flickr-Account haben, können Sie nicht nur Fotos und Videos hochladen, sondern auch die API nutzen. Um einen API-Schlüssel zu beantragen, rufen Sie am besten die folgende Adresse für den Flickr App Garden auf: *www.flickr.com/services*.

Bild 2.52: Fertige Web Apps, die mit der Flickr-API erstellt wurden, können im App Garden veröffentlicht werden.

Im App Garden sind viele Hundert Anwendungen und Webseiten gelistet, die die Flickr-API verwenden – ein guter Start, um nicht nur einen Schlüssel zu beantragen, sondern auch, um sich ein paar Ideen zu holen. Einen API-Schlüssel beantragen Sie mit einem Klick auf *Beantragen Sie einen API-Schlüssel* auf der rechten Seite.

Das bringt Sie zu einer Seite, auf der Sie entscheiden müssen, ob Sie einen nicht kommerziellen oder einen kommerziellen API-Schlüssel benötigen. Einen nicht kommerziellen API-Schlüssel erhalten Sie sofort, einen kommerziellen Schlüssel hingegen erhalten Sie nur nach Prüfung durch Flickr.

Die genauen Bedingungen für einen kommerziellen API-Schlüssel entnehmen Sie der Webseite, aber wann immer Sie eine Anwendung oder Webseite mit Flickr-Inhalten oder Flickr-Funktionalität planen, die entweder verkauft werden soll oder für eine größere Firma oder Marke erstellt wird, ist in der Regel ein kommerzieller Schlüssel erforderlich.

Allerdings benötigen Sie den kommerziellen Schlüssel in diesen Fällen immer erst dann, wenn Sie die Integration oder die Webseite veröffentlichen. Für den Zeitraum der Entwicklung reicht stets der nicht kommerzielle Schlüssel, sofern das Produkt noch nicht veröffentlicht wird. Private Webseiten oder private Projekte brauchen in der Regel keinen kommerziellen Schlüssel. Nähere Informationen finden Sie im Flickr App Garden.

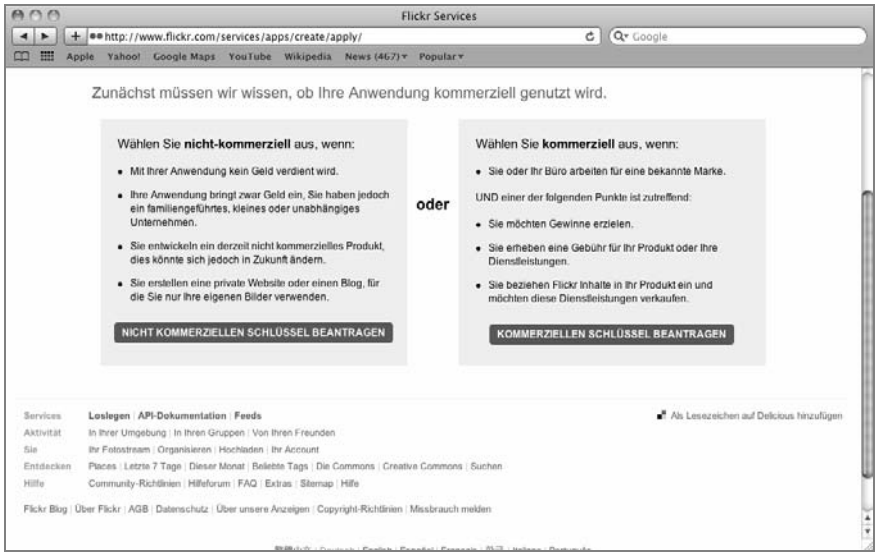


Bild 2.53: Flickr unterscheidet zwischen kommerziellen und nicht kommerziellen API-Schlüsseln. Nicht kommerzielle Schlüssel genügen während der Entwicklungsphase und werden sofort ausgestellt.

Wie bereits erwähnt: Selbst wenn Ihre Anwendung einen kommerziellen Schlüssel benötigt, können Sie bei der Entwicklung mit einem nicht kommerziellen Schlüssel arbeiten. Diesen beantragen Sie mit einem Klick auf *NICHT KOMMERZIELLEN SCHLÜSSEL BEANTRAGEN* und geben auf der folgenden Seite einige Informationen zu Ihrem geplanten Projekt ein:

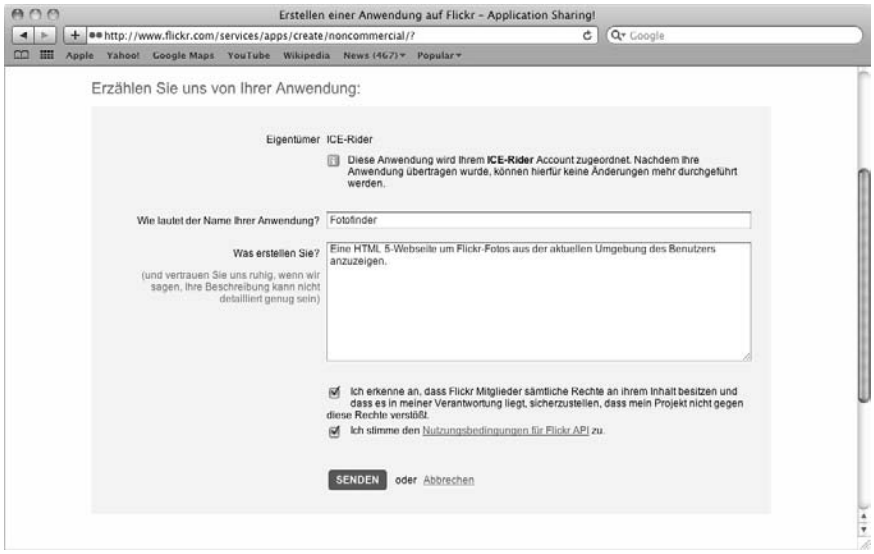


Bild 2.54: Bevor die API ausgestellt wird, fragt Flickr nach einigen Angaben zur geplanten Anwendung oder Webseite.

Wollen Sie die Angaben an Flickr übermitteln, lesen Sie sich bitte vorher die Nutzungsbedingungen der Flickr-API durch. Flickr legt großen Wert darauf, dass die Rechte der Fotos beim Fotografen liegen und diese Rechte auch von jedem Nutzer der API respektiert werden. Nachdem Sie die Nutzungsbedingungen gelesen und die richtigen Häkchen im Formular gesetzt haben, können Sie es an Flickr übermitteln und bekommen sofort, ohne weitere Überprüfung, einen neuen API-Schlüssel ausgestellt. Dabei erhalten Sie den API-Schlüssel sowie einen geheimen API-Schlüssel.

Bild 2.55: Für unser Beispiel benötigen Sie nur den normalen API-Schlüssel, nicht den geheimen Schlüssel.

Alle API-Anfragen bei Flickr benötigen den normalen API-Schlüssel. Der geheime API-Schlüssel wird lediglich bei API-Anfragen gebraucht, die auf persönliche Daten eines Flickr-Benutzers zugreifen. Für unser Beispiel werden wir nur öffentlich verfügbare Fotos über eine Suchanfrage abfragen und darstellen. Dafür reicht der normale API-Schlüssel aus.

Als Ausgangscode verwenden wir für das Beispiel die One-Shot-Anfrage, um die Position des Benutzers herauszufinden. Wie im One-Shot-Beispiel wird auch hier die Position in den Variablen `position.coords.longitude` und `position.coords.latitude` gespeichert. Die Google Maps-Karte wird ebenfalls zentriert um die Position des Benutzers aufgerufen – nahezu Zeile für Zeile wie im vorhergehenden One-Shot-Beispiel.

Anschließend kommt die Flickr-API ins Spiel. Um sie benutzen zu können, müssen wir den API-Schlüssel als Variable definieren:

```
var apiKey = 'Flickr API Key';
```

Als API-Call werden wir die Methode `flickr.photos.search` verwenden, die einfach die Bildsuchfunktion auf Flickr widerspiegelt. Sie können nicht nur nach Begriffen suchen, sondern über diese API auch eine Geoposition angeben und dann nach Fotos (oder auf Wunsch auch Videos oder Fotos und Videos) suchen, die in einem angegebenen Umkreis aufgenommen wurden:

```
var requestURL =
'http://api.flickr.com/services/rest/?method=flickr.photos.search&api_key='
+ apiKey + '&media=photos&lat=' + position.coords.latitude + '&lon=' +
```

```
position.coords.longitude + '&radius=2&extras=geo%2Curl_sq&format=
json&jsoncallback=apiCallback&per_page=20';
```

Mit den Ergebnissen von der Flickr-API wird dann in einem Fotocontainer für jedes empfangene Bild ein Thumbnail angezeigt:

```
imgElement = document.createElement('img');
imgElement.src = photos[i].url_sq
imgElement.hspace = 5
imgElement.vspace = 5
imgElement.alt = photos[i].title;
photoContainer.appendChild(imgElement);
```

Und dazu wird pro Foto noch eine Markierung auf die Google Maps-Karte gesetzt:

```
new google.maps.Marker({
   position: new google.maps.LatLng(photos[i].latitude,
photos[i].longitude),
      map: $i.map,
      title: photos[i].title});
```

Insgesamt ist das One-Shot-Beispiel nur um wenige Zeilen Code erweitert worden, bietet aber auf einmal nicht nur eine Landkarte, sondern auch völlig neue und relevante Inhalte. Nach dem Aufrufen der Webseite wird erst die Position des Benutzers abgefragt. Anschließend wird die Google Maps-Karte angezeigt, die Flickr-API aufgerufen und abhängig von den zurückkommenden Daten pro Bild ein kleines Thumbnail gezeigt sowie eine Markierung auf die Google-Karte gesetzt. Hier ist das Ergebnis in Form einer kompletten ausgeklappten Seite:

Bild 2.56: Mash-up von Flickr-Fotos und Google Maps anhand der aktuellen Position eines Benutzers.

Hier ist jetzt der gesamte Code des Beispiels. Bitte beachten Sie, dass Sie für den Flickr-API-Schlüssel einen eigenen Schlüssel in der Variablen `apiKey` spezifizieren müssen. Wie das geht, sehen Sie weiter oben. Andernfalls wird das Beispiel nicht funktionieren, und Sie bekommen keine Bilder.

```
<!-- Flickr / Geo-Positionsbeispiel
     Markus Spiering / Ross Harmes / Sven Haiges
-->
<!DOCTYPE html>
<html>
    <head>
        <title>Flickr-Fotos</title>
       <meta name="layout" content="webkit" />
      <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
      <link href="style.css" rel="stylesheet" media="screen" type="text/css"
/>

      <script type="text/javascript"
          src="http://maps.google.com/maps/api/js?sensor=true">
      </script>

      <script>

      (function my() {

         var $i = {};
         $i.vars = {};

         $i.init = function()
         {
            if ($i.hasGeo)
            {
               navigator.geolocation.getCurrentPosition($i.showLocation,
$i.handleError, {timeout:30000, maximumAge:300000, enableHighAccuracy:true})
            }
            else
               document.getElementById('message').innerHTML = 'Geolocation-
API nicht verfügbar!';
         };

         $i.showLocation = function(position)
         {
            document.getElementById('message').innerHTML = 'latitude: ' +
position.coords.latitude + ' / longitude: ' + position.coords.longitude + '
/ accuracy: ' + position.coords.accuracy + 'meters (acquired '+new
Date(position.timestamp)+')';

             var latlng = new google.maps.LatLng(position.coords.latitude,
position.coords.longitude);
             var myOptions = {
               zoom: 13,
               center: latlng,
               mapTypeId: google.maps.MapTypeId.ROADMAP
```

```
            };
            $i.map = new google.maps.Map(document.getElementById("map"),
myOptions);

            // Fotos im Umkreis von einem Kilometer holen

            var apiKey = 'Flickr-API-Schlüssel';
            var requestURL =
'http://api.flickr.com/services/rest/?method=flickr.photos.search&api_key='
+ apiKey + '&media=photos&lat=' + position.coords.latitude + '&lon=' +
position.coords.longitude +
'&radius=2&extras=geo%2Curl_sq&format=json&jsoncallback=apiCallback&per_page
=20';
            var scriptElement = document.createElement('script');
            scriptElement.src = requestURL;
            document.body.appendChild(scriptElement);
         }

         window.apiCallback = function(rsp) {

            var photos = rsp.photos.photo;

            if (!document.getElementById('photo-container')) {
               var photoContainer = document.createElement('div');
               photoContainer.id = 'photo-container';

   document.getElementById('message').appendChild(photoContainer);
            }
            else {
               var photoContainer = document.getElementById('photo-
container');
               photoContainer.innerHTML = '';
            }

            var imgElement;
            for (var i = 0, len = photos.length; i < len; i++) {
               if (photos[i].url_sq) {

                  // Thumbnail-Bild dem Container hinzufügen

                  imgElement = document.createElement('img');
                  imgElement.src = photos[i].url_sq
                  imgElement.hspace = 5
                  imgElement.vspace = 5
                  imgElement.alt = photos[i].title;
                  photoContainer.appendChild(imgElement);

                  // Markierung auf die Karte setzen

                  new google.maps.Marker({
                     position: new google.maps.LatLng(photos[i].latitude,
photos[i].longitude),
                     map: $i.map,
                     title: photos[i].title
```

```
                });
            }
         }
      };

      $i.handleError = function(error)
      {
         var msgNode = document.getElementById('message');

         switch (error.code)
         {
            case error.TIMEOUT:
               msgNode.innerHTML += 'Zeit &uuml;berschritten...';
               navigator.geolocation.getCurrentPosition($i.showLocation,
$i.handleError, {timeout:30000, maximumAge:300000});
               break;
            case error.PERMISSION_DENIED:
               msgNode.innerHTML += 'Benutzer hat Positionierung
abgelehnt.';
               break;
            case error.POSITION_UNAVAILABLE:
               msgNode.innerHTML += 'Geo-Position nicht verf&uuml;gbar.';
               break;
         }

         msgNode.innerHTML +=  ' - Fehlermeldung: ' + error.message;
      }

      $i.hasGeo = function()
      {
         if (navigator.geolocation)
            return true;
         else
            return false;
      };

      document.addEventListener("DOMContentLoaded", $i.init, false);

   }).call();


   </script>
 </head>
 <body>

   <div id="topbar">
      <div id="title">Fotos in ihrer N&auml;he</div>
   </div>
   <div id="content">
      <ul class="pageitem">
         <li class="textbox">
            <p>Geolocation API: <span id="message"></span></p>
```

```
            </li>
         </ul>
         <div style="width: 100%; height:200px" id="map"></div>
      </div>
      <div id="footer">
         <a class="noeffect" href="http://iwebkit.net">Powered by
iWebKit</a>
      </div>
    </body>
</html>
```

Dieses Beispiel hat Ihnen gezeigt, wie einfach es ist, eine Web-API anzusprechen, georeferenzierte Inhalte aus dem Netz zu holen und diese innerhalb seiner eigenen Webseite oder App zu verwenden. Stöbern Sie auch noch ein bisschen bei *www.flickr.com/services/api* und schauen Sie sich die anderen georelevanten API-Methoden an.

2.5.12 Weitere georelevante API-Methoden bei Flickr

Wir haben in unserem Beispiel die `flickr.photos.search`-API-Methode verwendet, die Geoinformationen als Eingangsparameter akzeptiert, aber noch keine der vielen »richtigen« Geo-API-Methoden bei Flickr darstellt.

Über die `flickr.photos.geo`-Methoden können Sie beispielsweise die Geoinformationen zu beliebten Fotos abrufen oder die georeferenzierten Fotos oder Videos eines bestimmten Benutzers zu einem bestimmten Ort erhalten. Auch die `flickr.places`-API-Methoden bieten Ihnen Geoinformationen, aber nahezu unabhängig von Fotos, wie beispielsweise Namen von Ländern, Regionen, Landkreisen, Städtenamen und Stadtteilen.

⊡ Lesezeichen

http:// www.flickr.com/services/api

Alle Flickr-API-Methoden finden Sie hier.

Dies ist ein weiterer Überblick über andere Anbieter, die nützliche georelevante APIs und Dienste anbieten:

Anbieter	*URL*	*Beschreibung*
SimpleGEO	*www.simplegeo.com*	Anstatt eine Vielzahl von verschiedenen Geodiensten einzeln zu verwenden, bietet SimpleGEO bereits eine Sammlung von georeferenzierten Daten aus diversen Quellen und bereitet diese für den Einsatz in Webanwendungen und Apps vor.
Cloudmade	*www.cloudmade.com*	Als Alternative zur Google Maps-API kann das Angebot von Cloudmade gesehen werden. Cloudmade bietet Zugriff auf modifizierte OpenStreet-Maps mit einer Vielzahl von zusätzlichen Geoinformationen.

Anbieter	*URL*	*Beschreibung*
Open Street Map	*www.openstreetmap.org*	Das Open Street Map-Projekt ist ein Kartendienst, der, ähnlich wie Wikipedia, von Benutzern aus der ganzen Welt bearbeitet wird. Open Street Map bietet in vielen Regionen der Welt eine bessere Kartendichte als beispielsweise Google Maps oder Bing Maps. Die offene API erlaubt es Entwicklern, Informationen von Open Street Maps zu bekommen, aber auch Daten zurück in die Datenbank zu speichern.
Foursquare	*www.foursquare.com/apps*	Das in New York ansässige Unternehmen bietet eine Plattform, über die Foursquare-Mitglieder in alle möglichen Orte »einchecken« können. Das Konzept ist recht spielerisch gehalten, und ein Benutzer kann immer sehen, wo seine Freunde gerade sind. Entwickler können die Foursquare-API beispielsweise dafür nutzen, populäre Locations rund um die Position des Benutzers vorzuschlagen.
Gowalla	*www.gowalla.com/api*	Alles, was wir gerade über Foursquare geschrieben haben, trifft mehr oder weniger auch auf Gowalla zu. Mittels des Diensts können Benutzer ebenfalls in diverse Locations »einchecken«, dies ihrem sozialen Zirkel mitteilen und eventuell Vergünstigungen erhalten. Die Konzepte und die API sind sehr ähnlich. Gowalla bietet beispielsweise im Gegensatz zu Foursquare bereits einen Fotoservice und ist in manchen Gegenden im Großen und Ganzen populärer als seine Konkurrenz aus New York.
Yelp	*www.yelp.de/developers*	Yelp wurde vor nicht allzu langer Zeit in Deutschland gestartet und bietet benutzergenerierte Bewertungen von jeglichen Diensten, beispielsweise von Restaurants, Läden, Ärzten etc. Die Adressen, die Geolocation und die Bewertungen von Dienstleistungsanbietern sind über die Yelp-API abrufbar.
Yahoo! PlaceFinder	*developer.yahoo.com/geo/placefinder*	Yahoo! PlaceFinder ist eine Alternative zur Google Maps-Reverse-Geocoding-API. Mit PlaceFinder können Koordinaten in Adressen und umgekehrt berechnet werden.

2.5.13 Geolocation mit jQTouch einbinden

Auch wenn wir versuchen, so wenig wie möglich vorgefertigte JavaScript-Bibliotheken in unseren Beispielen zu verwenden, möchten wir Ihnen dennoch die jQTouch-Funktion `jQT.updateLocation()` nicht vorenthalten. Sollten Sie ohnehin planen, Ihre

Webanwendung unter Nutzung von jQTouch zu erstellen, können Sie die aktuelle Position eines Benutzers mit nur wenigen Zeilen Code erhalten. Eine detaillierte Einführung zu jQTouch finden Sie am Anfang des Buchs.

Die Geofunktionaliät für jQTouch ist im Verzeichnis *Extensions* in der Datei *jqt.location.js* definiert. Diese muss mit in den HTML-Code geladen werden.

```
<!-- JavaScript-Erweiterung für Geolocation einbinden -->
<script type="application/x-javascript"
src="extensions/jqt.location.js"></script>
```

In der Datei *jqt.location.js* befindet sich die Funktion `updateLocation`, die wir mit einer Rückruffunktion versehen. Diese JavaScript-Funktion macht letztendlich genau das Gleiche, was wir in unserem vorhergehenden Beispielen gelernt haben, und erhält die Geoposition über den `navigator.geolocation.getCurrentPosition`-API-Aufruf. Weiterhin setzen wir die Variable `coords`, um festzustellen, ob wir von dem Gerät eine Geoposition erhalten können oder nicht. Die selbst definierte Funktion `setDisplay` ist nur für die Textausgabe verantwortlich.

Hier ist das volle Beispiel:

```
<html>
    <head>
        <title>Geo-Location mit jQTouch</title>

      <!-- jQuery direkt von Google-Code einbinden -->
      <script type="text/javascript"
src="http://www.google.com/jsapi"></script>
      <script type="text/javascript"> google.load("jqüry", "1.3.2"); </script>

      <!-- jQTouch einbinden -->
      <script type="text/javascript" src="jqtouch/jqtouch.js"></script>
      <link type="text/css" rel="stylesheet" media="screen"
href="jqtouch/jqtouch.css">
      <link type="text/css" rel="stylesheet" media="screen"
href="themes/jqt/theme.css">

      <!-- JavaScript-Erweiterung für Geolocation einbinden -->
        <script type="application/x-javascript"
src="extensions/jqt.location.js"></script>

       <!-- jQTouch initialisieren -->
      <script type="text/javascript">
         var jQT = $.jQTouch({ });
      </script>

        <script type="text/javascript" charset="utf-8">
            var jQT = new $.jQTouch({ });

            $(function(){
                function setDisplay(text) {
                    $('.info').empty().append(text)
                }

                var lookup = jQT.updateLocation(function(coords){
```

```
                    if (coords) {
                        setDisplay('Breitengrad: ' + coords.latitude + '<br
/>Längengrad: ' + coords.longitude);
                    } else {
                        setDisplay('Geo-Position wird nicht unterstützt.');
                    }
                });

                if (lookup) {
                    setDisplay('Aktuelle Position wird abgerufen');
                }
            });
        </script>

    </head>
    <body>
        <div>
            <div class="toolbar">
                <h1>Geo Location</h1>
            </div>
            <div class="info"></div>
        </div>
    </body>
</html>
```

Während das Beispiel noch versucht, die Geoposition zu erhalten, und die Variable `lookup` den Wert `true` enthält, wird der Benutzer darauf hingewiesen. Wurde die Position erfolgreich bestimmt, enthalten die Variablen `coords.latitude` den Breitengrad und `coords.latitude` den Längengrad.

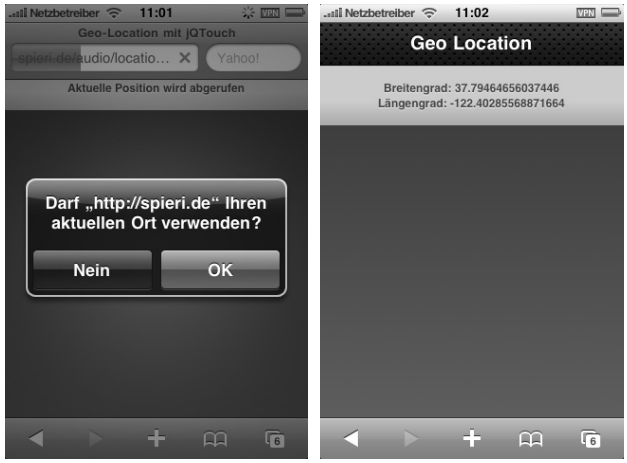


Bild 2.57: Nach dem Versuch, die Geoposition zu erhalten, wurde daneben die Position erfolgreich bestimmt. Breitengrad und Längengrad werden angezeigt.

2.5.14 Zusammenfassung und Ausblick

Dieser Teil des Buchs hat die W3C Geolocation-API vorgestellt, die Zugriff auf die aktuelle Position des Geräts ermöglicht. Es stehen zwei Methoden zur Verfügung, über die

die Position des Geräts einmalig oder ständig abgefragt werden kann – `getCurrentPosition` und `watchPosition`.

Eine erfolgreiche Positionsbestimmung hängt von mehreren Faktoren ab. Zunächst muss der Benutzer das notwendige Vertrauen in die Webapplikation haben und der Bestimmung durch die API zustimmen. Im nächsten Schritt müssen die Positionierungsparameter (`PositionOptions`), die an die oben genannten Methoden übergeben werden, passen.

Es bietet sich oftmals an, zunächst eine netzwerkbasierte Methode zu benutzen (`enableHighAccuracy=false`) und das Alter der Positionierung großzügig zu bemessen (`maximumAge=600000`). Falls diese Ortung glückt, kann im Hintergrund eine zweite Ortung durchgeführt werden, die dann eine höhere Genauigkeit und Aktualität hat.

Die Geolocation-API ist auf aktuellen Android-basierten Geräten sowie auf Apple iPhone und iPod touch (iPhone OS 3+) gleichermaßen gut implementiert. Insofern kann die Feature-Detection ganz einfach über die Existenz des `navigator.geolocation`-Objekts festgestellt werden. Auch die Methoden zum Abfragen der Position, `getCurrentPosition` und `watchPosition`, sind einheitlich implementiert. Da das derzeit der Fall ist, besteht wenig Grund, ein Toolkit speziell für Geolocation einzusetzen.

Dennoch gibt es bereits Geolocation-Toolkits, die beispielsweise unterschiedliche APIs abstrahieren. Im Desktop-Bereich ist Google Gears weit verbreitet und ermöglicht auch Internet Explorer-basierten Browsern den Zugriff auf Geolocation. Wenn Ihre Zielbrowser nicht nur WebKit-basiert sind, ist der Einsatz jedoch durchaus sinnvoll.

An dieser Stelle sei auf das Project *geo-location-javascript* (*http://bit.ly/ciWmAH*) hingewiesen, das u. a. Plattformen wie Nokias Web Run-Time (N97 etc.), BlackBerry Devices und Browser mit Google Gears durch eine einheitliche, der Geolocation-API nachempfundene API unterstützt.

Je nachdem, welche Funktionen Sie in Ihrer Webanwendung oder App planen, ist es gegebenenfalls sinnvoll, nicht die einzelnen APIs der verschiedenen Dienstanbieter aufzurufen, sondern beispielsweise auf SimpleGEO oder Cloudmade zurückzugreifen.

2.6 Web Storage

Ebenfalls ganz genau genommen nicht Teil der HTML5-Spezifikation, jedoch aktuell auf Android- und iPhone OS-basierten Geräten (und deren WebKit-basierten Browsern) verfügbar ist Web Storage. Web Storage ist eine W3C-Spezifikation und erlaubt das Speichern von beliebigen Daten bis zum Ende einer Session (Session Storage) oder darüber hinaus (Local Storage). Ganz wichtig dabei ist, dass es um die Speicherung von Daten auf Clientseite geht, ohne jegliche Serverunterstützung wie serverseitige Sessions. Zunächst scheint diese Art Speicher an Cookies zu erinnern, jedoch ist Web Storage einfacher zu benutzen, und der Speicherplatz pro »Origin« ist mit 5 MByte größer als die Menge der Daten, die mit Cookies gespeichert werden können.

Was ist ein Origin?
Ein Origin ist Tupel, bestehend aus »Schema, Host, Port« (*scheme/host/port*). Ein Beispiel ist *http://google.com*. Das Schema ist hier *http://*, der Host *google.com* und der implizite Standardport 80. Das Prinzip der Origins spielt bei Web Storage eine große Rolle, wenn es um den zur Verfügung stehenden Speicherplatz sowie die Sicherheit geht.
Das Storage-Limit pro Origin wird derzeit von den meisten Browsern einschließlich WebKit-basierten Browsern mit 5 MByte angegeben. Der Zugriff auf `localStorage` und `webStorage` wird ebenso per Origin bestimmt, d. h., alle Webseiten unterhalb eines Origins haben Zugriff auf den Web Storage. Durch DNS-Spoofing wird erreicht, dass ein Angreifer Zugriff auf die Daten des Web Storage bekommt. Um das zu verhindern, kann das Schema *https://* eingesetzt werden, wodurch DNS-Spoofing verhindert werden kann.
Web Storage sollte nicht eingesetzt werden, wenn sich mehrere Benutzer Webspeicher durch unterschiedliche Pfadnamen unterhalb eines Origins teilen. Beispielsweise kann der Benutzer von *http://meinserver.de/userA* auf sämtliche Daten des Benutzers unter *http://meinserver.de/userB* zugreifen, da der Origin in beiden Fällen *http://meinserver.de* ist.

2.6.1 Feature Detection: Darf Web Storage benutzt werden?

Web Storage und damit `sessionStorage` und `localStorage` stehen aktuell auf gängigen Android-basierten Geräten und auf iPhone OS zur Verfügung. Um über diese Grenzen hinaus Webseiten zu entwickeln oder einfach aus Gründen der Sicherheit, sollte jedoch per Feature Detection festgestellt werden, ob Web Storage benutzt werden kann:

```
function hasSessionStorage()
{
   return 'sessionStorage' in window;
};

function hasLocalStorage()
{
   return 'localStorage' in window;
};
```

Falls Sie ohnehin schon Modernizr verwenden:

```
if (!Modernizr.localstorage)
{
   alert ('No localStorage!');
   return;
}

...

if (!Modernizr.sessionstorage)
```

```
{
   alert ('No sessionStorage!');
   return;
}
```

2.6.2 sessionStorage und localStorage

Betrachten wir zunächst den `sessionStorage` genauer. Die Benutzung von `sessionStorage` und `localStorage` ist analog, sie unterscheidet sich lediglich in der Dauer der Datenspeicherung.

Der `sessionStorage` speichert alle Daten, die einer Browsersession zugehörig sind. Sobald das aktuelle Fenster oder Tab geschlossen wird, werden die damit verbundenen Daten des `sessionStorage` gelöscht. Pro Tab und Fenster stehen damit unabhängige Session Storages zur Verfügung, die sich gegenseitig (anders als Cookies) nicht beeinflussen.

Das Storage-Interface ist wie folgt definiert. Es ist exakt das Gleiche für `sessionStorage` und `localStorage`:

```
interface Storage {
  readonly attribute unsigned long length;
  getter DOMString key(in unsigned long index);
  getter any getItem(in DOMString key);
  setter creator void setItem(in DOMString key, in any data);
  deleter void removeItem(in DOMString key);
  void clear();
};
```

Beginnen wir mit der `getItem()`-Methode, die dazu verwendet wird, Daten zu speichern. Um unter dem Key `name` den Wert `WebStorage` abzulegen, würden wir Folgendes in JavaScript schreiben:

```
sessionStorage.setItem('name', 'WebStorage');
```

Recht einfach, oder? Um später (auf der gleichen Webseite auf einer Folgeseite) auf diesen Wert erneut zuzugreifen, schreiben wir:

```
var name = sessionStorage.getItem('name');
```

Anstatt der Methoden `setItem()`/`getItem()` kann auch die Kurznotation verwendet werden:

```
sessionStorage.name = 'WebStorage';
var name = sessionStorage.name;
```

Um die Anzahl der Keys abzurufen, kann das Attribut `length` des Storage verwendet werden:

```
var msg = 'Wir haben ' + sessionStorage.length + ' Elemente im
sessionStorage';
```

Dank des `length`-Attributs und der Methode `key()` können nun sehr einfach alle `key`/`value`-Paare des Storage abgerufen werden:

```
var i, key, value;
for (i = 0; i < sessionStorage.length; i++)
{
   key = sessionStorage.key(i);
   value = sessionStorage.getItem(key);
   existing.innerHTML += key + '=' + value + '<br/>';
}
```

`existing` zeigt hierbei auf ein bestehendes `div`-Element und wird pro Durchgang der `for`-Schleife um ein weiteres `key`/`value`-Paar angefüllt. Die Methode `key(n)` gibt also den Namen des Keys unter Position `n` zurück.

Um einen einzelnen Key und seinen Wert aus dem Speicher zu entfernen, wird die Methode `removeItem()` verwendet:

```
sessionStorage.removeItem('name');
alert('Die neue Größe des sessionStorage ist: ' + sessionStorage.length);
```

Um alle Elemente des Storage zu löschen, wird die Methode `clear()` aufgerufen:

```
sessionStorage.clear();
```

localStorage

Der `localStorage` wird dazu verwendet, Daten über die Grenzen der aktuellen Browsersession zu speichern. Alle Daten stehen auch nach einem Neustart des Browsers oder des Systems zur Verfügung. Ebenso stehen die Daten zur Verfügung, wenn das aktuelle Fenster geschlossen wird und der gleiche Origin später wieder geöffnet wird. `localStorage` bietet also sogenannten persistenten Speicher. Ein möglicher Anwendungsfall ist die Speicherung von Benutzereinstellungen für eine Webapplikation oder die Implementierung eines einfachen Zählers, um die Anzahl der Besuche eines einzelnen Benutzers zu zählen – wohlgemerkt, pro Origin, nicht pro Webseite.

Die Benutzung des `localStorage` ist identisch mit dem `sessionStorage`, daher erübrigt sich eine detaillierte Beschreibung. Um Daten im `localStorage` abzulegen und später wieder zu laden, schreiben wir also:

```
localStorage.setItem('visits', 5);
var visits = localStorage.getItem('visits');
```

Genau so wie beim `sessionStorage` werden auch die Methoden `removeItem()`, `clear()`, `key()` und `length` unterstützt.

clear() in sessionStorage und localStorage
Sowohl im `sessionStorage` als auch im `localStorage` führt `clear()` die gleiche Operation aus: Alle Elemente des angesprochenen Storage werden gelöscht. Im Fall des `sessionStorage` sind das jedoch nur die Elemente, die seit der laufenden Session im `sessionStorage` abgespeichert worden sind. Im Fall des `localStorage` kann das etwas weiter reichende Folgen haben, da damit alle Daten des `localStorage` eines Origins gelöscht werden. Wenn also eine Seite unter *http://meinserver.de/* eine Seite per JavaScript `clear()` aufruft, werden auch die `localStorage`-Daten der Seite *http://meinserver.de/andereSeite* gelöscht. Arbeiten also mehrere Personen gemeinsam an einer Website, sollte der Zugriff auf den `localStorage` koordiniert werden.
Ebenso sollte die Benutzung des `localStorage` generell aufgrund der 5-MByte-Beschränkung und die Vergabe der Keys koordiniert werden. Wenn `meinKey` bereits besteht, überschreibt `setItem('meinKey', 'meineDaten')` damit die alten Daten.

2.6.3 Beispiele zu localStorage und sessionStorage

Um `localStorage` und `sessionStorage` weiter zu verdeutlichen, folgt jetzt dazu je ein Beispiel. Um den `sessionStorage` zu demonstrieren, wurden zwei Webseiten erstellt, der Benutzer kann von Seite 1 auf Seite 2 surfen. Auf der ersten Seite wird per iWebKit-UI ein Toggle erstellt, das je nach Betätigung unter `sessionStorage.location` `true` oder `false` speichert:

Der Key `location` im `sessionStorage` wird mit `false` initialisiert. Die Liste der `sessionStorage`-Elemente zeigt lediglich dieses eine Element, nur wird der Location-Switch auf `on` geschaltet, entsprechend wird `location` nun mit `true` angezeigt.

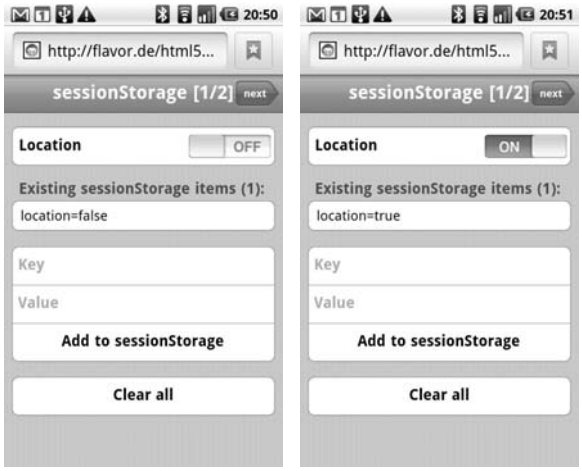


Bild 2.58: *Location* auf *OFF* und *Location* auf *ON*.

Es wird ein weiterer Wert hinzugefügt, der in der Liste erscheint. Mit dem Klick auf die nächste Seite wird die aktuelle Websession beibehalten, folglich kann auf den Wert von `location` zugegriffen werden.

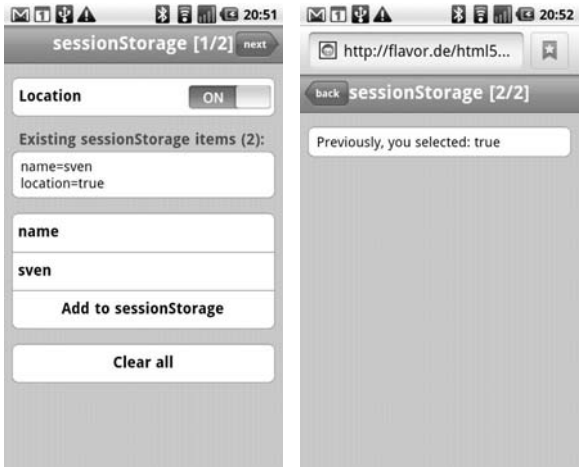


Bild 2.59: `sessionStorage` 1 von 2 und `sessionStorage` 2 von 2.

Hier ist der Code der ersten Seite:

```
<html>
    <head>
        <title>HTML5</title>
      <meta name="layout" content="webkit" />

      <style type="text/css" media="screen">

      </style>
      <script src="${resource(dir:'js',file:'modernizr-1.1.min.js')}"
type="text/javascript"></script>
      <script>

      (function my() {

         var $i = {};

         $i.init = function()
         {
            if (!Modernizr.sessionstorage)
            {
               alert ('No sessionStorage!');
               return
            }

            sessionStorage.location = false;
            document.getElementById('location').addEventListener('click',
function(e) {
               sessionStorage.location = e.target.checked;
               $i.updateDisplay();
            }, true);
```

```
            document.getElementById('count').textContent =
sessionStorage.length;

            //add key/values
            document.getElementById('add').addEventListener('click',
function(e){
               var sKey = document.getElementById('sKey').value;
               var sValue = document.getElementById('sValue').value;
               sessionStorage.setItem(sKey, sValue);
               $i.updateDisplay();
            }, true);

            //clear
            document.getElementById('clear').addEventListener('click',
function(e){
               sessionStorage.clear();
               $i.updateDisplay();
            }, true);

            $i.updateDisplay();

         };

         $i.updateDisplay = function() {
            var i, key;

            document.getElementById('count').textContent =
sessionStorage.length;

            var existing = document.getElementById('existing');
            existing.textContent = '';

            for (i = 0; i < sessionStorage.length; i++)
            {
               key = sessionStorage.key(i);
               Value = sessionStorage.getItem(key);
               existing.innerHTML += key + '=' + Value + '<br/>';
            }
         };



         document.addEventListener("DOMContentLoaded", $i.init, false);
      }).call();


      </script>
   </head>
   <body>

      <div id="topbar">
         <div id="title">sessionStorage [1/2]</div>
```

```
        <div id="rightnav"><a href="${createLink(action:'sessionstorage2')
}">next</a> </div>
      </div>
      <div id="content">
         <ul class="pageitem">
            <li class="checkbox"><span class="name">Location </span>
               <input id="location" type="checkbox" />
            </li>
         </ul>

         <span class="graytitle">Existing sessionStorage items (<span
id="count"></span>):</span>
         <ul class="pageitem">
            <li class="textbox">
               <p id="existing"></p>
            </li>
         </ul>

         <ul class="pageitem">
            <li class="bigfield">
               <input id="sKey" placeholder="Key" type="text" />
            </li>
            <li class="bigfield">
               <input id="sValue" placeholder="Value" type="text" />
            </li>
            <li class="button">
               <input id="add" type="submit" value="Add to sessionStorage" />
            </li>
         </ul>
         <ul class="pageitem">
            <li class="button">
               <input id="clear" type="submit" value="Clear all" />
            </li>
         </ul>
      </div>
    </body>
</html>
```

Und das ist der Code der zweiten Seite:

```
<html>
    <head>
        <title>HTML5</title>
      <meta name="layout" content="webkit" />

      <style type="text/css" media="screen">

      </style>
      <script src="${resource(dir:'js',file:'modernizr-1.1.min.js')}"
type="text/javascript"></script>
      <script>

      (function my() {
```

```
         var $i = {};

         $i.init = function()
         {
            if (!Modernizr.sessionstorage)
            {
               alert ('No sessionStorage!');
               return
            }

            document.getElementById('location').textContent =
sessionStorage.location;

         };

         document.addEventListener("DOMContentLoaded", $i.init, false);
      }).call();


      </script>
    </head>
    <body>

      <div id="topbar">
         <div id="title">sessionStorage [2/2]</div>
         <div id="leftnav"><a href="${createLink(action:'sessionstorage')
}">back</a> </div>
      </div>
      <div id="content">
         <ul class="pageitem">
            <li class="textbox">
               <p>Previously, you selected: <span id="location"></span></p>
            </li>
         </ul>
      </div>
    </body>
</html>
```

Das Beispiel für `localStorage` ist ähnlich aufgebaut, verzichtet jedoch auf eine zweite Webseite. Statt eines iWebKit-UI-Toggles, um die Location an- bzw. auszuschalten, wird die Anzahl der Seitenabrufe für diesen Benutzer hochgezählt. `localStorage.loadCount` wird ganz einfach so initialisiert:

```
if (!localStorage.loadCount)
   localStorage.loadCount = 0;
```

Um den Zähler `loadCount` um 1 zu erhöhen, wird zunächst der unter `loadCount` gespeicherte Wert per `parseInt()` in eine Zahl verwandelt und dann um 1 erhöht. Das HTML-Element mit der gleichnamigen ID `loadCount` dient dann als Container zur Anzeige des Werts.

```
localStorage.loadCount = parseInt(localStorage.loadCount, 10) + 1;
document.getElementById('loadCount').textContent = localStorage.loadCount;
```

Wenn der Key `loadCount` nicht vorhanden ist, wird er mit 1 initialisiert. Die Liste zeigt entsprechend den Wert an. Die Seite wird nun zwei weitere Male geladen, wodurch `loadCount` je um 1 erhöht wird und somit 3 ist. Es wurden nun per JavaScript zwei weitere Key/Value-Paare hinzugefügt, die in der Liste angezeigt werden.

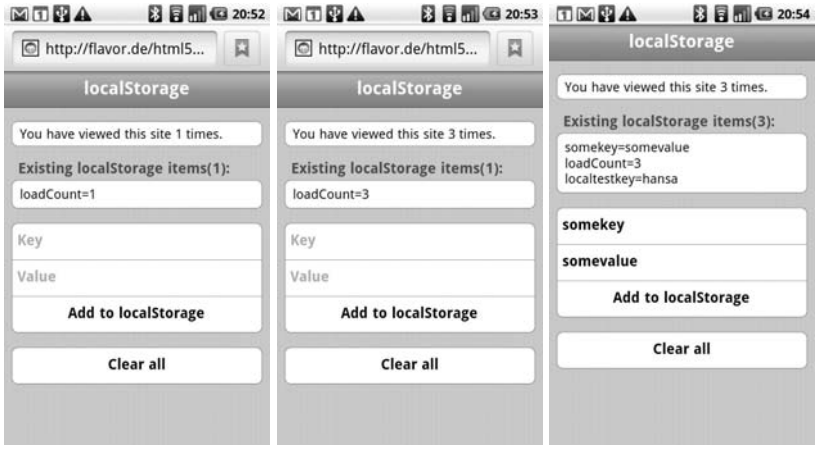


Bild 2.60: Beispiel einer `loadCount`-Operation.

Hier das vollständige Beispiel zur `localStorage`-Seite:

```
<html>
    <head>
        <title>HTML5</title>
      <meta name="layout" content="webkit" />

      <style type="text/css" media="screen">

      </style>
      <script src="${resource(dir:'js',file:'modernizr-1.1.min.js')}"
type="text/javascript"></script>
      <script>

      (function my() {

         var $i = {};

         $i.init = function()
         {
            if (!Modernizr.localstorage)
            {
               alert ('No localStorage!');
               return
            }

            if (!localStorage.loadCount)
               localStorage.loadCount = 0;

            localStorage.loadCount = parseInt(localStorage.loadCount, 10) + 1;
            document.getElementById('loadCount').textContent =
localStorage.loadCount;
```

```
            //add key/values
            document.getElementById('add').addEventListener('click',
function(e){
               var lKey = document.getElementById('lKey').value;
               var lValue = document.getElementById('lValue').value;
               localStorage.setItem(lKey, lValue);
               $i.updateDisplay();
            }, true);

            //clear
            document.getElementById('clear').addEventListener('click',
function(e){
               localStorage.clear();
               $i.updateDisplay();
            }, true);

            $i.updateDisplay();

         };

         $i.updateDisplay = function() {
            var i, key;

            document.getElementById('count').textContent =
localStorage.length;

            var existing = document.getElementById('existing');
            existing.textContent = '';

            for (i = 0; i < localStorage.length; i++)
            {
               key = localStorage.key(i);
               value = localStorage.getItem(key);
               existing.innerHTML += key + '=' + value + '<br/>';
            }
         };


         document.addEventListener("DOMContentLoaded", $i.init, false);
      }).call();

      </script>
   </head>
   <body>

      <div id="topbar">
         <div id="title">localStorage</div>
      </div>
      <div id="content">
         <ul class="pageitem">
            <li class="textbox">
               You have viewed this site <span id="loadCount">0</span> times.
            </li>
         </ul>
```

```
        <span class="graytitle">Existing localStorage items(<span
id="count"></span>):</span>
        <ul class="pageitem">
           <li class="textbox">
              <p id="existing"></p>
           </li>
        </ul>

        <ul class="pageitem">
           <li class="bigfield">
              <input id="lKey" placeholder="Key" type="text" />
           </li>
           <li class="bigfield">
              <input id="lValue" placeholder="Value" type="text" />
           </li>
           <li class="button">
              <input id="add" type="submit" value="Add to localStorage" />
           </li>
        </ul>
        <ul class="pageitem">
           <li class="button">
              <input id="clear" type="submit" value="Clear all" />
           </li>
        </ul>

     </div>
    </body>
</html>
```

2.6.4 Storage-Event

Sobald sich `localStorage` oder `sessionStorage` ändert, wird vom Browser ein Storage-Event gesendet. Dieses Event kann abgefangen werden, und so kann mit einer eigenen Logik darauf reagiert werden. Je nach Aktivität (neues Element unter Key X, Element unter X gelöscht etc.) sind unterschiedliche Attribute dieses Event-Objekts belegt. Hier zunächst das Interface des Storage-Events:

```
interface StorageEvent : Event {
  readonly attribute DOMString key;
  readonly attribute any oldValue;
  readonly attribute any newValue;
  readonly attribute DOMString url;
  readonly attribute Storage storageArea;
  void initStorageEvent(in DOMString typeArg, in boolean canBubbleArg, in
boolean cancelableArg, in DOMString keyArg, in any oldValueArg, in any
newValueArg, in DOMString urlArg, in Storage storageAreaArg);
};
```

Die Methode `initStorageEvent()` kann dazu verwendet werden, ein Storage-Event zu erstellen, was normalerweise nicht notwendig ist. Falls die Webapplikation jedoch Events emulieren soll, kann es hilfreich sein.

Viel wichtiger ist zunächst das Attribut `storageArea`, das immer gesetzt sein muss und auf den aktuell veränderten Storage zeigt: `localStorage` oder `webStorage`. Das Attribut `key` repräsentiert den aktuellen Key, der verändert worden ist.

Sofern ein Wert unter diesem Key vorhanden war, enthält `oldValue` eine Kopie dieses Werts, `newValue` repräsentiert den neuen Wert, während `url` die URL des aktuellen Dokuments enthalten sollte. In unseren Tests war `url` jedoch sowohl auf iPhone OS 4 als auch auf Android 2.1 immer `undefined`.

Um das Storage-Event zu empfangen, muss zunächst ein Listener registriert werden:

```
window.addEventListener('storage', function(e){
   var msgNode, msg = '';
   msgNode = document.getElementById('message');
   msg += 'StorageArea: ' + ((e.storageArea == localStorage) ?
"localStorage" : "sessionStorage") + '<br/>';
   msg += 'Document URL: ' + e.url + '<br/>';
   msg += 'Key: ' + e.key + '<br/>';
   msg += 'OldValue: ' + e.oldValue + '<br/>';
   msg += 'NewValue: ' + e.newValue + '<br/>';

   msgNode.innerHTML = msg;
}, true);
```

Schauen Sie sich online das Beispiel zum Storage-Event an, um dessen Ausprägungen zu verstehen. Unter sehen Sie drei Screenshots, die dieses Beispiel zeigen.

Durch Anklicken des ersten Buttons wird ein Datumsobjekt unter dem Key `myKey` im `sessionStorage` abgelegt. Das erneute Drücken überschreibt den Key `myKey`, somit stand ein `oldValue` zur Verfügung. Der Aufruf von `clear()` durch Anklicken von *Clear all* löscht alle Elemente und ist dadurch zu erkennen, dass `key` leer ist und `oldValue` und `newValue` null sind.

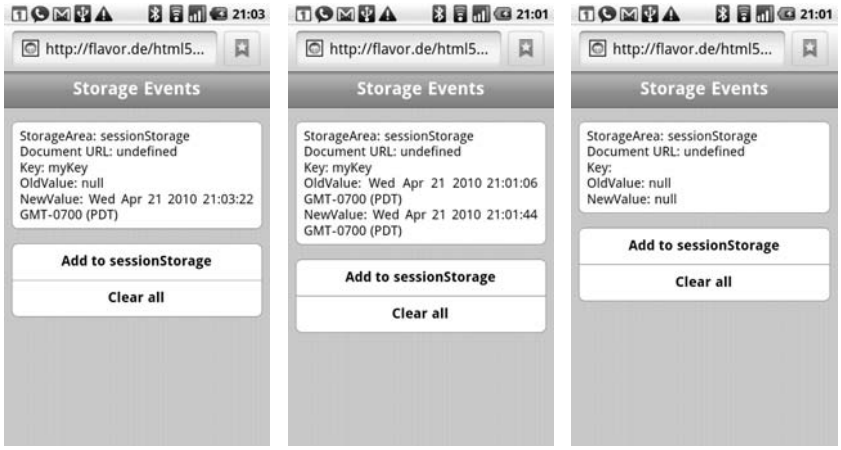


Bild 2.61: Ein Beispiel zum Storage-Event.

Hier der Code zum eben dargestellten Beispiel:

```
<!--
   Storage: storage events
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
<html>
<head>
        <title>HTML5</title>
      <meta name="layout" content="webkit" />

      <style type="text/css" media="screen">

      </style>
      <script src="${resource(dir:'js',file:'modernizr-1.1.min.js')}"
type="text/javascript"></script>
      <script>

      (function my() {

         var $i = {};

         $i.init = function()
         {
            if (!Modernizr.sessionstorage)
            {
               alert ('No sessionStorage!');
               return
            }

            window.addEventListener('storage', function(e){
               var msgNode, msg = '';
               msgNode = document.getElementById('message');
               msg += 'StorageArea: ' + ((e.storageArea == localStorage) ?
"localStorage" : "sessionStorage") + '<br/>';
               msg += 'Document URL: ' + e.url + '<br/>';
               msg += 'Key: ' + e.key + '<br/>';
               msg += 'OldValue: ' + e.oldValue + '<br/>';
               msg += 'NewValue: ' + e.newValue + '<br/>';

               msgNode.innerHTML = msg;
            }, true);

            //add
            document.getElementById('add').addEventListener('click',
function(e){
               sessionStorage.setItem('myKey', new Date().toString());
            }, true);

            document.getElementById('clear').addEventListener('click',
function(e){
               sessionStorage.clear();
            }, true);

         };
```

```
        document.addEventListener("DOMContentLoaded", $i.init, false);
      }).call();

      </script>
    </head>
    <body>

      <div id="topbar">
         <div id="title">Storage Events</div>
      </div>
      <div id="content">
         <ul class="pageitem">
            <li class="textbox">
               <p id="message"></p>
            </li>
         </ul>

         <ul class="pageitem">
            <li class="button">
               <input id="add" type="submit" value="Add to sessionStorage" />
            </li>
            <li class="button">
               <input id="clear" type="submit" value="Clear all" />
            </li>
         </ul>

      </div>
    </body>
</html>
```

2.7 Web SQL Database

Unter der Spezifikation Web SQL Database definiert das W3C eine clientseitige Datenbankschnittstelle. Genau genommen, hat die darin definierte API nichts mit HTML5 zu tun, jedoch fassen viele diese und andere Features wieder einmal unter dem Dach HTML5 zusammen. Unterstützung für diese API steht sowohl unter aktuellen Android-Geräten als auch auf iPhone OS-basierten Geräten zur Verfügung, und deshalb darf sie an dieser Stelle natürlich nicht fehlen.

Die Möglichkeit, per JavaScript auf eine im Browser definierte Schnittstelle zu einer relationalen Datenbank zugreifen zu können, ist mit Sicherheit eine der faszinierendsten APIs, die auf aktuellen mobilen Endgeräten zur Verfügung stehen. Allerdings ist dieses Thema auch ziemlich komplex, und zum Thema SQL (Structured Query Language) – der Sprache der relationalen Datenbanken – sind genügend dicke Bücher im Handel.

Um Neulingen dennoch einen kurzen Überblick zu geben, beginnen wir dieses Kapitel zunächst mit der Vorstellung der wichtigsten Konzepte relationaler Datenbanken und

SQL-Statements, um Tabellen zu erstellen und zu löschen, um Daten in eine Datenbanktabelle einzufügen, zu verändern und zu löschen.

Im nächsten Teil folgt dann die eigentliche Vorstellung der Web SQL Database-API. In unseren Tests haben wir zahlreiche Features entdeckt, die derzeit noch nicht implementiert sind. Daher beschränken wir uns auf die Vorstellung der Teile, die gut und konsistent implementiert sind. Die Vorstellung der API basiert auf einer kleinen Demoapplikation, mit der innerhalb einer einzigen HTML-Seite simple textbasierte Notizen verwaltet werden können. Den Code finden Sie weiter unten im Kapitel.

2.7.1 Grundlagen relationaler Datenbanken

Tabellen einer relationalen Datenbank enthalten die Daten in Zeilen und Spalten, ähnlich wie Excel-Worksheets. Die Zeilen stellen üblicherweise die Einträge dar, während die Spalten angeben, welche Informationen pro Eintrag gespeichert werden können. Um jeden Eintrag eindeutig identifizieren zu können, wird eine besondere Spalte als Primärschlüssel definiert, meist mit dem Namen »ID«. Eine einfache Notiztabelle könnte also so dargestellt werden:

ID	*Notiz*	*Datum*
1	Meine erste Notiz ...	01.05.2010 10:00:00
2	Notiz 2	01.05.2010 10:05:00

Diese Tabelle speichert also zwei Notizen, wobei jede Notiz eine eindeutige Primärschlüssel-ID hat sowie den Text der Notiz und ein Datum, wodurch die Notizen beispielsweise nach der neuesten Notiz zuerst sortiert werden können.

Um komplexere Daten zu speichern, bedienen sich relationale Datenbanken mehrerer Tabellen, die sich untereinander per Fremdschlüssel referenzieren. Ein Fremdschlüssel ist vereinfacht ausgedrückt der Primärschlüssel einer Tabelle, der jedoch innerhalb einer anderen Tabelle verwendet wird, um Daten zu referenzieren. Auf die unterschiedlichen Relationen, die durch Fremdschlüsselbeziehungen aufgebaut werden können, soll an dieser Stelle nicht eingegangen werden, da es hierzu bereits umfassende Literatur gibt.

2.7.2 SQL-Basics

Die Sprache der relationalen Datenbanken ist SQL, die Structured Query Language. SQL definiert die Syntax, um beispielsweise Tabellen in einer relationalen Datenbank anzulegen und Daten in diese Tabellen zu speichern. SQL kann grob aufgeteilt werden in Befehle, die einerseits Daten und Strukturen definieren und andererseits Daten erstellen, verändern oder löschen. Obwohl SQL ein Standard ist, gibt es je nach Datenbanksystem teils große Unterschiede und zahlreiche Erweiterungen. Zu wissen, welches Datenbanksystem genau verwendet wird, ist unerlässlich.

Derzeit wird jedoch praktisch nur ein einziges Datenbanksystem von allen Browsern verwendet: SQLite22. Obwohl das aus Entwicklersicht eine erfreuliche Nachricht ist, steht die W3C-Spezifikation dadurch derzeit leider still. Das Problem besteht darin, dass sich

sämtliche Implementierungen für SQLite entschieden haben, wodurch die Web SQL Database-Spezifikation nicht zum Standard erhoben werden kann. Es fehlt der Nachweis, dass die so geschriebene Spezifikation nicht nur von SQLite umgesetzt werden kann.

Das soll uns zunächst aber nicht kümmern. Betrachten wir nun eine kleine Auswahl aus SQL/SQLite, um grundlegende Datenbankoperationen verstehen zu können.

⊡ Lesezeichen

http://bit.ly/Jvnq

Eine vollständige Liste aller von SQLite verstandenen Befehle finden Sie hier.

CREATE TABLE

Erzeugt eine neue Datenbanktabelle.

```
CREATE TABLE [IF NOT EXISTS] tabellen_name (
 spalten_name spalten_typ [constraints],
 ...
);
```

Der Zusatz `IF NOT EXISTS` erzeugt die Tabelle nur, wenn es noch keine Tabelle mit exakt diesem Namen gibt. Die Spaltendefinitionen werden durch Kommata getrennt und sind nach dem Muster `spalten_name`, `spalten_typ`, `constraints` aufgebaut.

`constraints` gibt dabei weitere Integritätsbedingungen für die aktuelle Spalte an, häufige Werte sind `PRIMARY KEY` (Primärschlüssel), `NOT NULL` (keine NULL-Werte), `UNIQUE` (Wert muss eindeutig in der aktuellen Spalte sein), `DEFAULT` (ein Standardwert wird angegeben) und `AUTOINCREMENT` (Wert wird automatisch erhöht).

Die Tabelle der Notizenapplikation wird durch folgendes SQL erstellt:

```
CREATE TABLE IF NOT EXISTS notes (
   id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
   note TEXT,
   date DATE NOT NULL DEFAULT CURRENT_TIMESTAMP
);
```

INSERT

Fügt eine neue Zeile in eine Datenbanktabelle ein.

```
INSERT INTO tabellen_name (spalte_1, ...)
   VALUES (wert_1, ...);
```

Um eine neue Notiz in der `notes`-Tabelle zu speichern, muss nur eine Spalte angegeben werden. Die Spalten `id` und `date` beziehen ihren Wert automatisch, da das mit `CREATE TABLE` so definiert wurde. Um also eine neue Notiz zu speichern, schreiben wir in SQL:

```
INSERT INTO notes (note) VALUES ('Meine erste Notiz...');
```

SELECT

Liefert als Rückgabewert eine Anzahl Zeilen aus einer Tabelle.

```
SELECT spalte_1 [, ...] from tabelle_1 [, ...] [WHERE ausdruck]
   [ORDER BY spalte_1 ASC|DESC];
```

Die `WHERE`-Klausel kann dazu verwendet werden, die zurückgegebenen Spalten zu beschränken. Beispielsweise würde `WHERE id > 5` nur die Zeilen zurückliefern, die einen ID-Wert höher als 5 haben. Wenn Sie alle Spalten als Rückgabe erhalten möchten, kann auch `*` angegeben werden. Folgende `SELECT`-Query liefert alle Spalten und alle Zeilen unserer `notes`-Tabelle, geordnet nach der Spalte `date` absteigend:

```
SELECT * FROM notes ORDER BY date DESC;
```

UPDATE

Verändert die Werte einer bestehenden Zeile.

```
UPDATE tabellen_name SET spalte_1=wert_1 [, ...]
 [WHERE expression];
```

Die neuen Werte werden dabei in der Form `spalte=wert` angegeben und durch Kommata getrennt. Die `WHERE`-Klausel kann dazu benutzt werden, die Werte exakt einer Zeile und nicht aller Zeilen zu verändern. Um eine Notiz in der `notes`-Tabelle zu aktualisieren, schreiben wir:

```
UPDATE notes set note='Aktualisierte Notiz', date=CURRENT_TIMESTAMP WHERE
id=5;
```

DELETE

Löscht Zeilen aus einer Tabelle.

```
DELETE FROM tabellen_name [WHERE ausdruck];
```

Die optionale `WHERE`-Klausel gibt dabei an, welche Zeilen gelöscht werden sollen. Wird diese Klausel weggelassen, wird die gesamte Tabelle gelöscht:

```
DELETE FROM notes;
```

Um eine einzelne Zeile zu löschen, müssen Sie das `DELETE`-Statement weiter eingrenzen:

```
DELETE FROM notes WHERE id = 5;
```

DROP TABLE

Löscht eine komplette Tabelle inklusive ihrer Definition.

```
DROP TABLE tabellen_name;
```

Um also die `notes`-Tabelle komplett zu entfernen, schreiben wir:

```
DROP TABLE notes;
```

Eine so gelöschte Tabelle könnte nun mit dem `CREATE TABLE`-Statement wieder angelegt werden.

2.7.3 Transaktionen

Transaktionen sind ein wichtiger Bestandteil relationaler Datenbanken. Eine Transaktion stellt eine atomare Einheit dar und besteht aus mehreren Befehlen, die entweder alle erfolgreich ausgeführt werden können oder alle zurückgenommen werden.

Transaktionen verhindern also, dass eine zusammengehörige Kette von Operationen nur halb durchgeführt wird und damit die Datenbank in einem unzulässigen und fehlerhaften Zustand hinterlässt. Die Web SQL Database-API unterstützt Transaktionen. Noch besser: Sämtliche SQL-Befehle werden immer innerhalb einer Transaktion ausgeführt.

2.7.4 Database-API

Bevor wir uns die umfangreiche Web SQL Database-API ansehen, sollten wir zunächst noch ein paar Fakten zu dieser API und den damit zusammenhängenden Datenbanken betrachten.

Zunächst gilt für den Zugriff auf die Datenbanken ähnlich wie beim `session`- und `localStorage` der Web Storage-API das Origin-Prinzip. Alle Webseiten desselben Origins können also in vollem Umfang auf die Daten zugreifen, diese modifizieren und löschen.

Jede Datenbank, die mit der Database-API erzeugt wird, hat einen Zugriffsnamen und eine Version. Anhand der Version kann später das Datenbankschema auf eine neuere Version migriert werden, da bestehenden Tabellen eventuell neue Spalten hinzugefügt wurden.

Um eine Datenbank zu öffnen, muss immer der Name der Datenbank bekannt sein. Es gibt keine Möglichkeit, sämtliche Datenbanken eines Origins durch diese API aufzulisten oder zu löschen. Sie können zwar alle Tabellen einer Datenbank droppen. Ist die eigentliche Datenbank jedoch einmal angelegt, kann sie nicht über diese API gelöscht werden.

Beginnen wir damit, die Web SQL Database-API zu erforschen. Um eine vorhandene Datenbank zu öffnen oder eine neue Datenbank entsprechenden Namens anzulegen, benutzen Sie die `openDatabase`-Methode, die wie nachfolgend definiert ist. (Die Web SQL-Spezifikation sieht weitere Methoden vor, beispielsweise für den synchronen Zugriff auf die Datenbanken. Diese Methoden scheinen derzeit weder auf WebKits in Android 2.1 noch bei iPhone OS 4 implementiert zu sein und werden deshalb hier nicht vorgestellt.)

```
Database openDatabase(in DOMString name, in DOMString version,
 in DOMString displayName, in unsigned long estimatedSize,
 in optional DatabaseCallback creationCallback);
```

Um die `notes`-Datenbank zu öffnen, schreiben wir also das:

```
var db = openDatabase('notes', '1.0', 'Offline Notes Storage', 5*1024*1024);
```

Hiermit öffnen oder erstellen Sie eine SQLite-Datenbank mit dem Namen `notes` und der Version 1.0. Der dritte Parameter gibt die Beschreibung der Datenbank an, während

der vierte Parameter die erwartete Größe der Datenbank darstellt. In diesem Beispiel werden 5 MByte angegeben; dadurch zeigen viele Browser keinen zusätzlichen Hinweis an, da 5 MByte die in der Spezifikation empfohlene maximale Größe der Datenbanken darstellt. Sie können mehr Speicherplatz erhalten, jedoch kann der Browser in diesem Fall dem User einen Hinweis einblenden, dem dieser erst zustimmen muss.

2.7.5 Neue Datenbank in den iPhone-Einstellungen

Die nun erstellte SQLite-Datenbank kann jetzt auf iPhone, iPod touch oder iPad unter *Einstellungen/Safari/Datenbanken/notes* gefunden werden. Zwar ist das interessant zu beobachten, aber für die Entwicklung ist diese Ansicht leider wenig hilfreich. Die einzigen Informationen, die Sie hier entnehmen können, sind der vorhandene und der belegte Speicherplatz. Ebenso wie jeder andere Benutzer später können Sie, falls gewünscht, die Datenbanken auch löschen.

Bild 2.62: Neue Datenbank unter iOS und unter Android.

Unter Android (2.2) haben Sie derzeit leider noch weniger Einsicht in die auf dem Gerät erstellten SQLite-Datenbanken. Sobald vom aktuellen Origin eine Datenbank erstellt wurde, können Sie lediglich über *Menü/Mehr/Einstellungen/Cache löschen* den gesamten Webcache inklusive aller Datenbanken löschen.

Wie Ihnen sicherlich aufgefallen ist, haben wir das optionale `creationCallback` nicht angegeben, da es derzeit von keiner Implementierung unterstützt wird. Das `creationCallback` sollte eine JavaScript-Funktion darstellen, die nur dann aufgerufen wird, wenn die Datenbank soeben erstellt wurde. Es würde sich dann anbieten, in dieser Funktion das Schema der Datenbank (`CREATE TABLE`) durchzuführen, aber wie gesagt, dies ist leider nicht implementiert.

Allerdings müssen Sie nicht umständlich erkennen, ob Tabellen in Ihrer Datenbank bereits vorhanden sind. Glücklicherweise lassen sich die `CREATE TABLE`-Statements mit dem Zusatz `IF EXISTS` versehen, wodurch eine Tabelle nur dann angelegt wird, wenn

sie noch nicht existiert. Somit kann mit jedem Zugriff auf die HTML-Seite der gleiche JavaScript-Code ausgeführt werden. In unserem Fall sieht die Erzeugung der `notes`-Tabelle (in der `notes`-Datenbank) so aus:

```
var errorCallback = function(error)
{
   alert('errorCallBack: ' + error.code + ' / message: ' + error.message);
}

db.transaction(function(t){
   t.executeSql('CREATE TABLE IF NOT EXISTS notes
      (id INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT,
       note TEXT, date DATE NOT NULL DEFAULT CURRENT_TIMESTAMP);');

}, errorCallback);
```

Um SQL im Rahmen einer Transaktion auszuführen, benutzen Sie die Methode `transaction` der Datenbankreferenz. Der erste Parameter stellt eine Callback-Funktion dar, an die ein Transaktionsobjekt übergeben wird (hier `t`). Die Methode `executeSql()` kann nun dazu benutzt werden, beliebiges SQL auszuführen. In unserem Fall erzeugen wir die `notes`-Tabelle, falls sie noch nicht existiert.

Wie Sie ebenfalls sehen können, haben wir einen zweiten Parameter angegeben, der auch eine Callback-Funktion darstellt. Das `errorCallback` oben wird aufgerufen, sobald die Transaktion missglückt ist. In diesem Fall wird ein `SQLError`-Objekt an die Funktion übergeben, das zwei wichtige Attribute hat: `code` und `message`. `message` stellt eine (englische) Beschreibung des Fehlers dar und sollte nur während der Entwicklungszeit verwendet werden. Der folgenden Tabelle können Sie alle Fehlercodes entnehmen:

Konstante	*Code*	*Bedeutung*
`UNKNOWN_ERROR`	0	Ein unbekannter Fehler ist aufgetreten, der nicht weiter spezifiziert werden kann.
`DATABASE_ERR`	1	Das SQL-Statement ist fehlgeschlagen, Grund kann ein fehlerhaftes Statement sein, das so von der Datenbank (SQLite) nicht verstanden wird.
`VERSION_ERR`	2	Ein Fehler ist aufgetreten, weil die Version der Datenbank nicht der Version entsprach, die erwartet wurde.
`TOO_LARGE_ERR`	3	Die zurückgelieferte Datenmenge ist zu groß, es muss eventuell der Modifizierer `LIMIT` benutzt werden, um die Anzahl der Zeilen zu beschränken.
`QUOTA_ERR`	4	Das Statement schlug fehl, da die maximale Größe der Datenbank überschritten wurde. Sofern der Benutzer nach Zustimmung zur Erhöhung der Speichermenge gefragt wurde, hat er abgelehnt.

Konstante	*Code*	*Bedeutung*
SYNTAX_ERR	5	Das SQL-Statement schlug aufgrund der Syntax fehl, oder die Anzahl der Argumente im SQL-Statement stimmt nicht mit der Anzahl der Platzhalter überein.
COSTRAINT_ERR	6	Ein SQL-Statement schlug aufgrund eines Datenbank-Constraints fehl.
TIMEOUT_ERR	7	Das Transaktionsobjekt konnte nicht innerhalb der vorgesehenen Zeit erzeugt werden.

Betrachten wir nochmals die `transaction()`-Methode der Datenbankreferenz. An diese Methode kann falls gewünscht auch ein drittes Callback übergeben werden, das im Erfolgsfall nach der Transaktion aufgerufen wird. Diese Methode kann praktisch sein, wenn Sie innerhalb der Transaktion beispielsweise neue Zeilen in eine Tabelle einfügen. Somit muss nach geglückter Transaktion oftmals die UI aktualisiert werden, was dann hier veranlasst werden kann. Die Definition der `transaction()`-Methode sei nun nochmals vorgestellt:

```
void transaction(in SQLTransactionCallback callback,
 in optional SQLTransactionErrorCallback errorCallback,
 in optional SQLVoidCallback successCallback);
```

2.7.6 Daten speichern

Um Daten zu speichern, müssen die `SQL INSERT`-Statements an die Datenbank übergeben werden. Um beim Notizenbeispiel zu bleiben, müssen Sie den Text der neuen Notiz in das `SQL INSERT`-Statement aufnehmen:

```
var statementSuccessCallback = function(t, resultSet){
   displayNotes(); //aktualisiert die UI
}

var statementErrorCallback = function(t, error){
   alert('errorCallBack: ' + error.code + ' / message: ' + error.message);

}

db.transaction(function(t){
   t.executeSql('INSERT INTO notes (note) VALUES (?);',
      [newNoteText],
      statementSuccessCallback,
      statementErrorCallback);
});
```

Rufen Sie die bereits bekannte `transaction()`-Methode auf und benutzen Sie die `executeSql()`-Methode der übergebenen Transaktion. In diesem Fall haben wir für die `transaction()`-Methode selbst keine Callbacks angegeben. An dieser Stelle werden die Callbacks der `executeSql()`-Methode benutzt, die im Gegensatz zur `transaction()`-Methode verwirrenderweise in umgekehrter Reihenfolge angegeben werden.

Die Funktion `statementSuccessCallback` wird also nach geglücktem Ausführen des SQL-Statements aufgerufen. Da Sie innerhalb dieser Transaktion nur ein SQL-Statement ausführen, hätten Sie genauso gut das `successCallback` der Transaktion benutzen können. Das `statementErrorCallback` wird natürlich aufgerufen, wenn das Statement nicht geglückt ist.

Während beide Callbacks die aktuelle Transaktion (durch die z. B. erneut SQL ausgeführt werden kann, eventuell um einen Fehler in einer Fehlertabelle zu loggen) übergeben bekommen, wird an das `statementSuccessCallback` als zweiter Parameter ein `SQLResultSet`-Objekt übergeben, an das `statementErrorCallback` jedoch ein `SQLError`-Objekt.

Das `SQLResultSet`-Objekt ist in obigem Beispiel nicht von Bedeutung, da wir die Datenbank nicht abgefragt haben. Im nächsten Abschnitt gehen wir jedoch darauf ein.

Betrachten wir zunächst noch dieses SQL-Statement, das Ihnen sicherlich aufgefallen ist:

```
INSERT INTO notes (note) VALUES (?);
```

Das `?` stellt dabei einen Platzhalter dar, eine Art Makro, das später von der API aufgelöst wird. An die Stelle des `?` wird ein Wert eingefügt, der in einem JavaScript-Array des zweiten Parameters an gleicher Position enthalten sein muss. In diesem Beispiel wird nur ein einziger Platzhalter benutzt, also darf das Array auch nur ein Element enthalten:

```
var newNoteText = 'Neue Notiz...';
...
t.executeSql('INSERT INTO notes (note) VALUES (?);',
 [newNoteText],
 statementSuccessCallback,
 statementErrorCallback);
```

2.7.7 SQL Injection

Um die neue Notiz in die Datenbank zu speichern, haben wir uns bewusst für ein SQL-Statement mit Platzhalter `(?)` entschieden. Die Platzhalter werden der Reihe nach durch die Elemente eines im zweiten Parameter angegebenen Arrays ersetzt.

```
t.executeSql('INSERT INTO notes (note) VALUES (?);', [newNoteText], ...
```

Im Vergleich dazu hätten wir das `INSERT`-Statement auch durch Zusammenfügen erstellen können:

```
t.executeSql('INSERT INTO notes (note) VALUES (\'+newNoteText+\');', null,
...
```

Das birgt zweierlei Probleme: Zunächst wird das SQL-Statement durch das nun notwendige Escaping der `'` relativ unleserlich. Das ließe sich noch teilweise lösen, indem für den eigentlichen String in JavaScript doppelte Anführungszeichen verwendet würden:

```
t.executeSql("INSERT INTO notes (note) VALUES ('"+newNoteText+"');", null,
...
```

Leider gibt es aber immer noch ein gravierendes Problem. Was, wenn der Benutzer beispielsweise dies als `newNoteText` in eine Textbox eintippt:

```
'); DROP TABLE notes; --
```

Sie haben es erahnt: Durch diese Eingabe würde neben dem ersten `INSERT`-Statement, das hier irrelevant ist, ein zweites Stück SQL ausgeführt – in diesem Fall `DROP TABLE notes;`, das die gesamte `notes`-Tabelle löscht. Die beiden Bindestriche am Ende der Eingabe sorgen dafür, dass der restliche Text des Statements als Kommentar von SQLite ignoriert wird.

Wenn die Daten, die Sie in der Datenbank speichern, nicht mit einem Server synchronisiert werden, schadet sich der Nutzer damit selbst. Das mag eventuell noch kein Problem darstellen, was aber, wenn die clientseitigen Daten synchronisiert und serverseitig weitere Operationen ausgeführt werden?

Prinzipiell sollten Sie daher vom Benutzer erlangte Informationen wie Texte aus Eingabefeldern nur mithilfe von Platzhaltern in SQL-Statements einfügen. SQLite kann so die Textdaten automatisch escapen, falls notwendig.

2.7.8 Daten aktualisieren

Um bestehende Daten zu aktualisieren, müssen Sie das `SQL UPDATE`-Statement verwenden. Der Gebrauch unterscheidet sich im Wesentlichen nur durch das auszuführende SQL. Allerdings können wir zum ersten Mal von dem an das `statementSuccessCallback` übergebenen `SQLResult`-Objekt Gebrauch machen, wie der Kommentar andeuten soll:

```
var statementSuccessCallback = function(t, resultSet){
   //resultSet.rowsAffected == 1
}

var statementErrorCallback = ...

db.transaction(function(t){
   t.executeSql('UPDATE notes set note=?, date=CURRENT_TIMESTAMP WHERE
id=?;', [newNoteText, selected], statementSuccessCallback,
statementErrorCallback);
});
```

Innerhalb das `UPDATE SQL`-Statements benutzen Sie nun zwei Platzhalter. Der erste wird den aktualisierten Text der Notiz enthalten, während der zweite Platzhalter in der `WHERE`-Klausel des SQL-Statements benutzt wird, um eindeutig die zu aktualisierende Notiz per `id` zu bestimmen. Das Array, der zweite Parameter an die `executeSQL()`-Methode, enthält nun folglich auch zwei Elemente: den neuen Text sowie die `id` der zu aktualisierenden Zeile in der Tabelle.

Einen ersten Gebrauch können wir nun auch von dem `resultSet`-Objekt machen. Eines der Attribute ist `rowsAffected`, das die Anzahl der Tabellenzeilen zurückliefert, die verändert worden sind (das trifft also nur auf `INSERT`-, `UPDATE`- und `DELETE`-Statements zu).

2.7.9 Datenbank-Queries

Wir haben in den vorangegangenen Beispielen bereits oft davon gesprochen, dass die UI aktualisiert wird. In diesem Beispiel bedeutet das, dass sämtliche Notizen sortiert nach Datum aus der Datenbank geholt und angezeigt werden müssen. Schauen wir uns also dieses `SELECT`-Statement genauer an und werfen wir auch einen Blick darauf, wie das `SQLResult`-Objekt die Daten unserer Abfrage enthält:

```
var statementErrorCallback = ...
```

```
db.transaction(function(t){
   t.executeSql('SELECT * FROM notes ORDER BY date DESC;', null,
function(transaction, resultSet){
      var i, row, id, note, date;

      for (i = 0; i < resultSet.rows.length; i++)
      {
         row = resultSet.rows.item(i);

         id = row.id;
         note = row.note;
         date = row.date;

         //create HTML and update UI
      }
   }, statementErrorCallback);
});
```

Zunächst wird folgendes `SELECT`-Statement ausgeführt, wodurch alle Zeilen der `notes`-Tabelle für unser Ergebnis selektiert werden:

```
SELECT * FROM notes ORDER BY date DESC;
```

Da wir keine Platzhalter verwenden, muss auch kein zweiter Parameter mit den Ersetzungen angegeben werden. Der zweite Parameter ist also somit null.

Spannend ist nun das `statementSuccessCallback`, da wir für das `SQLResultSet`-Objekt zum ersten Mal die Daten unserer Abfrage erwarten können. Das `SQLResultSet`-Objekt wird wie folgt definiert:

```
interface SQLResultSet {
  readonly attribute long insertId;
  readonly attribute long rowsAffected;
  readonly attribute SQLResultSetRowList rows;
};
interface SQLResultSetRowList {
  readonly attribute unsigned long length;
  getter any item(in unsigned long index);
};
```

Das Attribut `rows` zeigt auf ein weiteres Objekt vom Typ `SQLResultSetRowList`. Dies ist das eigentliche Objekt, das unsere Daten enthält. Da das Ergebnis einer SQL-Data-

base zeilenorientiert ist, können wir nun jede Zeile einzeln in einer `for`-Schleife abfragen. Um auf die einzelnen Spalten der Zeilen zuzugreifen, kann ganz bequem die Dot-Notation verwendet werden:

```
for (i = 0; i < resultSet.rows.length; i++)
{
   row = resultSet.rows.item(i);

   id = row.id;
   note = row.note;
   date = row.date;

   //create HTML and update UI
}
```

2.7.10 Daten löschen

Um Daten zu löschen, bedienen wir uns des `DELETE`-Statements. In den meisten Fällen soll nicht die gesamte Tabelle gelöscht werden, also wird das `DELETE`-Statement durch eine `WHERE`-Klausel eingeschränkt:

```
db.transaction(function(t){
   t.executeSql('DELETE FROM notes WHERE id = ?',
      [selected],
      function(t, resultSet){
         //UI aktualisieren
      },
      statementErrorCallback);
});
```

2.7.11 Schemamigration

Jede Webanwendung entwickelt sich weiter, und somit können Veränderungen des Datenbankschemas notwendig sein. Aus diesem Grund hat jede Web-SQL-Database ein `version`-Attribut, mit dessen Hilfe zunächst die Version der Datenbank abgefragt werden kann. Die Version ist dabei ein einfacher String, beispielsweise `'1.0'`.

Wurde die gewünschte Version im `openDatabase()`-Aufruf weggelassen, wird einfach die aktuell vorhandene Version der Datenbank geöffnet (oder eine neue Datenbank zunächst angelegt). Man kann dann über das `version`-Attribut die Version der aktuell auf dem Endgerät vorhandenen Datenbank feststellen, und der JavaScript-Code kann bei Bedarf eine Schemamigration durchführen.

```
var db = openDatabase('notes', '', 'Offline Notes Storage', 5*1024*1024);
var version = db.version; //z. B. '1.0'
```

Folgendes Beispiel demonstriert, wie die Datenbank angelegt wird und nach Bedarf später aktualisiert werden kann. Wir machen es uns dabei zunutze, dass eine ohne Version geöffnete Datenbank, sofern sie tatsächlich noch nicht existiert, als Version

einen leeren String hat. Somit kann die initiale Migration auf die aktuelle Version für neue Nutzer direkt erfolgen.

Für Nutzer, die bereits eine Datenbank angelegt hatten, liefert der Aufruf der `openDatabase()`-Methode die aktuelle Datenbank zurück, die dann mittels des `version`-Attributs ebenso auf Aktualität geprüft werden kann. Eine notwendige Migration wird dann analog zur Migration auf die Version 1.0 durchgeführt.

Um die Version einer Datenbank zu verändern, muss die Methode `changeVersion()` der Datenbank benutzt werden. Diese Methode nimmt insgesamt fünf Parameter:

```
changeVersion(alteVersion, neueVersion, migrationAlt_Neu,
migrationAlt_Neu_Fehler, migrationAlt_Neu_OK);
```

Der erste Parameter, `alteVersion`, stellt die aktuell vorhandene Version dar. `neueVersion` stellt die Version dar, die abgespeichert werden soll, wenn die Migration erfolgreich war. Das eigentliche Migrationsskript wird durch die Funktion `migrationAlt_Neu` durchgeführt. An diese Funktion wird ein `transaction`-Objekt übergeben, auf dem dann `executeSql('sql')` wie bereits bekannt aufgerufen werden kann. Ebenso können ein Fehler-Handler sowie eine Funktion angegeben werden, die im Erfolgsfall aufgerufen wird. Folgendes Beispiel demonstriert die Integration von Datenbankmigrationen (ab der Erstellung der Datenbank):

```
function cv_0_0_1_0(transaction)
{
   transaction.executeSql('CREATE TABLE IF NOT EXISTS notes (id INTEGER NOT
NULL PRIMARY KEY AUTOINCREMENT, note TEXT);');
}

function cv_failure_0_0_1_0(error)
{
   alert('Fehler in Migration von 0.0 -> 1.0 conversion. ' + error.message);
   return true;
}

function cv_success_0_0_1_0()
{
   alert('DB auf 1.0 migriert');
}


function prepareDatabase()
{
   try
   {
      var db = openDatabase('someDB', '', 'someDB Beschreibung',
5*1024*1024);

      if (db.version == '')
      {
         //neu db, die DB muss auf die erste Version 'migriert' werden
         db.changeVersion("", "1.0", $i.cv_0_0_1_0, $i.cv_failure_0_0_1_0,
$i.cv_success_0_0_1_0);
      }
```

```
      else if (db.version == '1.0')
      {
         alert('habe v1.0, alles OK');
         //sobald sich die DB verändert, wird hier db.changeVersion
aufgerufen
         //gleichzeitig wird die Erzeugung der DB so verändert, dass sofort
die aktuellste Version erstellt wird
         //db.changeVersion("1.0", "2.0", $i.cv_1_0_2_0,
$i.cv_failure_1_0_2_0, $i.cv_success_1_0_2_0);
      }

      return db;
   }
   catch(e)
   {
       alert('Konnte DB nicht öffnen/migrieren. Aktuelle Version: ' +
db.version);
   }
}
```

Wie immer hier das komplette Beispiel im HTML-Framework:

```
<!-- HTML5-SQL-Datenbanken / DB-Migration
     Markus Spiering / Sven Haiges
-->
<!DOCTYPE html>
<html>
    <head>
        <title>DB-Migration</title>
       <meta name="layout" content="webkit" />
      <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
      <link href="style.css" rel="stylesheet" media="screen" type="text/css" />

      <script>
      (function my() {

         var $i = {};

         $i.hasSQL = function()
         {
            return typeof window.openDatabase == "function" ? true : false;
         };

         $i.init = function()
         {
            if (!$i.hasSQL())
            {
               alert('No SQL Database, no fun!');
               return;
            }

            //get current db
            $i.db = $i.prepareDatabase();
```

```
        }

        $i.errorStatementCallback = function(transaction, error)
        {
           alert('errorCallBack: ' + error.code + ' / message: ' +
error.message);
           return true; //true = roll back transaction, false = execution
continues
           //transaction kann zum error loggign verwendet werden.
        };

        $i.cv_0_0_1_0 = function(transaction)
        {
              transaction.executeSql('CREATE TABLE IF NOT EXISTS notes (id
INTEGER NOT NULL PRIMARY KEY AUTOINCREMENT, note TEXT);');
        };

        $i.cv_failure_0_0_1_0 = function(error)
        {
           alert('Fehler in Migration von 0.0 -> 1.0 conversion. ' +
error.message);
           return true;
        };

        $i.cv_success_0_0_1_0 = function()
        {
           alert('DB auf 1.0 migriert');
        };


        $i.prepareDatabase = function()
        {
           try
           {
              var db = openDatabase('someDB', '', 'Migration Test',
5*1024*1024);

              if (db.version == '')
              {
                 //brand new, creating current schema and setting version
                 db.changeVersion("", "1.0", $i.cv_0_0_1_0,
$i.cv_failure_0_0_1_0, $i.cv_success_0_0_1_0);
              }
              else if (db.version == '1.0')
              {
                 alert('got v1.0, all good');
                 //sobald sich die DB veraendert, wird hier
db.changeVersion aufgerufen
                 //gleichzeitig wird die Erzeugung der DB so veraendert,
dass sofort die aktuellste version erstellt wird
                 //db.changeVersion("1.0", "2.0", $i.cv_1_0_2_0,
$i.cv_failure_1_0_2_0, $i.cv_success_1_0_2_0);

              }
```

```
            return db;
          }
          catch(e)
          {
              alert('Konnte DB nicht öffnen. Aktuelle Version: ' +
db.version);
          }

        };

        addEventListener('DOMContentLoaded', $i.init, true);

      }).call();

      </script>
    </head>
    <body>

      <div id="topbar">
        <div id="title">DB Migration</div>
      </div>

      <div id="content">

        <ul class="pageitem">
          <li class="textbox"><span id="msg">Beim ersten Aufruf wird die
DB someDB auf die Version 1.0 'migriert' und es wird das initiale Schema
erstellt. Sobald sich die Anwendung &auml;ndert, kann dann analog eine
Migration auf die n&auml;chste Version stattfinden.</span></li>
        </ul>

      </div>
      <div id="footer">
        <a class="noeffect" href="http://iwebkit.net">Powered by
iWebKit</a>
      </div>
    </body>
</html>
```

Anzeige	*Bedeutung*
	Der Benutzer muss der Erstellung der Datenbank erst zustimmen.

Anzeige	*Bedeutung*
	Die Datenbank wird dann erstellt und ist in den Safari-Einstellungen unter *Datenbanken* sichtbar.
	Beim ersten Aufruf wird die Datenbank `someDB` auf die Version 1.0 migriert, und das initiale Schema wird erstellt. Sobald sich die Anwendung ändert, kann dann analog eine Migration auf die nächste Version stattfinden.
	Wird die Seite nochmals geladen, wird überprüft, ob die Version 1.0 der Datenbank immer noch verfügbar ist.

2.7.12 Web-SQL-Database: Notes-Beispielanwendung

Die bisherigen Beispiele wurden recht nüchtern und ohne Web-UI präsentiert. Der Grund dafür ist der, dass für diese UI eine Menge HTML und JavaScript vonnöten ist, was die Vorstellung der API nicht vereinfacht hätte. Daher soll zum Schluss auf eine Beispielanwendung hingewiesen werden, deren Quellcode wir nun komplett vorstellen.

Die bereits bekannte Notes-Notizverwaltung wird anhand der folgenden Screenshots kurz vorgestellt.

Anzeige	*Bedeutung*
	Der Ausgangsbildschirm. Es wurden noch keine Notizen in der Datenbank abgelegt. Die Datenbank selbst wurde durch `openDatabase()` zu diesem Zeitpunkt jedoch bereits erstellt.
	Eine neue Notiz wird durch Anklicken des Buttons *Neue Notiz* ausgelöst. Der Benutzer fügt einen Text hinzu und klickt auf *Speichern*.
	Die Notiz wurde abgespeichert. Das Datum wurde automatisch hinzugefügt. Es wurde ein `SQL INSERT` in die `notes`-Tabelle durchgeführt.

Anzeige	Bedeutung
	Um eine Notiz zu bearbeiten oder zu löschen, kann der Benutzer auf die Notiz klicken. Der bisherige Text wird dann in der Textbox angezeigt.
	Hier wurden drei Notizen angelegt ...
	... die nun allesamt gelöscht werden.

Hier der Beispielcode:

```
<!-- HTML5-SQL-Datenbanken / Notes
     Markus Spiering / Sven Haiges
-->
<!DOCTYPE html>
<html>
    <head>
        <title>Notes</title>
```

```
       <meta name="layout" content="webkit" />
      <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
      <link href="style.css" rel="stylesheet" media="screen" type="text/css" />

      <style>
      .menu .delete {
         position: absolute;
         width: 70px!important;
         height: 30px!important;
         right: 10px;
         top: 6px;
         margin: 0!important;
         background: url("../images/delete.png") 0 0 norepeat
      }

      .hidden {
         display: none;
      }
      </style>

      <script>
      (function my() {

         var $i = {};

         $i.hasSQL = function()
         {
            return typeof window.openDatabase == "function" ? true : false;
         };

         $i.init = function()
         {
            if (!$i.hasSQL())
            {
               alert('No SQL Database, no fun!');
               return;
            }

            document.getElementById('createButton').
addEventListener('click', function(e){
               document.getElementById('newnoteTitle').style.display = "block";
               document.getElementById('newnoteForm').style.display = "block";
            }, true);

            document.getElementById('save').addEventListener('click',
function(e){
               var newNoteText = document.getElementById('note').value;

               if (!$i.selected)
               {
                  $i.db.transaction(function(t){
                     t.executeSql('INSERT INTO notes (note) VALUES (?);',
[newNoteText], function(t, resultSet){
```

```
                        $i.displayNotes();
                    }, $i.errorStatementCallback);
                });

            }
            else
            {
                $i.db.transaction(function(t){
                    t.executeSql('UPDATE notes set note=?,
date=CURRENT_TIMESTAMP WHERE id=?;', [newNoteText, $i.selected], function(t,
resultSet){
                        $i.displayNotes();
                        $i.selected = false;
                    }, $i.errorStatementCallback);
                });
            }

            //hide form, reset
            document.getElementById('note').value = '';
            document.getElementById('deleteLi').style.display = "none";
            document.getElementById('newnoteTitle').style.display = "none";
            document.getElementById('newnoteForm').style.display = "none";
        }, true);

        document.getElementById('delete').addEventListener('click',
function(e){
            var ok = confirm("Wirklich?");

            if (!ok)
                return;

            $i.db.transaction(function(t){
                t.executeSql('DELETE FROM notes WHERE id = ?',
[$i.selected], function(t, resultSet){
                    //alert(resultSet.rowsAffected);
                    $i.displayNotes();
                    $i.selected = false;
                    document.getElementById('note').value = '';
                    document.getElementById('deleteLi').style.display =
"none";
                    document.getElementById('newnoteTitle').style.display =
"none";
                    document.getElementById('newnoteForm').style.display =
"none";
                }, $i.errorStatementCallback);
            });
        }, true);

        document.getElementById('deleteAll').addEventListener('click',
function(e){
            var ok = confirm("Alle Notizen löschen?");

            if (!ok)
                return;
```

```
                $i.db.transaction(function(t){
                   t.executeSql('DELETE FROM notes', null, function(t,
resultSet){
                      alert(resultSet.rowsAffected + ' Notizen
geloescht.');
                      $i.displayNotes();

                   }, $i.errorStatementCallback);
                });
            }, true);

            //get current db or create
            $i.db = $i.prepareDatabase();

            $i.displayNotes();
         }

         $i.editNote = function(e)
         {
            var id = parseInt(e.currentTarget.id, 10);

            $i.db.transaction(function(t){
               t.executeSql('SELECT * FROM notes WHERE id = ?', [id],
function(transaction, resultSet){
                  //alert(resultSet.rows.length);
                  var row = resultSet.rows.item(0);
                  document.getElementById('note').value = row.note;
                  $i.selected = row.id;
               }, $i.errorStatementCallback)
            });

            document.getElementById('deleteLi').style.display = "block";
            document.getElementById('newnoteTitle').style.display = "block";
            document.getElementById('newnoteForm').style.display = "block";

         }

         $i.displayNotes = function(){
            var existingNode = document.getElementById('existing');
            existingNode.innerHTML = '';

            $i.db.transaction(function(t){
               t.executeSql('SELECT * FROM notes ORDER BY date DESC;', null,
function(transaction, resultSet){
                  var i, row, newLi, newSpan, commentSpan;
                  //alert("got " + resultSet.rows.length + ' rows in the db');
                  for (i = 0; i < resultSet.rows.length; i++)
                  {
                     row = resultSet.rows.item(i);
                     //<li class="menu note" id="note1"><span
class="name">Meine erste...</span></li>
                     newLi = document.createElement('li');
                     newLi.setAttribute('class', 'menu note');
                     newLi.setAttribute('id', row.id);
                     newSpan = document.createElement('span');
```

```
                    newSpan.setAttribute('class', 'name');
                    newSpan.textContent = row.note.substring(0,10) + '...';
                    newLi.appendChild(newSpan);
                    commentSpan = document.createElement('span');
                    commentSpan.setAttribute('class', 'comment');
                    commentSpan.textContent = row.date;
                    newLi.appendChild(commentSpan);

                    newLi.addEventListener('click', $i.editNote, false);

                    document.getElementById('existing').appendChild(newLi);
                 }
              }, $i.errorStatementCallback);
           }/*errorCallback, successCallback*/);
        };

        $i.errorCallback = function(error)
        {
           alert('errorCallBack: ' + error.code + ' / message: ' +
error.message);
        };

        $i.errorStatementCallback = function(transaction, error)
        {
           alert('errorCallBack: ' + error.code + ' / message: ' +
error.message);
           return true; //true = roll back transaction, false = execution
continues
           //transaction kann zum error loggign verwendet werden.
        };


        $i.prepareDatabase = function()
        {

           //migration using:
           /*
           var db = openDatabase(shortName, "", displayName, maxSize);

           var version = db.version; // For example, "1.0"
           */
           //or try/catch

           try
           {
              var db = openDatabase('notes', '1.0', 'Offline Notes
Storage', 5*1024*1024);

              db.transaction(function(t){
                 t.executeSql('CREATE TABLE IF NOT EXISTS notes (id INTEGER
NOT NULL PRIMARY KEY AUTOINCREMENT, note TEXT, date DATE NOT NULL DEFAULT
CURRENT_TIMESTAMP);');
              }, $i.errorStatementCallback);
```

```
            //if (db.changeVersion)
            // alert('can change version');

            return db;
         }
         catch(e)
         {
             if (e == 2)
                 alert("Invalid database version.");
            else
               alert("Unknown error "+e+".");
         }

      };

      addEventListener('DOMContentLoaded', $i.init, true);

   }).call();


   </script>
  </head>
  <body>

   <div id="topbar">
      <div id="title">Notes</div>
   </div>
   <span id="msg"></span>
   <div id="content">
      <span class="graytitle">Bestehende Notizen</span>
      <ul class="pageitem" id="existing">
         <li class="menu note" id="note1"><span class="name">Meine
erste...</span></li>
      </ul>
      <span id="newnoteTitle" style="display:none;"
class="graytitle">Notiz bearbeiten</span>
      <ul id="newnoteForm" style="display:none;" class="pageitem"
id="createEditNote">
         <li class="textbox"><textarea id="note"
rows="4"></textarea></li>
         <li class="button" style="display:none;" id="deleteLi"><input
id="delete" type="submit" value="L&ouml;schen..." /></li>
         <li class="button"><input id="save" type="submit"
value="Speichern" /></li>
      </ul>

      <ul class="pageitem" id="actions">
            <li class="button"><input id="createButton" type="submit"
value="Neue Notiz..." /></li>
            <li class="button"><input id="deleteAll" type="submit"
value="Alle l&ouml;schen..." /></li>
```

```
        </ul>

      </div>
      <div id="footer">
         <a class="noeffect" href="http://iwebkit.net">Powered by
iWebKit</a>
      </div>
    </body>
</html>
```

2.7.13 Datenbanken im Safari Web Inspector und in Google Chrome

Wir haben die Web Inspector-Ansicht von Safari und dem Google Chrome-Browser bereits weiter oben im Buch vorgestellt, dabei aber ausgelassen, dass nicht nur der HTML-Code einer Seite untersucht werden kann, sondern die Ansichten auch Einblicke in die Struktur von angelegten Datenbanken, Local Storage und Session Storage bieten.

Um zur Datenbankansicht zu gelangen, gehen Sie einfach in das Menü *Entwickler/Developer* im Safari-Browser und rufen dort *Show Web Inspector* auf. Als eine der Inspector-Ansichten wird anschließend *Databases* angeboten.

Im Notes-Beispiel wurde die Datenbank *notes* angelegt, die sichtbar wird, wenn man die Beispielanwendung im Safari-Browser ausführt. Die Datenbank *notes* enthält die Felder *note* und *date*. Die Inhalte der beiden Datenfelder, also die einzelnen Datensätze, können Sie ebenfalls über den Web Inspector ansehen.

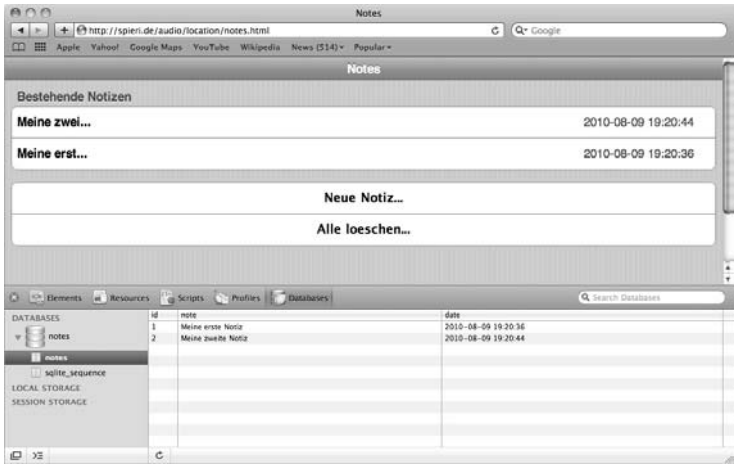


Bild 2.63: Datenbank im Safari Web Inspector.

Die gleiche Funktionalität ist auch im Google Chrome-Browser verfügbar. Dort wird der vergleichbare *Web Inspector* unter *View/Developer/Developer Tools* aufgerufen.

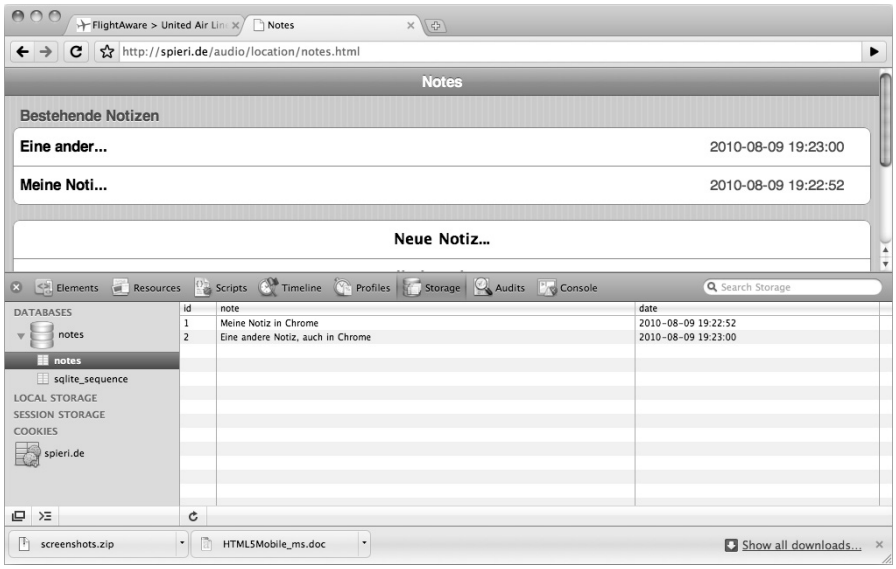


Bild 2.64: Datenbank im Google Chrome-Web Inspector.

2.8 Offline Web Applications

Ein sehr spannendes Feature der HTML5-Spezifikation nennt sich »Offline Web Applications«. Kurz bedeutet das, dass eine beliebige HTML5-Webseite inklusive aller Ressourcen (wie Bilder, Medien), JavaScript und CSS komplett auf dem Client dauerhaft gecacht werden kann und dadurch die Webseite komplett offline verfügbar wird.

Beim ersten Aufruf der Webseite lädt der Browser im Hintergrund alle in einem sogenannten Manifest-File aufgeführten Dateien herunter. Beim nächsten Aufruf der Webseite wird dann, falls eine Internetverbindung vorhanden ist, dieses Manifest kurz auf Änderungen überprüft. Falls Änderungen vorhanden sind, wird die gecachte Version angezeigt, jedoch werden im Hintergrund bereits die neuen Dateien heruntergeladen. Wenn keine Änderungen vorhanden sind oder der Browser offline ist, wird einfach die gecachte Version genommen.

Aktuell implementiert iPhone OS diesen sogenannten »Application Cache«, auf Android 2.1 scheint die grundlegende Implementierung zwar vorhanden zu sein, jedoch funktioniert das Aktualisieren des Caches derzeit nicht. Insofern ist dieses HTML5-Feature derzeit auf Android leider nicht nutzbar. Es wird hier dennoch beschrieben, da eine vollständige und spezifikationskonforme Implementierung auf Android in nächster Zeit als wahrscheinlich gilt.

2.8.1 Grundlagen der Cache-Manifestdatei

Um die aktuelle HTML5-Webseite einschließlich der benötigten Ressourcen speichern zu können, benötigt der Browser die Information darüber, was genau diese Ressourcen sind. Die einzelnen Dateien, die lokal gespeichert werden sollen, werden in einer soge-

nannten Cache-Manifestdatei angegeben. Die eigentliche Webseite, die diese Datei referenziert, ist implizit immer dabei, d. h. sie muss nicht explizit im Manifest angegeben werden. Folgendes Beispiel zeigt ein einfaches Cache-Manifest:

```
CACHE MANIFEST

/js/functions.js
/css/style.css
/images/logo.jpg
```

Diese Manifestdatei muss nun auf einem Server verfügbar gemacht werden, üblicherweise im selben Verzeichnis wie auch die HTML5-Seite. Innerhalb des HTML-Tags dieser Webseite muss nun die Manifestdatei referenziert werden:

```
<html manifest="demo.manifest">
```

Wichtig ist hierbei, dass das Manifest `demo.manifest` mit dem Content-Type `text/cache-manifest` vom Server ausgeliefert wird. Das kann entweder dadurch erreicht werden, dass die eigentliche Manifestdatei dynamisch, etwa von einem Skript auf dem Server, erstellt wird oder ein Eintrag in die *.htaccess*-Datei des aktuellen Verzeichnisses des (Apache-)Webservers gemacht wird. Dieser Eintrag in die (eventuell noch zu erstellende) *.htaccess*-Datei sieht so aus:

```
AddType text/cache-manifest .manifest
```

Hiermit wird der Server angewiesen, alle Dateien mit der Endung `.manifest` mit dem Content-Type `text/cache-manifest` auszuliefern. Sobald die vom Browser geladene HTML-Seite über das `manifest`-Attribut eine Referenz auf eine Manifestdatei herstellt und diese Datei mit dem Content-Type `text/cache-manifest` ausgeliefert wird, versucht der Browser, die im Manifest aufgeführten Dateien für das nächste Aufrufen der Webseite zu speichern. Wenn diese Speicherung abgeschlossen ist und der Nutzer die Seite erneut lädt, wird die vorab gespeicherte Version verwendet.

Im Hintergrund wird zusätzlich bei jedem Aufruf der Seite versucht, das Manifest auf Änderungen zu überprüfen. Das geschieht durch einen byteweisen Vergleich der Manifestdatei. Nicht das Modifikationsdatum der Datei ist also ausschlaggebend, sondern der Inhalt. Um die oben aufgeführte Manifestdatei zu verändern, könnte beispielsweise ein Kommentar hinzugefügt werden:

```
CACHE MANIFEST
#hier ein Kommentar
/js/functions.js
/css/style.css
/images/logo.jpg
```

Sobald nicht alle Dateien heruntergeladen werden konnten, wird automatisch der gesamte Vorgang abgebrochen, und ein eventuell vorhandener älterer Cache bleibt der aktuelle Cache. Der Download kann also entweder komplett abgeschlossen werden, oder sämtliche Dateien werden verworfen. Wenn der Download der Dateien erfolgreich war, wird der vorhandene Cache ausgetauscht und beim nächsten Laden der Webseite automatisch verwendet. An dieser Stelle sollte aber betont werden, dass mit dem Abschluss des Down-

loads der Dateien in der Manifestdatei nicht automatisch die aktuelle Webseite aktualisiert wird. Erst beim nächsten Laden dieser Seite werden die neuen Dateien verwendet.

2.8.2 Feststellen, ob der Browser online ist

Seit HTML5 kann der Zustand des Browsers durch `navigator.onLine` abgerufen werden. Es wird `false` zurückgegeben, wenn der Browser definitiv offline ist, und `true`, falls der Browser eine Verbindung haben könnte. Wichtig ist hierbei, dass nicht ständig im Hintergrund aktiv überprüft wird, ob der Browser online oder offline ist. Definitiv offline ist der Browser beispielsweise, wenn keine WLAN-Verbindung hergestellt werden konnte und auch kein Mobilfunknetz benutzt werden kann. Eine einfache Art, diesen Zustand herzustellen, ist, das Handy in den Flugmodus zu versetzen, wodurch alle drahtlosen Verbindungen unterbrochen werden.

Es werden auch die Online- und Offline-Events in HTML5 definiert. In der Praxis scheint das jedoch weder auf dem iPhone OS noch auf Android derzeit implementiert zu sein. Sobald das der Fall ist, sollte folgender JavaScript-Code tatsächlich funktionieren:

```
window.addEventListener("online", function() {
  alert('Du bist online!');
}, true);

window.addEventListener("offline", function() {
  alert('Du bist offline!');
}, true);
```

2.8.3 Online-Whitelist und Fallback-Sektion

Die Manifestdatei kann laut Spezifikation aus bis zu drei Sektionen bestehen, die durch `CACHE:`, `FALLBACK:` und `NETWORK:` auf einer Zeile geschrieben eingeleitet werden. Die `CACHE:`-Sektion haben Sie bereits kennengelernt, da das die Standardsektion des Cache-Manifests ist. Unser obiges Beispiel könnten wir also auch expliziter so schreiben:

```
CACHE MANIFEST
#hier ein Kommentar

CACHE:
/js/functions.js
/css/style.css
/images/logo.jpg
```

In diesem Beispiel wurde zusätzlich noch eine Leerzeile eingefügt, die von den Browsern einfach ignoriert wird.

Die `NETWORK:`-Sektion kann dazu verwendet werden, Dateien aktiv vom Caching auszuschließen. Es wird also immer versucht, die hier aufgeführten Dateien über das eventuell vorhandene Netzwerk zu laden:

```
CACHE MANIFEST
#hier ein Kommentar
```

```
CACHE:
/js/functions.js
/css/style.css
/images/logo.jpg

NETWORK:
/dynamic/currentUsageChart.png
```

Die Datei *currentUsageChart.png* wird nun immer vom Browser angefragt, auch wenn keine Netzwerkverbindung zur Verfügung steht. Das hat zur Folge, dass dieses Bild eventuell nicht geladen werden kann und somit in der UI des Browsers als kaputter Link dargestellt wird.

Soll das verhindert werden, kann man mit der `FALLBACK:`-Sektion angeben, welche (gecachte) Datei angezeigt werden soll, wenn die bevorzugte Datei nicht verfügbar ist:

```
CACHE MANIFEST
#hier ein Kommentar

CACHE:
/js/functions.js
/css/style.css
/images/logo.jpg

NETWORK:
/dynamic/currentUsageChart.png

FALLBACK:
/dynamic/currentUsageChart.png /offline/noChart.png
```

Sobald der Browser nun online ist, wird *currentUsageChart.png* benutzt. Wenn er offline ist, wird *noChart.png* angezeigt. Um Fallback-Dateien gleich für mehrere Einträge anzugeben, kann auch ein partieller Pfadname verwendet werden. Beispielsweise wird durch folgenden Eintrag in der Manifestdatei jede nicht verfügbare Ressource im */images*-Verzeichnis durch *fallback.png* ersetzt:

```
FALLBACK:
/images/ offline/fallback.png
```

Wie wird ein Cache gelöscht?
Je nach Plattform geschieht das auf unterschiedliche Art und Weise und ist natürlich gerade für die Entwicklung der Offlineapplikationen von Bedeutung. Um auf dem iPhone OS den Cache zu löschen, gehen Sie so vor: *Einstellungen/Safari/Cache löschen*. Auch wenn die derzeitige Implementierung auf Android noch dürftig ist, hier analog die Vorgehensweise dazu: vom WebKit-Browser und der aktuell gecachten Webseite aus *Menü/Mehr/Einstellungen/Cache löschen*. Sobald das erledigt ist, kann die Webseite neu geladen werden, wodurch das initiale Caching ausgelöst wird.

2.8.4 Dynamische Erzeugung des Cache-Manifests

In realen Applikationen müssen meist weitaus mehr Dateien gecacht werden, als in den bisherigen Beispielen aufgeführt. Daher bietet es sich an, alle von der aktuellen Webseite benötigten Dateien in einem Verzeichnis auf dem Server zu hosten und dann das Cache-Manifest dynamisch auf Basis des aktuellen Verzeichnisinhalts zu erstellen. Da es eine Menge unterschiedlichster Sprachen gibt, die dazu serverseitig verwendet werden können, kann an dieser Stelle kein allgemeingültiges Beispiel gezeigt werden. Mit dem Grails-Framework, einem auf der Groovy-Skriptsprache basierendes Web-Framework, lässt sich eine dynamische Manifestdatei in wenigen Zeilen Code innerhalb einer Grails-Action erstellen:

```
def manifest = {
   def pathList = []
   new File("./web-app/images").eachFile {
      if (!it.name.startsWith('.') && !it.directory)
         pathList << 'html5' + it.path[1..-1]
   }
   render(contentType:'text/cache-manifest', text:pathList.join('\n'));
   }
```

Das `manifest`-Attribut der HTML-Seite kann nun verwendet werden, um im Browser die dynamisch erstellte Manifestdatei zu referenzieren. Es muss lediglich eine korrekte URL zu der Ressource angegeben werden, die die Manifestdatei mit dem korrekten Content-Type zurückgibt. Konkret in unserer Beispielanwendung wäre das:

```
<!DOCTYPE html>
<html manifest="/html5/offline/manifest">
```

2.8.5 Die ApplicationCache-API

Zusätzlich kann der Offlinecache über eine JavaScript-API angesprochen werden. Es stehen auch zahlreiche Events zur Verfügung, mit deren Hilfe der aktuelle Zustand des Caches abgerufen werden kann. Das ApplicationCache-Interface ist wie folgt definiert und global unter `applicationCache` verfügbar:

```
interface ApplicationCache {
  readonly attribute unsigned short status;

  // updates
  void update();
  void swapCache();
};
```

Diese Möglichkeiten werden nun der Reihe nach betrachtet. Unter `applicationCache.status` wird der aktuelle Status des Caches abgefragt. Im `applicationCache`-Objekt sind dadurch auch einige Konstanten definiert, die in folgender Tabelle erklärt werden:

Konstante	*Bedeutung*
`UNCACHED`	Derzeit steht für die aktuelle Webseite kein Cache zur Verfügung, etwa weil kein `manifest`-Attribut im HTML-Tag angegeben wurde.
`IDLE`	Ein Cache ist vorhanden, aber derzeit wird keine Operation auf dem Cache ausgeführt.
`CHECKING`	Ein Cache ist vorhanden, und derzeit wird die Manifestdatei auf Aktualität überprüft.
`DOWNLOADING`	Ein Cache ist vorhanden, und neue Ressourcen werden derzeit heruntergeladen.
`UPDATEREADY`	Ein vorheriger Download eines neuen Caches wurde abgeschlossen, und der neue Cache könnte nun durch Aufrufen der Methode `swapCache()` eingesetzt werden. Falls das nicht geschieht, wird der neue Cache beim nächsten Aufruf der Webseite automatisch benutzt.
`OBSOLETE`	Ein Cache ist vorhanden, der jedoch nicht mehr verwendet wird. Eventuell hat der Entwickler die Manifestdatei vom Server entfernt.

Um mittels dieser Konstanten also zu überprüfen, in welchem Zustand sich der Cache befindet, könnten wir Folgendes schreiben:

```
var cache = window.applicationCache;
if (cache.status == cache.UPDATEREADY)
   cache.swapCache();
```

`swapCache()` kann dann verwendet werden, wenn der Cache im Zustand `UPDATEREADY` ist. Dadurch werden die neu gecachten Ressourcen ausgetauscht, was ansonsten erst mit dem erneuten Laden der Webseite geschehen würde.

An dieser Stelle sei auch die Methode `update()` kurz erklärt. Ein Aufruf der Methode hat zur Folge, dass ein Check der Manifestdatei erzwungen wird. Dadurch wird programmatisch gesteuert, ob zusätzlich zum Check während des Aufrufs der Webseite ein Update gemacht wird.

Die ApplicationCache-API definiert glücklicherweise auch zahlreiche Events, durch die ein umständliches Polling des Status des Caches vermieden werden kann. Diese Events sind in folgender Tabelle kurz erklärt:

Event-Typ	*Beschreibung*	*Darauffolgende Events*
`checking`	Der Browser überprüft die Manifestdatei auf Aktualität oder lädt sie zum ersten Mal herunter. Das ist immer das erste Event.	`noupdate`, `downloading`, `obsolete`, `error`
`noupdate`	Die Manifestdatei hat sich nicht verändert.	-
`downloading`	Der Browser hat eine Veränderung in der Manifestdatei festgestellt und wird nun die Ressourcen erneut laden, oder die Ressourcen werden zum ersten Mal heruntergeladen.	`progress`, `error`, `cached`, `updateready`

Event-Typ	*Beschreibung*	*Darauffolgende Events*
`progress`	Der Browser lädt eine Ressource der Manifestdatei herunter. Dieses Event wird einmal pro Ressource gesendet.	`progress`, `error`, `cached`, `updateready`
`cached`	Die Ressourcen in der Manifestdatei wurden heruntergeladen, und die Webseite ist gecacht.	-
`updateready`	Die Ressourcen wurden aktualisiert, und das Skript kann über `swapCache()` den aktuellen Cache austauschen.	-
`obsolete`	Die Manifestdatei kann nicht mehr gefunden werden, und somit wird der aktuelle Cache gelöscht (Response Code 404 – Not Found oder 410 – Gone).	-
`error`	Das `error`-Event kann aus zahlreichen Gründen gesendet werden: Es war bisher kein Cache vorhanden, jedoch wird per `manifest`-Attribut auf einen Cache verwiesen. Dieser kann nicht geladen werden, da die Anfrage mit 404 oder 410 beantwortet wird. Ein fataler Fehler ist aufgetreten: Die Manifestdatei wird während des Updates verändert. In diesem Fall sollte der Browser unmittelbar einen erneuten Download starten.	-

Um ein Gefühl für alle diese Events sowie deren Reihenfolge und Bedeutung zu bekommen, bietet es sich an, ein einfaches Beispiel zu entwickeln. Dieses Beispiel wird in den folgenden Abschnitten schrittweise erklärt.

```
<!--
   HTML5 Offline Apps
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
<html manifest="/html5/offline/sampleManifest">
    <head>
        <title>HTML5 Offline Apps</title>

      <!-- iOS-Web-App-spezifische Metadaten festlegen -->
      <meta name="apple-mobile-web-app-capable" content="yes"/>
      <meta name="apple-mobile-web-app-status-bar-style" content="black-
translucent"/>
      <link rel="apple-touch-startup-image" href="startup.png" />
      <link rel="apple-touch-icon" href="icon.png"/>

      <meta content="minimum-scale=1.0, width=device-width, maximum-
scale=0.6667, user-scalable=no" name="viewport" />
      <link href="style.css" rel="stylesheet" media="screen" type="text/css" />
```

```
      <script src="javascript/functions.js" type="text/javascript"></script>

      <!-- Feature-Detection mit Modernizr -->
      <script src="js/modernizr-1.5.js" type="text/javascript"></script>

      <script>

      (function my() {

         var $i = {};
         $i.vars = {};

         $i.init = function()
         {

            if (!$i.hasApplicationCache())
            {
               alert('No applicationCache!');
               return;
            }

            //online offline - aktuell nicht unterstützt
            window.addEventListener("online", function() {
              alert('Du bist online!');
            }, true);

            window.addEventListener("offline", function() {
              alert('Du bist offline!');
            }, true);

            document.getElementById('message').innerHTML =
(navigator.onLine) ? 'online' : 'offline';

            var cache = window.applicationCache;

            //setTimeout(function() {
            // cache.update();
            //}, 5000);


            document.getElementById('message').innerHTML += ' / status: ' +
cache.status;

            //setTimeout(function() {
            // document.getElementById('message').innerHTML += ' /
status(2s): ' + cache.status;
            //}, 2000);

            var handler = function(e) {
               var eventsNode = document.getElementById('events');

               eventsNode.innerHTML += e.type + '<br/>';

               if ((e.type == 'checking' || e.type == 'progress') &&
!$i.startTimestamp) //on Android: no checking event
                  $i.startTimestamp = e.timeStamp;
               else if (e.type == 'noupdate' || e.type == 'updateready' ||
e.type == 'cached')
```

```
                eventsNode.innerHTML += 'total time (s): ' +
Math.abs((e.timeStamp - $i.startTimestamp) / 1000);

            if (e.type == 'updateready')
               window.applicationCache.swapCache();
         }

         cache.addEventListener('checking', handler, false);

         cache.addEventListener('error', handler, false);
         cache.addEventListener('noupdate', handler, false);

         cache.addEventListener('downloading', handler, false);

         cache.addEventListener('progress', handler, false);

         cache.addEventListener('updateready', handler, false);

         cache.addEventListener('cached', handler, false);

         cache.addEventListener('obsolete', handler, false);

      };

      $i.hasApplicationCache = function()
      {
         if (Modernizr.applicationcache)
            return true;
         else
            return false;
      };

      document.addEventListener("DOMContentLoaded", $i.init, false);
   }).call();

   </script>
  </head>
  <body>

   <div id="topbar">
      <div id="title">Offline Web App</div>
   </div>
   <div id="content">
      <ul class="pageitem">
         <li class="textbox">
            <p>Ihr Browser ist <span id="message"></span></p>
         </li>
         <li class="textbox">
            <p id="events"></p>
         </li>

      </ul>
   </div>
  </body>
</html>
```

Betrachten wir zunächst die iWebKit-basierte UI dieser Webseite, die durch ein von dieser Seite referenziertes Cache-Manifest offline verfügbar gemacht wird. Wie Sie sich vorstellen können, müssen zahlreiche Ressourcen (JavaScript, CSS, Bilder), die im Zusammenhang mit iWebKit stehen, gecacht werden. Die beiden folgenden Screenshots zeigen die Webseite in Aktion.

Beim ersten Aufruf der Webseite wird eine Manifestdatei vom Browser festgestellt, und der Download der Ressourcen wird eingeleitet. Das abschließende Event wird `cached` sein, da die Ressourcen zum ersten Mal heruntergeladen werden. Danach wird die Webseite erneut geladen, der Browser stellt jedoch eine unveränderte Manifestdatei fest. Dadurch wird nur noch das Event `noupdate` gesendet.

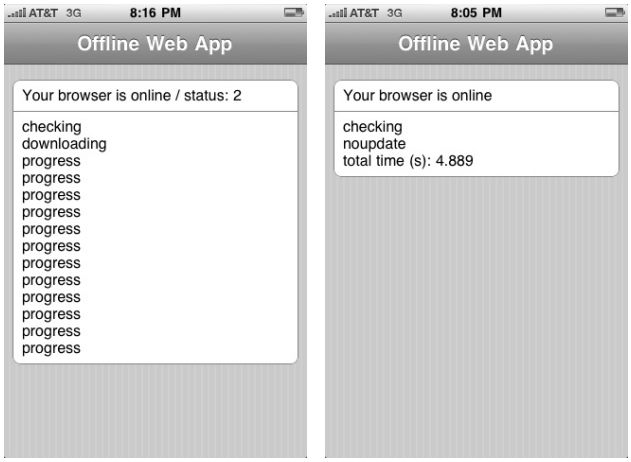


Bild 2.65: Der Browser stellt eine neue und in der rechten Abbildung eine unveränderte Manifestdatei fest.

Ein weiteres Szenario ist, dass eine veränderte Manifestdatei mit bereits bestehendem Cache vorliegt. In diesem Fall wird zunächst ein `checking`-Event, gefolgt von `downloading`, `progress` und abschließend bei geglücktem Download ein `updateready`-Event gesendet.

Um diese Events in der UI sichtbar zu machen, wird folgendes JavaScript verwendet:

```
var cache = window.applicationCache;

var handler = function(e) {
   var eventsNode = document.getElementById('events');

   eventsNode.innerHTML += e.type + '<br/>';

   if ((e.type == 'checking' || e.type == 'progress') && !$i.startTimestamp)
//on Android: no checking event
      $i.startTimestamp = e.timeStamp;
   else if (e.type == 'noupdate' || e.type == 'updateready' || e.type ==
'cached')
      eventsNode.innerHTML += 'total time (s): ' + Math.abs((e.timeStamp -
$i.startTimestamp) / 1000);
```

```
   if (e.type == 'updateready')
      window.applicationCache.swapCache();
}

cache.addEventListener('checking', handler, false);

cache.addEventListener('error', handler, false);

cache.addEventListener('noupdate', handler, false);

cache.addEventListener('downloading', handler, false);

cache.addEventListener('progress', handler, false);

cache.addEventListener('updateready', handler, false);

cache.addEventListener('cached', handler, false);

cache.addEventListener('obsolete', handler, false);
```

Wie Sie sehen, wird jeweils die Funktion, die in der Variablen `handler` referenziert wird, auf die verschiedenen Events registriert. Sobald das Event gesendet wird, wird es zunächst in der über `eventsNode` referenzierten Node in die UI übertragen. Im nächsten Schritt wird dann versucht, die Dauer der jeweiligen Operation festzustellen, indem bei den Events `checking` und `progress` ein Zeitstempel genommen wird und bei Eintreten der abschließenden Events `noupdate`, `updateready` und `cached` die Differenz berechnet wird. Des Weiteren wird `swapCache()` aufgerufen, sobald das `updateready`-Event gesendet wird.

2.9 Zusammenfassung

Wenn Sie es bis hierher geschafft haben, haben Sie bereits das nötige Wissen, um Ihre eigenen Webanwendungen zu erstellen. Sie haben gelernt, welche Aufbauarten und Designgrundsätze es für mobile Webanwendungen gibt, wie JavaScript- und CSS-Frameworks einzusetzen sind, und können nun Web Apps mit diversen HTML5-Funktionen erstellen. Folgendes können Sie jetzt:

- Benutzereingaben mit intelligenten Eingabefeldern gestalten.
- Dynamische Grafiken und Charts mit dem `canvas`-Element erstellen.
- Die aktuelle Position des Benutzers in einer Web App verwenden und Google Maps oder auch Flickr-Inhalte dazu anbieten.
- Videos und Audiodateien richtig erstellen und innerhalb der Anwendung einsetzen.
- Webanwendungen erstellen, die auch ohne Internetverbindung funktionieren.
- Seiten und Web Apps erstellen, die Datenbanken nicht auf einem Server, sondern direkt auf dem mobilen Gerät anlegen.

Sie können jetzt Ihrer Fantasie freien Lauf lassen, indem Sie Webanwendungen entwickeln und auch veröffentlichen. Lesen Sie dazu das Kapitel zum Publizieren von Web Apps, »Webanwendungen und native Apps veröffentlichen«, am Ende des Buchs. Als Nächstes möchten wir Ihnen Wege und Technologien zeigen, wie Sie Ihr HTML-, CSS- und JavaScript-Wissen für eine native Anwendung nutzen können und wie Sie eine solche Anwendung für iOS und Android erstellen.

3 Native Anwendungen erstellen

Wir haben es schon mehrfach im Buch angesprochen, und die Möglichkeit ist einfach atemberaubend: Um native Anwendungen zu erstellen, brauchen Sie nicht unbedingt Objective-C oder Java zu beherrschen, sondern Sie können richtige iPhone- oder Android-Anwendungen mit dem bislang erlernten Wissen und den Webtechnologien HTML, CSS und JavaScript erstellen. Diese Anwendungen verhalten sich nicht nur wie richtige native Apps, sondern Sie können sie auch im iTunes App Store oder im Android Market verkaufen – was logischerweise nicht mit einer Webseite funktioniert.

Die Technologie hat natürlich auch ihre Grenzen: Wenn es Ihr Ziel ist, superschnelle Spiele mit 3-D-Grafik zu programmieren, sollten Sie sich lieber schnell ein Objective-C- oder C++-Buch besorgen. Hochperfomante Spiele und Programme, die sehr komplexe und schnell wechselnde Grafiken benötigen, lassen sich mit diesen Webtechnologien weniger komfortabel schreiben. Nahezu alle anderen Anwendungsgebiete sind aber durchaus möglich.

Die Hilfsmittel, die zur Erstellung von nativen Anwendungen aus »Webseiten« verwendet werden, überbrücken noch ein weiteres Problem: den Zugriff auf telefontypische Funktionen wie beispielsweise die Kamera, den Sie von einem Browser aus bislang noch nicht haben. Mit den vorgestellten Frameworks haben Sie die Möglichkeit, per API auf diese Funktionen zuzugreifen und sie innerhalb Ihrer Anwendung zu benutzen.

3.1 Titanium vs. PhoneGap

Es gibt zwei primäre Anbieter von Produkten für das Vorhaben »Webtechnologien-zu-nativen-Anwendungen«: Titanium von Appcelerator und PhoneGap von Nitobi.

Beide machen augenscheinlich das Gleiche: Sie helfen dabei, native Anwendungen aus HTML, CSS und JavaScript zu bilden, bieten Zugriff auf Gerätefunktionen, die nicht über den Browser erreichbar sind, und liefern ein Endprodukt, das eine »richtige« Anwendung ist und über den iTunes App Store oder den Android Market vertrieben werden kann.

Technologisch gesehen, besteht der große Unterschied darin, dass eine native Anwendung mit PhoneGap tatsächlich immer noch eine Webanwendung ist, die in einem nativen Browserfenster läuft. Das ist vergleichbar mit einer Webanwendung, die Sie durch die richtigen Header sowie Startup-Icon und -Bildschirm auf dem Home-Bildschirm speichern können.

PhoneGap generiert nativen Code für jede der unterstützten Plattformen, aber größtenteils nur für die Komponente, die die eigentliche Webanwendung anzeigt. Die

Anwendung selbst ist immer noch HTML, CSS und JavaScript mit dem Unterschied, dass PhoneGap eine Brücke zwischen den Webtechnologien und nativen Funktionen bietet. Hier liegt auch die Stärke von PhoneGap: Die gerätespezifischen APIs für den Zugriff auf beispielsweise die Geoposition oder die Bewegungssensoren sind ähnlich, wenn nicht sogar die gleichen, für eine Vielzahl der von PhoneGap unterstützten Plattformen (iOS, Android, BlackBerry). Ein weiterer Vorteil von PhoneGap ist, dass sich mit den bereits erlernten Techniken wesentlich schneller und einfacher Apps erstellen lassen, als es mit anderen Frameworks oder Technologien möglich ist.

Etwas anders verhält es sich bei Titanium, das nicht nur für mobile Plattformen Anwendungen erstellen kann, sondern die gleiche Technologie für Windows, Linux und Mac OS X-Anwendungen bietet. Zahlreiche zum Teil auch größere Unternehmen verwenden die Titanium-Plattform, um ihre Desktop-Anwendungen für Windows und Apple aus einem Quellcode heraus zu erstellen. Der für uns wichtige Unterschied ist aber, dass Titanium einen Großteil des JavaScript-Codes in nativen Code umwandelt und im Gegensatz zu PhoneGap mehr nativen Code produziert. Auf einem iPhone wird beispielsweise aus dem `TableView`-JavaScript-Element eine native `TableView`-Komponente aus dem Apple UI-Kit produziert. Wie auch PhoneGap bietet Titanium die Möglichkeit, native Hardwarefunktionen und -komponenten direkt anzusprechen.

Man könnte übertrieben behaupten, dass man mit PhoneGap sehr einfach und effektiv Apps erstellen kann, die wie Anwendungen aussehen und anwendungsähnliche Funktionen besitzen, aber immer noch Webseiten sind. Auch Titanium benutzt Webview-Fenster, generiert aber wesentlich mehr nativen Code und produziert Anwendungen, die zu großen Teilen auch wirklich native Anwendungen sind. Das hat dann auch seinen Preis: Entwicklung mit Titanium ist sehr JavaScript-lastig, und erste Ergebnisse sind schwieriger zu realisieren als mit PhoneGap.

Ein anderes Problem, das Titanium mitbringt, ist, dass jeder Bildschirm eine neue WebView-Ansicht darstellt und jeglicher JavaScript-Kontext nicht von einem Bildschirm auf den anderen übernommen wird. Das stellt in der Regel kein großes Problem für die meisten Anwendungen dar, jedoch werden Sie insbesondere bei umfangreicheren Anwendungen eine schlechtere Performance gegenüber einer nativen Objective-C-App feststellen.

Wir möchten noch betonen, dass weder Titanium noch PhoneGap eine »Write once, run everywhere«-Lösung anbietet. Theoretisch können Sie einfachen Webcode schreiben und diesen mithilfe der beiden Lösungen als native Anwendung für iPhone, iPad, Android und, sofern Sie PhoneGap benutzen, auch für BlackBerry, Palm, Symbian und Windows Mobile konvertieren. In der Praxis besitzt aber jede dieser Plattformen andere Oberflächen, sehr oft auch verschiedene API-Aufrufe für die nativen Hardwarefunktionen und andere Bedienkonzepte. Selbst mit Titanium oder PhoneGap empfehlen wir Ihnen, beispielsweise für eine iPhone- und eine Android-Anwendung unterschiedliche Codegrundlagen zu verwenden.

In diesem Kapitel dreht es sich hauptsächlich um PhoneGap. PhoneGap ist näher dran am Thema dieses Buchs, deckt die wichtigsten mobilen Plattformen ab und erlaubt einen leichteren Einstieg als Titanium. Falls Sie sich für die Entwicklung mit Titanium interessieren, empfehlen wir Ihnen, die Quick-Start-Tutorials auf der Titanium-Webseite durchzuarbeiten. Diese zeigen Ihnen, wie Sie schnell eine einfache Titanium-

Hello World-Anwendung erstellen. Sie werden aber schon anhand eines einfachen Beispiels sehen, dass beide Technologien unterschiedliche Ansätze verfolgen – ein einfaches `<p>Hello World!</p>` wie bei PhoneGap funktioniert dort nicht ohne JavaScript.

Ebenfalls ein für uns wichtiger Ansatz ist, dass Sie bei einer Entwicklung mit PhoneGap eine mobile Seite entwickeln und diese in eine App transformieren. Bei Titanium entwickeln Sie spezifisch eine App mit Webtechnologien, aber Sie entwickeln eine Anwendung, die sich nicht als Webseite nutzen lässt. Das ist der große Vorteil von PhoneGap: Egal, ob Sie eine App in einem App Store haben und egal, ob sie erfolgreich ist oder nicht: Sie werden auf jeden Fall eine mobile Webanwendung besitzen, die auf mehr als nur einer Plattform läuft.

3.2 SDKs, IDEs und ADTs

Bevor Sie sich an die native App-Entwicklung wagen können, muss zuerst der Computer mit der nötigen Software und den einzelnen SDKs (Software Development Kits) ausgestattet werden. Die Unterschiede sind immens: Die Entwicklungsumgebung für iOS ist schnell installiert, aber wenn Sie alle Funktionen nutzen wollen, nicht kostenlos. Android ist kostenlos, bedarf aber vieler Schritte, bis die Entwicklungsumgebung für PhoneGap und Titanium schließlich einsatzbereit ist.

3.2.1 Entwicklungsumgebung für iOS-Entwicklung einrichten

Wenn Sie Anwendungen für die iOS-Plattform, also für iPhone, iPod touch oder iPad, erstellen möchten, benötigen Sie:

- einen Apple-Rechner mit mindestens Mac OS X 10.6.3 (Leopard),
- das aktuelle iOS SDK mit der Xcode-Entwicklungsumgebung.

Das iOS SDK erhalten Sie bei Apple unter der Adresse *http://developer.apple.com/iphone*. Sie müssen sich vorher aber als Entwickler bei Apple registrieren. Das erledigen Sie durch den *Register*-Link im iPhone Dev Center. Die Registrierung ermöglicht Ihnen nicht nur den Zugriff auf iPhone-relevante Dokumentationen, sondern auch auf die Entwicklerseiten für den Safari-Browser – vieles ist dort auch für den mobilen Safari relevant – und für Mac OS X-Anwendungen

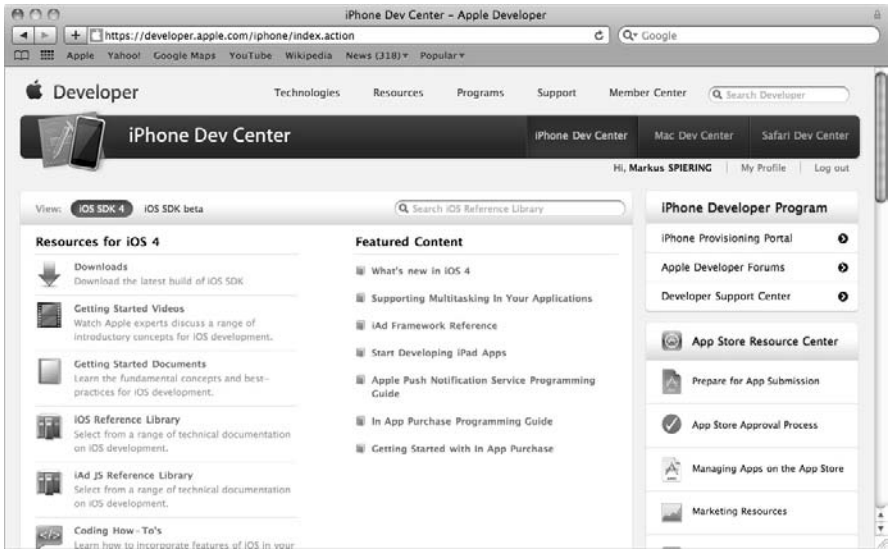


Bild 3.1: Das iPhone Dev Center bietet Zugriff auf alle Entwicklerdokumentationen und das SDK.

Um sich als Entwickler anzumelden, benötigen Sie eine Apple ID. Wenn Sie iTunes-Benutzer sind, ist die Wahrscheinlichkeit schon sehr groß, dass Sie eine solche ID besitzen. Es ist die gleiche ID, mit der Sie in iTunes Musik, Filme oder Apps kaufen. Falls Sie noch keine Apple ID besitzen, kann sie während des Registrierungsprozesses beantragt werden.

Nun gibt Ihnen ein kostenloses Apple Developer-Konto Zugriff auf das SDK, allerdings …

- … können Sie Ihre Anwendungen nicht im App Store veröffentlichen.
- … können Sie nur im iOS-Simulator, aber nicht auf echten Geräten testen.
- … haben Sie keinen Zugriff auf Betaversionen des SDK.
- … haben Sie keinen Anspruch auf Apple-Support.

All diese Dinge gibt es natürlich, aber nicht kostenlos. Um aktiv Anwendungen im App Store zu verkaufen bzw. die Möglichkeit zu haben, auf einem richtigen Gerät zu testen, müssen Sie sich am iPhone Developer-Programm anmelden. Das können Sie als Einzelperson für derzeit 99 Dollar im Jahr oder als Firma für 299 Dollar pro Jahr tun.

Wenn Sie sich als Einzelperson anmelden, haben nur Sie über Ihre Apple ID Zugriff auf die Entwicklerinformationen und Dokumente im iPhone Dev Center. Außerdem wird Ihr Name fest als »Verkäufer« im App Store gelistet. Sie haben keine Möglichkeit, eine Firmenbezeichnung anzugeben.

Möchten Sie iOS-Anwendungen im Namen Ihrer Firma veröffentlichen oder mehreren Entwicklern Zugriff auf die Entwicklerinformationen im iPhone Dev Center geben, müssen Sie sich für einen Firmen-Account anmelden. Dieser kostet nicht nur 200 Dollar mehr im Jahr, sondern erfordert auch Belege dafür, dass Sie rechtlich in der Lage sind, im Namen der Firma am Entwicklerprogramm teilzunehmen. Für alle Anmeldungen zum iPhone Developer-Programm gilt, dass Apple generell eine Überprüfung der

Anmeldung durchführt. Diese kann bei Einzelpersonanmeldungen ein paar Stunden dauern. Bei Firmenanmeldungen kann sich der Prozess mehrere Tage hinziehen.

Teilnahme am kostenpflichtigen Apple-Entwicklerprogramm?
Sie müssen für die meisten Beispiele im Buch wie auch für eigene Projekte, die Sie nicht im iTunes App Store vertreiben möchten, nicht am kostenpflichtigen Apple-Entwicklerprogramm teilnehmen. Allerdings beschäftigt sich ein Teil der folgenden Beispiele mit Funktionen, die Sie nicht im Simulator testen können. Das betrifft beispielsweise den Zugriff auf die Kamera, auf die Bewegungssensoren und den Vibrationsalarm. Auch wenn viele Funktionen kostenfrei auf der Android-Plattform mit einem Android-Gerät testbar sind, gibt es immer noch API-Calls, die nur unter iOS funktionieren. Falls Sie ohnehin planen, eine iOS-Anwendung für das iPhone, das iPad oder den iPod touch zu schreiben und zu veröffentlichen, empfehlen wir Ihnen auch für die Beispiele im Buch die Teilnahme am Developer-Programm.

Im iPhone Dev Center starten Sie den Download des aktuellen iOS SDK. Das fertig installierte SDK benötigt über 6 GByte Speicherplatz und sollte unbedingt im *Developer*-Verzeichnis installiert werden. Dieses Verzeichnis wird als Standardverzeichnis benutzt, und Xcode erwartet, dass sich bestimmte Komponenten dort befinden.

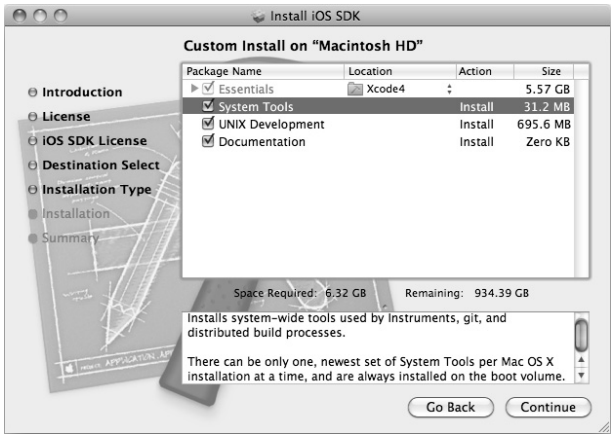


Bild 3.2: Während der Installation werden Sie aufgefordert, die Installationskomponenten auszuwählen. Damit Sie PhoneGap verwenden können, müssen Sie auch die *UNIX Development*-Tools mit installieren.

3.2.2 Entwicklungsumgebung für Android-Entwicklung einrichten

Bei Android gibt es keine Anmeldung, und Google erhebt auch keine Gebühren, um Android-Anwendungen zu veröffentlichen. Allerdings ist der Installationsprozess des Android SDK nicht mit ein paar Mausklicks wie beim iOS SDK erledigt, sondern wesentlich aufwendiger. Ebenfalls im Gegensatz zur iOS-Entwicklung gibt es das Android SDK für Windows (ab Windows XP), für Mac OS (ab Version 10.5.8 und Intel-Macs) sowie für Linux.

Neben dem SDK benötigen Sie auch eine separate Entwicklungsumgebung – diese ist bei iOS mit Xcode bereits dabei. Android unterstützt die beliebte Eclipse-Umgebung

(*http://www.eclipse.org/downloads*), weiterhin JDK (Java Development Kit) 5 oder JDK 6 – wenn Sie nicht mit einem Mac arbeiten – und Oracle (*http://www.oracle.com/technetwork/java/javase/downloads*) oder Apache Ant (*http://ant.apache.org*).

Gegebenenfalls ist eine JDK-Version auf Ihrem Rechner bereits installiert. Apple liefert Mac OS X mit einer eigenen Java-Version aus. Mac-Benutzer müssen daher kein JDK-Paket und auch für die Arbeit mit PhoneGap kein Apache Ant installieren.

Für Windows-Benutzer: Bitte überprüfen Sie, ob Sie entweder die JDK-Version 5 oder 6 installiert haben. Wenn nicht oder wenn sich noch gar kein JDK auf Ihrem Computer befindet bzw. Sie unsicher sind, müssen Sie die Installation jetzt nachholen und sie von der oben genannten Oracle-Seite durchführen.

Wir werden hier mit der Eclipse-Umgebung arbeiten. Die Eclipse-Downloadseite bietet viele Versionen der Umgebung an. Für die Android-Entwicklung können Sie folgende Eclipse-Versionen herunterladen und installieren:

- Eclipse IDE for Java Developers (empfohlen)
- Eclipse IDE for Java EE Developers
- Eclipse for RCP/Plug-in Developers
- Eclipse Classic (ab Version 3.5.1)

Falls Sie sich entschieden haben, mit Eclipse zu arbeiten, empfehlen wir Ihnen, auch das Android Development Toolkit (ADT) für Eclipse zu installieren. Das ADT erweitert die Eclipse-Oberfläche, um komfortabel für Android entwickeln zu können. Sie können beispielsweise leichter Android-Projekte aufsetzen, die Benutzeroberfläche gestalten und auch fertige Programme für den Vertrieb vorbereiten. Das ADT muss innerhalb von Eclipse installiert werden:

1. Starten Sie Eclipse und wählen Sie aus dem *Help*-Menü die Option *Install New Software*. Im darauffolgenden Fenster klicken Sie auf *Add*.

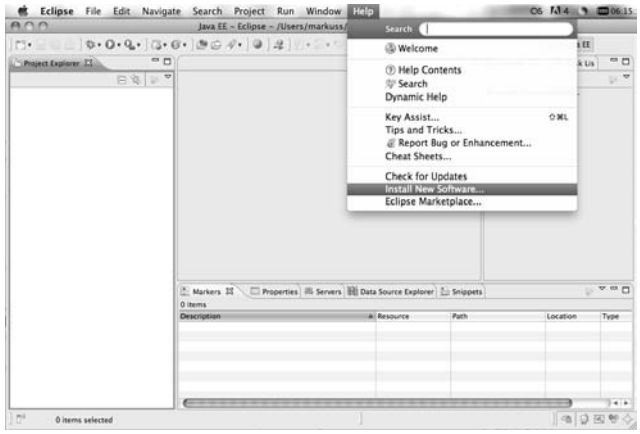
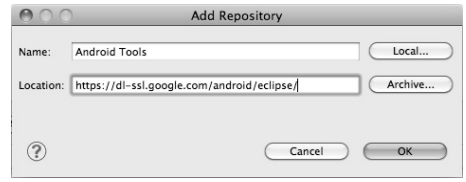


Bild 3.3:
Installation neuer Software für Eclipse.

2. Geben Sie anschließend unter *Name* eine Bezeichnung wie beispielsweise *Android Tools* ein, geben Sie unter *Location* die Webadresse *https://dl-ssl.google.com/android/eclipse/* an und klicken Sie dann auf *OK*.



   Falls Probleme auftreten, versuchen Sie es ohne SSL und verwenden nur *http://* anstatt *https://*.

3. Im Bereich *Available Software* markieren Sie das Kontrollfeld *Developer Tools*. So ist sichergestellt, dass sowohl die Android Development Tools als auch das Android DDMS (Dalvik Debug Monitor Service) mit installiert werden. Klicken Sie anschließend auf *Next*.

   Danach werden beide Komponenten noch einmal angezeigt. Bestätigen Sie mit *Weiter*.

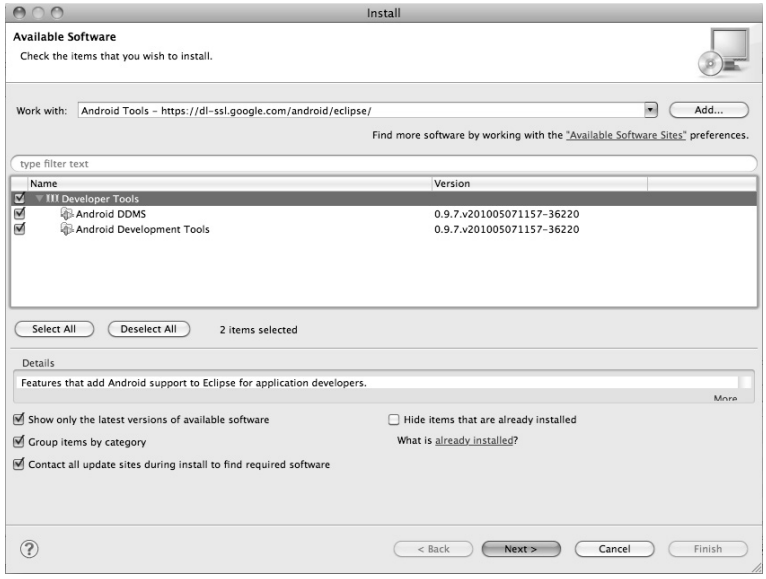


Bild 3.4: Unter *Developer Tools* stellen Sie sicher, dass die zentralen Komponenten installiert werden.

4. Im letzten Schritt müssen Sie noch den Lizenzbedingungen zustimmen und dann, nachdem die einzelnen Komponenten installiert wurden, Eclipse neu starten.

Jetzt haben Sie erst mal die Grundvoraussetzungen geschaffen, um das Android SDK benutzen zu können. Das »Starter Package« des SDK können Sie unter folgender Adresse herunterladen: *http://developer.android.com/sdk/index.html*.

Das Starter Package hat seinen Namen erhalten, da es noch nicht das vollständige SDK ist. Je nachdem, wie umfangreich Ihre Erfordernisse sind, können Sie weitere SDK-Komponenten aus dem Starterpaket hinzufügen. Um es zu installieren, wählen Sie einfach die für Ihr System richtige Datei aus und speichern sie auf Ihrem Computer.

Obwohl Sie den Android-SDK-Ordner überall auf Ihrem Computer speichern können, empfehlen wir, ihn in Ihr Benutzerverzeichnis auf der Festplatte zu sichern.

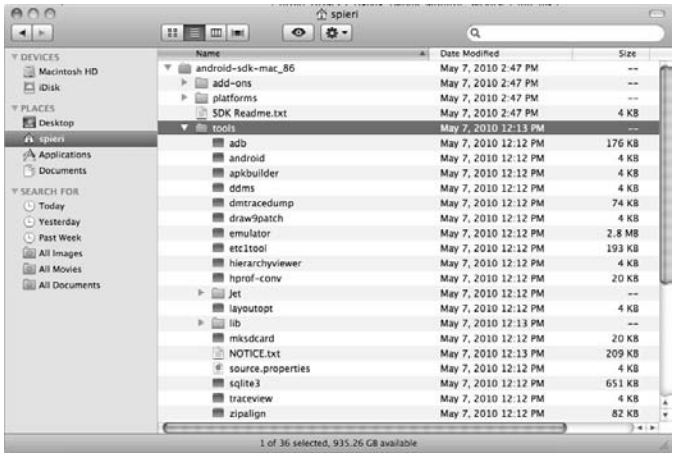


Bild 3.5: Wird das Android SDK in dem persönlichen Benutzerordner abgespeichert, ist es von der Konsole bzw. der Terminalanwendung leicht auffindbar.

Wie Sie im letzten Bild sehen können, befinden sich im *tools*-Verzeichnis des Android SDK viele kleine ausführbare UNIX-Programme wie beispielsweise *adb* oder *android*. Um diese von der Konsole aus aufrufen zu können, muss das System wissen, dass es bei Befehlseingaben auch in dieses Android SDK-Verzeichnis hineinsehen muss, wenn eingegebene Befehle erst einmal unbekannt erscheinen.

Mac OS X

Unter Mac OS X müssen Sie Ihren Texteditor aufrufen und im aktuellen Benutzerverzeichnis nach einer Datei namens *.bash_profile* schauen. Diese werden Sie nur in einem Dateiauswahlfenster sehen, wenn Sie die Option *Versteckte Dateien anzeigen* auswählen, da alle Systemdateien mit einem Punkt beginnen und oft von Texteditoren ausgeblendet werden. Ist diese Datei nicht vorhanden, muss sie mit folgendem Inhalt erstellt werden:

```
export PATH=${PATH}:~/android-sdk-mac_86/tools/
```

Auch wenn die Datei bereits vorhanden ist, müssen Sie die eben genannte Zeile in die Datei einfügen und sie neu abspeichern.

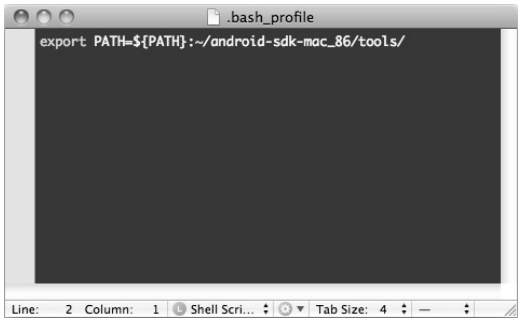


Bild 3.6: Die Datei *.bash-profile*, in TextMate neu angelegt.

Windows

Unter Windows brauchen Sie nur mit der rechten Maustaste auf *Mein Computer* zu klicken und dann *Eigenschaften* auszuwählen. Anschließend öffnen Sie den Bereich *Erweitert*, klicken auf den Button *Environment-Variablen* und doppelklicken auf *Pfad* (unter *Systemvariablen*). Geben Sie dort den vollständigen Pfad zum *tools*-Verzeichnis an.

Jetzt sind Sie fast fertig. Sie haben nun einen Teil des Android SDK bereits auf der Festplatte, aber die wichtigste Komponente, nämlich eine Android-Version, in der wir entwickeln, fehlt noch.

1. Starten Sie dazu Eclipse, wählen Sie aus dem Menü *Eclipse* den Punkt *Preferences* und anschließend im Einstellungsfenster *Android* aus. Hier können Sie angeben, in welchem Verzeichnis Sie das Android SDK gespeichert haben.

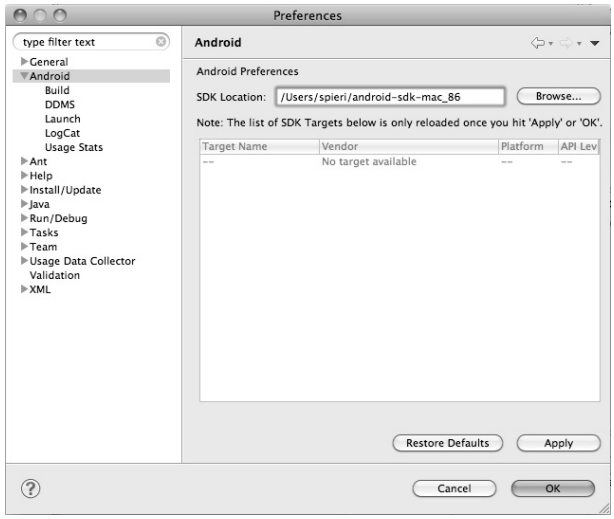


Bild 3.7: Auswahl des Verzeichnisses für das Android-SDK

2. Anschließend rufen Sie im Menü *Window* den Punkt *Android SDK and AVD Manager* auf.

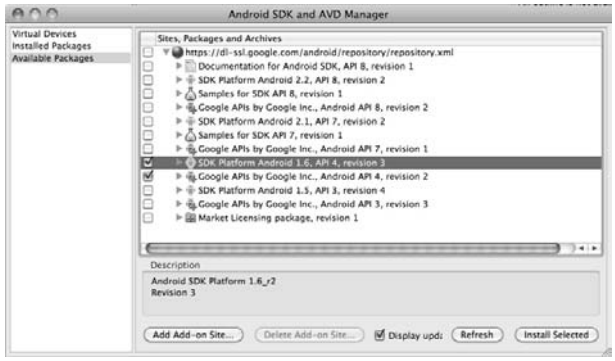


Bild 3.8: Aufruf der Android-Packages im SDK and AVD Manager.

3. Klicken Sie dort auf *Available Packages* und wählen Sie anschließend die folgenden Komponenten aus:
   - *SDK Platform Android 1.6, API 4, Revision 3*
   - *SDK Platform Android 2.2, API 8, Revision 2*
   - *Google APIs by Google Inc., Android API 4, Revision 2*
   - *Google APIs by Google Inc., Android API 8, Revision 2*
4. Klicken Sie nach der Auswahl auf *Install Selected*, dann im nächsten Fenster auf *Accept All* und auf *Install*. Die Komponenten werden nun heruntergeladen und installiert. Das *Android SDK and AVD Manager*-Fenster sollte diese Komponenten jetzt unter *Installed Packages* auflisten:

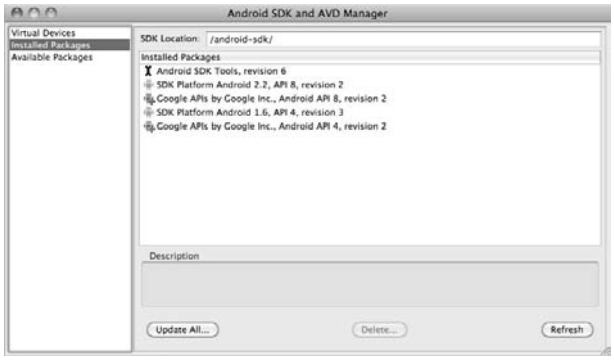


Bild 3.9: Alternativ können Sie auch alle Komponenten auswählen, die im *Android SDK and AVD Manager* angeboten werden.

Wie Sie sehen, ist die Installation recht umständlich, aber mit dem letzten Schritt haben Sie alle benötigten Komponenten für die Android-Entwicklung installiert. Als Nächstes beginnen Sie damit, das erste der nativen Frameworks zu installieren. Während die Installation von Titanium relativ einfach ist, wird Ihnen die PhoneGap-Installation für Android, aber auch für iOS, möglicherweise wieder etwas Kopfschmerzen bereiten.

3.2.3 PhoneGap installieren

Nachdem Sie die SDKs für beide Plattformen installiert haben, können Sie sich an den Download und die Installation von PhoneGap machen. Wenn Sie denken, dass die Installation des Android SDK schon anspruchsvoll war, dann warten Sie jetzt mal die PhoneGap-Installationen ab!

Um die jeweils aktuelle Version von PhoneGap zu erhalten und die notwendigen Installationsskripte am sichersten auszuführen, sollten Sie sich bei GitHub anmelden. GitHub ist ein sehr beliebter Onlinedienst, der dabei hilft, Code mit passendem Versionsmanagement online zu speichern, zu tauschen und generell online verfügbar zu machen. Sobald Sie Mitglied bei GitHub sind und GitHub auf Ihrem Computer installiert haben, können Sie über die Kommandozeile aktuelle Programmpakete einfach auf Ihrem Computer installieren. Sie können sich dabei immer sicher sein, dass Sie die aktu-

elle Version einer Anwendung aus dem Internet herunterladen. Bevor Sie GitHub installieren, ist es ratsam, sich unter *www.github.com* kostenlos anzumelden.

Mit den folgenden Schritten installieren Sie GitHub:

1. Wenn Sie einen Mac-Rechner verwenden, können Sie die aktuelle Version von GitHub unter der Adresse *http://code.google.com/p/git-osx-installer* herunterladen. Auf dieser Seite müssen Sie nur auf *Download the packages here* klicken, dann auf der Folgeseite die für Ihr System passende Version herunterladen und das Installationsprogramm ausführen. Den Rest übernimmt GitHub automatisch.

2. Sind Sie Windows-Benutzer, ist der Prozess ähnlich einfach: Besuchen Sie die Seite *http://code.google.com/p/msysgit,* und wählen Sie einen der angebotenen *.exe*-Downloads aus der *Feature Downloads*-Spalte zur Installation von GitHub für Windows. Sie können während des Installationsprozesses alle Standardeinstellungen so lassen und die Dialoge immer bestätigen.

3. Um eine sichere Verbindung zwischen Ihrem Rechner und GitHub herzustellen, sollten Sie nach erfolgreicher GitHub-Installation SSH-Schlüsselpaare anlegen. Der Prozess ist für Windows und Mac OS X gleich, nur dass Sie unter Windows als Konsole nicht das normale Kommandozeilenfenster (*cmd*) benutzen, sondern die Bit-Bash-Konsole, die Sie über das Startmenü im *GitHub*-Ordner finden. Auf einem Mac können Sie die Terminalanwendung verwenden, die Sie im Order *Programme/Dienstprogramme* finden.

4. Wenn Sie noch nie einen SSH-Schlüssel erstellt haben, geben Sie in der Bit-Bash-Konsole (Windows) oder in der Terminalanwendung (Mac OS) folgende Zeile ein:

```
ssh-keygen -t rsa -C "ihre@emailadresse.de"
```

   Den letzten Teil der Zeile müssen Sie natürlich durch Ihre E-Mail-Adresse ersetzen. Bei der ersten Abfrage drücken Sie einfach Enter, bei der zweiten müssen Sie sich ein Passwort ausdenken und es dann noch einmal wiederholen.

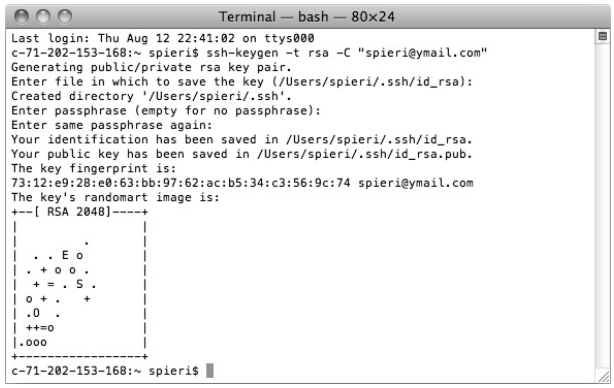


Bild 3.10: So müsste Ihr Terminalfenster aussehen, wenn der SSH-Key erfolgreich erstellt wurde.

   Anschließend wird Ihr Schlüssel erstellt, und Sie können ihn über die folgende Zeile in die Zwischenablage kopieren:

```
cat ~/.ssh/id_rsa.pub | pbcopy
```

5. Sobald sich der Schlüssel in der Zwischenablage befindet, rufen Sie in Ihrem Browser die Seite *https://github.com/account#ssh_bucket* auf, klicken dort auf *Add Public Key* und kopieren im anschließenden Fenster den Schlüsselinhalt aus der Zwischenablage in das Feld *Key*.

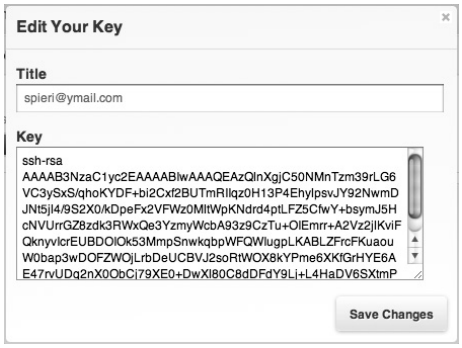


Bild 3.11: Der Schlüssel wird in das Feld *Key* kopiert, das Titelfeld kann auch freigelassen werden. GitHub füllt es dann mit der E-Mail-Adresse aus, mit der der Schlüssel im ersten Schritt generiert wurde.

6. Wenn Sie alles richtig gemacht haben, können Sie jetzt testweise eine SSH-Verbindung zu GitHub aufbauen. Geben Sie dazu in der Terminalanwendung bzw. unter Windows in der Bit Bash-Konsole folgende Zeile ein:

```
ssh git@github.com
```

7. Anschließend geben Sie, wenn es Ihre erste SSH-Verbindung ist, Ihr selbst gewähltes SSH-Passwort ein. Passt es, erhalten Sie die Meldung, dass Sie erfolgreich erkannt wurden, GitHub aber keinen Shell-Zugang erlaubt:

Bild 3.12: GitHub erlaubt noch keinen Zugriff.

Da Sie sich jetzt mit der Konsole bzw. dem Terminalprogramm schon etwas warm gemacht haben, können wir als Nächstes PhoneGap für Xcode installieren.

PhoneGap für die iOS-Entwicklung installieren

Auch hier sind die ersten Schritte der PhoneGap-Installation für iOS nicht per Mausklick gemacht, sondern erfordern Eingaben in die Terminalkonsole. Rufen Sie die Terminalanwendung auf, und gehen Sie zu einem Verzeichnis, in dem Sie den Phone-

Gap-Ordner installieren wollen. Wir empfehlen Ihnen, dieses Verzeichnis nicht auf den Desktop zu legen. Hier haben wir es in den Hauptordner des Benutzers gelegt.

1. Starten Sie den Kopiervorgang der neuesten PhoneGap-Software direkt von GitHub mit folgendem Befehl von der Konsole:

```
git clone git://github.com/phonegap/phonegap-iphone.git
```

2. Dadurch werden alle Komponenten heruntergeladen und in den Ordner *phonegap-iphone* gespeichert. Ist dieser Vorgang beendet, navigieren Sie in den neu erstellten Ordner:

```
cd phonegap-iphone
```

3. Sind Sie in den neuen Ordner gewechselt, geben Sie folgende Zeile ein und drücken anschließend Enter:

```
git submodule init
```

4. Und anschließend, gefolgt ebenfalls mit Enter:

```
git submodule update
```

Ihr Terminalfenster müsste jetzt wie folgt aussehen:

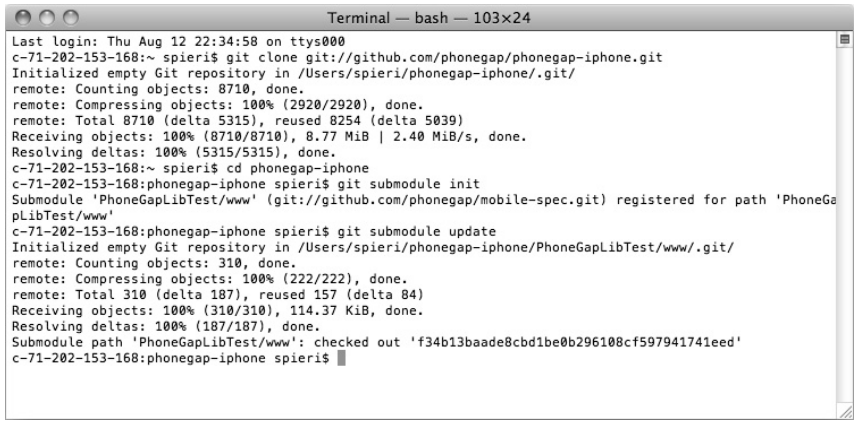


Bild 3.13: Blick auf das Terminalfenster: Soweit ist noch alles in Ordnung.

5. Jetzt kommt der Moment der Wahrheit, und es wird sich herausstellen, ob Sie bei der Installation des iPhone SDK auch die UNIX-Komponenten mit installiert haben. Geben Sie, ebenfalls im Konsolenfenster, einfach den folgenden Begriff mit einem anschließenden Enter ein:

```
make
```

Dieser Befehl führt nun ein Skript aus, das die Datei *PhoneGapLibInstaller.pkg* im Verzeichnis *phonegap-iphone* erstellt:

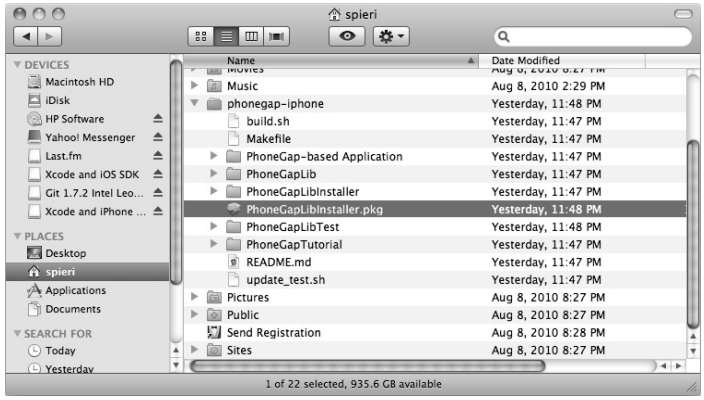


Bild 3.14: Die Datei *PhoneGap-LibInstaller.pkg* wird erstellt

6. Stellen Sie jetzt sicher, dass Xcode nicht im Hintergrund läuft, und starten Sie dann das Installationsprogramm *PhoneGapLibInstaller.pkg* mit einem Doppelklick. Das Programm installiert nun unter anderem die PhoneGapLib und die PhoneGap Xcode-Vorlage (Template) im Xcode-Bereich, sodass Sie beim nächsten Start von Xcode auch ein PhoneGap-Projekt beginnen können.

7. Um zu überprüfen, ob bislang alles richtig verlief, führen Sie nach Beendigung der Installation das Programm Xcode aus. Xcode befindet sich zusammen mit den anderen iPhone-Entwicklungsprogrammen aus dem iOS SDK im Hauptverzeichnis der Festplatte (Macintosh HD) unter *Developer* und *Applications*.

8. Nach dem Programmstart werden Sie gefragt, was Sie tun möchten. Wählen Sie im Startbildschirm entweder *New Xcode Project* aus oder gehen Sie in das Xcode-Menü *File/New Project*. Sollte in dem nun erscheinenden Bildschirm auf der linken Seite unter *User Templates* die PhoneGap-Vorlage vorhanden sein und das Fenster ungefähr wie folgt aussehen, ist das definitiv ein gutes Zeichen und zeigt, dass die Installation erfolgreich war.

Bild 3.15: Bei erfolgreicher Installation sehen Sie ein PhoneGap-Template als auswählbare Vorlage in Xcode.

PhoneGap für die Android-Entwicklung installieren

Jetzt, da Sie bereits so gut in Übung sind, können Sie auch noch die Android-Version von PhoneGap auf Ihrem Computer installieren.

Die Installation der wichtigsten Android-Komponenten – JDK, Android SDK, Android-Komponenten, Eclipse und die Android Developer Tools (ADT) für Eclipse – haben wir bereits beschrieben. Für die Android-PhoneGap-Entwicklung benötigen Windows-Benutzer noch zwei weitere Komponenten:

- Apache Ant, das ebenfalls kostenfrei von der Adresse *http://ant.apache.org* heruntergeladen werden kann.
- Ruby, das einfach mit einem Windows-Installer auf einem Windows-Rechner installiert werden kann. Der Windows-Installer ist unter *http://rubyinstaller.org* erhältlich.

1. Unter Mac OS X müssen die beiden Komponenten nicht zusätzlich installiert werden. Um jetzt die Android-Version zu PhoneGap zu bringen, geben Sie in der Konsole bzw. auf einem Mac im Terminalprogramm folgende Zeile ein:

```
git clone git://github.com/phonegap/phonegap-android.git
```

2. Stellen Sie sicher, dass Sie zuerst in das Verzeichnis navigieren, in dem Sie die Android-Version von PhoneGap installieren möchten. In unserem Beispiel nehmen Sie wieder das Hauptverzeichnis des derzeit angemeldeten Benutzers.

3. Nach der Installation wechseln Sie in das gerade erstellte Verzeichnis:

```
cd phonegap-android
```

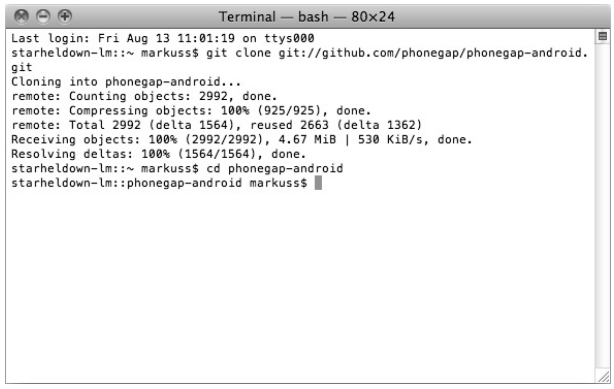


Bild 3.16: So sollte Ihr Terminalfenster nach erfolgreicher Installation von PhoneGap für Android aussehen.

4. Bevor Sie ein neues Android-Projekt erstellen, müssen Sie Ruby aus der Konsole bzw. aus dem Terminal aufrufen, um eine *PhoneGap.jar*-Datei sowie ein einfaches Android-Eclipse-Projekt zu erstellen. Um zu testen, ob Sie alle Komponenten richtig installiert haben, bietet PhoneGap eine Beispielanwendung namens *droidgap*. Diese gilt es mit dem folgenden Ruby-Befehl aufzurufen und zu generieren:

```
ruby ./droidgap ~/android-sdk-mac_86 test_app com.spieri.testapp example/
~/beispiel/test_droid_phonegap
```

Wenn keine Fehler aufgetreten sind, sollte Ihr Terminalfenster folgenden Inhalt anzeigen:

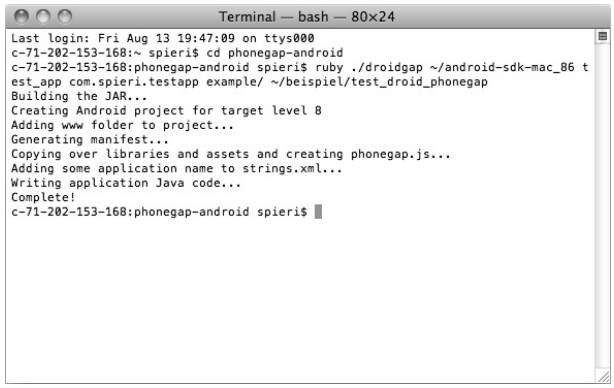


Bild 3.17: Die Beispielanwendung droidgap ist fertig.

5. Jedes Mal, wenn Sie ein neues PhoneGap-Android-Projekt starten, müssen Sie diesen Ruby-Befehl ausführen, um die Basiskomponenten erstellen und mit Eclipse kompilieren zu können. Hier ist die Syntax des Ruby-Befehls:

```
ruby ./droidgap "[Pfad zum Android SDK]\" [App-Name] [Paketname] "[www]"
"[Zielpfad]"
```

Parameter	*Beschreibung*
./droidgap	Da hier die Beispielanwendung *droidgap* aus dem PhoneGap-Android-Paket generiert wird, setzen Sie *./droidgap* als Ausgangsparameter für die Quelldateien.
Pfad zum Android SDK	Hier spezifizieren Sie den Ordner, in dem Sie das Android SDK installiert bzw. hineinkopiert haben. Beachten Sie bei der Angabe des Parameters, dass Sie keine Leerzeichen im Pfad verwenden.
App-Name	Der Name Ihrer Anwendung.
Paketname	Der in der Entwicklung verwendete Paketname.
www	Pfad zum *www*-Ordner. Im sogenannten *www*-Ordner befinden sich die Quelldateien eines PhoneGap-Projekts, also die HTML-, CSS- und JavaScript-Dateien zusammen mit allen Ressourcen, die Sie einzubinden wünschen.
Zielpfad	Der Pfad, in dem Sie die Projektdateien generieren möchten. Ganz wichtig: Der Pfad darf sich nicht innerhalb des von Eclipse benutzten Workspace befinden. Zur Erinnerung: Den Workspace mussten Sie beim Start von Eclipse festlegen.

Falls der Ruby-Befehl nicht erfolgreich ausgeführt wurde und Sie Fehlermeldungen erhalten haben, kann das unter anderem folgende Ursachen haben:

1. Überprüfen Sie, ob die Pfadvariable zum Android SDK-Ordner richtig gesetzt wurde. Sagt Ihnen der Fehler, dass der Befehl `android` nicht erkannt wurde, liegt das an der fehlenden Zuweisung. Sie müssen das Android SDK-Verzeichnis im Terminalfenster aufgelistet sehen, wenn Sie folgenden Befehl in der Konsole ausführen:

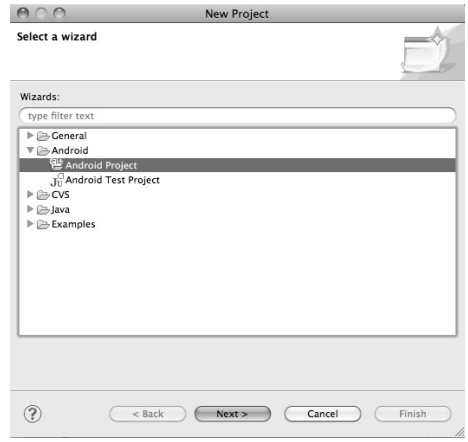
```
echo_android_pfad.PNG
echo $PATH
```

2. Überprüfen Sie, wie Sie den Verzeichnisnamen zum Android SDK im Ruby-Befehl angegeben haben. Er darf keine Leerzeichen enthalten. Unter Windows sollte die Verzeichnisangabe mit doppelten Anführungszeichen erfolgen:

```
"C:/development/android/"
```

Wenn wir später auf die Entwicklung einer eigenen Android-Anwendung mit PhoneGap eingehen, werden wir Ihnen auch noch zeigen, welche Schritte Sie für ein eigenes Projekt durchführen müssen. Bislang geht es erst einmal darum, die Installation zu verifizieren.

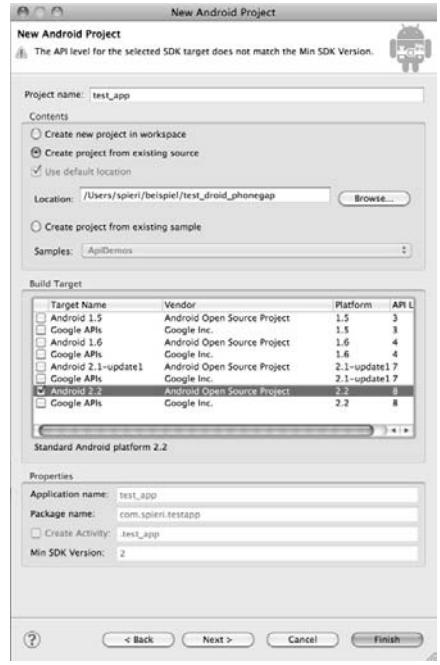
1. Um die gerade erstellten Dateien als erste Android-Anwendung zu sehen, müssen Sie jetzt die Eclipse-Entwicklungsumgebung starten. Gehen Sie dort in das Menü *File/New,* und klicken Sie auf den Menüeintrag *Project.* Im darauffolgenden Fenster wählen Sie unter *Android* die Unterkategorie *Android Project* aus (siehe Bild rechts):



2. Klicken Sie auf *Next* und im folgenden Fenster auf die Option *Create project from existing source.* Neben dem *Location*-Texteingabefeld steuern Sie über den *Browse*-Button das gerade erstellte Verzeichnis an.

3. Wenn Sie die gleichen Werte wie in unserem Beispiel verwendet haben, befindet sich das Verzeichnis im Hauptverzeichnis des aktuellen Benutzers unter *beispiel/test_droid_phonegap*. Wählen Sie es aus, und bestätigen Sie mit *Open*.

4. Als Letztes müssen Sie noch eine SDK-Version als Build Target auswählen. In diesem Beispiel verwenden Sie Android 2.2. Die Warnung am oberen Fensterrand können Sie in diesem Fall ignorieren. Nach einem Klick auf *Finish* wird das Projekt erstellt. Nach diesem kurzen Prozess werden am unteren Rand des Eclipse-Fensters drei Fehlermeldungen angezeigt. Auch diese können Sie großzügig ignorieren.



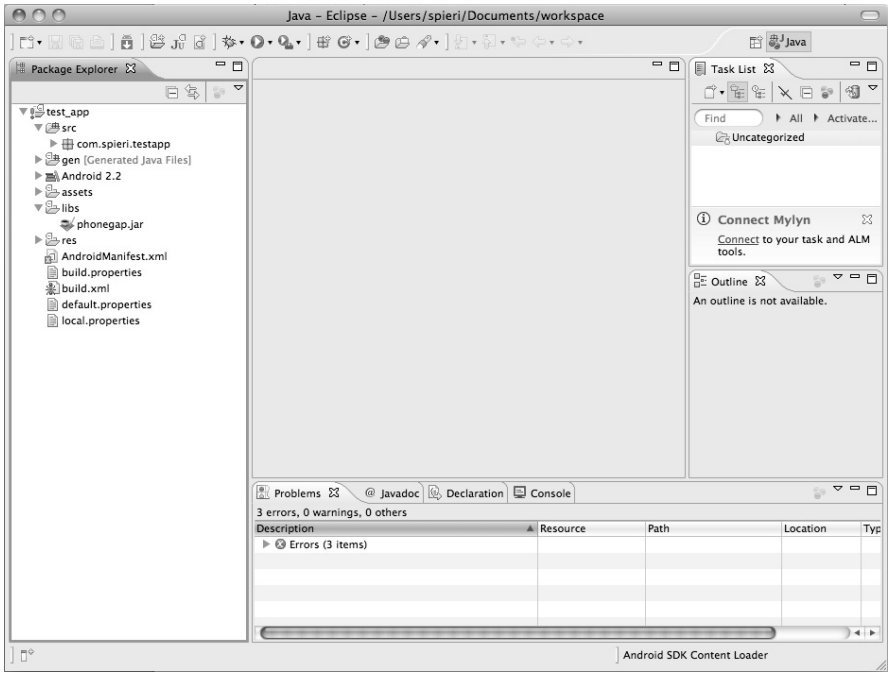


Bild 3.18: So sollte Ihr Eclipse-Fenster aussehen, nachdem Sie das Beispielprojekt erstellt und den kleinen Pfeil neben *test_app* im *Package Explorer* angeklickt haben.

5. Im *Package Explorer* navigieren Sie nun in das Verzeichnis *test_app/libs* zur Datei *phonegap.jar*. Markieren Sie die Datei mit der rechten Maustaste, und wählen Sie aus dem Kontextmenü *Build Path* und den darin befindlichen Menüpunkt *Add to Build Path*:

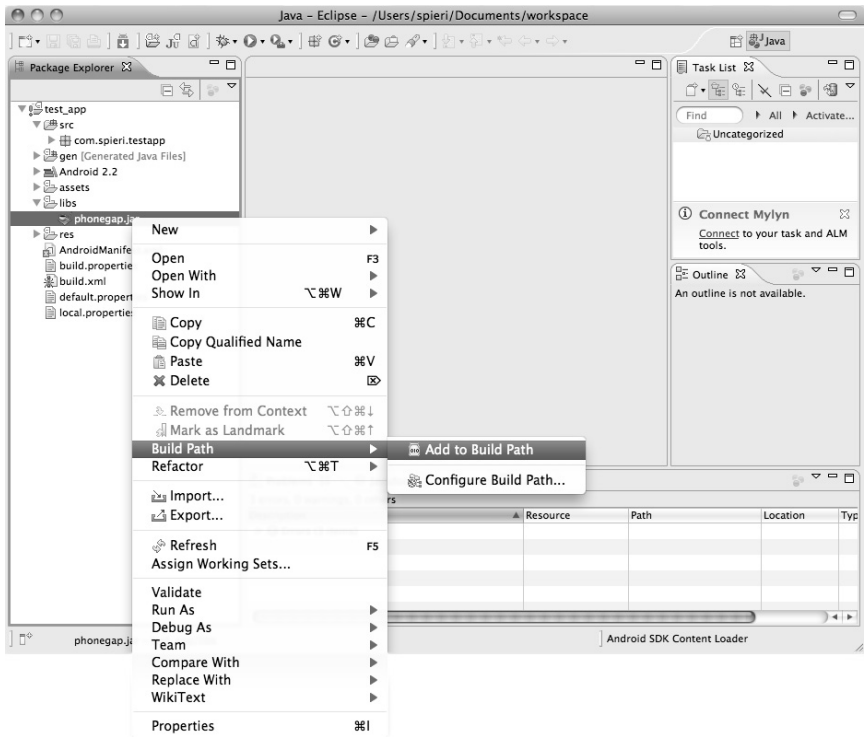


Bild 3.19: Der richtige Weg zum BuildPath.

Dieser Schritt ist unbedingt notwendig, da hiermit der PhoneGap-Java-Code in den Kompilierungsprozess mit eingebunden wird. Sobald dieser Schritt durchgeführt ist, sind auch die zuerst angezeigten Fehler verschwunden.

6. Bevor das Beispiel ausgeführt werden kann, muss ein virtuelles Android-Gerät – also ein passender Simulator – eingerichtet werden. Rufen Sie dazu wieder den *Android SDK and AVD Manager* aus dem *Help*-Menü auf. Klicken Sie auf den Bereich *Virtual Devices* und anschließend auf *New*.

7. Im nächsten Fenster können Sie die Eigenschaften für ein neues, virtuelles Android-Telefon eingeben. Wählen Sie bei *Name* eine griffige Bezeichnung, und geben Sie unter *Target* die Android-Version an, die Sie auch als Build Target bei der Anlage des Projekts gewählt haben. In diesem Beispiel war das *Android 2.2 – API Level 8*. Die anderen Angaben können Sie bislang noch vernachlässigen:

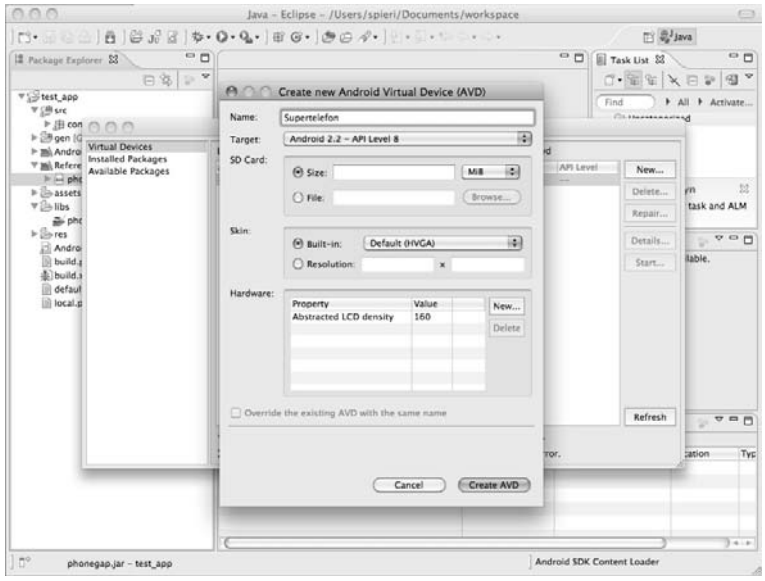


Bild 3.20: Die Eigenschaften des virtuellen Android-Supertelefons

8. Um das neue virtuelle Telefon zu speichern, klicken Sie einfach auf *Create AVD* und schließen den *Android SDK and AVD Manager*.

9. Klicken Sie im *Package Explorer* jetzt auf den obersten Eintrag, also auf den Anwendungsnamen *test_app*, und anschließend auf den grünen Play-Button in der oberen Werkzeugleiste. Im Dialogfenster *Run As* wählen Sie die Option *Android Application*:

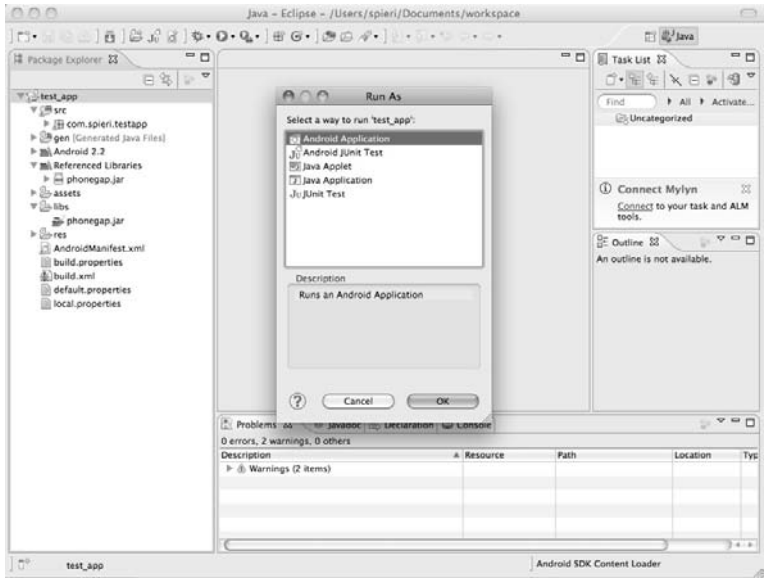


Bild 3.21: Wahl des Applikationstyps

Anschließend wird das neu angelegte virtuelle »Supertelefon« geladen. Der Zeitraum vom Start des Simulators bis zur fertigen Installation der Android-Anwendung kann mehrere Minuten betragen. Zum Schluss sollte folgender Bildschirm erscheinen:

Bild 3.22: Willkommensbildschirm des Supertelefons.

Sehen Sie ihn, haben Sie bislang alles richtig gemacht. Wenn Sie ein wenig mit der Beispielanwendung herumspielen, werden Sie merken, dass die Funktionen zum Accelerometer, der Geoposition und der Beep-Funktion nicht funktional sind. Das liegt aber nicht an PhoneGap oder dem Beispielprogramm, sondern daran, dass der Simulator diese Funktionen nicht wiedergeben kann.

3.3 iOS-Apps mit PhoneGap entwickeln

Die richtige Entwicklungsumgebung einzurichten, war ein hartes Stück Arbeit, und vielerorts können sich während der Installation, gerade für die Android-Entwicklung, Fehler einschleichen. Bleibt zu hoffen, dass PhoneGap in Zukunft einfacher zu bedienende Methoden findet. Dafür wird jetzt die eigentliche Entwicklungsarbeit und das Konvertieren von Webanwendungen in native Anwendungen mittels PhoneGap wieder einfacher.

3.3.1 Eine Anwendung für iOS erstellen

Eine eigene iOS-Anwendung wird durch die Anlage eines Xcode-Projekts mit einer PhoneGap-Vorlage begonnen.

1. Starten Sie dazu Xcode, gehen Sie in das Menü *File* und anschließend auf *New Project*. Im folgenden Fenster wählen Sie unter *User Templates* den Punkt *PhoneGap* aus und markieren im Hauptteil des Fensters das Symbol mit dem Namen *PhoneGap-based Application*.

2. Klicken Sie dann auf *Choose,* und geben Sie den Namen des Projektverzeichnisses an, unter dem PhoneGap die notwendigen Dateien anlegen soll:

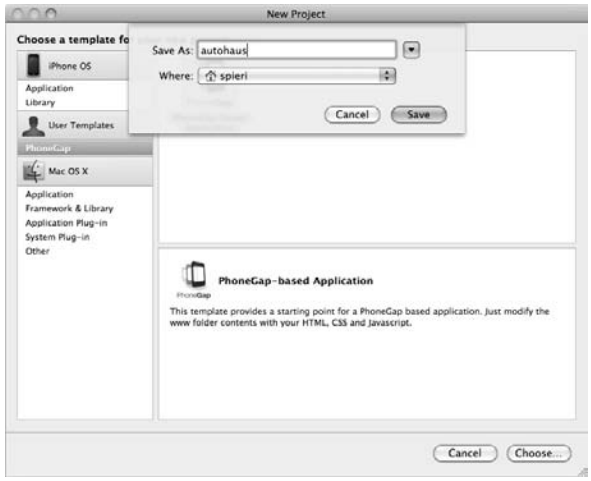


Bild 3.23: Das neue iPhone-Projekt anlegen.

3. Da wir das schon einmal verwendete Beispiel mit dem Oldtimer-Autohaus auch gern als iPhone-Anwendung hätten, nehmen Sie im Beispiel den Verzeichnisnamen *autohaus*. Nachdem Sie die Auswahl mit *Save* bestätigt haben, fängt Xcode an, alle notwendigen Dateien in diesem neuen Verzeichnis anzulegen und weitere Unterverzeichnisse zu generieren. Das Xcode-Fenster sollte für Sie wie folgt aussehen:

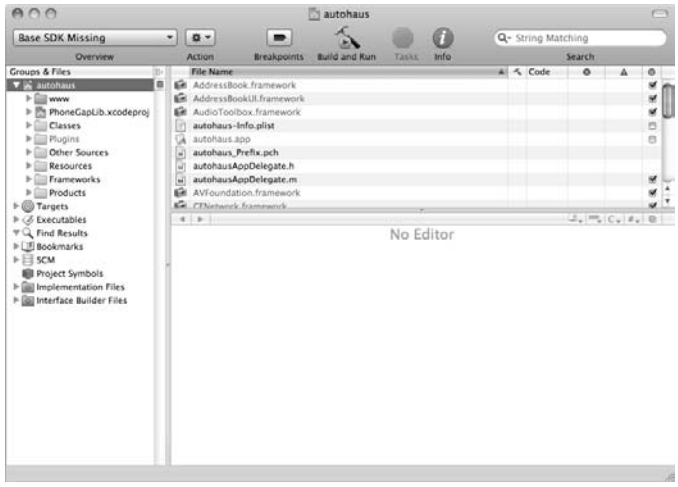


Bild 3.24: Xcode legt das Applikationsgerüst für Sie an.

4. Im neu angelegten Verzeichnis finden Sie nicht nur zahlreiche iOS-spezifische Dateien, Icons und die Xcode-Projektdatei *autohaus.xcodeproj*, sondern auch ein Verzeichnis mit dem Namen *www*.

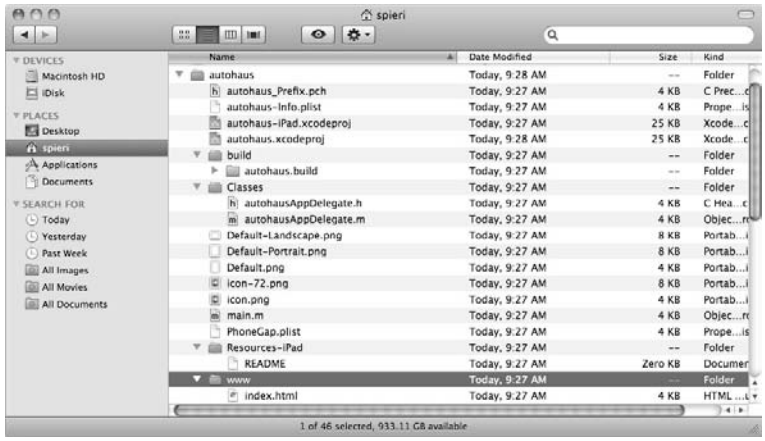


Bild 3.25: In das Verzeichnis www gelangen alle Webdateien.

5. Wie bereits erwähnt, ist dies das Verzeichnis, in das all Ihre Ressourcen und Webdateien kommen. Die dort befindliche *index.html* ist die Datei, die zuerst beim Programmstart aufgerufen wird. Sie hat bei einer »frischen Installation« den folgenden Inhalt:

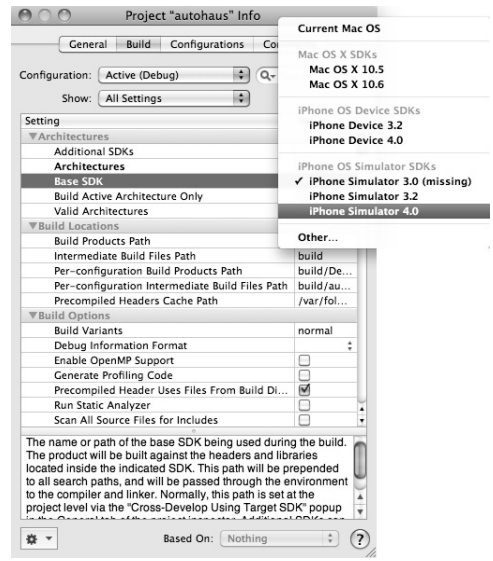
```
<html>
  <head>
   <!-- Change this if you want to allow scaling -->
    <meta name="viewport" content="width=default-width; user-scalable=no" />

    <meta http-equiv="Content-type" content="text/html; charset=utf-8">

    <title>autohaus</title>

   <!-- iPad/iPhone specific css below, add after your main css >
   <link rel="stylesheet" media="only screen and (max-device-width: 1024px)"
href="ipad.css" type="text/css" />
   <link rel="stylesheet" media="only screen and (max-device-width: 480px)"
href="iphone.css" type="text/css" />
   -->
   <script type="text/javascript" charset="utf-8"
src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

   // If you want to prevent dragging, uncomment this section
   /*
   function preventBehavior(e)
   {
      e.preventDefault();
    };
   document.addEventListener("touchmove", preventBehavior, false);
   */

   function onBodyLoad()
   {
```

```
      document.addEventListener("deviceready",onDeviceReady,false);
   }

   /* When this function is called, PhoneGap has been initialized and is
ready to roll */
   function onDeviceReady()
   {
      // do your thing!
   }

    </script>
  </head>
  <body onload="onBodyLoad()">

  </body>
</html>
```

Sie sehen, es handelt sich um eine ganz gewöhnliche HTML-Datei mit CSS-Verweisen und etwas JavaScript. Auf die einzelnen Befehle und Funktionen wird noch genauer eingegangen. Um aber überhaupt etwas auf dem Bildschirm zu sehen, schreiben Sie die folgende Zeile am Ende der Datei *index.html* zwischen die `body`-Tags:

```
<h1>Ich bin eine App!</h1>
```

Jetzt lassen Sie uns diesen einfachen Code schon einmal in eine native Anwendung konvertieren. Zurück in Xcode, bemerken Sie wahrscheinlich, dass noch keine SDK-Version in der oberen linken Ecke ausgewählt ist und Sie dort *Base SDK Missing* sehen.



Um die SDK-Version festzulegen, gehen Sie im *Project*-Menü in Xcode auf den Menüpunkt *Edit Project Settings* und im daraufhin sich öffnenden Fenster in den Bereich *Build*. Unter *Architecture* können Sie dort ein *Base SDK* festlegen:

Falls in dem oberen linken Menü noch *Device* aufgelistet ist, klicken Sie bitte darauf und verwenden folgende Auswahl:

```
Simulator

Debug

iPhone Simulator
```

Unter Android (2.2) haben Sie derzeit leider noch weniger Einsicht in die auf dem Gerät erstellten SQLite-Datenbanken. Sobald vom aktuellen Origin eine Datenbank erstellt wurde, können Sie lediglich über das Menü *Mehr/Einstellungen/Cache löschen* den gesamten Webcache einschließlich aller Datenbanken löschen.

Jetzt ist es so weit: Sie können auf *Build and Run* in der Mitte der Xcode-Werkzeugleiste klicken, sich zurücklehnen und nach dem Laden des iPhone-Simulators Ihre erste eigene iPhone-Anwendung bestaunen.

Kurz nach dem Start des iPhone Simulators wird der folgende Bildschirm angezeigt. Dieser hat noch nichts mit unserem HTML-Code zu tun und – keine Angst – wird auch nicht bei jeder mit PhoneGap erstellten Anwendung angezeigt. Das ist, wie bei Web Apps auf dem iPhone auch, der uns bekannte Startup-Bildschirm, den wir gleich modifizieren werden.

Die Zeile `<h1>Ich bin eine App!</h1>`, die wir der *index.html* noch hinzugefügt haben, ist bislang der einzige Inhalt unserer Anwendung.

Bild 3.26: Die erste eigene iPhone-Anwendung.

Schauen wir uns jetzt noch einmal den Code in der Datei *index.html* genauer an:

```
<!-- Change this if you want to allow scaling -->
    <meta name="viewport" content="width=default-width; user-scalable=no" />
```

Diese Zeile sollte Ihnen bekannt vorkommen. Die `meta`-Angabe unterbindet, dass der Inhalt in der gesamten verfügbaren Bildschirmbreite angezeigt wird, der Benutzer ihn aber nicht vergrößern oder verkleinern kann.

Für den Fall, dass Sie unterschiedliche CSS-Definitionen für das iPhone und das iPad verwenden möchten, kommen die folgenden Zeilen zum Einsatz. Beide sind bislang noch ausgeklammert, und die jeweiligen CSS-Dateien *ipad.css* und *iphone.css* sind nicht mit im Lieferumfang von PhoneGap enthalten, sondern müssen von Ihnen selbst erstellt werden.

```
<!-- iPad/iPhone specific css below, add after your main css >
   <link rel="stylesheet" media="only screen and (max-device-width: 1024px)"
href="ipad.css" type="text/css" />
   <link rel="stylesheet" media="only screen and (max-device-width: 480px)"
href="iphone.css" type="text/css" />
   -->
```

Die nächste JavaScript-Funktion ist besonders wichtig, wenn Sie Ihre Anwendung so gestalten möchten, dass der Benutzer den Inhalt nicht nach oben oder unten verschieben kann. In der *index.html*-Datei ist die Funktion auskommentiert, also nicht wirksam. Entfernt man die Kommentare, erhält man nicht nur folgenden Code, sondern auch das entsprechende Verhalten:

```
function preventBehavior(e)
   {
      e.preventDefault();
    };
   document.addEventListener("touchmove", preventBehavior, false);
```

Ohne die `preventBehavior()`-Funktion kann der Benutzer mit dem Finger den in der App angezeigten Inhalt nach oben und nach unten ziehen. Das wird notwendig, wenn Sie eine Anwendung gestalten, deren Inhalt über die normale Bildschirmdimension hinausgeht.

Bild 3.27: Ohne die `preventBehavior()`-Funktion.

Mit der `preventBehavior()`-Funktion kann der Benutzer lange versuchen, den Bildschirm mit den Fingern nach oben oder unten zu bewegen, es wird ihm nicht gelingen. Bedingung für eine gute Bedienung: Alle Bildschirmelemente der Anwendung müssen in die Bildschirmdimensionen passen.

Bild 3.28: Mit der `preventBehavior()`-Funktion.

Die restlichen JavaScript-Funktionen dienen dazu, zu überprüfen, ob PhoneGap richtig initialisiert wurde und die Anwendung ausgeführt werden kann.

Wir werden jetzt damit beginnen, die Beispielwebseite vom Anfang des Buchs in das Projektverzeichnis zu kopieren und die Seite für eine native Anwendung umzuschreiben. Dazu ist es erforderlich, dass Sie alle dafür benötigten Dateien in das neue *www*-Verzeichnis des *autohaus*-Projekts kopieren. Das Verzeichnis des ersten Autohaus-Beispiels, in dem Sie auch erstmalig iWebKit als Oberfläche verwendet haben, sah wie folgt aus:

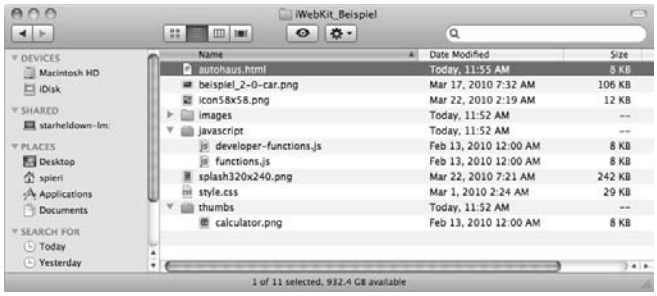


Bild 3.29: Kopieren Sie den gesamten Inhalt in das *www*-Verzeichnis Ihres PhoneGap-Projekts, inklusive der iWebKit-Framework-Ordner für Grafiken, der CSS-Definitionen und JavaScript.

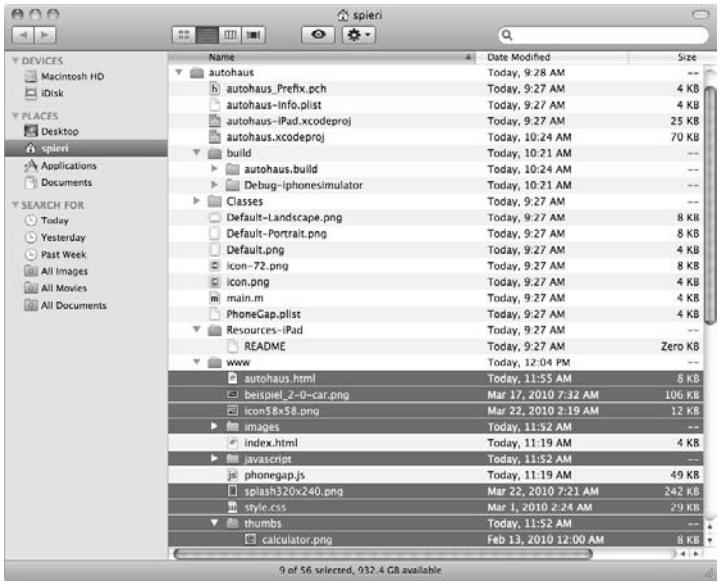


Bild 3.30:
Die neuen Dateien werden durch eine Markierung hervorgehoben.

Jetzt geht es darum, die bestehende *index.html* von PhoneGap und die neue Seitendatei namens *autohaus.html* intelligent zu verbinden. Folgende Zeilen aus dem Header-Bereich werden in die bestehende PhoneGap-*index.html*-Datei übernommen:

```
<link rel="stylesheet" type="text/css" href="style.css">
```

Dieser Verweis wird die im Verzeichnis befindlichen iWebKit-Stylesheets mitladen. Da Sie diese Unterscheidung jetzt nicht benötigen, löschen Sie auch die beiden vorgegebenen CSS-Verlinkungen:

```
<!-- iPad/iPhone specific css below, add after your main css >
 <link rel="stylesheet" media="only screen and (max-device-width: 1024px)"
href="ipad.css" type="text/css" />
 <link rel="stylesheet" media="only screen and (max-device-width: 480px)"
href="iphone.css" type="text/css" />
-->
```

Da wieder runde Ecken für das Startup-Bild verwendet werden sollen, können in der PhoneGap-Indexdatei nicht nur CSS-Definitionen eingeladen, sondern auch direkt im Code spezifiziert werden:

```
<style>
 <!--
  #textimg img {
   border: 1px solid #222222;
   -webkit-border-radius: 5px;
   }
 -->
 </style>
```

Allein durch die Zuweisung der iWebKit-CSS-Definitionen besitzt die neue iOS-Anwendung den gestreiften Hintergrund:

Bild 3.31: iOS-Anwendung mit gestreiftem Hintergrund.

Die `preventBehavior()`-Funktion können Sie weiter auskommentiert lassen, da der Inhalt etwas länger als die Bildschirmlänge sein wird und der User scrollen muss. Jetzt können Sie sich an den eigentlichen Inhalt der Seite wagen und den Quellcode aus dem `body`-Bereich zwischen die `body`-Tags der PhoneGap-Indexdatei kopieren.

Aus:

```
<body onload="onBodyLoad()">
   <h1>Ich bin eine App!</h1>
</body>
```

wird jetzt:

```
<body onload="onBodyLoad()">

<div id="topbar" class="transparent" >
 <div id="title">Oldtimer</div>
 <div id="bluerightbutton"><a href="kontakt.html">Kontakt</a></div>
</div>

<div class="searchbox">
 <form action="http://..." method="get">
  <fieldset>
   <input id="suche" placeholder="Oldtimer-Datenbank durchsuchen"
type="text" />
   <input id="submit" type="hidden" />
  </fieldset>
 </form>
</div>
```

```
<div id="content">

 <span class="graytitle">Spitzen-Oldtimer von VW und Porsche</span>
  <ul class="pageitem">
   <li class="textbox">
    <p>Wir f&uuml;hren und reparieren Old- und Youngtimer der Marken VW und
Porsche.</p>
   </li>
  </ul>

 <ul class="pageitem">
  <li class="menu"><a href="angebote.html"><span class="name">Aktuelle
Angebote</span><span class="arrow"></span></a></li>
  <li class="menu"><a href="volkswagen.html"><span class="name">Unsere
VWs</span><span class="arrow"></span></a></li>
  <li class="menu"><a href="porsche.html"><span class="name">Unsere
Porsche</span><span class="arrow"></span></a></li>
  <li class="menu"><a href="kontakt.html"><span
class="name">Reparaturen</span><span class="arrow"></span></a></li>
  <li class="menu">
   <a href="preisrechner.html">
    <img alt="Description" src="thumbs/calculator.png" />
   <span class="name">Preisrechner</span>
   <span class="arrow"></span>
   </a>
  </li>
 </ul>

 <span class="graytitle">Wir importieren aus Kalifornien!</span>

 <ul class="pageitem">
  <li class="textbox">
   <div id="textimg"><p align="center"><img src="beispiel_2-0-car.png"
alt="Karmann Ghia Auto"></p></div>
   <p>Neu im Angebot: Volkswagen Karmann Ghia</p>
  </li>
</ul>

</div>

<div id="footer">
   <p><b>Oldtimer Spiering</b><br>Beispielgasse 1, 12345 Bad
Beispielstadt</p>
</div>

</body>
```

Falls Sie den iPhone-Simulator noch im Hintergrund laufen haben, beenden Sie ihn jetzt, da Sie sonst mit großer Wahrscheinlichkeit eine Fehlermeldung von Xcode erhalten, sollten Sie versuchen, Ihre Veränderungen noch einmal mit *Build and Run* zu

kompilieren und zu überprüfen. Sobald Sie aber diesen Prozess ausführen, startet der iPhone-Simulator wieder und zeigt Ihnen nachfolgende Bildschirme.

Obwohl die Datei *splash320x240.png* für den Startup-Bildschirm mit in das *www*-Verzeichnis kopiert wurde, wird die Zeile

```
<link rel="apple-touch-startup-image" href="splash320x240.png" />
```

im linken Bild nicht mit in den Code übernommen, da sie auch nicht funktioniert hätte. Das Startup-Bild muss an einer anderen Stelle geändert werden. Bis dahin startet die Autohaus-Anwendung immer noch mit dem PhoneGap-Bildschirm.

Im rechten Bild sehen Sie die Webseite nahezu eins zu eins als native iPhone-Anwendung, ohne auch nur eine einzige Zeile Objective-C programmiert zu haben.

Bild 3.32: Unsere vormalige Web-App als richtige iOS-Anwendung.

Startup-Bildschirm ändern

Jetzt soll die Anwendung aber nicht mehr mit dem PhoneGap-Bildschirm starten, sondern einen eigenen Bildschirm verwenden.

1. Dazu müssen Sie einfach die Datei *splash320x460.png* in das Hauptverzeichnis des PhoneGap-Projekts verschieben, die dort befindliche *Default.png* löschen und unsere Datei in *Default.png* umbenennen.
2. Klicken Sie anschließend in Xcode auf *Build and Run*. Wie immer startet kurze Zeit später der Simulator mit unserer Anwendung. Aber: Unter Umständen sehen Sie noch das alte Bild.
3. Um den Build-Prozess schneller ausführen zu können, speichert Xcode bestimmte Dateien und vorkompilierte Header. Das hat zur Folge, dass die aktuell gebaute Anwendung nicht alle neuen Dateien oder Verweise enthält. Um das zu verhindern, empfiehlt es sich vor dem Bauen eines neuen Builds, in das Xcode-Menü *Build* zu gehen und dort *Clean* auszuwählen.

In der darauffolgenden Dialogbox lassen Sie die Häkchen an den Checkboxen gesetzt und klicken einfach auf *Clean*. Führen Sie *Build and Run* nochmals aus.

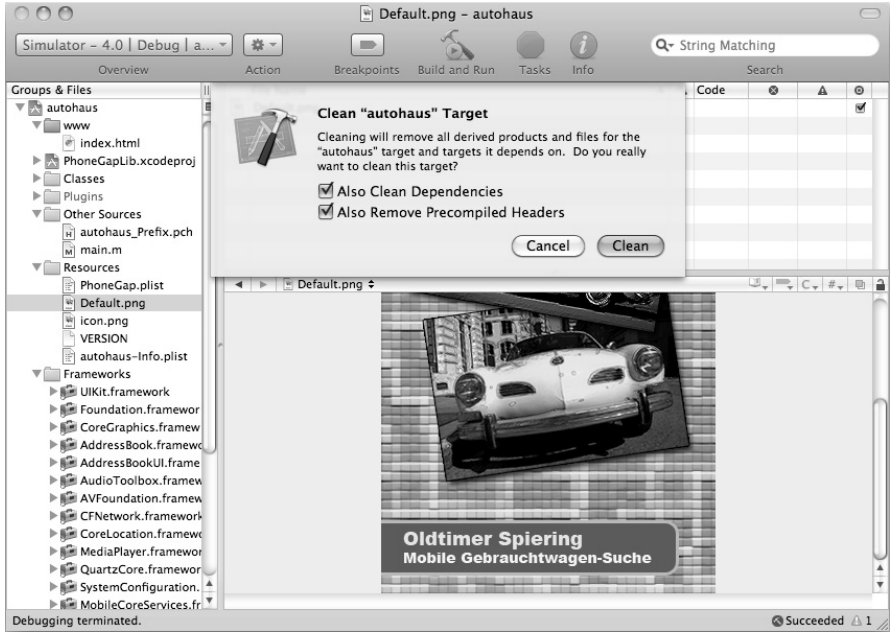


Bild 3.33: Wenn Sie alles richtig gemacht haben, sehen Sie jetzt das neue Startup-Bild.

4. Leider werden Sie feststellen, dass das Startup-Bild beim Ausblenden etwas wackelt. Das kommt daher, dass in diesem Beispiel die Kopfleiste mit berücksichtigt werden musste. Bei einer nativen iPhone-Anwendung brauchen Sie diese 20 Pixel nicht mit zu berücksichtigen und müssen stattdessen eine Startup-Bilddatei mit einer Dimension von 320 x 480 Bildpunkten im PNG-Format verwenden. Gönnen Sie Ihrer Bilddatei noch 20 Pixel mehr, tauschen Sie sie aus, und starten Sie den Build-Prozess noch einmal. Jetzt sollte alles wie gewünscht aussehen.

Programm-Icon ändern

Die gleiche Prozedur nehmen Sie vor, um das Standard-Icon von PhoneGap mit unserem Programm-Icon auszutauschen.

1. Dazu verschieben Sie wieder das Beispiel-Icon in der Datei *icon58x58.png* in das Hauptverzeichnis des Projekts, löschen das dort befindliche Piktogramm in der Datei *icon.png* und geben der Datei anschließend den Namen *icon.png*.

2. Zurück in Xcode, säubern Sie alles erst wieder mit *Clean* bzw. *Clean All Targets* und erstellen eine neue Version. Das Ergebnis sieht wie folgt aus:

Auf einem iPhone 3G oder 3GS (links) sieht das neue Icon mit der Auflösung von 57 x 57 Bildpunkten richtig gut aus. Auf einem iPhone 4 mit seinem hochauflösenden Display (rechts) wird das 57-x-57-Icon zum Problem. Im Gegensatz zu den Apple-Logos sieht es blass und unscharf aus.

3. Sie können das selbst feststellen, da der iPhone Simulator auch den Bildschirm eines iPhone 4 simulieren kann. Bei laufendem Simulator wählen Sie einfach im Menüpunkt *Hardware* unter *Device* den Eintrag *iPhone 4* aus.

Icon und Startup-Bildschirm für Retina-Bildschirm anpassen

Glücklicherweise können iPhone und iPod touch Systemgrafiken, also Icons und Startup-Bildschirme, auf kleinere Auflösungen herunterrechnen, sodass Sie mit den vollen iPhone 4-Retina-Elementgrößen arbeiten können. Bislang wurden die für das iPhone 4 zu kleinen Grafiken einfach hochgerechnet, was funktioniert, aber auf den neuen Geräten nicht gut aussieht.

Alternativ können Sie statt des 57-x-57-Icons eine Icon-Grafik mit den Ausmaßen von 114 x 114 erstellen.

Bild 3.34: Das neue Icon mit 114 x 114 Bildpunkten.

1. Die kleinere Icon-Datei im Projekthauptverzeichnis wird einfach mit der neuen Datei mit dem gleichen Namen überschrieben. Sie erreichen dann in den unterschiedlichen Auflösungen folgendes Ergebnis:

Auf einem Nicht-iPhone 4 (links) sieht das Icon auch wieder sehr gut aus. Der große Unterschied wird auf einem iPhone 4-Display sichtbar (rechts). Das Icon ist jetzt genauso scharf wie die Apple-Systemlogos.

2. Wenn Sie das Icon auf dem iPhone mit der Originaldatei vergleichen, sehen Sie, dass iOS einen halbkreisförmigen Glanz in der oberen Hälfte des Icons hinzufügt. In unserem Beispiel stört dieser Glanz aber eher, und wir können ihn mit folgender Einstellung in Xcode entfernen.

   Markieren Sie in der linken *Groups & Files*-Spalte in Xcode die Datei *autohaus-Info.plist* im Ordner *Resources*. In der Hauptansicht sehen Sie jetzt eine Tabelle mit allerlei Einstellungen. Am Ende dieser Liste klicken Sie in den letzten Eintrag und anschließend auf das neben dem letzten Eintrag erscheinende Plussymbol. Das fügt dieser Einstellungsliste eine Zeile hinzu.

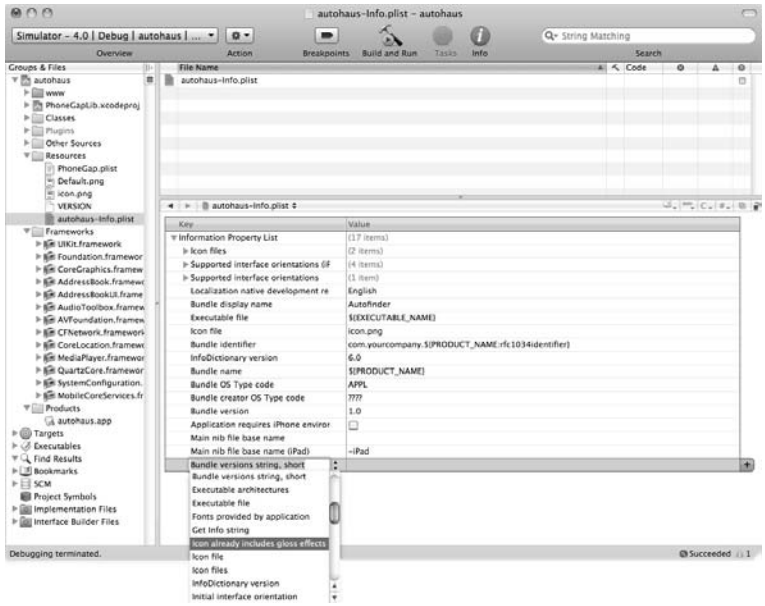


Bild 3.35: Glanzeffekte von den Icons entfernen.

3. Aus der Auswahlliste wählen Sie *Icon already includes gloss effects* aus und setzen ein Häkchen in die dann erscheinende Checkbox. Ein kurzes *Clean* mit anschließendem *Build and Run* zeigt Ihnen Ihr Icon mit den für iOS abgerundeten Ecken, jedoch ohne jeglichen zusätzlichen Glanz.

Das iPad-Icon

PhoneGap erstellt in jedem Projektverzeichnis auch eine Icon-Datei mit der Größe 72 x 72. Dieses Icon ist für iPad-Anwendungen reserviert, da für dieses Gerät die Standard-Icon-Größe 72 x 72 beträgt. Standardmäßig ist Ihr PhoneGap-Projekt für das iPad deaktiviert, wenn Sie die bisherigen Schritte mitverfolgt haben.

Um Ihre Seite auf einem virtuellen iPad mit dem anderen Logo zu sehen, müssen Sie nur das Häkchen aus der *Application requires iPhone environment*-Checkbox in der *Info.plist*-Datei entfernen.

Für den Startup-Bildschirm gilt das Gleiche wie für das Icon: Anstatt die Bildschirmdimensionen von 340 x 480 Bildpunkten zu nehmen, können Sie jetzt eine Bilddatei mit den Ausmaßen 640 x 960 Pixel verwenden.

3.3.2 Einstellungen in der Info.plist-Datei

Die gerade eben entdeckte *Info.plist*-Datei erlaubt projektweite Einstellungen und ermöglicht uns, die Anwendung noch weiter zu verbessern.

Bundle display name

Hier können Sie einen alternativen Namen für Ihre iOS-Anwendung vergeben. Standardmäßig wird der Anwendungsname genommen, den Sie beim Erstellen des Projekts angegeben haben. In unserem Fall sieht das kleingeschriebene *autohaus* nicht besonders gut aus, also löschen Sie den derzeitigen Inhalt des Felds und geben *Autofinder* ein.

Application requires iPhone environment

Ist dieses Feld markiert, wird Ihre Anwendung nicht auf einem iPad oder einem iPod touch laufen. Wählen Sie dieses Feld aus, legen Sie explizit fest, dass nur ein iPhone die von Ihnen generierte Anwendung ausführen kann. Obwohl das standardmäßig in PhoneGap markiert ist, ist es in den meisten Fällen wenig sinnvoll. Nur Anwendungen, die unbedingt eine Telefoniefunktion benötigen, sollten dieses Feld markiert haben.

Status bar style

Eine der `meta`-Zeilen haben Sie aus dem ursprünglichen Quellcode des Oldtimer-Beispiels nicht mit in die neue PhoneGap-*index.html* übernommen:

```
<meta name="apple-mobile-web-app-status-bar-style" content="black"/>
```

Aus gutem Grund: Es hätte nämlich nicht funktioniert und keine Auswirkungen auf die Statusleiste gehabt. Anstatt ein `meta`-Tag im HTML-Code zu verwenden, kann diese Einstellung in der *Info.plist*-Datei vorgenommen werden. Die Variable `Status bar style` bietet folgende Auswahlmöglichkeiten:

Menüauswahl	*Beschreibung*
Grey style (default)	Die normale iOS-Statusleiste in Grau. Sie wird auch verwendet, wenn Sie die *Status bar style*-Einstellung nicht angeben.
Opaque black style	Schwarze, undurchsichtige Statusleiste.
Transparent black style (alpha of 0.5)	Schwarze Statusleiste. 50 % transparent und somit durchsichtig.

Status bar is initially hidden

Insbesondere bei Spielen wollen Sie möglicherweise komplett auf die Statusleiste verzichten. Falls Sie ein Häkchen bei *Status bar is initially hidden* setzen, wird keine Statusleiste angezeigt. Diese Einstellung überschreibt auch die Einstellung *Status bar style*.

Bild 3.36: Links die Standardstatusleiste im klassischen iPhone-Grau. Darf's etwas dunkler sein? In der Mitte die Statusleiste in Schwarz, beliebt und gut aussehend bei iPhone-Anwendungen mit dunklen bzw. schwarzen Hintergrundfarben.

Bild 3.37: Üblich bei Spielen: Mit der *Info.plist*-Einstellung *Status bar is initially hidden* kann die Statusleiste auch komplett ausgeblendet werden.

Supported interface orientations

In den *Info.plist*-Einstellungen können Sie ebenfalls festlegen, ob Ihre Anwendung im Hoch- oder im Querformat laufen kann. Dabei wird in Xcode und somit auch in unseren PhoneGap-Projekten sogar zwischen iPhone/iPod touch und dem iPad unterschieden.

Supported interface orientations (iPad)	(4 items)
Item 0	Portrait (bottom home button)
Item 1	Landscape (left home button)
Item 2	Portrait (top home button)
Item 3	Landscape (right home button)
Supported interface orientations	(2 items)
Item 0	Portrait (top home button)
Item 1	Portrait (bottom home button)

Bild 3.38: Die *Supported interface orientations* für das iPad.

Mit *Supported interface orientations (iPad)* legen Sie fest, in welcher Geräteausrichtung Ihre App laufen bzw. ob sie diese Ausrichtung unterstützen soll. In obigem Beispiel kann das iPad in jede der vier Richtungen gedreht werden, und die Anwendung wird sich immer dem Bildschirm anpassen.

Anders bei der nächsten Einstellung: Unter *Supported interface orientations* werden die Einstellungen für das iPhone bzw. den iPod touch festgelegt. Würden Sie die Einstellungen so belassen, könnte die App nur im Hochformat dargestellt werden – egal ob der Home-Button oben oder unten wäre.

Bei Spielen sehen Sie beispielsweise sehr oft, dass sie nur im Querformat laufen. Das können Sie erzwingen, indem Sie die Werte von *Portrait* (Hochformat) auf *Landscape* (Querformat) umstellen. Eine so eingestellte Anwendung oder ein so eingestelltes Spiel würde sich wie folgt verhalten:

Bild 3.39: Die Anwendung wird vom Start weg im Querformat ausgeführt. Haben Sie in den *Supported interface orientations* der App keine *Portrait*-Unterstützung, bleibt sie auch im Hochformat in der eingestellten *Landscape*-Position.

Initial interface orientation

Wenn Ihre App das Hoch- und das Querformat eines iPhones oder iPads unterstützt, können Sie diese Einstellung dazu verwenden, dem System mitzuteilen, in welchem Modus die Anwendung zuerst starten soll. Falls Ihre App nur im Hochformat läuft und Sie das in den *Supported interface orientations*-Einstellungen bereits so festgelegt haben, ist die Angabe dieser Einstellung überflüssig.

jQTouch-Besonderheiten

Alles, was Sie bis jetzt gelernt haben, können Sie auch auf unser jQTouch-Beispiel anwenden. Auch wenn wir im Laufe des Buchs iWebKit aufgrund seines einfachen Aufbaus und seiner Kompatibilität zwischen den verschiedenen Plattformen bewusst für unsere Beispiele gewählt haben, ist es durchaus sehr sinnvoll, jQTouch für native iPhone Apps mit PhoneGap zu verwenden.

Im Gegensatz zu iWebKit 5 enthält jQTouch bereits eine ganze Menge an Animationen, die von PhoneGap auch voll unterstützt werden. Unseren Beispielcode können Sie fast unverändert lassen, allerdings sollten Sie – damit unser Beispiel auch komplett offline funktioniert – die jQuery-JavaScript-Bibliothek nicht dynamisch von Google laden, sondern in die Anwendung integrieren.

Dazu müssen Sie den `head`-Bereich des Beispiels wie folgt ändern:

```
<head>
   <meta name="layout" content="webkit" />
   <title>Oldtimer Spiering</title>

   <!-- jQuery jetzt lokal laden -->
   <script type="text/javascript" src="jQuery.js"></script>

   <!-- jQTouch auch lokal einbinden -->
   <script type="text/javascript" src="jqtouch/jqtouch.js"></script>
   <link type="text/css" rel="stylesheet" media="screen"
href="jqtouch/jqtouch.css">
```

```
   <link type="text/css" rel="stylesheet" media="screen"
href="themes/jqt/theme.css">

    <!-- jQTouch initialisieren / auch in der App notwendig -->
   <script type="text/javascript">
      var jQT = $.jQTouch({ });
   </script>
</head>
```

Bevor das funktioniert, muss die *jQuery.js* von der Seite *http://jquery.com* heruntergeladen und im Verzeichnis der *index.html*-Datei im *www*-Verzeichnis des PhoneGap-Projekts gespeichert werden. Wenn Sie jQuery nicht lokal speichern wollen, muss es, wie bei der vorgestellten Webseite, direkt von Google aus dem Internet geladen werden. Das hat den Nachteil, dass Ihre Anwendung zum Start eine Internetverbindung immer benötigt und somit iPod touch-Benutzer und Besitzer eines WLAN-iPads Ihre App nicht ausführen können, wenn sie nicht gerade in der Nähe eines WLAN-Hotspots sind. Um dieses jQTouch-Beispiel als iPhone-Anwendung laufen zu lassen, müssen Sie wieder:

- Ein neues Xcode-Projekt mit PhoneGap-Vorlage erstellen und somit einen neuen Ordner anlegen.
- Alle jQTouch-Komponenten, wie das *jqtouch*-Verzeichnis, das *themes*-Verzeichnis und optional das *extensions*-Verzeichnis in das *www*-Verzeichnis kopieren.
- Den HTML-Code des jQTouch-Beispiels mit der Datei *index.html* im *www*-Verzeichnis wie gelernt verbinden.
- In Xcode in den *Project Settings* ein Build Target auswählen.
- Schließlich mit Xcode die Anwendung kompilieren und im iPhone-Simulator zum Laufen bringen.

Wenn Sie das alles befolgt haben, sehen Sie im iPhone-Simulator Folgendes:

Wenn Sie den Bildschirm genau betrachten, gibt es an seinem unteren Ende einen leeren, schwarzen Bereich. Dieser entsteht dadurch, dass jQTouch immer noch denkt, dass es in Safari laufe, den Vollbildmodus nicht erkennt und den Bereich dafür reserviert, die Webadresse einzugeben, wie in Safari üblich.

Um das zu ändern, müssen Sie in die Datei *jqtouch.css* im Verzeichnis *jqtouch* gehen, die beim Laden der *index.html* mit eingebunden wird. Dort finden Sie relativ weit vorne folgende Zeilen:

```
body > * {
    -webkit-backface-visibility: hidden;
    -webkit-box-sizing: border-box;
    display: none;
    position: absolute;
```

```
    left: 0;
    width: 100%;
    -webkit-transform: translate3d(0,0,0) rotate(0) scale(1);
    min-height: 420px !important;
}
body.fullscreen > * {
    min-height: 460px !important;
}
body.fullscreen.black-translucent > * {
    min-height: 480px !important;
}
```

Diese CSS-Definitionen bestimmen unter anderem die Mindesthöhe für den `body`-Bereich und sind leider innerhalb von PhoneGap gleich mehrfach falsch.

Wenn Sie in Ihrer jQTouch-Anwendung eine graue oder schwarze Statusleiste verwenden – das definieren Sie innerhalb der *Info.plist*-Datei –, sollte das oben stehende CSS durch die folgenden Definitionen ausgetauscht werden:

```
body > * {
    -webkit-backface-visibility: hidden;
    -webkit-box-sizing: border-box;
    display: none;
    position: absolute;
    left: 0;
    width: 100%;
    -webkit-transform: translate3d(0,0,0) rotate(0) scale(1);
    min-height: 460px !important;
}
```

Richtig, die Definitionen für *body.fullscreen* und *body.fullscreen.back-translucent* können Sie löschen, da sie von PhoneGap und Xcode ohnehin nicht erkannt werden.

Verwenden Sie in Ihrer jQTouch-Anwendung keine Statusleiste – *Status bar is initially hidden*-Einstellung in der Datei *Info.plist* – oder eine transparente, schwarze Statusleiste, ersetzen Sie die oben stehenden CSS-Definitionen durch folgenden Code:

```
body > * {
    -webkit-backface-visibility: hidden;
    -webkit-box-sizing: border-box;
    display: none;
    position: absolute;
    left: 0;
    width: 100%;
    -webkit-transform: translate3d(0,0,0) rotate(0) scale(1);
    min-height: 480px !important;
}
```

Speichern Sie diese Änderungen ab, säubern Sie Ihr Xcode-Projekt mit *Clean,* und rufen Sie anschließend *Build and Run* auf. Der Hintergrund sollte bis zum unteren Rand reichen und wie nebenstehend aussehen.

Wenn Sie jetzt die Animationen testen, die mit jQTouch möglich sind, sieht man bereits überhaupt nicht mehr, dass hier nicht mit Objective-C, sondern einfach nur mit ein bisschen HTML, CSS und JavaScript programmiert wurde.

Code im Xcode Editor bearbeiten

Sie haben bislang immer einen externen Texteditor verwendet, wenn Sie HTML-, CSS- oder JavaScript-Code bearbeitet oder neu erstellt haben. Xcode enthält bereits einen eingebauten Editor, der gerade bei den folgenden Beispielen das Arbeiten erleichtert. Sie müssen nicht immer zwischen Ihrem externen Texteditor und Xcode hin- und herschalten, sondern können jeglichen Code innerhalb eines Programms schreiben, verändern und schließlich auch ausführen.

Um eine Datei in Xcode zu bearbeiten, wählen Sie sie einfach links unter *Groups & Files* aus. Der Code wird anschließend automatisch im Hauptfenster von Xcode angezeigt:

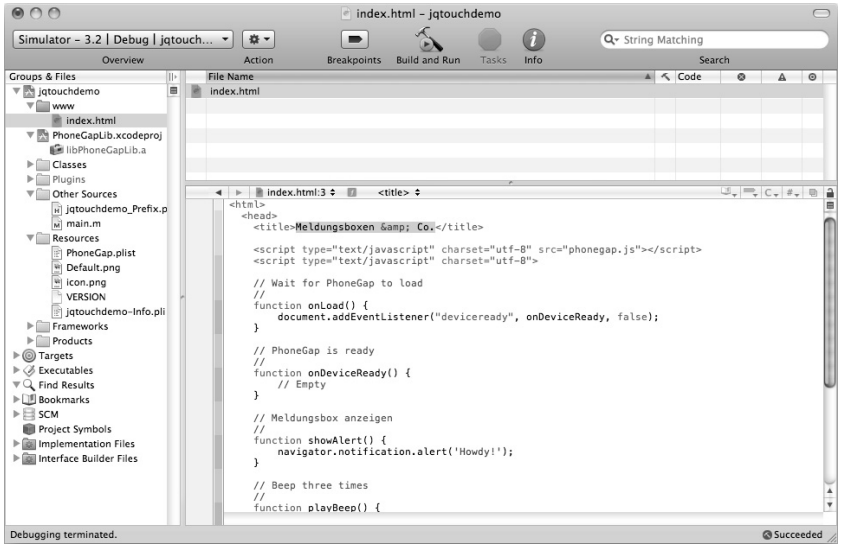


Bild 3.40: Die verschiedenen Graustufen auf der linken Seite des Codefensters markieren einzelne Codebereiche, also in sich geschlossene Funktionen oder Bereiche, in denen ein bestimmtes Tag gilt. Sobald Sie mit der Maus in dieses Feld gehen, können Sie einzelne Codebereiche ein- und ausblenden und so nur die Codeteile sehen, die für Sie gerade relevant sind.

3.3.3 Mit PhoneGap und Xcode direkt auf einem iOS-Gerät testen

Sie haben bisher gesehen, wie Sie aus einer Webanwendung eine eigenständige iOS-App entwickeln können. Dabei werden alle HTML-Funktionen, die Sie bislang gelernt haben, unterstützt, da PhoneGap, wie am Anfang dieses Kapitels beschrieben, zwar nativen Code erstellt, die Inhalte aber in einem WebView-Fenster anzeigt, das die gleichen Eigenschaften und Fähigkeiten wie der mobile Safari-Browser besitzt. PhoneGap kann aber noch mehr und bietet Ihnen eine Brücke zwischen diversen Gerätefunktionen wie Vibration, Kamerazugriff, Bewegungssensoren und noch einiges mehr. Diese Funktionen können Sie in unseren Webcode über PhoneGaps JavaScript-APIs mit einbinden, allerdings im iOS Simulator nur begrenzt testen. Aus diesem Grund sehen wir uns jetzt kurz an, wie man die mit Xcode erstellten iOS-Anwendungen auf ein Gerät bekommt.

Eines gleich vorweg: Um iOS-Anwendungen auf dem iPhone, dem iPad oder iPod touch zu testen, ist die kostenpflichtige Teilnahme am iPhone Developer-Programm erforderlich. Wir sind bei der Beschreibung des iOS SDK bereits darauf eingegangen – siehe Kapitel 3.2.1, »Entwicklungsumgebung für iOS-Entwicklung einrichten«.

Haben Sie die Mitgliedschaft für das iPhone Developer-Programm beantragt und sind Sie nach Zahlung der Jahresgebühr zugelassen, erhalten Sie von Apple eine E-Mail, mit der Sie das Developer-Programm aktivieren können. Ab sofort stehen Ihnen dann im iPhone Dev Center – *https://developer.apple.com/iphone* – erweiterte Funktionen zur Verfügung.

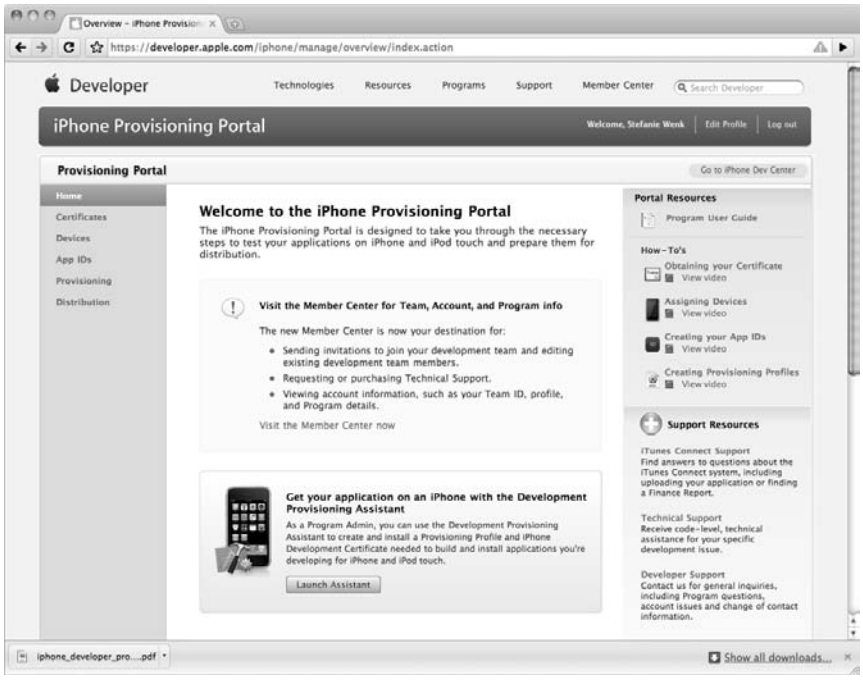


Bild 3.41: Eine Provisionsdatei (Provisioning File) stellt sicher, dass Sie als Entwickler mit Ihrer App und Ihrer Entwicklungsumgebung (Xcode) identifizierbar sind. Die Provisionsdatei muss auf dem von Ihnen genutzten mobilen Gerät und gleichzeitig auch auf Ihrem Mac installiert werden.

Eine dieser neuen Optionen ist ein Link zum *iPhone Provisioning Portal – http://developer.apple.com/iphone/manage/overview*. Ein Klick darauf bringt Sie zur gleichnamigen Seite und lässt Sie Ihre physischen iOS-Geräte für Testinstallationen konfigurieren. Ähnlich strikt wie bei der Anmeldung zum Entwicklerprogramm ist Apple auch, wenn es darum geht, die Anwendung auf einem Gerät zu testen bzw. letztendlich in den App Store zu stellen. Das alles klingt unheimlich kompliziert, ist es aber nicht. Im iPhone Provisioning-Portal finden Sie am unteren Ende der Seite den *Development Provisioning Assistent*. Sobald Sie Ihr iPhone, iPad oder Ihren iPod touch inklusive USB-Kabel zur Hand haben, kann es mit einem Klick auf *Launch Assistent* losgehen. Im Assistenten selbst können Sie sich mittels *Continue* und *Go Back* Schritt für Schritt weiterbewegen.

1. Zuerst werden Sie gebeten, eine Beschreibung zu einer App ID anzugeben. Eine App ID ist eine eindeutige Bezeichnung für eine Anwendung im Apple-App-Universum. Jede Anwendung im App Store hat ihre eigene App ID. Wenn eine App ID einmal angelegt ist, kann sie im Anschluss nicht mehr bearbeitet werden. Das gilt im Übrigen auch für die Beschreibung der App ID: Zwar muss sie nicht eindeutig sein, bearbeiten oder gar löschen können Sie sie dennoch nicht. Seien Sie daher relativ spezifisch in der Auswahl eines geeigneten Namens. Die eigentliche App ID erhalten Sie von Apple zugeteilt.

Bild 3.42: Für unsere Oldtimer-Anwendung schlagen wir den Namen *Oldtimer Markt* als App ID-Beschreibung vor.

2. Im nächsten Schritt geht es um Ihr mobiles Gerät: Sie müssen zuerst eine kurze Beschreibung zu diesem Gerät eingeben. Wenn Sie nur ein iPhone haben und mit diesem entwickeln möchten, dürfte die Wahl nicht so schwer fallen. Falls Sie aber planen, auf diversen Geräten zu testen, ist es hier sinnvoll, über griffige Namen nachzudenken, anhand deren sich die jeweiligen Geräte identifizieren lassen.

Bild 3.43: Eingeben der Gerätebeschreibung

3. Jetzt geben Sie den Identifier Ihres Geräts ein. Jedes Apple-Gerät hat eine eindeutige ID, die über Xcode abgerufen werden kann. Schließen Sie Ihr Apple-Gerät per USB-Kabel an den Mac an, und gehen Sie bei laufendem Xcode in das Menü *Window* und dort auf den Menüpunkt *Organizer*. Den 40-stelligen Identifier-Code können Sie mit der Maus markieren, kopieren und in den Development Provisioning-Assistanten kopieren.

4. Jetzt geht es um Ihren Rechner. Unter *Programme/Dienstprogramme* finden Sie die Anwendung *Schlüsselbundverwaltung*, die viele auf Ihrem Computer verwendete Passwörter speichert – sofern Sie das autorisiert haben. Nach dem Start der Anwendung wählen Sie aus der *Schlüsselbundverwaltung* das Untermenü *Zertifikatsassistent* und darunter den Punkt *Zertifikat einer Zertifizierungsinstanz anfordern*.

5. Danach geben Sie Ihre E-Mail-Adresse sowie Ihren Namen an und wählen aus, dass das generierte Zertifikat auf Ihrer Festplatte gespeichert wird. Mit einem Klick auf *Weiter* bekommen Sie die Gelegenheit, die Datei *certSigningRequest* auf Ihrem Desktop zu speichern. Wieder zurück im *Development Provisioning Assistant*, wählen Sie im nächsten Schritt die soeben gespeicherte Datei aus.

6. Bevor Sie Ihr *Provisioning Profile* erhalten, müssen Sie ihm noch einen passenden Namen geben. Wir sind in unserem Beispiel nicht übermäßig kreativ und wählen einfach *Oldtimer Markt Profil*. Den Zertifikatsnamen können Sie nicht ändern, da er an den Namen des Developer-Programm-Kontos gebunden ist. Im nächsten Schritt wird die Profildatei erstellt, und im darauffolgenden Schritt kann diese auf Ihren Mac heruntergeladen werden.

7. Rufen Sie jetzt wieder Xcode in den Vordergrund, und ziehen Sie die soeben generierte und heruntergeladene Datei in das *Organizer*-Fenster – in dem Sie sich bereits den Geräte-Identifier geholt haben – oder lassen Sie das Dateisymbol wie im Assistenten gezeigt einfach auf das Xcode-Symbol fallen. In der Geräteansicht im Organizer sehen Sie jetzt den Namen des Provisioning Profile unter *Provisioning* aufgelistet.

Bild 3.44: Das Provisioning-Profil im Organizer

8. Zurück im jetzt schon bekannten *Development Provisioning Assistant*, können Sie wieder eine Datei herunterladen. Diesmal ist es das Entwicklungszertifikat, das wir auf dem iPhone, iPod touch oder iPad installieren. Die zum Download angebotene CER-Datei kann nach dem Herunterladen vom Assistenten mit der Maus angeklickt werden, um die Keychain Access-Anwendung aufzurufen und mit *Add* das Zertifikat zu speichern.

   Wenn Sie in das Hauptfenster der Keychain Access-Anwendung gehen und dort auf die Unterkategorie *Keys* klicken, sollten der private und der öffentliche Schlüssel sowie innerhalb des privaten Schlüssels Ihr iPhone Development-Zertifikat sichtbar sein.

9. Nach diesen vielen Schritten haben Sie es fast geschafft: Sie können jetzt den Xcode Organizer schließen und ein PhoneGap-Projekt öffnen. In dem Drop-down-Menü in der linken oberen Ecke wählen Sie jetzt statt *Simulator* den Punkt *Device* aus. Haben Sie bislang nur ein Gerät provisioniert, wird es automatisch ausgewählt. Falls Sie mehrere Geräte besitzen und angeschlossen haben, können Sie im gleichen Menü noch das gewünschte Gerät auswählen. Der bereits bekannte *Build and Run*-Button ruft ab sofort nicht mehr den Simulator auf dem Bildschirm auf, sondern installiert Ihre neue iPhone-PhoneGap-App direkt auf dem angeschlossenen Gerät.

3.3.4 Native iOS-Systemfunktionen in PhoneGap einbinden

Bislang müssen Sie noch mit den Beschränkungen einer Webseite leben, obwohl Sie bereits eine richtige iOS-App erstellt haben: Die Kamera ist für uns weiterhin nicht erreichbar, Sie können keinen Vibrationsalarm auslösen, nicht auf das Adressbuch zugreifen und auch den Bewegungssensor nicht nutzen. Glücklicherweise bietet PhoneGap eine ganze Reihe von JavaScript-APIs, mit denen Sie den Webcode ergänzen und auf diese nativen Funktionen zugreifen können.

Nutzen Sie native Funktionen, wenn Sie in den App Store möchten
Uns ist keine feste Regel bekannt, aber wir haben von Fällen gehört, in denen Apps, die beispielsweise mit PhoneGap erstellt wurden, nicht in den App Store aufgenommen wurden, weil sie keine nativen Systemfunktionen hatten und sich nicht von einer Webanwendung unterschieden haben. Nun kann man Apple keinen Vorwurf daraus machen, dass der App Store für Anwendungen und nicht für Webseiten gedacht ist. Dennoch sollten Sie überlegen, wie gut Sie die im folgenden Abschnitt vorgestellten nativen Funktionen innerhalb Ihres Projekts sinnvoll verwenden können, wenn Sie die Absicht haben, Ihr PhoneGap-App im App Store zu verkaufen.

Navigationsleisten

Eines der markantesten Elemente einer iOS-Anwendung ist die Navigationsleiste am unteren Bildschirmrand. Sie kennen sie von einigen mitgelieferten Apps, beispielsweise von der iPod- bzw. Musikanwendung, in der Sie zwischen *Listen*, *Interpreten*, *Titel*, *Alben* und *Mehr* umschalten können.

Die untere Navigationsleiste, im Englischen »Tab Bar« genannt, wird in iOS-eigenen Anwendungen wie hier innerhalb der Musikanwendung verwendet. Das gleiche Konzept findet sich aber auch in vielen Apps von Drittanbietern wieder und erlaubt das schnelle und immer präsente Umschalten zwischen diversen Hauptbildschirmen der App.

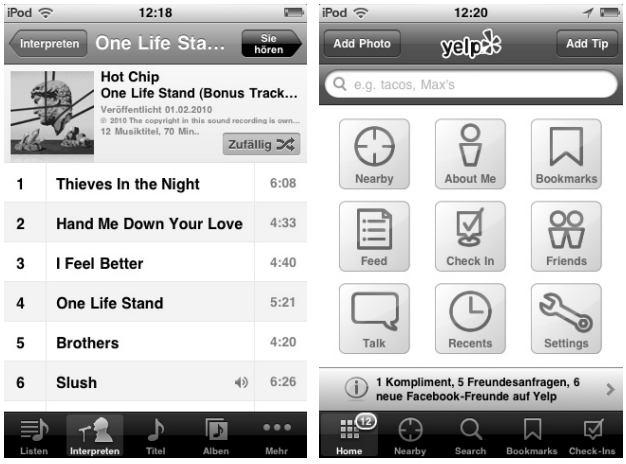
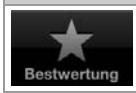
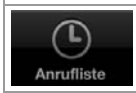
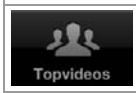
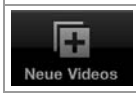
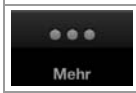
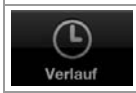
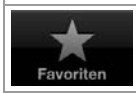
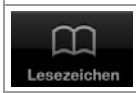


Bild 3.45: Wenn neue Inhalte in einer der Ansichten zur Verfügung stehen, wird das oft mit einem kleinen Indikator angezeigt. Im rechten Beispiel gibt es zwölf Neuigkeiten auf dem Home-Bildschirm der Anwendung.

Theoretisch könnte eine derartige Leiste auch recht umständlich innerhalb einer Webanwendung mit CSS und JavaScript nachgebaut werden, jedoch macht PhoneGap es einfacher und bietet Ihnen entsprechende JavaScript-Funktionen, um diese nativen Bedienelemente in Ihre Anwendungen einzubauen.

Zuerst sollte man wissen, dass das iOS uns bereits verschiedene Symbole und Namen zur Verfügung stellt. Falls eines dieser Icons gut zu den von Ihnen geplanten Funktionen und Bildschirmansichten innerhalb Ihrer App passt, empfehlen wir, so häufig wie möglich die System-Icons zu verwenden. Benutzer wissen in der Regel schon, was sie bedeuten, und Ihre App wird vom Anwender schneller verstanden.

Symbol	*Deutsche Bezeichnung*	*Englische Bezeichnung*	*PhoneGap JavaScript-Parameter*
	Bestwertung	Top Rated	`tabButton:TopRated`
	Suche	Search	`tabButton:Search`
	Anrufliste	Recents	`tabButton:Recents`
	Topvideos	Most Viewed	`tabButton:MostViewed`
	Neue Videos	Most Recent	`tabButton:MostRecent`
	Mehr	More	`tabButton:More`
	Verlauf	History	`tabButton:History`
	Highlights	Featured	`tabButton:Featured`
	Favoriten	Favorites	`tabButton:Favorites`
	Downloads	Downloads	`tabButton:Downloads`
	Kontakte	Contacts	`tabButton:Contacts`
	Lesezeichen	Bookmarks	`tabButton:Bookmarks`

Die englischen Begriffe sind viel weiter gefasst als die deutschen Übersetzungen. Leider können Sie, wenn Sie in PhoneGap auf die iOS-Symbole zurückgreifen, die Bezeichnungen nicht ändern. Daher müssen Sie aufpassen, wenn Sie beispielsweise den vordefinierten Button *Most Recent* wählen, dass dieser sich im Deutschen nur auf neue Videos bezieht und daran gebunden ist.

Sie müssen aber nicht auf die vorgefertigten Icons zurückgreifen, sondern können auch eigene Piktogramme erstellen. Diese müssen folgendes Format und folgende Eigenschaften besitzen:

- 30 x 30 Bildpunkte für Bildschirme in niedriger Auflösung.
- 60 x 60 Bildpunkte für den iPhone 4-Retina-Bildschirm.
- PNG-Format mit transparentem Hintergrund.
- Reinweißer Vordergrund ohne Schattenwurf etc.

Wir empfehlen Ihnen, generell das 60-x-60-Icon zu verwenden, da dieses Icon von älteren iOS-Geräten auf die benötigte Größe heruntergerechnet wird. Die Bezeichnung des Icons brauchen Sie nicht mit in die Grafik zu übernehmen. Im Gegensatz zu den vorgefertigten System-Icons können Sie hier die Bezeichnung frei im JavaScript-Aufruf wählen. Mit dem nächsten Beispielcode wollen wir Folgendes erreichen:

Am unteren Bildschirmrand unseres iWebKit-Beispiels soll eine native Navigationsleiste mit vier antippbaren Bereichen angezeigt werden (linkes Bild). Drei der verwendeten Icons sollen vom System kommen, das Testauto-Icon wurde individuell erstellt. Wenn man auf *Bestbewertung* tippt, soll eine Meldungsbox erscheinen, die das bestätigt, und ein kleiner Zähler soll über dem Icon anzeigen, wie oft dieses Feld bereits angetippt wurde (mittleres Bild). Tippt man auf *Testautos*, wird die gleiche Seite neu geladen, allerdings ohne wieder eine ganz neue (und in dem Fall zusätzliche) Navigationsleiste anzuzeigen (rechtes Bild).

Bild 3.46: Nach dem Antippen von »Bestbewertung« wird das dazugehörige Symbol mit einem Zähler ausgezeichnet.

Hier der Beispielcode:

```
<!--
   Autohaus-Beispiel in PhoneGap mit Navigationsleiste
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
<html>
  <head>
    <title>Navigationsleiste</title>

    <!-- iWebKit 5 laden -->
    <link rel="stylesheet" type="text/css" href="style.css">

   <!-- PhoneGap JavaScript laden -->
    <script type="text/javascript" charset="utf-8" src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

    // Warten, bis PhoneGap geladen ist
    function onLoad() {
        document.addEventListener("deviceready", onDeviceReady, false);
    }

    // Sobald PhoneGap da ist, lade folgende Funktionen:
    function onDeviceReady() {

     window.uicontrols.createTabBar();

     var toprated = 0;
     window.uicontrols.createTabBarItem("toprated", "Egal was hier steht",
"tabButton:TopRated", {
       onSelect: function() {
       navigator.notification.alert("Bestwertung angetippt");
       window.uicontrols.updateTabBarItem("toprated", { badge: ++toprated });
       }
     });

     window.uicontrols.createTabBarItem("suche", "Auch egal",
"tabButton:Search", {
       onSelect: function() {
       navigator.notification.alert("Suche angetippt");
       window.uicontrols.updateTabBarItem("recents", { badge: ++recents });
       }
     });

     window.uicontrols.createTabBarItem("test", "Testautos",
"/www/button.png", {
       onSelect: function() {
       document.location.href="index.html";
       }
     });

     window.uicontrols.createTabBarItem("anderes", "Buch",
"tabButton:Bookmarks", {
       onSelect: function() {
```

```
       navigator.notification.alert("Lesezeichen angetippt");
       }
     });

     if (window.innerHeight == 480) {
     window.uicontrols.showTabBar();
     window.uicontrols.showTabBarItems("toprated", "suche", "test",
"anderes");
     }
    }

    </script>
  </head>
  <body onload="onLoad()">

<div id="topbar" class="transparent" >
 <div id="title">Oldtimer</div>
 <div id="bluerightbutton"><a href="kontakt.html">Kontakt</a></div>
</div>

<div class="searchbox">
 <form action="http://..." method="get">
  <fieldset>
   <input id="suche" placeholder="Oldtimer-Datenbank durchsuchen"
type="text" />
   <input id="submit" type="hidden" />
  </fieldset>
 </form>
</div>

<div id="content">

 <span class="graytitle">Spitzen-Oldtimer von VW &amp;amp; Porsche</span>
  <ul class="pageitem">
   <li class="textbox">
    <p>Wir f&uuml;hren und reparieren Old- und Youngtimer der Marken VW und
Porsche.</p>
   </li>
  </ul>

 <ul class="pageitem">
  <li class="menu"><a href="angebote.html"><span class="name">Aktuelle
Angebote</span><span class="arrow"></span></a></li>
  <li class="menu"><a href="volkswagen.html"><span class="name">Unsere
VWs</span><span class="arrow"></span></a></li>
  <li class="menu"><a href="porsche.html"><span class="name">Unsere
Porsche</span><span class="arrow"></span></a></li>
  <li class="menu"><a href="kontakt.html"><span
class="name">Reparaturen</span><span class="arrow"></span></a></li>
  <li class="menu">
   <a href="preisrechner.html">
    <img alt="Description" src="thumbs/calculator.png" />
   <span class="name">Preisrechner</span>
   <span class="arrow"></span>
```

```
    </a>
   </li>
  </ul>

   </body>
</html>
```

3.3.5 Lokalisierte Button-Bezeichnungen

Wenn Sie diesen Beispielcode mit den bisherigen Xcode-Einstellungen ausführen, werden Sie mit großer Wahrscheinlichkeit die englischen Bezeichnungen *Top Rated*, *Search* und *Bookmarks* für die System-Icons erhalten. Das liegt daran, dass die App noch nicht für die Lokalisierung vorbereitet ist.

Um das zu ermöglichen, müssen Sie einfach mit der Maus wieder auf die *Info.plist*-Datei klicken, dort die Einstellung *Localized resources can be mixed* mit in die Liste aufnehmen und durch einen Klick auf die Checkbox aktivieren. Speichern Sie die *Info.plist*-Datei, säubern Sie anschließend über *Build/Clean* Ihr Projekt, und lassen Sie es neu kompilieren. Wenn Sie Deutsch oder eine andere Sprache auf Ihrem virtuellen (oder richtigen) iOS-Gerät eingestellt haben, erscheinen jetzt die lokalisierten Bezeichnungen unterhalb der System-Icons.

Analysieren wir jetzt, was im Code des vorhergehenden Beispiels gemacht wurde: Als Erstes wurde der JavaScript-Code in die Funktion `onDeviceReady()` gepackt, damit sichergestellt ist, dass die nativen JavaScript-Aufrufe erst dann ins Spiel kommen, wenn die Anwendung und PhoneGap komplett gestartet sind.

Eine Navigationsleiste wird im Wesentlichen durch zwei JavaScript-Aufrufe:

```
window.uicontrols.createTabBar();
```

und eine bestimmte Anzahl von:

```
window.uicontrols.createTabBarItem("ID", "Label", "Button", { Anweisung };
```

bestimmt. Mit `window.uicontrols.createTabBar()` wird die eigentliche Leiste initialisiert. Diese Leiste sitzt am unteren Bildschirmrand und hat weder Icons noch Bezeichnungen. Diese werden durch einzelne Angaben von `window.uicontrols.createTabBarItem()` mit den jeweiligen Parametern hinzugefügt.

Parameter	*Beschreibung*
`ID`	Eine selbst gewählte ID, die Sie später im Code verwenden, um das jeweilige Element innerhalb der Navigationsleiste anzusprechen.
`Label`	Eine frei wählbare Bezeichnung, die unterhalb des Icons angezeigt wird. Beachten Sie, dass, wenn Sie ein System-Icon verwenden, die Angaben unter `Label` ignoriert werden.

Parameter	*Beschreibung*
`Button`	Entweder der Link zu einer eigenen Buttongrafik oder die Angabe des System-Buttontyps, wie beispielsweise `tabButton:Search`. Falls Sie eine eigene Grafik verwenden, muss hier der gesamte Pfad angegeben werden. Liegt die Grafik *button.png* beispielsweise im *www*-Verzeichnis, muss die Angabe des Links wie folgt aussehen: */www/button.png*.
`Anweisung`	Über ein `onSelect()`-Event legen Sie hier fest, welche JavaScript-Anweisungen beim Antippen des jeweiligen Elements auf der Navigationsleiste auszuführen sind.

In diesem Beispiel verwenden wir zwei verschiedene Varianten der `window.uicontrols.createTabBarItem()`-Angabe:

```
var toprated = 0;
window.uicontrols.createTabBarItem("toprated", "Egal was hier steht", "tab-
Button:TopRated", {

 onSelect: function() {

  navigator.notification.alert("Bestwertung angetippt");
  window.uicontrols.updateTabBarItem("toprated", { badge: ++toprated });

   }
});
```

Zuerst definieren Sie die Variable `toprated` und setzen sie auf null. Dann rufen Sie `window.uicontrols.createTabBarItem` mit der ID `toprated` auf. Die ID ist unabhängig von der gewählten Variablen. Als Label-Text verwenden Sie irgendeinen Text, der ohnehin nicht angezeigt wird, da wir als Nächstes mit `tab-Button:TopRated` einen System-Button verwenden.

Als Anweisung verwenden wir das Event `onSelect()`. Sobald der Benutzer auf das Navigationselement tippt, wird eine Meldungsbox angezeigt (zur Funktion `navigator.notification.alert()` kommen wir im übernächsten PhoneGap-Abschnitt). Diese Meldungsbox bestätigt noch einmal, dass das richtige Feld angetippt wurde. Als Nächstes finden Sie die Zeile:

```
window.uicontrols.updateTabBarItem("toprated", { badge: ++toprated });
```

Mit dieser Angabe wird das Navigationselement mit der ID `toprated` aktualisiert. In den geschweiften Klammern stehen die JavaScript-Anweisungen, die bewirken, dass die Variable `toprated` um eins erhöht wird und der kleine Zahlenindikator dann mit dem neuen Wert angezeigt wird.

Die zweite Variante der `window.uicontrols.createTabBarItem()`-Angabe ist diese:

```
window.uicontrols.createTabBarItem("test", "Testautos", "/www/button.png", {
    onSelect: function() {
    document.location.href="index.html";
   } });
```

Hier wird ein Element mit der ID `test` und dem Label `Testautos` definiert. Diesmal wird das Label angezeigt, da kein System-Icon im nächsten Feld angegeben wird, sondern der Pfad und der Dateiname zu einer PNG-Grafik.

Als Anweisung wird wieder ein `onSelect()`-Event verwendet, allerdings mit der Angabe, dass jetzt eine HTML-Seite aufgerufen wird und nicht nur eine Meldungsbox. In unserem Beispiel rufen wir die Datei *index.html* auf – also wieder die Seite, die die Definitionen für die Navigationsleiste enthält. Würden wir nicht einen besonderen Trick verwenden, würde sich bei jedem Antippen dieses Elements und dem Neuladen der Seite eine neue Navigationsleiste auf die andere legen, und zwar so lange, bis der Bildschirm mit Navigationsleisten gefüllt ist.

Das umgehen wir durch die folgende Angabe, in der es darum geht, die mit den bisherigen JavaScript-Aufrufen definierten Elemente und die Navigationsleiste auch anzuzeigen:

```
if (window.innerHeight == 480) {
  window.uicontrols.showTabBar();
  window.uicontrols.showTabBarItems("toprated", "suche", "test", "anderes");
 }
```

Die Funktionen `window.uicontrols.showTabBar()` und `window.uicontrols.showTabBarItems()` bewirken also die Anzeige aller vorher festgelegten Elemente – um genau zu sein, aller Elemente, die bei `window.uicontrols.showTabBarItems()` mittels ihrer ID angegeben wurden. Beide Funktionen werden aber nur aufgerufen, wenn auf dem Bildschirm noch 480 Bildpunkte an Höhe frei sind. Hätten Sie zwei Navigationsleisten definiert, würde der verfügbare Bildausschnitt kleiner als 480 Bildpunkte sein, und die Anzeige einer weiteren Navigationsleiste würde durch die `if`-Bedingung unterbunden.

Jetzt haben Sie sicher gemerkt, dass wir das Beispiel ohne die Statusleiste kopiert haben. Wenn Sie die schwarze oder graue Statusleiste innerhalb der *Info.plist*-Datei anschalten, müssen Sie auch den `window.innerHeight`-Parameter verkleinern und auf mindestens 460 Bildpunkte setzen.

Native Kopfzeile

Die von uns verwendete Kopfzeile ist Teil der Webseite und wird beim Scrollen der Seite aus dem Bildschirm herausgeschoben. Viele native Anwendungen verwenden feste und statische Kopfzeilen im Zusammenhang mit oder auch ohne Navigationsleiste. PhoneGap bietet Ihnen ebenfalls die Möglichkeit, eine solche Kopfleiste mit zwei ähnlichen JavaScript-Funktionen zu definieren:

```
window.uicontrols.createToolBar();
window.uicontrols.setToolBarTitle("Anwendungstitel");
```

Leider ist dieses Element nicht besonders stabil integriert. Insbesondere im Zusammenhang mit der Navigationsleiste treten oft Probleme und Inkompatibilitäten auf. Möchten Sie eine feste Leiste am oberen Bildschirmrand mit dem Anwendungstitel verwenden, sind die beiden JavaScript-Aufrufe eine gute Möglichkeit. In allen anderen Fällen empfehlen wir aber, auf HTML- und CSS-Definitionen zurückzugreifen.

Native Systemmeldungen: Meldungsboxen, Vibration und Audio-Beep

Sie haben bereits im letzten Beispiel mit der systemeigenen Meldungsbox Bekanntschaft gemacht, als wir uns bestätigen ließen, auf welches Element in der Navigationsleiste der Benutzer getippt hatte. Zu den nativen Systemmeldungen gehören aber auch noch der Vibrationsalarm sowie die Beep-Funktion, die auf einem iPhone eine im *www*-Verzeichnis befindliche Audiodatei (im WAV-Format) abspielt.

Das Beispiel soll vier antippbare Flächen besitzen, die entweder das iPhone vibrieren lassen, eine WAV-Datei abspielen oder eine der beiden Systemmeldungen anzeigen:

Bild 3.47: Wenn Sie schnell etwas anzuzeigen haben: Titel und Button-Label sind vom System vorgegeben – den eigentlichen Meldungstext können Sie selbst festlegen (linkes Bild). Die Angabe des Titels und die des Button-Labels sind zwar optional, lassen aber den Dialog verständlicher wirken (rechtes Bild).

Hier der Beispielcode:

```
<!--
   Native Systemmeldungen in PhoneGap
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
<html>
  <head>
    <title>Meldungsboxen &amp; Co.</title>

   <!-- iWebKit einladen -->
    <link rel="stylesheet" type="text/css" href="style.css">

   <!-- PhoneGap JavaScript laden -->
    <script type="text/javascript" charset="utf-8"
src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

    // Warten, bis PhoneGap geladen ist
    function onLoad() {
        document.addEventListener("deviceready", onDeviceReady, false);
    }

    function onDeviceReady() { }

    // Standard-Meldungsbox anzeigen
    function simpleShowAlert() {
```

```
        navigator.notification.alert('Hallo! Ich bin eine einfache Alert-Box');
    }

    //  Selbst definierte Meldungsbox anzeigen
    function complexShowAlert() {
        navigator.notification.alert('Erwischt. Du hast schon wieder auf dem
iPhone rumgetippt!','Das gibts ja nicht!',
    'Ok - ich machs nicht wieder');
   }

    // Melodie spielen
    function playBeep() {
        navigator.notification.beep();
    }

    // Vibrieren
    function vibrate() {
        navigator.notification.vibrate(1000);
    }

    </script>
  </head>
  <body onload="onLoad()">

  <div id="topbar" class="transparent" >
   <div id="title">Meldungsboxen &amp;amp; Co.</div>
  </div>

  <ul class="pageitem">
   <li class="menu"><a href="#" onclick="simpleShowAlert(); return
false;"><span class="name">Standard-Alert-Box</span></a></li>
  </ul>
  <ul class="pageitem">
   <li class="menu"><a href="#" onclick="complexShowAlert(); return
false;"><span class="name">Selbstdefinierte Box</span></a></li>
  </ul>
  <ul class="pageitem">
   <li class="menu"><a href="#" onclick="playBeep(); return false;"><span
class="name">Melodie spielen</span></a></li>
  </ul>
  <ul class="pageitem">
   <li class="menu"><a href="#" onclick="vibrate(); return false;"><span
class="name">Vibrrrrrieren</span></a></li>
  </ul>

  </body>
</html>
```

Wie Sie bemerkt haben, ruft die JavaScript-Funktion `navigator.notification.alert()` eine systemeigene Meldungsbox auf. Diese muss mindestens den Meldungstext als Parameter beinhalten:

```
navigator.notification.alert('Hallo! Ich bin eine einfache Alert-Box');
```

Sie kann aber optional auch Definitionen für Titel und Button-Label im folgenden Format enthalten:

```
navigator.notification.alert('Meldungstext','Titel','Button-Label');
```

Falls Sie von Ihrer Anwendung aus in bestimmten Situationen ein Audiosignal abspielen möchten, geht das mit der im Code sichtbaren Zeile `navigator.notification.beep`. Auf einem iPhone wird in diesem Fall eine Audiodatei abgespielt, die

- sich im *www*-Verzeichnis befindet,
- im WAV-Format vorliegt und
- nicht länger als 29 Sekunden lang sein darf.

Das Abspielen einer WAV-Datei erfolgt nur beim iPhone. Wird diese Funktion mit PhoneGap für Android verwendet, spielt das Telefon den eingestellten Standardklingelton für Benachrichtigungen ab.

PhoneGap enthält bereits eine mitgelieferte *beep.wav*-Datei, die sich im Verzeichnis *phonegap-iphone/PhoneGapTutorial/www/* befindet. Kopieren Sie diese Datei einfach in das *www*-Verzeichnis Ihres Projekts, und schon klingelt Ihr iPhone beim Aufruf der `playBeep()`-Funktion.

Zum Abschluss macht die JavaScript-Anweisung `navigator.notification.vibrate()` genau das, was sie auch namentlich verspricht: Sie lässt das Telefon vibrieren. Auch hier gibt es wieder einen Unterschied zwischen dem iPhone und einem Android-Gerät. Auf einem iPhone wird der Wert in der Klammer, der die Vibrationslänge in Millisekunden angibt, nicht unterstützt. Das iPhone hat einen festen internen Wert und ignoriert diesen Parameter. Anders Android: Hier kann man die Länge der Vibration in Millisekunden angeben.

Systemseitige Ladeanzeige

Wir haben alle schon zu lang auf diesen Bildschirm geschaut, und als iPhone-Anwender kennen wir ihn auch alle: den Bildschirm mit dem aus kleinen Linien bestehenden Kreis, der sich so schön dreht, wenn wir wieder einmal auf Informationen aus dem Netz warten und diese geladen werden. Mit etwas JavaScript kann diese Ladeanzeige in verschiedenen Darstellungsweisen auch in PhoneGap-Anwendungen mit eingebaut werden.

Die systemseitige Ladeanzeige kann zum einen als kleines Overlay auf der aktuellen Seite dargestellt werden. Das ist hier schwer zu sehen: Aber der Overlay-Hintergrund ist nicht komplett schwarz, sondern sehr dunkel transparent. Alternativ kann eine Ladeanzeige auch den gesamten Bildschirm einnehmen. Das ist besonders dann sinnvoll, wenn nicht nur Teile der Seite nachgeladen werden müssen, sondern sich nach dem fertigen Laden besonders viel auf der Seite ändert oder sogar komplett neuer Inhalt angezeigt wird.

Bild 3.48: Anzeigemöglichkeiten der systemseitigen Ladeanzeige.

Wir haben das wieder anhand eines kleinen Beispiels verdeutlicht:

```
<!--
   Nativer Ladebildschirm (iOS) in PhoneGap
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
<html>
  <head>
    <title>Ladebildschirm</title>

    <!-- iWebKit 5 laden -->
    <link rel="stylesheet" type="text/css" href="style.css">

   <!-- PhoneGap JavaScript laden -->
    <script type="text/javascript" charset="utf-8"
src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

    // Warten, bis PhoneGap geladen ist
    function onLoad() {
        document.addEventListener("deviceready", onDeviceReady, false);
    }

    // Sobald PhoneGap da ist, lade folgende Funktionen:
    function onDeviceReady() { }

    function LadeVollbild(dauer)
       {
          optionen = {
                   'duration': dauer,
            'backgroundOpacity': 0.4,
            'labelText': "Laden..."
          };
          navigator.notification.loadingStart(optionen);
        }

    function LadeInline(dauer)
       {
```

```
          optionen = {
                   'duration': dauer,
             'bounceAnimation': true,
             'fullScreen': false,
                 'labelText': "Laden..."
          };
          navigator.notification.loadingStart(optionen);
      }

     </script>
  </head>
  <body onload="onLoad()">

<div id="topbar" class="transparent" >
 <div id="title">Ladebildschirm</div>
</div>

 <ul class="pageitem">
  <li class="menu"><a href="#" onclick="LadeVollbild(10); return
false;"><span class="name">Ladebildschirm Vollbild</span></a></li>
 </ul>

 <ul class="pageitem">
  <li class="menu"><a href="#" onclick="LadeInline(10); return false;"><span
class="name">Ladebildschirm Overlay</span></a></li>
 </ul>

  </body>
</html>
```

Das Beispiel ist zwar nicht besonders zweckdienlich, verdeutlicht aber sehr gut, wie vielseitig ein Ladebildschirm aufgerufen werden kann. Der eigentliche Aufruf der Ladeanzeige erfolgt mit `navigator.notification.loadingStart();`, wobei der Funktion zahlreiche Optionen mit auf den Weg gegeben werden können:

Parameter	*Beispielwert*	*Beschreibung*
`duration`	`10`	Die Anzeigedauer des Ladebildschirms in Sekunden.
`bounceAnimation`	`true`	Die Ladeanzeige kann mit einer Animation auf den Bildschirm gerufen werden. Obwohl die Option auch bei der Vollbildladeanzeige zur Verfügung steht, sieht sie beim Overlay besser aus.
`fullScreen`	`false`	Standardmäßig ist die Vollbildladeanzeige definiert, sie kann aber mit dieser Option auch auf ein Overlay verkleinert werden.
`strokeOpacity`	`0.2`	Hiermit wird der Kreisstrich bestimmt.

Parameter	*Beispielwert*	*Beschreibung*
`strokeColor`	`FF0000`	Hiermit wird die Farbe der Schrift bestimmt.
`backgroundOpacity`	`0.4`	Hiermit wird die Transparenz des Hintergrunds bestimmt.
`labelText`	`Laden`	Wenn Sie diesen Parameter nicht angeben, wird der englische Begriff `Loading` angezeigt. Allerdings ist er nicht in die Systemlokalisierung eingebunden und wird auch bei eingestellter deutscher Sprache in Englisch angezeigt. Deshalb ist es ratsam, mit diesem Parameter einen alternativen Text anzuzeigen.

In diesem Beispiel werden einige unterschiedliche Parameter je nach gewählter Funktion zusammengefasst und dann mit `navigator.notification.loadingStart()` aufgerufen. In einem wirklich nützlichen Szenario würden wir aber den Benutzer nicht für eine feste Anzahl von Sekunden auf einen Ladebildschirm sehen lassen, sondern im Hintergrund wirklich noch etwas tun und erst nach erledigter Arbeit die Ladeanzeige ausblenden.

Das geschieht mit dem Aufruf von `navigator.notification.loadingStop()`. Sobald diese Funktion aufgerufen wird, ist die Ladeanzeige nicht mehr sichtbar.

3.3.6 Ladeanzeige in der Statusleiste

Insbesondere wenn Daten im Hintergrund geladen werden, ist das durch ein kleinen, sich drehenden Kreis in der Statusleiste sichtbar. Auch PhoneGap kann diese native Funktion über JavaScript und die folgenden Aufrufe ansprechen:

- `navigator.notification.activityStart()` – startet die Ladeanzeige.
- `navigator.notification.activityStop()` – blendet die Ladeanzeige wieder aus.

Das funktioniert natürlich nur unter einer Bedingung: Die Statusleiste, egal ob in Grau oder Schwarz, muss eingeschaltet und darf nicht über *Status bar is initially hidden* in der *Info.plist*-Datei deaktiviert sein.

3.3.7 Adressbucheinträge neu anlegen und abrufen

Ein weiterer Bereich, der für eine Webanwendung unerreichbar ist, für eine mobile Anwendung jedoch sehr nützlich sein kann, ist die Kontaktliste bzw. das Adressbuch des Telefons. PhoneGap bietet vollen Zugriff auf das Adressbuch durch diverse JavaScript-APIs.

Eintrag anlegen

Im ersten Beispiel erstellen wir eine einfache Oberfläche, in der der Benutzer den Vor- und Nachnamen sowie die Telefonnummer eines neuen Kontakts eingeben kann. Das Formular enthält einen *Speichern*-Button, der die Daten direkt ins Adressbuch schreibt und mit einer Meldungsbox bestätigt. Da PhoneGap auch leere Einträge in das Adress-

buch speichern würde, stellen wir vorher sicher, dass der Benutzer mindestens den Vor- oder Nachnamen eingegeben hat.

Bild 3.49: Links: Über eine einfache Oberfläche kann der Benutzer Vorname, Nachname und Telefonnummer eingeben und im Adressbuch speichern.
Mitte: ob der Eintrag erfolgreich war, wird dadurch überprüft, dass die neuen Einträge wieder aus dem Adressbuch abgerufen und angezeigt werden.
Rechts: Bei einem erfolgreichen Eintrag ist der neue Kontakt auch im iOS-Adressbuch sichtbar.

```
<!--
   Kontakte im iOS-Adressbuch anlegen mit PhoneGap
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
<html>
  <head>
    <meta name="viewport" content="width=default-width; user-scalable=no" />
    <title>Kontakte</title>
   <link rel="stylesheet" type="text/css" href="style.css">
   <script type="text/javascript" charset="utf-8"
src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">
   function onBodyLoad() {
    }
function addContact()
{
    // Variablen aus dem Form-Feld lesen
    var neuvorname = document.kontaktinfo.vorname.value;
   var neuname = document.kontaktinfo.nachname.value;
   var neutelefon = document.kontaktinfo.telefon.value;
    if (neuvorname !="" || neuname !="") {
   // Parameter bilden
```

```
   var neuadresse = { 'firstName': neuvorname, 'lastName': neuname,
'phoneNumber': neutelefon };

   navigator.contacts.newContact(neuadresse, addContact_Return);

   }
   else {
   navigator.notification.alert("Mindestens Vor- oder Nachnamen eingeben");
   }
}

function addContact_Return(contact)
{
        if (contact) {
                navigator.notification.alert(contact.firstName + " " +
contact.lastName, "Neu im Adressbuch", "Danke");
        }
}

    </script>
  </head>
  <body onload="onBodyLoad()">

   <div id="topbar" class="transparent" >
    <div id="title">Kontakt hinzuf&uuml;gen</div>
   </div>

    <form name="kontaktinfo">
      <fieldset>
      <ul class="pageitem">
       <li class="smallfield"><span class="name">Vorname</span><input
name="vorname" placeholder="Manfred" type="text"/></li>
       <li class="smallfield"><span class="name">Name</span><input
name="nachname" placeholder="Mustermann" type="text"/></li>
          <li class="smallfield"><span class="name">Telefon</span><input
name="telefon" placeholder="(030) 12121212" type="tel" /></li>
      </ul>
      <ul class="pageitem">
       <li class="button">
        <input type="button" value="Speichern" onClick="addContact(); return
false;">
       </li>
      </ul>

      </fieldset>
    </form>
   </body>
</html>
```

Zuerst erstellen Sie ein HTML-Formular und geben ihm die ID bzw. den Namen `kontaktinfo`. Wird es durch Antippen des Buttons abgeschickt, wird über das JavaScript-Event `onClick` die Funktion `addContact()` aufgerufen.

In dieser Funktion erhalten Sie über das Document Object Model die Werte, die im Formular eingegeben wurden, und weisen folgende neue Variablen zu:

```
// Variablen aus dem Form-Feld lesen
var neuvorname = document.kontaktinfo.vorname.value;
var neuname = document.kontaktinfo.nachname.value;
var neutelefon = document.kontaktinfo.telefon.value;
```

Nur wenn die Variable `neuvorname` oder die Variable `neuname` nicht leer ist, wird der für den JavaScript-Aufruf notwendige Parameter aus Vorname, Nachname und Telefonnummer gebildet …

```
if (neuvorname !="" || neuname !="") {

// Parameter bilden
var neuadresse = { 'firstName': neuvorname, 'lastName': neuname,
'phoneNumber': neutelefon };
```

… und letztendlich an die JavaScript-API übergeben, die den neuen Eintrag im Adressbuch erstellt und anschließend die JavaScript-Funktion `addContact_Return()` aufruft:

```
navigator.contacts.newContact(neuadresse, addContact_Return);
```

Die Funktion macht nichts anderes, als den gerade erstellten Eintrag aus der Datenbank zu holen und den gespeicherten Vor- und Nachnamen in einer Meldungsbox anzuzeigen.

Einträge abrufen

Jetzt wollen wir die gespeicherten Einträge wieder aus dem iPhone-Adressbuch herausholen und eine Liste mit allen im Adressbuch befindlichen Vornamen und Nachnamen erstellen. Zusätzlich wollen wir die von der JavaScript-API zur Verfügung gestellten optionalen Einstellungen ebenfalls berücksichtigen.

Bild 3.50: Links: Das Beispiel besitzt einen einfachen Bildschirm, über den alle Kontakte innerhalb des Adressbuchs abgerufen werden können.

Rechts: Verändert man keinen der vorgegebenen Werte, werden die Vor- und Nachnamen der ersten 20 Kontakte aus Ihrem Adressbuch gelesen und dargestellt.

Dass es sich um die ersten 20 und nicht etwa um die ersten 50 Einträge handelt, wird im Beispiel mit dem Wert unter *Einträge/Seite* festgelegt. Wollen Sie die ersten 20 Einträge überspringen, können Sie unter *Startseite* in unserer Beispielanwendung eine *2* eingeben.

Bild 3.51: Links: Ein sehr nützliches Feature ist der Suchfilter. Er arbeitet nicht nur mit expliziten Anweisungen, sondern versteht auch Platzhalter wie * am Ende des Namens.

Rechts: Im gezeigten Beispiel liefert die Suchanfrage *wen** alle Namen, die mit »wen« beginnen.

Im folgenden Beispielcode sehen Sie, wie wir das gemacht haben:

```
<!--
   Kontakte aus dem iOS-Adressbuch lesen mit PhoneGap
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
<html>
  <head>
    <meta name="viewport" content="width=default-width; user-scalable=no" />

    <title>Kontakte</title>
   <link rel="stylesheet" type="text/css" href="style.css">
   <script type="text/javascript" charset="utf-8"
src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

   function onBodyLoad() {
    }

    // Initiale Variablen festlegen
    var nameFilter = "";
    var pageSize = 20;
    var pageNumber = 1;

    function kontakteAnzeigen() {

    // Variablen durch eventuelle Formulareingaben aktualisieren
    nameFilter = document.kontaktsuche.suchfilter.value;
   pageSize = document.kontaktsuche.maxContacts.value;
   pageNumber = document.kontaktsuche.startpage.value;

    // Aufruf der selbst definierten getContacts-Funktion
    getContacts(parseInt(pageSize), parseInt(pageNumber), nameFilter);
    }

   function getContacts(pageSize, pageNumber, nameFilter) {
    debug.log("getContacts");

    var fail = function() {};
```

```
    var options = {};

    // Die Anzahl von Eintraegen pro Seite kann mit der Option pageSize
festgelegt werden
    if (pageSize)
      options.pageSize = pageSize;

    // Als Option kann auch die anzuzeigende Seitenzahl angegeben werden
    if (pageNumber)
      options.pageNumber = pageNumber;

    // Der Namensfilter kann mit * einen Platzhalter beinhalten
    if (nameFilter)
       options.nameFilter = nameFilter;

    // JavaScript-Aufruf, um Kontaktdaten zu erhalten
   navigator.contacts.getAllContacts(getContacts_Return, fail, options);
   }

   function getContacts_Return(contactsArray)
   {

    // So lange die Schleife durchlaufen, bis alle Kontakte empfangen wurden

     for (var i = 0; i < contactsArray.length; i++) {
       var firstName = contactsArray[i].firstName;
       var lastName = contactsArray[i].lastName;

       if (i + 1 != contactsArray.length)

       // Nachname und Vorname in einem neuen div-Element
       // und Listenelement definieren
       newDiv = document.createElement("div");
      newDiv.ClassName = "content"
       newDiv.innerHTML = '<ul class="pageitem"><li class="menu"><span
class="name"><b>' + lastName + "</b>, " + firstName +'</span></li></ul>';

       // Neues Element und Inhalt in DOM uebernehmen
      listen_div = document.getElementById("footer");
      document.body.insertBefore(newDiv, listen_div);

       }
navigator.notification.alert("Alle Namen importiert", "Fertig", "Vielen
Dank");
}

    </script>
  </head>
  <body onload="onBodyLoad()">

   <div id="topbar" class="transparent" >
    <div id="title">Kontakte anzeigen</div>
   </div>

    <form name="kontaktsuche">
      <fieldset>
      <ul class="pageitem">
```

```
       <li class="smallfield"><span class="name">Suchfilter</span><input
name="suchfilter" placeholder="* = Platzhalter" type="text"/></li>
       <li class="smallfield"><span
class="name">Eintr&auml;ge/Seite</span><input name="maxContacts"
placeholder="20" type="tel"/></li>
       <li class="smallfield"><span class="name">Startseite</span><input
name="startpage" placeholder="1" type="tel"/></li>
      </ul>
      <ul class="pageitem">
       <li class="button">
        <input type="button" value="Anzeigen" onClick="kontakteAnzeigen();
return false;">
       </li>
      </ul>

      </fieldset>
    </form>

   <div id="footer">
   </div>

   </body>
</html>
```

Zuerst haben wir wieder ein iPhone-typisches Formular mit iWebKit angelegt, um dort einige für die API optionale Parameter abzufragen. Falls der Benutzer im Formular keine anderen Werte eingibt, werden die am Anfang des Codes definierten Werte benutzt:

```
var nameFilter = "";
var pageSize = 20;
var pageNumber = 1;
```

Die JavaScript-API `getAllContacts()`, die wir später im Beispiel verwenden, gibt uns nicht nur die Adressinformationen aus dem Adressbuch wieder, sondern die Ausgabe lässt sich über diese drei Parameter auch anpassen:

Variable	*Bedeutung*
`nameFilter`	Über die Variable `nameFilter` kann ein Suchbegriff übermittelt werden, der die API veranlasst, nur Namen zurückzugeben, die mit dem Suchbegriff übereinstimmen. Die Angabe eines Platzhalters am Ende eines Namens mittels eines *-Zeichens ist ebenfalls möglich.
`pageSize`	Ein Adressbuch kann ziemlich lang und komplex sein. Manche Benutzer haben Hunderte, wenn nicht sogar Tausende Adresseinträge auf ihrem Telefon. Wenn Sie die API aufrufen, ist es sinnvoll, diese in verschiedene Seiten zu unterteilen. Über `pageSize` legen Sie fest, wie viele Einträge pro Seite zurückgegeben werden sollen.
`pageNumber`	Mittels `pageNumber` können Sie angeben, welche Seite Sie aus dem Adressbuch abfragen und von welcher Sie Adressdaten erhalten möchten. Haben Sie beispielsweise ein Adressbuch mit 60 Einträgen und `pageSize = 20` definiert, erhalten Sie bei der Angabe von `pageNumber = 2` die Adressbucheinträge 21 bis 40 von der API.

Das Formular ruft beim Antippen des *Anzeigen*-Buttons die Funktion `kontakteAnzeigen()` auf und übergibt über das Document Object Model die einzelnen Formularwerte an die Variablen. Anschließend wird die selbst definierte `getContacts()`-Funktion aufgerufen:

```
getContacts(parseInt(pageSize), parseInt(pageNumber), nameFilter);
```

Interessant bei diesem Aufruf ist, dass die Variablen `pageSize` und `pageNumber` auf einen Integer-Ganzzahlwert gebracht werden. Nachdem die Variablen als Optionsparameter zusammengefasst wurden, erfolgt der eigentliche JavaScript-API-Aufruf `getAllContacts()`, der bei Erfolg die Funktion `getContacts_Return (contactsArray)` aufruft.

Die Variable `contactsArray.length` bestimmt, wie viele Adresseinträge von der API empfangen wurden, und eine `for`-Schleife läuft so lange durch, bis alle Vor- und Nachnamen erfasst sind.

```
for (var i = 0; i < contactsArray.length; i++) {
   var firstName = contactsArray[i].firstName;
   var lastName = contactsArray[i].lastName;
```

Bei jedem Durchlauf der Schleife wird der aktuelle Vorname in die Variable `firstName` und der Nachname in die Variable `lastName` übertragen und auch gleich auf dem Bildschirm ausgegeben. Um direkt von dieser JavaScript-Funktion auf den Bildschirm zu schreiben, benutzen wir wieder das Document Object Model, erstellen einen neuen `div`-Container über:

```
newDiv = document.createElement("div");
```

… und weisen diesem Container die Klasse `content` und den folgenden Inhalt zu:

```
newDiv.ClassName = "content"
newDiv.innerHTML = '<ul class="pageitem"><li class="menu"><span
class="name"><b>' + lastName + "</b>, " + firstName +'</span></li></ul>';
```

Pro Durchlauf wird also ein neuer `ul`- und `li`-Bereich mit dem Nachnamen und dem Vornamen erstellt, über folgende Zeilen oberhalb des `footer` `div`-Containers geschrieben und somit angezeigt:

```
// Neues Element und Inhalt in DOM übernehmen
listen_div = document.getElementById("footer");
document.body.insertBefore(newDiv, listen_div);
```

Ziemlich einfach, oder? Etwas komplizierter ist es, die E-Mail-Adressen oder Telefonnummern eines Kontakts auszulesen. Grund dafür ist, dass ein Kontakt mehrere Telefonnummern oder E-Mail-Adressen besitzen kann. Pro Kontakt wird deshalb für Telefon und E-Mail ein neues Array angelegt. Würden wir der ersten Schleife, in der der Vor- und Nachname abgefragt wird, noch die folgende Schleife hinzufügen, würden wir zu einem Kontakt auch die Telefonnummer erhalten:

```
for (var j = 0; j < contactsArray[i].phoneNumbers.length; ++j) {
 var phoneNumber = contactsArray[i].phoneNumbers[j].label + ": " +
contactsArray[i].phoneNumbers[j].value;
```

```
 navigator.notification.alert(phoneNumber, lastName + ', ' + firstName,
"Aha. Danke.")
}
```

Dabei ist anzumerken, dass wir in `contactsArray[i].phoneNumbers[j].value` die Nummer erhalten, in `contactsArray[i].phoneNumbers[j].label` jedoch auch die dazugehörige Kategorie sehen und damit bestimmen können, ob es sich um eine Privat-, Arbeits- oder Mobilnummer handelt.

Bild 3.52: Über die JavaScript-API erhalten Sie nicht nur die Telefonnummern eines Kontakts, sondern auch die dazu passenden Nummerntypen (Arbeit, privat etc.). Das Gleiche gilt für die verschiedenen E-Mail-Adressen eines Kontakts.

Einen nahezu identischen Code können Sie verwenden, um die E-Mail-Felder abzufragen:

```
for (var k = 0; k < contactsArray[i].emails.length; ++k) {
 var email = contactsArray[i].emails[k].label + " : " +
contactsArray[i].emails[k].value;
 navigator.notification.alert(email, lastName + ', ' + firstName, "Aha.
Danke.")
}
```

3.3.8 Kamera und Fotobibliothek

Jetzt kommen wir zu einem sehr spannenden Bereich: der Kamera und dem Zugriff auf die Fotobibliothek durch die PhoneGap JavaScript-API. Auch wenn die Steuerung oder zumindest der Zugriff auf die Kamera keinen Platz in der HTML5-Spezifikation gefunden hat, ist das mit Sicherheit eines der Features, die in den nächsten Jahren ihren Stammplatz in vielen Browsern erhalten werden. Google hat bereits in der Entwicklerkonferenz I/O 2010 angekündigt, dass eine der zukünftigen Android-Versionen mit einem Webbrowser ausgeliefert wird, der vollen Zugriff auf die eingebaute Kamera bietet. Bis wir aber in dieser Zukunft angekommen sind, können wir mittels PhoneGap ebenfalls die Kamera in unseren Anwendungen nutzen.

Unser Beispiel *Foto-Betrachter* besteht aus sehr einfachem Beispielcode und lässt dem Benutzer die Wahl, ob er ein Foto aus der Fotobibliothek auswählen oder direkt eins mit der Kamera aufnehmen möchte.

Um Fotos aus der Fotobibliothek auszuwählen, wird der iOS-Standardbildschirm aufgerufen. Sie brauchen ihn nicht zu programmieren, denn er wird automatisch durch den API-Aufruf auf den Bildschirm gebracht. Nach der Auswahl erscheint das Foto dann im schwarz markierten Fotobereich unserer kleinen Beispielanwendung.

Bild 3.53: Ein markiertes Fotos erscheint nach der Auswahl im Fotobereich der Beispielanwendung.

Nicht viel anders sieht es bei der Benutzung der Kamera aus: Auf die Kamerafunktionen selbst haben Sie keinen Einfluss, und es wird die iOS-Kameraanwendung aufgerufen. Nachdem Sie ein Bild aufgenommen und für gut befunden haben, wird es ebenfalls im schwarzen Fotobereich unseres Foto-Betrachters dargestellt.

Bild 3.54: Ein aufgenommenes Bild wird im Fotobereich des Foto-Betrachters dargestellt.

Wie immer hier erst einmal der Beispielcode:

```
<!--
   Kamera (Android und iOS) bzw. Fotobibliothek auslesen (iOS)
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
```

```
<html>
  <head>
    <meta name="viewport" content="width=default-width; user-scalable=no" />

    <title>Foto-Betrachter</title>
   <link rel="stylesheet" type="text/css" href="style.css">
   <script type="text/javascript" charset="utf-8"
src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

   function onBodyLoad() {
     var sourceType;
    }

    function holeFoto(auswahl) {

    // Kamera oder Fotobibliothek?
    if (auswahl == 'bibliothek') {
     sourceWahl = 0;   }
   else {
     sourceWahl = 1;
    }

    var options = { quality: 50 }
    options["sourceType"] = sourceWahl;

    // Kamera- bzw. Fotobibliothek-API-Aufruf
    navigator.camera.getPicture(onSuccess, onFail, options);
    }

    // Wird nur aufgerufen, wenn ein Bild erfolgreich geholt wurde
    function onSuccess(imageData) {
    document.getElementById("bild").src = "" +
imageData;
    }

    // Wird im Fehlerfall aufgerufen
    function onFail(meldung) {
        navigator.notification.alert('Folgender Fehler aufgetreten: ' +
meldung);
    }

    </script>
  </head>
  <body onload="onBodyLoad()">

   <div id="topbar" class="transparent" >
    <div id="title">Foto-Betrachter</div>
   </div>

    <form name="sourcetyp">
      <fieldset>
      <ul class="pageitem">
       <li class="button">
        <input type="button" value="Foto aus Bibliothek ausw&auml;hlen"
onClick="holeFoto('bibliothek'); return false;">
       </li>
```

```
       <li class="button">
        <input type="button" value="Foto mit Kamera aufnehmen"
onClick="holeFoto('kamera'); return false;">
       </li>
      </ul>
      </fieldset>
    </form>

     <div>
       <ul class="pageitem" style="background-color:#000000">
          <center><img style="width:200px;height:300px" id="bild" src=""
/></center>
       </ul>
     </div>

   </body>
</html>
```

Wie Sie sehen, ist ein Bildbetrachter, der die native Fotobibliothek und die iPhone-Kamera benutzt, keine hohe Kunst, sondern insgesamt nur ein API-Aufruf. Den Rest macht PhoneGap fast von allein. Doch der Reihe nach: Wir stellen den Benutzer mit zwei Buttons vor die Wahl, ob er das Foto aus der Bibliothek auswählen oder mit der Kamera aufnehmen möchte.

Falls Sie das Beispiel im Simulator, auf einem iPod touch oder einem iPad ausprobieren, wird der Button zur Kameraaufnahme nicht funktionieren. Die Anwendung stürzt weder ab, noch gibt es eine Fehlermeldung – es passiert eben einfach gar nichts, da diese Geräte wie auch der Mac-Simulator keine Kamera besitzen.

Die Variable `sourceType` bestimmt, wo das Foto ausgewählt wird. Wird `sourceType` auf `0` gesetzt, wird beim API-Aufruf die Fotobibliothek aufgerufen. Wird im Gegenzug der Wert auf `1` gesetzt, wird die Kamera benutzt. Eine weitere Option ist der `quality`-Parameter: Dieser kann auf einen Wert von 0 bis 100 gesetzt werden und bestimmt die Qualität des gespeicherten Fotos. Mit der folgenden Zeile wird die API aufgerufen, und die Optionsparameter werden übergeben:

```
navigator.camera.getPicture(onSuccess, onFail, options);
```

Geht alles gut und ist der API-Aufruf erfolgreich, wird die selbst definierte Funktion `onSuccess()`, anderenfalls die Funktion `onFail()` aufgerufen.

```
function onSuccess(imageData) {
    document.getElementById("bild").src = "" +
imageData;
    }
```

Die Funktion `onSuccess` enthält als Ergebnis des `navigator.camera.getPicture()`-API-Calls die Bilddaten in der Variablen `imageData`. Die Bilddaten eines jeden Fotos sind mit Base64 codiert, also einem Verfahren zur Codierung von 8-Bit-Binärdateien in eine normale Zeichenfolge. Die Bilddaten in der Variablen `imageData` sind also nichts weiter als ein normaler String.

Diesen String können Sie entweder im lokalen Speicherbereich (Local Storage) speichern, an einen Server übermitteln oder einfach als Bild innerhalb unserer Anwendung

anzeigen. Da wir in PhoneGap arbeiten, nehmen Sie dazu das HTML-Tag `img` und konstruieren den dazugehörigen `source`-Parameter, indem wir das Datenformat `data:image/jpeg;base64` sowie die Bilddaten über den erstellten String in der Variablen `imageData` miteinander verbinden.

Zum Schluss legen Sie mittels CSS für den Bereich, in dem das Foto angezeigt werden soll, einen schwarzen Hintergrund fest und wählen eine vordefinierte Anzeigegröße aus:

```
<ul class="pageitem" style="background-color:#000000">
   <center><img style="width:200px;height:300px" id="bild" src=""
/></center>
</ul>
```

Würden Sie die CSS-Größenangaben im `img`-Tag nicht mit angeben, würde das Bild in Originalgröße angezeigt werden.

3.3.9 Geolocation

Die aktuelle Position eines Benutzers können Sie bereits mit jedem modernen WebKit-Browser erhalten. Glücklicherweise brauchen Sie Ihren Webcode für eine iOS-Anwendung mit PhoneGap auch kaum umzuschreiben, da PhoneGap genau die gleichen JavaScript-API-Calls wie HTML5 verwendet:

Methode	*Beschreibung*
`geolocation.getCurrentPosition()`	Einfache One-Shot-Anfrage, um die aktuelle Position des Benutzers einmalig zu erfragen.
`geolocation.watchPosition()`	Übermittelt die aktuelle Position des Benutzers in einer bestimmten Frequenz.
`geolocation.clearWatch()`	Stoppt die automatische Übermittlung der aktuellen Position des Benutzers.

Neben dem Längen- und dem Breitengrad gibt die PhoneGap-API noch weitere Werte aus:

- Längengrad
- Breitengrad
- Höhe
- Genauigkeit
- Richtung
- Geschwindigkeit
- Zeit- und Datumsstempel

Bild 3.55: Einfache Geolocation-Anfrage über PhoneGap.

Das folgende kleine Beispiel gibt genau diese Werte auf einer Seite aus:

```
<!--
   Kamera (Android und iOS) bzw. Fotobibliothek auslesen (iOS)
   Sven Haiges / Markus Spiering
-->
<!DOCTYPE>
<html>
  <head>
    <meta name="viewport" content="width=default-width; user-scalable=no" />

    <title>One-Shot Geo</title>
   <link rel="stylesheet" type="text/css" href="style.css">
   <script type="text/javascript" charset="utf-8"
src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

   function onBodyLoad() {
    }

    function holePosition() {

    // Geo-One-Shot-API-Aufruf
    navigator.geolocation.getCurrentPosition(onSuccess, onFail);
    }

    // Wird nur aufgerufen, wenn die Position erfolgreich geholt wurde
    function onSuccess(position) {
    var element = document.getElementById('geolocation');

   element.innerHTML = 'Breitengrad: '  + position.coords.latitude + '<br>' +
    'L&auml;ngengrad: ' + position.coords.longitude  + '<br>' +
     'H&ouml;he: '  + position.coords.altitude  + '<br>' +
     'Richtung: '   + position.coords.heading  + '<br>' +
     'Geschwindigkeit: '  + position.coords.speed  + '<br>'
```

```
    }

    // Wird im Fehlerfall aufgerufen
    function onFail() {
        navigator.notification.alert('Immer geht was schief!');
    }

    </script>
  </head>
  <body onload="onBodyLoad()">

   <div id="topbar" class="transparent" >
    <div id="title">One-Shot als App</div>
   </div>

     <div id="content">
      <ul class="pageitem">
       <li class="button">
        <input type="button" value="Wo bin ich?" onClick="holePosition();
return false;">
       </li>
        </ul>

      <ul class="pageitem">
       <li class="textbox">
         <p id="geolocation">Suche Position...</p>
       </li>
      </ul>
    </div>

   </body>
</html>
```

Das Beispiel ist schnell erklärt: Das Antippen des Buttons *Wo bin ich?* ruft die Funktion `holePosition()` auf, die dann den schon bekannten Geolocation-API-Aufruf vornimmt:

```
navigator.geolocation.getCurrentPosition(onSuccess, onFail);
```

Bekommen wir die Geoposition, wird die Funktion `onSuccess()` ausgeführt, falls nicht, wird die Funktion `onFail()` aufgerufen. Beides sind selbst definierte JavaScript-Funktionen. Die Funktion `onSuccess` enthält die Positionsdaten in der Variablen `position`. Wir nehmen uns dann die einzelnen Werte vor und generieren einen HTML-String, der im `body`-Bereich der Seite angezeigt wird:

```
var element = document.getElementById('geolocation');
element.innerHTML = 'Breitengrad: '  + position.coords.latitude + '<br>' +
'Längengrad: ' + position.coords.longitude  + '<br>' +
'Höhe: '  + position.coords.altitude  + '<br>' +
'Richtung: '   + position.coords.heading  + '<br>' +
'Geschwindigkeit: '  + position.coords.speed  + '<br>'
```

3.3.10 Bewegungssensoren

Als eine der letzten nativen APIs möchten wir Ihnen die API zum Auslesen des Accelerometers, also des Beschleunigungs- und Bewegungsmessers, vorstellen. Die Art und Weise, wie wir die API aufrufen werden, ist eng verwandt mit dem Aufrufen und Verwenden der Geoposition. PhoneGap kennt drei verschiedene JavaScript-APIs:

Funktion	*Erläuterung*
`accelerometer.watchAcceleration()`	Übermittelt die aktuelle Lage des Geräts im dreidimensionalen Raum in einer festgelegten Frequenz.
`accelerometer.clearWatch()`	Stoppt die automatische Übermittlung der Lage des Geräts.
`accelerometer.getCurrentAcceleration()`	Übermittelt die aktuelle Lage des Geräts im Raum als One-Shot-Anfrage. Dieser API-Aufruf ist dem iPhone unbekannt, funkioniert aber beispielsweise auf Android-Geräten. Unter iOS müssen Sie immer die `accelerometer.watchAcceleration()`-API verwenden.

Als Beispiel dient eine Art virtuelle Wasserwaage, die keine Wasserblasen, sondern Koordinaten anzeigt. Wenn sich das Gerät allerdings genau im Hochformat befindet, also der Untergrund genau gerade ist, wird durch den `navigator.notification.beep()`-API-Call ein Ton ausgegeben.

Wir verwenden wieder eine simple Oberfläche mit zwei Buttons, die selbst definierte JavaScript-Funktionen aufrufen, die wiederum entweder die `accelerometer.watchAcceleration()`-API oder die `accelerometer.clearWatch()`-API beinhalten.

Alle 0,1 Sekunden wird die Lage des Telefons im dreidimensionalen Raum ausgewertet und die jeweilige Position auf der x-, y- und z-Achse angezeigt.

Bild 3.56: Auf einer einfachen Oberfläche wird alle 0,1 Sekunden die Lage des Telefons im dreidimensionalen Raum ausgewertet.

```
<!--
   Bewegungssensoren auslesen
   Sven Haiges / Markus Spiering
-->
```

```
<!DOCTYPE>
<html>
  <head>
    <title>Virtuelle Wasserwaage</title>

    <meta name="viewport" content="width=default-width; user-scalable=no" />
   <link rel="stylesheet" type="text/css" href="style.css">
    <script type="text/javascript" charset="utf-8"
src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

    // Festlegen der watchID
    var watchID = null;

    // Warten, bis PhoneGap geladen ist
    function onLoad() {
        document.addEventListener("deviceready", onDeviceReady, false);
    }

    function onDeviceReady() {
    }

    function startAccelerometer() {

        // Als Aktualisierungsfrequenz legen wir 0,1 Sekunden fest
        var options = { frequency: 100 };

      // API-Aufruf
        watchID = navigator.accelerometer.watchAcceleration(onSuccess,
onFail, options);
    }

    // Automatische Uebermittlung der Geraetelage beenden
    function stopAccelerometer() {
        if (watchID) {
            navigator.accelerometer.clearWatch(watchID);
            watchID = null;
        }
    }

    // Wird nur aufgerufen, wenn Beschleunigungswerte erhalten wurden
    function onSuccess(acceleration) {
        var element = document.getElementById('accelerometer');
        element.innerHTML = 'Beschleunigung/Lage X: ' + acceleration.x +
'<br />' +
         'Beschleunigung/Lage Y: ' + acceleration.y + '<br />' +
         'Beschleunigung/Lage Z: ' + acceleration.z + '<br />';

    // Virtuelle Wasserwaage - Sende beep, wenn x- und y-Achse gerade sind
    if ( acceleration.x <= 0.01 && acceleration.x >= -0.01 && acceleration.y
<= -0.99 && acceleration.y >= -1.01 ) { navigator.notification.beep(); }

    }

    // Wird im Fehlerfall aufgerufen
    function onFail() {
        alert('Immer geht was schief!');
```

```
    }

    </script>
  </head>
<body onload="onBodyLoad()">

   <div id="topbar" class="transparent" >
    <div id="title">3D-Bewegungsbestimmung</div>
   </div>

     <div id="content">
      <ul class="pageitem">
       <li class="button">
        <input type="button" value="Start" onClick="startAccelerometer();
return false;">
       </li>
       <li class="button">
        <input type="button" value="Stop" onClick="stopAccelerometer();
return false;">
       </li>
        </ul>

      <ul class="pageitem">
       <li class="textbox">
         <p id="accelerometer">Die Lage im 3-D-Raum ist...</p>
       </li>
      </ul>
    </div>

   </body>
</html>
```

Die virtuelle Wasserwaage besteht aus drei wichtigen JavaScript-Funktionen. Die erste dieser selbst definierten Funktionen ist `startAccelerometer()`, die durch das Antippen des *Start*-Buttons aufgerufen wird. Als einzige Option für den kommenden API-Aufruf kann die Frequenz angegeben werden, die festlegt, wie oft nach einer neuen Raumlage des Geräts geschaut werden soll. Bei einer Wasserwaage wollen wir ständige Aktualisierungen erhalten und setzen aus diesem Grund den Wert auf 0,1 Sekunden. Die Variable `frequency` wird, genau wie bei den automatischen Geopositionsaktualisierungen, in Millisekunden angegeben. Anschließend erfolgt der eigentliche API-Aufruf:

```
watchID = navigator.accelerometer.watchAcceleration(onSuccess, onFail,
options);
```

Der Aufruf wird in der Variablen `watchID` festgehalten. Das ist wichtig, da Sie diese `watchID` dann wieder angeben müssen, wenn Sie z. B. die automatische Übermittlung der Raumpositionsdaten deaktivieren möchten. Wie bei den meisten anderen PhoneGap-APIs geben wir auch hier eine Funktion `onSuccess` an, die bei einer erfolgreichen Datenerfassung aufgerufen wird, und eine Funktion `onFail`, die im Fall eines Fehlers oder Problems aufgerufen wird.

Die Beschleunigungsdaten werden in den Variablen `acceleration.x`, `acceleration.y` und `acceleration.z` gespeichert und alle 0,1 Sekunden auf einer HTML-Seite ausgegeben:

```
var element = document.getElementById('accelerometer');
        element.innerHTML = 'Beschleunigung/Lage X: ' + acceleration.x +
'<br />' +
         'Beschleunigung/Lage Y: ' + acceleration.y + '<br />' +
         'Beschleunigung/Lage Z: ' + acceleration.z + '<br />';
```

Die folgende Grafik verdeutlicht noch einmal, welche Werte Sie in welcher Lage erwarten können und wie die Achsen beispielsweise bei einem iPhone liegen:

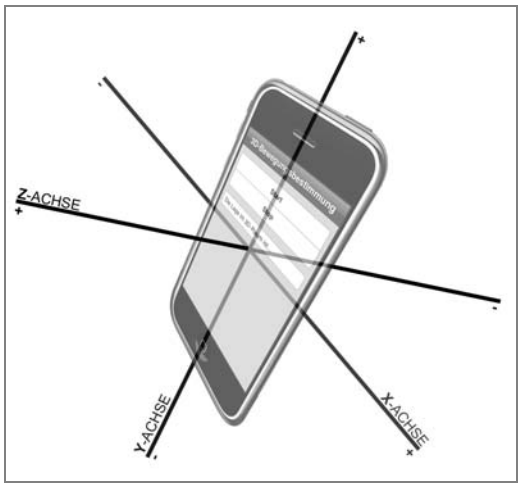


Bild 3.57: Die Achsen des iPhones im Raum

Wenn Sie das Telefon ganz flach auf den Tisch legen, sollten Sie die Idealwerte von x = 0, y = 0 und z = –1 erhalten. Halten Sie das Telefon dagegen ganz genau hochkant, also die Unterseite gerade auf den Boden und genau senkrecht, soll sich die akustische Wasserwaage melden:

```
if ( acceleration.x <= 0.01 && acceleration.x >= -0.01 && acceleration.y <=
-0.99 && acceleration.y >= -1.01 ) { navigator.notification.beep(); }
    }
```

Mit dieser `if`-Abfrage schauen wir, ob sich x- und y-Wert im Bereich von ca. x = 0 und y = –1 befinden. Wenn ja, führen Sie den PhoneGap JavaScript-API-Call `navigator.notification.beep` aus und hören, wenn sich die Datei *beep.wav* im *www*-Verzeichnis des Projekts befindet, einen Piepton.

3.3.11 iPad-Apps mit PhoneGap erstellen

Wenn Sie den Anweisungen im Buch bislang gefolgt sind, haben Sie Anwendungen für das iPhone und den iPod touch erstellt. Diese laufen natürlich auch auf einem iPad, allerdings in dem Modus, in dem Sie über einen Vergrößerungsbutton die Anwendung auf iPad-Größe hochskalieren können. In diesem Modus verliert nahezu jede Anwendung ihren Reiz und ihr besonderes Aussehen, und auch für die Benutzer eines iPads sind derartige Anwendungen eher Notlösungen.

Bild 3.58: Unsere Beispiele wurden bisher als iPhone-Programme kompiliert und laufen auf einem iPad in einem kleinen Fenster. Das Fenster daneben kann auf die doppelte Größe hochskaliert werden, doch der Charme einer Anwendung geht dadurch verloren, und der größere Bildschirm wird nicht effektiv genutzt.

Da Xcode auch zur Erstellung von iPad-Anwendungen genutzt wird, lassen sich mit PhoneGap und entsprechendem HTML-Code ebenfalls vollwertige iPad-Anwendungen erstellen. Wir nehmen dazu in Xcode das oben gezeigte Beispiel zur Darstellung der Navigationsleiste und werden es jetzt als iPad-Anwendung kompilieren.

1. In Xcode gehen Sie in das Menü *Project* und wählen dort den Unterpunkt *Edit Project Settings* aus. Im daraufhin erscheinenden Fenster müssen Sie in den *Build*-Bereich wechseln und unter *Configuration* im Drop-down-Menü den Eintrag *All Configurations* auswählen.

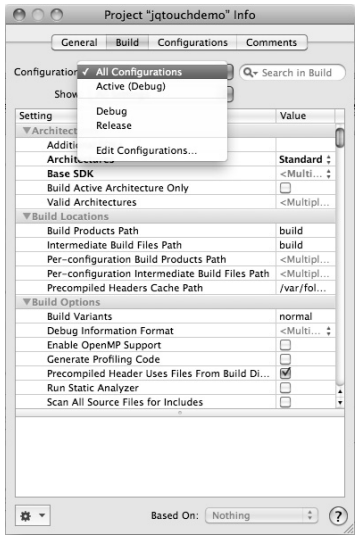


Bild 3.59: Auswahl der Option All Configurations.

2. Wählen Sie anschließend als *Base SDK* die neueste iOS-Version, in unserem Fall *iPhone Device 4.0* unter *iPhone OS Device SDKs* aus:

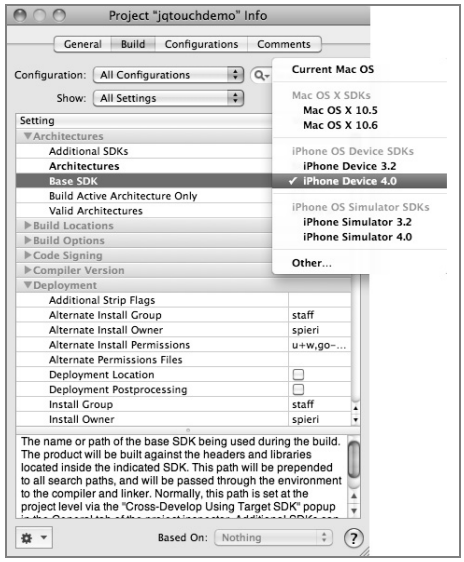


Bild 3.60: Auswahl der Version4 des iPhones-SDK

3. Scrollen Sie dann in den Bereich *Deployment* und geben Sie dort unter *Targeted Device Family* einfach *iPad* an:

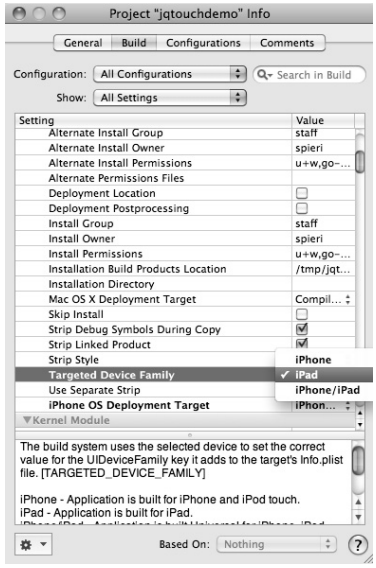


Bild 3.61: Auswahl des iPads als Zielsystem.

4. Sie können jetzt das Einstellungsfenster schließen, noch einmal sicherstellen, dass im oberen linken Menü in der Xcode-Hauptansicht der iPad Simulator ausgewählt ist, und anschließend auf *Build and Run* klicken. Sie werden sehen, dass aus der unproportionierten iPhone-App, die im iPhone Simulator auf einem iPad angezeigt wurde,

nun eine richtige iPad-App geworden ist, die den vollen Bildschirm des iPads nutzen kann.

In den *Projekteinstellungen* können Sie unter *Targeted Device Family* auch *iPhone/iPad* auswählen, um eine Anwendung zu kompilieren, die auf dem iPhone und iPod touch, aber auch auf einem iPad läuft. Möchten Sie das tun, empfiehlt es sich, den folgenden Code mit in den Header einzubauen, um für die jeweilige Geräteklasse unterschiedliche CSS-Definitionen zu benutzen:


```
<!-- iPad-/iPhone-spezifische CSS-
Anweisungen -->
<link rel="stylesheet" media="only
screen and (max-device-width:
1024px)" href="ipad.css"
type="text/css" />
<link rel="stylesheet" media="only
screen and (max-device-width: 480px)"
href="iphone.css" type="text/css" />
```

Wenn Sie eine Webanwendung bzw. App für das iPad erstellen, beachten Sie bitte folgende Punkte:

- Um den Bildschirm des iPads voll auszunutzen, empfehlen wir, Ihr Webdesign auf die Bildschirmdimensionen des iPads auszurichten: 768 x 1.024 Bildpunkte im Hochformat und 1.024 x 768 Bildpunkte im Querformat.
- Das Programmsymbol einer iPad-App hat eine Bildgröße von 72 x 72 Bildpunkten. Die im PNG-Format vorliegende Datei kann in der *Info.plist*-Datei unter *Icon File* angegeben werden.
- Auch der Startbildschirm einer iPad-App muss sich an der Bildschirmgröße orientieren. PhoneGap erstellt mit jedem neuen Projekt bereits zwei passende PNG-Dateien: *Default-Landscape.png* für einen Querformatstartbildschirm und *Default-Portrait.png* für die Hochformatvariante.

Mit den Änderungen in den Projekteinstellungen haben wir zwar den gesamten iPad-Bildschirm zur Verfügung, allerdings nimmt die Navigation immer noch die gesamte Bildschirmbreite ein. Ein beliebtes Navigationskonzept auf dem iPad ist eine statische Navigationsleiste auf der linken Seite und die Anzeige von dynamischen Inhalten auf der rechten Seite.

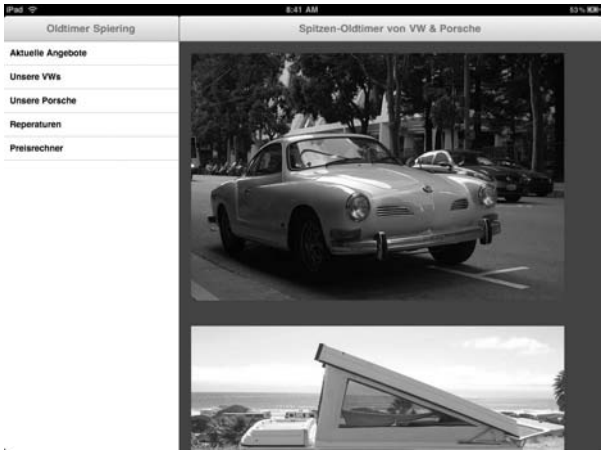


Bild 3.62: Die Oldtimer-Autohaus-Seite-App, für das iPad optimiert. Die Navigationsleiste bleibt statisch, während der Inhalt auf der rechten Seite scrollbar ist.

Was einfach aussieht, ist relativ umständlich zu programmieren. Vor einigen Jahren hätte man hierzu HTML-Frames benutzt, heute würde man für die Navigation und den Inhalt `div`-Container mit der CSS-Definition `position:fixed` erstellen. Diese CSS-Definition wird aber von mobilen Browsern aus guten Gründen nicht unterstützt.

Die im letzten Screenshot gezeigte Navigationsleiste wurde mittels `iScroll` von Matteo Spinelli realisiert. Nach dem Einbinden der von ihm erstellten JavaScript-Bibliothek und den im iPad-Beispiel sichtbaren CSS-Definitionen müssen für die Navigationsleiste und den Inhaltsbereich nur unterschiedliche `div`-Container erstellt werden. In unserem Beispiel verwenden wir für die Leiste:

```
<div id="sidebar">
   <header>Oldtimer Spiering</header>
   <nav>
      <ul id="navScroller">
         <li><b>Aktuelle Angebote</b></li>
         <li><b>Unsere VWs</b></li>
```

… und für den Inhaltsbereich:

```
<div id="content">
   <header>Spitzen-Oldtimer von VW &amp;amp; Porsche</header>
   <article>
      <div id="contentScroller" style="background:#424242;">
      ...
```

⊡ Lesezeichen

http://cubiq.org/iscroll

Hier können Sie den `iScroll`-Code inklusive einiger Beispiele kostenlos herunterladen.

3.4 Android Apps mit PhoneGap entwickeln

Wir haben die bisherigen Beispiele mit Xcode aus dem iOS SDK in native iOS-Anwendungen für iPhone, iPod touch und iPad umgewandelt. Fast alle Funktionen, die Sie dabei kennengelernt haben, lassen sich auch für Android umsetzen.

1. Zu Beginn werden wir uns wieder mit unserem Standardbeispiel aus der iWebKit-Vorstellung, der Oldtimer-HTML-Seite, beschäftigen. Dazu haben wir die Hauptbestandteile dieser Seite wieder in ein neues *www*-Verzeichnis innerhalb des *phonegap-android*-Ordners kopiert:

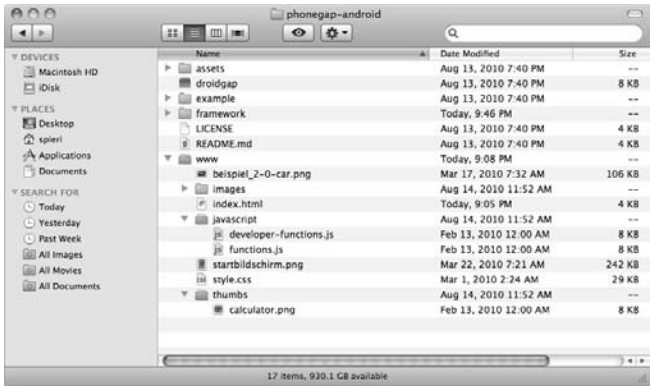


Bild 3.63: Die Beispiele im *www*-Verzeichnis.

Anders als in einer iOS-Anwendung erstellen Sie das *www*-Verzeichnis zuerst manuell, es wird nicht durch die Entwicklungsumgebung erzeugt. In dieses *www*-Verzeichnis gehören aber wie auch für iOS-Apps alle Bestandteile Ihrer mobilen Seite: HTML-Seiten, lokale Ressourcen (Bilder, Videos), CSS- und JavaScript-Dateien. Sie können theoretisch den *www*-Ordner überall auf Ihrem Rechner anlegen, der Einfachheit halber empfiehlt sich die temporäre Anlage allerdings in dem Ordner, in dem PhoneGap für Android installiert wurde.

2. Am Anfang jeder neuen Android-App steht nicht nur das passende HTML, sondern auch die recht umständliche Prozedur in der Konsole bzw. im Terminalprogramm. Rufen Sie dieses Programm auf und navigieren Sie in das Verzeichnis *phonegap-android*. Von dort aus rufen Sie den folgenden Befehl (nur mit Ihren Parametern) auf:

```
./droidgap ~/android-sdk-mac_86 oldtimermarkt com.spiering.automarkt www/
~/autohaus/androidapp
```

Sie haben diesen Aufruf schon einmal bei der Anlage des PhoneGap-Beispiels für Android kennengelernt. Die Angaben bedeuten:

Parameter	*Erläuterung*
`~/android-sdk-mac_86`	In diesem Verzeichnis befindet sich die Installation des Android SDK. Je nachdem, welche Android-Version Sie installieren, kann sich die Angabe des Verzeichnisses ändern.
`oldtimermarkt`	Der Name der Android-App.

Parameter	*Erläuterung*
`com.spiering.automarkt`	Das ist der Paketname der Anwendung. Der Paketname ist eine einzigartige ID Ihres Programms und sollte aus den Bestandteilen *com.firmenname.programmname* bestehen. Unter iOS wird das »Bundle ID« genannt. Mehr dazu können Sie im Abschnitt zum Verkauf Ihrer iOS-App in iTunes lesen.
`www/`	Der Verzeichnisname unseres *www*-Verzeichnisses mit dem HTML-Code, JavaScript, CSS und allen Grafiken. Da der *www*-Ordner in das *phonegap-android*-Verzeichnis kopiert wurde, müssen Sie hier keine langen Pfadnamen angeben.
`~/autohaus/androidapp`	Das Zielverzeichnis, in dem »droidgap« alle Dateien und Verzeichnisstrukturen für das Android-Projekt anlegen soll.

Die Anweisungen im Terminal sehen bei erfolgreicher Ausführung wie folgt aus:

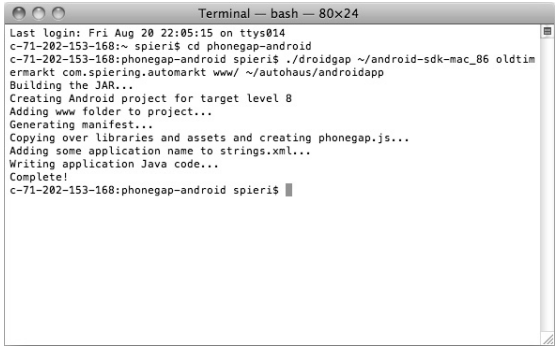


Bild 3.64: Blick auf das Terminalprogramm.

Dadurch wird diese Verzeichnisstruktur erstellt:

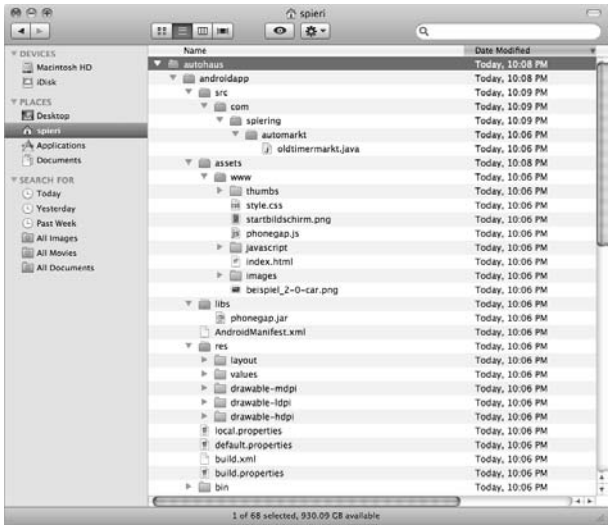
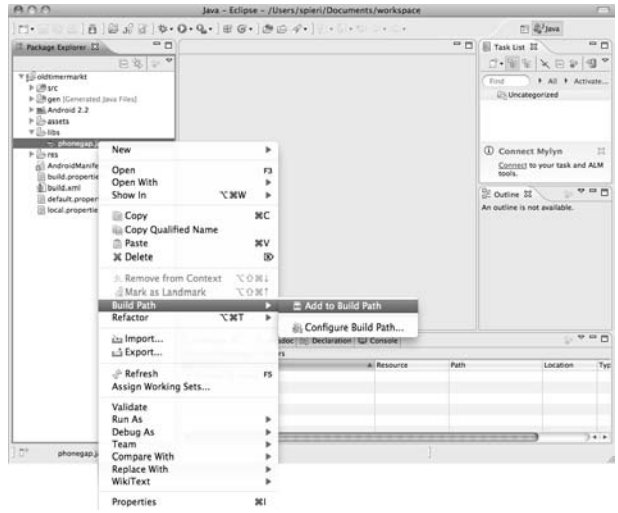


Bild 3.65: Die erstellte Verzeichnisstruktur.

3. Sie können jetzt das Terminalprogramm bzw. die Konsole schließen und Eclipse öffnen. Wie auch beim ersten Android-Beispiel gehen Sie hier in das Menü *File* und wählen dort *New Project* aus.

4. Im folgenden Fenster wählen Sie im Ordner *Android* den Assistenten *Android Project* aus und bestätigen die Auswahl mit *Next*. Da wir unsere Quelldaten bereits erstellt haben, können wir unter *New Android Project* wieder *Create project from existing source* auswählen und das Verzeichnis angeben, das Sie gerade über Droidgap haben erstellen lassen. Zum Schluss wählen Sie in diesem Fenster eine Android-Version aus – in diesem Beispiel Android 2.2.

5. Dass Sie nun eventuell in der *Problems*-Anzeige eine oder mehrere Fehlermeldungen sehen, ist normal. Wie auch bei der vorherigen Installation müssen Sie jetzt im seitlichen *Package Explorer* unter unseren App-Namen den Ordner *libs* aufklappen und dort per rechter Maustaste und *Build Path*/*Add to Build Path* die Datei *phonegap.jar* in die Liste der Dateien aufnehmen, die beim Erstellen einer App berücksichtigt werden.



6. Nun können Sie erneut den App-Namen im *Package Explorer* markieren, auf den grünen Play-Button drücken und *Android Application* auswählen, da wir ja bereits bei der Installation des Android SDK und PhoneGap für Android unser virtuelles »Supertelefon« definiert hatten. Nach einer kurzen Weile sollte dann der Android-Emulator starten und unsere eigene Android App zeigen.

Bevor wir damit beginnen, einige der nativen Funktionen in unserer Android-Anwendung zu testen, müssen noch zwei Dinge erledigt werden: die Android-JavaScript-Bibliothek von PhoneGap in den Code einbinden und – um die nativen Funktionen auch ordentlich testen zu können – die App auf ein richtiges Gerät übertragen.

Die erste Aufgabe, die JavaScript-Bibliothek einzutragen, ist einfach: Wie in Xcode können Sie auch in Eclipse die Dateien komfortabel bearbeiten. Navigieren Sie im *Package Explorer* in das Verzeichnis *assets* und *www* (hier sind alle unsere HTML-, CSS- und JavaScript-Dateien hineinkopiert wurden) und wählen Sie aus dem Kontextmenü für die *index.html* die Option *Open With/Text Editor*:

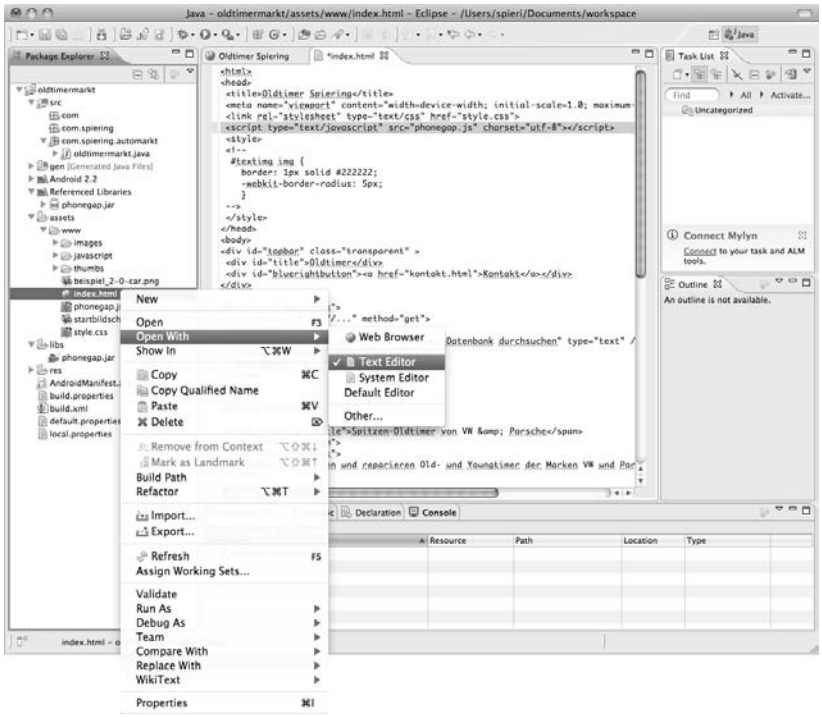


Bild 3.66: Öffnen der JavaScript-Bibliothek

In den `head`-Bereich der *index.html* tragen Sie einfach die folgende Zeile ein:

```
<script type="text/javascript" src="phonegap.js" charset="utf-8"></script>
```

3.4.1 Mit PhoneGap und Eclipse direkt auf einem Android-Gerät testen

Das Testen Ihrer Android App auf einem »echten« Gerät ist ein recht einfacher Prozess. Zwar können Sie bereits vieles mit dem Emulator testen, doch gerade wenn es um bestimmte Hardwarefunktionen wie beispielsweise Geoposition oder die Bewegungssensoren geht, kommen Sie um das Testen auf einem richtigen Android-Telefon nicht herum.

Als ersten Schritt müssen Sie auf Ihrem Handy *USB-Debugging* aktivieren. Diese Option finden Sie in der Einstellungsanwendung unter dem Punkt *Anwendungen* und *Entwicklung:*

Bild 3.67: USB-Debugging aktivieren.

Bevor Sie Ihr Gerät an den Computer anschließen, müssen Sie, wenn Sie unter Windows arbeiten, zuerst einen USB-Treiber installieren. Den Treiber können Sie von der Windows-USB-Treiber-Seite – *http://bit.ly/xvSSl* – herunterladen und die weiteren Anweisungen auf der Seite befolgen. Wenn Sie unter Mac OS X arbeiten, können Sie diesen Schritt überspringen.Schließen Sie jetzt Ihr Telefon über das USB-Kabel an den Rechner an. Da Sie das USB-Debugging aktiviert haben, erscheint unmittelbar nach dem Anstecken ein kleiner Käfer und die Meldung *USB-Debugging verbunden* auf dem Handybildschirm. Zurück in Eclipse, müssen Sie wie gewohnt nur auf den grünen Play-Button klicken. Die Anwendung wird in Sekundenschnelle auf dem Telefon installiert und ausgeführt. Einfach, oder? Sobald Sie das Telefon vom Rechner trennen und erneut auf *Play* klicken, führt Eclipse automatisch wieder den Emulator aus.

3.4.2 Eigene Programmsymbole erstellen und verwenden

Da wir noch keine neuen Icon-Dateien benannt und erstellt haben, verwendet unser Android-Beispiel noch das typische blaue Phone-Symbol, das sich gleich drei Mal in unserem Projektordner befindet:

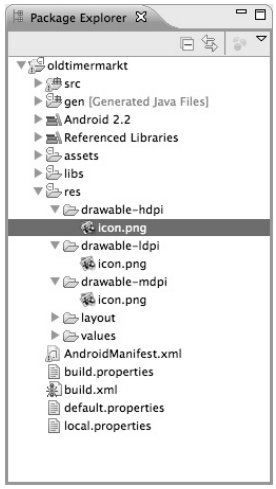


Bild 3.68: Im Verzeichnis *res* befinden sich die drei Ordner *drawable-hdpi* für hochauflösende Bildschirme, *drawable-mdpi* für mittelauflösende Bildschirme und *drawable-ldpi* für Bildschirme mit niedriger Auflösung.

In diese Ordner werden jeweils die einzelnen nativen Grafikelemente einer Anwendung platziert, damit Android sich je nach verwendetem Gerät die am besten passende Grafik aussuchen kann. Wird nur eine Grafikgröße zur Verfügung gestellt, kann das Android-System die Grafik hoch- oder herunterrechnen, doch für eine bestmögliche Anzeige empfiehlt es sich, drei verschiedene Größen zu erstellen.

Wie Sie sich sicher schon gedacht haben, repräsentiert die Datei *icon.png* das jeweilige Programmsymbol in den Ordnern. Dabei hat es sich PhoneGap etwas einfach gemacht und einfach das Symbol aus der iPhone-Version in der gleichen Größe (57 x 57 Pixel) in alle drei Ordner kopiert. Das werden wir jetzt korrigieren, indem wir drei Symbolgrößen entsprechend den Android-Richtlinien erstellen:

Bildschirmauflösung	*Gesamtgröße (a)*	*Symbolgröße (b)*	*Größe eines quadratischen Symbols (c)*
drawable-hdpi	72 x 72	60 x 60	56 x 56
drawable-mdpi	48 x 48	40 x 40	38 x 38
drawable-ldpi	36 x 36	30 x 30	28 x 28

Jetzt haben Sie nicht nur drei verschiedene Bildschirmkategorien, sondern auch noch pro Kategorie drei unterschiedliche Größen.

Keine Angst – Sie brauchen nur eine Icon-Größe pro Datei zu wählen – und zwar die Gesamtgröße (a). Allerdings geben die Android-Richtlinien vor, dass sich Ihr eigentliches Symbol je nach Form an der Symbolgröße (b) oder der quadratischen Symbolgröße (c) orientieren sollte.

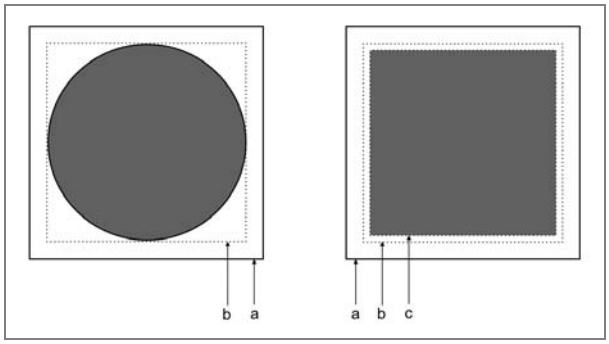


Bild 3.69: Die Rahmen für die Programmsymbole

Ein rundes oder nicht quadratisches Programmsymbol sollte demnach in den Rahmen (b) passen, wobei die gesamte Symboldatei die Größe (a) besitzen muss. Ein quadratisches Programmsymbol sollte im Gegensatz dazu in den Rahmen (c) passen, da ein quadratisches Symbol optisch größer und massiver erscheint als ein Symbol, das keine quadratische Form besitzt.

Programmsymbole mit einer Photoshop-Vorlage erstellen
Falls Sie Photoshop benutzen, können Sie von der offiziellen Android-Seite eine Datei mit Photoshop-Vorlagen für Programmsymbole herunterladen: *http://bit.ly/aQD089*. Diese Vorlage enthält diverse für Programmsymbole empfohlene Materialstrukturen und hilft bei der Gestaltung eines eigenen Icons.

Der innere Bereich kann auch mit einem Schlagschatten versehen werden, der auf dem (a)-Bereich sichtbar wird. Hier sind die empfohlenen Android-Photoshop-Schatteneinstellungen für die hdpi-Auflösung:

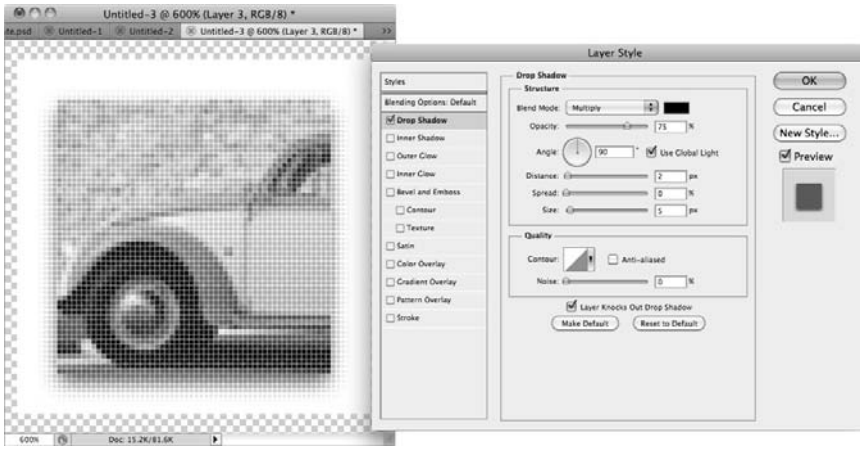


Bild 3.70: Schatten für hochauflösende Displays

Und hier die für die mdpi-Auflösung:

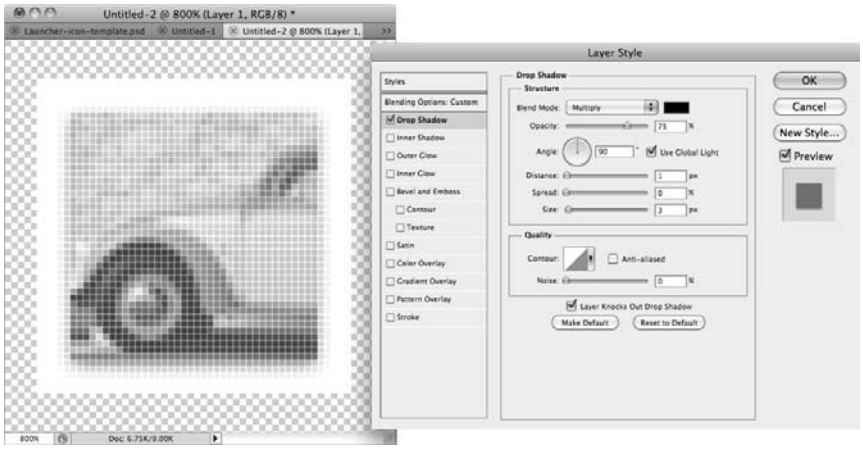


Bild 3.71: Schatten für mittelauflösende Displays

Schlagschatten werden für ldpi-Symbole aufgrund der geringen Icon-Größe nicht empfohlen. Haben Sie Ihre Symbole in den drei verschiedenen Größen erstellt, können

Sie die in den drei Verzeichnissen im *res*-Ordner befindlichen *icon.png*-Dateien mit den neuen Symbolen ersetzen.

Bevor Sie die neuen Symbole im Emulator oder auf Ihren Android-Telefon sehen, müssen Sie in Eclipse im Menü *Project* den Punkt *Clean* auswählen und dann erst den grünen Play-Button drücken.

3.4.3 Einstellungen in der Datei AndroidManifest.xml

Im Laufe Ihrer Android App-Karriere werden Sie immer wieder Kurztrips zur Datei *AndroidManifest.xml* unternehmen. Innerhalb von Eclipse finden Sie die für die Android-Entwicklung wichtige Datei im Hauptverzeichnis Ihres Projekts. Um sie zu bearbeiten, führen Sie einfach einen Doppelklick auf den Dateinamen im *Package Explorer* aus.

Die Manifestdatei enthält alle wichtige Informationen und Einstellungen, die zur Ausführung einer Android-App benötigt werden. Bevor auch nur eine Zeile Code Ihrer Android-Anwendung ausgeführt wird, werden die im Manifest zementierten Einstellungen gelesen und beachtet. In Eclipse können Sie im Manifest-Editor zwischen diversen Ansichten umschalten: *Manifest*, *Application*, *Permissions*, *Instrumentation* und dem XML-Bearbeitungsmodus *AndroidManifest.xml*.

App auf bestimmte Bildschirmgrößen beschränken

Wenn Sie in Ihrer App relativ statische Grafiken in einer bestimmten Größe verwenden, kann es mitunter sinnvoll sein, die App für ganz kleine oder ganz große Bildschirmgruppen nicht zur Verfügung zu stellen. Im *Manifest*-Bereich können Sie diese Einstellung vornehmen. Klicken Sie unter *Manifest Extras* auf *Supports Screens* und wählen Sie anschließend in der rechten Seite aus, ob bestimmte Bildschirmgruppen unterstützt (*true*) werden sollen oder nicht (*false*).

Start im Querformat erzwingen

Möchten Sie beispielsweise Ihre Android App unbedingt im Querformat starten lassen, ist das eine Einstellung, die Sie im Manifest vornehmen können. Klicken Sie dazu auf den Bereich *Application* am unteren Fensterrand, und wählen Sie dann unter *Application Nodes* Ihren Anwendungsnamen aus. Auf der rechten Seite erscheint dann eine ganze Reihe von Einstellungsmöglichkeiten, die sich global auf Ihre Anwendung auswirken.

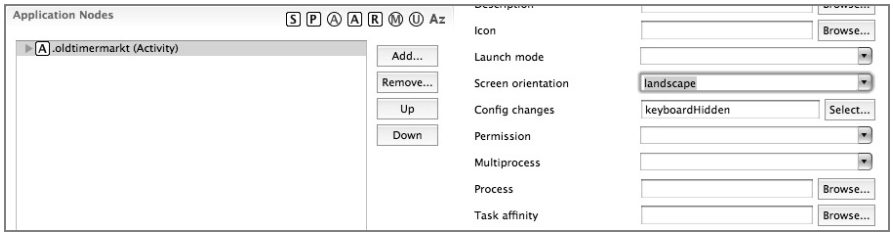


Bild 3.72: Diese Einstellungen erzwingen den Start der App im Querformat.

Setzen Sie *Screen orientation* auf *landscape*, klicken Sie bei *Config changes* auf *Select* und entfernen Sie das Häkchen bei *orientation*. Sie legen somit fest, dass die App auf jeden Fall im Querformat startet und der Benutzer die *orientation*, also die Ausrichtung, auch nicht ändern kann. Selbst wenn der Benutzer das Android-Telefon hochkant hält, wird sich der Bildschirm nicht mitdrehen.

Android-Titelleiste ausblenden

Die Android-Titelleiste ist der Bereich am oberen Rand des Bildschirms, in dem der App-Name steht und der direkt unterhalb der Statusleiste liegt. Bei kleineren PhoneGap-Android-Apps bekommen Sie sie nur für einen kurzen Augenblick zu sehen, nämlich nachdem PhoneGap gestartet ist und bevor der Inhalt der *index.html* aus dem *www*-Ordner angezeigt wird.

Über das Manifest können Sie die Anzeige der Titelleiste unterbinden. Hierzu bleiben Sie in den gleichen *Application Nodes* wie zur Einstellung des Querformats, geben aber unter *Theme* die folgende Zeile ein:

```
@android:style/Theme.NoTitleBar
```

3.4.4 Android-App mit Menüfunktionen versehen

Der Umstand, dass PhoneGap eine native Android-Anwendung generiert, die im Wesentlichen Webinhalte innerhalb dieser Anwendung darstellt, macht die Programmierung leicht, beinhaltet aber eine Einschränkung, die wir gekonnt überbrücken müssen. Nahezu alle Android-Anwendungen besitzen Programmmenüs, die der Benutzer über die Menütaste aufrufen kann und die Zugriff auf oft benutzte Programmteile oder -funktionen bieten.

Bei einer Webseite berücksichtigt man diese Menüs nie, da beim Antippen der Menütaste das Browsermenü erscheint und somit diese Funktion bereits durch den Browser belegt ist. Auch beim iPhone oder generell unter iOS spielt das keine Rolle, da es einfach keine Menütaste und auch keine klassischen Menüs gibt.

In der PhoneGap-Community werden deshalb schon seit einiger Zeit Lösungen diskutiert, die bislang noch nicht komplett in die Codebasis von PhoneGap für Android eingeflossen sind. Daher müssen wir uns kurz selbst helfen, indem wir im Eclipse Package Explorer die Datei *phonegap.js* ansteuern und sie im internen Texteditor öffnen.

Suchen Sie in dieser Datei nach dem Ausdruck *menuKeyDown*. Falls er da ist, wurde zumindest ein Teil des Codes in die offzielle PhoneGap-Codebasis übernommen. Falls nicht, fügen Sie einfach folgende Zeilen der *phonegap.js* hinzu:

```
KeyEvent.prototype.menuTrigger = function()
{
  var e = document.createEvent('Events');
  e.initEvent('menuKeyDown');
  document.dispatchEvent(e);
}
```

Gehen Sie dann zur *index.html* des aktuellen Beispiels und schreiben Sie die folgenden Zeilen in den `head`-Bereich unterhalb der Zeile, mit der wir die *phonegap.js*-Datei in den Code laden:

```
<script type="text/javascript" charset="utf-8">

// Menüleiste einblenden
   document.addEventListener("menuKeyDown", function(){
                            alert('Jemand hat den Menü-Button angetippt!')
                                }, false);

    // Warten, bis PhoneGap geladen ist
    function onLoad() {
        document.addEventListener("deviceready", onDeviceReady, false);
    }
function onDeviceReady() {
}
</script>
```

Bei der iOS-Entwicklung mit PhoneGap haben Sie bereits das Event `deviceready` kennengelernt, das die JavaScript-Funktion `onDeviceReady` aufruft, sobald PhoneGap komplett geladen ist. Das Event `menuKeyDown` ist ähnlich aufgebaut. Sobald ein Benutzer das Menüfeld oder die Menütaste auf einem Android-Gerät antippt, wird die definierte Funktion ausgeführt. In diesem Fall ist es einfach eine native Meldungsbox.

Somit können wir zumindest feststellen, ob der Benutzer die Menütaste gedrückt hat, und eine entsprechende Funktion ausführen. Wenn wir uns sicher sind, dass unsere Android-Anwendung nicht über die vertikale Bildschirmgröße hinausgeht, können Sie sich auch mit diesem einfachen Skript behelfen, das auf Menütastendruck ein CSS-Overlay anzeigt und es bei nochmaligem Druck wieder verschwinden lässt.

Im `head`-Bereich definieren Sie zusätzliches CSS für die Box, in der das Menü dargestellt werden soll:

```
<style type="text/css">
     #menuoverlay {
     background-color:rgba(255, 255, 255, 0.9);
     border:2px solid;
     border-color:#736d73;
     left:0px; bottom:0px; width:100%; height:100px;
     z-index:99;
     visibility:hidden;
     }
</style>
```

Weisen Sie das CSS der ID `menuoverlay` zu, und wählen Sie eine weiße Hintergrundfarbe mit 10 % Transparenz. Der Rand soll in einer dunkelgrauen Farbe erscheinen, und als Position wurde die linke untere Ecke gewählt, von der aus die 100 Pixel Höhe berechnet werden.

Da das CSS-Overlay jeweils an- oder ausgeschaltet werden soll, sobald der Benutzer auf die Menütaste tippt, speichern Sie den aktuellen Anzeigestatus in der Variablen *toggleID*:

```
// Festlegen des Toggle-Status
var toggleID = 0;
```

Die Funktion `showMenu()` ruft die Funktion `show()` mit dem jeweiligen `visible`- oder `hidden`-Parameter auf, je nach Wert der `toggleID`-Variablen. Mit der Funktion `show()` sprechen Sie den später im `body`-Bereich definierten `div`-Container an, der durch die CSS-Definitionen am Anfang der Seite auf `hidden`, also unsichtbar, gesetzt wurde und hiermit angezeigt oder versteckt wird.

```
// Menü anzeigen
function showMenu() {
  if (toggleID == 0) {
       toggleID = 1;
       show('visible');
   }
   else {
       toggleID = 0;
       show('hidden');
     }
   }

// Menüfunktion definieren
 function show(status){
    e=document.getElementById('menuoverlay');
    e.style.visibility=status;
  }
```

Ganz zum Schluss, innerhalb des `body`-Bereichs, definieren Sie noch ein ganz einfaches `div`, das den Inhalt des kleinen Menüs enthält:

```
<div id="menuoverlay"><center>
        <img src="menu_mapmode.png" alt="" border="0"><br />
        <small>Karte aufrufen</small>
        </center>
 </div>
```

Wenn Sie den folgenden Code in das Beispiel integriert haben und eine entsprechende Grafik mit in das *www*-Verzeichnis kopieren, sollte Ihr Android-Bildschirm wie nebenstehend aussehen.

Sollten Sie umfangreichere Android-Seiten erstellen und ein komplexeres Menü benötigen, empfehlen wir Ihnen, einen Blick auf das JavaScript-Paket von »Slide-in menu« zu werfen. Dieses frei benutzbare Paket kann ein Menü per JavaScript ein- und ausblenden. Allerdings ist der Code derzeit nur für ein Menü am oberen Bildschirmrand konzipiert. Ein Update, das das Menü wahlweise an allen Seiten anzeigen kann, wurde vom Autor bereits in Aussicht gestellt. Weitere Informationen und den Code finden Sie unter *http://cubiq.org/slide-in-menu*.

3.4.5 Original-Android-Grafiken in Ihren Projekten verwenden

Wäre es nicht prima, wenn Sie genau die gleichen Menügrafiken in Ihren Projekten verwenden könnten, die ein Benutzer auch in den normalen Android-Anwendungen findet? Kein Problem – Sie haben diese Grafiken bereits versteckt auf Ihrer Festplatte. In Ihrem Android SDK-Ordner befindet sich das Verzeichnis *platforms*, das die heruntergeladenen Plattformversionen enthält. Wählen Sie dort den Ordner mit der neuesten Version aus, und legen Sie eine Kopie der darin befindlichen *android.jar*-Datei an. Sie erhalten dann eine neue Datei im gleichen Verzeichnis (oder in einem Verzeichnis Ihrer Wahl), bei der Sie die Dateiendung von *.jar* in *.zip* umbenennen. Eine JAR-Datei ist nämlich nichts anderes als ein ZIP-Archiv, nur unter einem anderen Namen.

Bild 3.73: Entpacken Sie die ZIP-Datei, und navigieren Sie in das Verzeichnis *res*, in dem Sie im Ordner *drawable-hdpi* alle vom System verwendeten Grafiken und Symbole im PNG-Format finden.

3.4.6 Native Android-Systemfunktionen integrieren

In den folgenden Abschnitten gehen wir auf die verschiedenen nativen Android-Funktionen ein, die über PhoneGap und JavaScript von Ihrem Webcode aus ansteuerbar sind. Wir werden uns dabei mit den optischen und funktionalen Unterschieden zwischen Android und iOS beschäftigen, da sich viele der APIs nicht nur ähneln, sondern teilweise identisch aufgerufen werden und funktionieren wie unter PhoneGap für iOS. Falls Sie gleich hierher gesprungen sind und nicht das iOS-Kapitel zu diesem Thema gelesen haben, empfehlen wir Ihnen, es als Vorbereitung zu überfliegen.

Systemmeldungen, Vibration und Klingeltonalarm

Die Systemmeldungsbox haben wir bereits im Menübeispiel kennengelernt, wobei wir sie nur mit dem simplen JavaScript-Aufruf `alert();` aufgerufen haben. Selbst wenn wir die PhoneGap-API mit folgendem Beispielcode nehmen, erhalten wir nur das unten angezeigte Ergebnis:

```
navigator.notification.alert('Jemand hat den Menü-Button
angetippt!','Titel','Button')
```

Bild 3.74: Die Werte für den Titel oder die Buttons werden ignoriert.

Selbst auf einem in deutscher Sprache eingestellten Android-Telefon wird der Titel immer *Alert* und die beiden Auswahlbuttons *OK* und *Cancel* benannt.

Bessere Nachrichten gibt es zum Vibrationsalarm: Im Gegensatz zum iPhone können Sie hier die Länge der Vibration in Millisekunden angeben:

```
navigator.notification.vibrate(5000);
```

Dieser API-Aufruf lässt unser Android-Handy insgesamt fünf Sekunden vibrieren.

Während Sie unter iOS eine Tondatei mit in das *www*-Verzeichnis kopieren mussten, damit ein Ton für den Klingeltonalarm gespielt wurde, ist das unter PhoneGap für Android nicht notwendig. Hier wird automatisch der Klingelton ausgewählt, den der Benutzer in den Geräteeinstellungen unter *Töne* als *Benachrichtigungston* eingestellt hat. Der Aufruf im Code könnte kaum einfacher sein:

```
navigator.notification.beep();
```

Fotos von der Kamera einbinden

Auch unter Android können Sie auf die Kamera zugreifen, Bilder direkt in Ihre Anwendung laden und dort weiterverwenden. Was bislang nicht möglich ist, ist der Zugriff auf die Fotobibliothek. Benutzer können somit nur Bilder innerhalb Ihrer Android App aufnehmen und keine bereits aufgenommenen Fotos verwenden.

Bild 3.75: Von unserer Beispielanwendung aus können wir wieder die native Android-Kamera ansteuern. Auf die Elemente in der Kameraanwendung haben wir keinen Einfluss.

Bild 3.76: Sobald das Bild aufgenommen ist, wird es wieder in einer Variablen gespeichert, die wir in unserem Beispiel sichtbar machen.

3.4.7 Android und Kameraanwendungen

Bei der Betrachtung der Kamera-APIs unter Android müssen Sie beachten, dass jeder Gerätehersteller seine eigene Kameraanwendung schreiben kann. Unter iOS gibt es eine Kameraanwendung, nämlich die von Apple. In der Android-Welt wollen sich die Hersteller mit ihren eigenen Versionen von Android oft voneinander unterscheiden. Sehr häufig wird die Kameraanwendung deshalb von verschiedenen Herstellern geändert oder neu geschrieben.

Beispielsweise hat ein Google Nexus One eine etwas andere Anwendung als ein HTC-Gerät mit der Sense-UI. Das resultiert daraus, dass die einzelnen Kameraanwendungen unterschiedlich stabil sind und unterschiedlich gut mit PhoneGap zusammenarbeiten. Auf den meisten von uns getesteten Telefonen hatten wir keine Probleme, wir wissen aber von anderen, dass die Kamera auf manchen Telefonen abstürzt oder keine Ergebnisse liefert.

Das sollte Sie nicht unbedingt davon abhalten, diese Funktion zu benutzen, da sie, wie gesagt, auf den meisten derzeitigen Android-Geräten funktioniert. Weil wir im Beispiel nicht mehr die Fotobibliothek berücksichtigen müssen, können wir unseren Beispielcode für Android wesentlich vereinfachen:

```
<html>
  <head>
    <meta name="viewport" content="width=default-width; user-scalable=no" />

    <title>Foto-Betrachter</title>
   <link rel="stylesheet" type="text/css" href="style.css">
   <script type="text/javascript" charset="utf-8"
src="phonegap.js"></script>
    <script type="text/javascript" charset="utf-8">

   function onBodyLoad() {
    }

    function holeFoto() {
```

```
    // Kamera bzw. Fotobibliothek-API-Aufruf
    navigator.camera.getPicture(onSuccess, onFail, { quality: 50 } );
    }

    // Wird nur aufgerufen, wenn ein Bild erfolgreich geholt wurde
    function onSuccess(imageData) {
    document.getElementById("bild").src = "" +
imageData;
    }

    // Wird im Fehlerfall aufgerufen
    function onFail(meldung) {
        navigator.notification.alert('Folgender Fehler aufgetreten: ' +
meldung);
    }

    </script>
  </head>
  <body onload="onBodyLoad()">

   <div id="topbar" class="transparent" >
    <div id="title">Foto-Betrachter</div>
   </div>

    <form name="sourcetyp">
      <fieldset>
      <ul class="pageitem">
       <li class="button">
        <input type="button" value="Foto mit Kamera aufnehmen"
onClick="holeFoto(); return false;">
       </li>
      </ul>
      </fieldset>
    </form>

     <div>
       <ul class="pageitem" style="background-color:#000000">
          <center><img style="width:200px;height:300px" id="bild" src=""
/></center>
       </ul>
     </div>

   </body>
</html>
```

Wie Sie sehen, haben wir jede Menge Code aus dem iPhone-Beispiel entfernt. Auch im API-Aufruf selbst können Sie auf den `source`-Parameter verzichten, da nur die direkte Ansprache der Kamera möglich ist. Der `quality`-Parameter ist auch unter Android gültig, allerdings haben wir diese Option direkt in den API-Aufruf eingebaut:

```
// Kamera bzw. Fotobibliothek-API-Aufruf
navigator.camera.getPicture(onSuccess, onFail, { quality: 50 } );
```

Geoposition

Die Geopositions-API entspricht auch unter Android dem W3C-Standard und wird mit den gleichen Parametern wie unter PhoneGap für iOS aufgerufen. Somit stehen unter Android die bereits bekannten drei Methoden zur Verfügung:

- `geolocation.getCurrentPosition`
- `geolocation.watchPosition`
- `geolocation.clearWatch`

Im Beispiel für iOS hatten wir die aktuelle Position mit fünf verschiedenen Variablen ausgewertet. Es scheint, dass `navigator.geolocation.getCurrentPosition` keine Ergebnisse für die Höhe liefert, sondern dass wir nur die aktuellen Positionskoordinaten erhalten. Die selbst definierte JavaScript-Funktion `onSuccess()` können wir demnach minimieren:

```
function onSuccess(position) {
var element = document.getElementById('geolocation');

element.innerHTML = 'Breitengrad: '  + position.coords.latitude + '<br>' +
'L&auml;ngengrad: ' + position.coords.longitude
}
```

Bewegungssensoren

Die virtuelle Wasserwaage können wir aus dem iOS-Beispiel nahezu ohne Codeänderungen übernehmen. PhoneGap für Android verwendet auch hier die gleichen Aufrufe. Allerdings sind die Daten, die in der Funktion `onSuccess()` angezeigt werden, anders als auf einem iPhone:

```
function onSuccess(acceleration) {
    var element = document.getElementById('accelerometer');
    element.innerHTML = 'Beschleunigung/Lage X: ' + acceleration.x + '<br />' +
     'Beschleunigung/Lage Y: ' + acceleration.y + '<br />' +
     'Beschleunigung/Lage Z: ' + acceleration.z + '<br />';
}
```

Um den gleichen Effekt wie unter iOS zu erzielen, müssen wir die `if`-Bedingung anpassen:

```
if ( acceleration.x <= 0.1 && acceleration.x >= -0.1 && acceleration.y <=
9.7 && acceleration.y >= 9.5 ) { navigator.notification.beep(); }
```

3.4.8 Zusammenfassung

Insgesamt kann man sagen, dass PhoneGap für Android sicher hier und da noch Nachholbedarf gegenüber der iOS-Variante besitzt. Dies bekommt man vor allem zu spüren, wenn man in seiner Anwendung auf native UI-Elemente wie beispielsweise Menüs oder Pop-up-Menüs zurückgreifen möchte. Hier müssen Sie heute noch mit viel Kreativität herangehen. Zahlreiche Frameworks wie jQTouch, Sencha Touch (*http://www.sencha.com/products/touch/*) und vor allen Dingen auch jQuery Mobile (*http://jquerymobile.com*) haben bereits vernünftige Anwendungen erstellen lassen.

4 Webanwendungen und native Apps veröffentlichen

Wenn Sie diese Zeilen lesen, haben Sie tatsächlich bereits Ihre erste mobile HTML5-Webanwendung oder native Anwendung erstellt. Herzlichen Glückwunsch! Jetzt wollen wir Ihnen natürlich auch zeigen, wie Sie Ihre Arbeit veröffentlichen können. Dabei liegt ein großes Augenmerk natürlich auf der Veröffentlichung von nativen Anwendungen im iTunes App Store und im Android Market, doch auch für reine Webanwendungen gibt es Distributionsmöglichkeiten, die wir Ihnen jetzt zeigen werden.

4.1 Apple Web Apps-Katalog

Wenn Sie eine Webanwendung erstellt haben, die auf einem iPod touch und auf einem iPhone läuft und gut aussieht, lassen Sie sie doch von Apple bewerben. In den meisten Märkten außerhalb der USA ist das relativ unbekannt, doch Apple besitzt auf seiner globalen Webseite *apple.com* einen Katalog mit mehreren Tausend HTML5-Webanwendungen. Sie erreichen diesen Katalog unter *http://www.apple.com/webapps/* auf Ihrem Desktop-Browser.

Dieser Katalog ähnelt dem Downloadkatalog für Mac-Software auf der Apple-Seite und unterteilt die vielen Web Apps in verschiedene Kategorien. Entstanden ist der Katalog 2007, als das erste iPhone veröffentlicht wurde und es noch keine Möglichkeit für Entwickler gab, Anwendungen für das Telefon zu schreiben. Die einzige Möglichkeit für Entwickler, anwendungsähnliche Inhalte für das iPhone aufzubereiten, waren Web Apps. Zwar ist der Katalog dadurch schon etwas älter, allerdings finden Sie nahezu jeden Tag viele neue Einträge.

Im Apple Web Apps-Katalog ist keine Shoppingfunktionalität enthalten. Das wäre auch wenig sinnvoll, da sich Ihre Web App öffentlich im Internet befindet und sie theoretisch jeder aufrufen kann. Den Katalog gibt es auch nur in englischer Sprache. Um im Web Apps-Katalog gelistet zu werden, müssen Sie als Apple Developer registriert sein. Sie benötigen nicht die kostenpflichtige Mitgliedschaft im iPhone Developer-Programm, die kostenlose Anmeldung als Entwickler ist ausreichend.

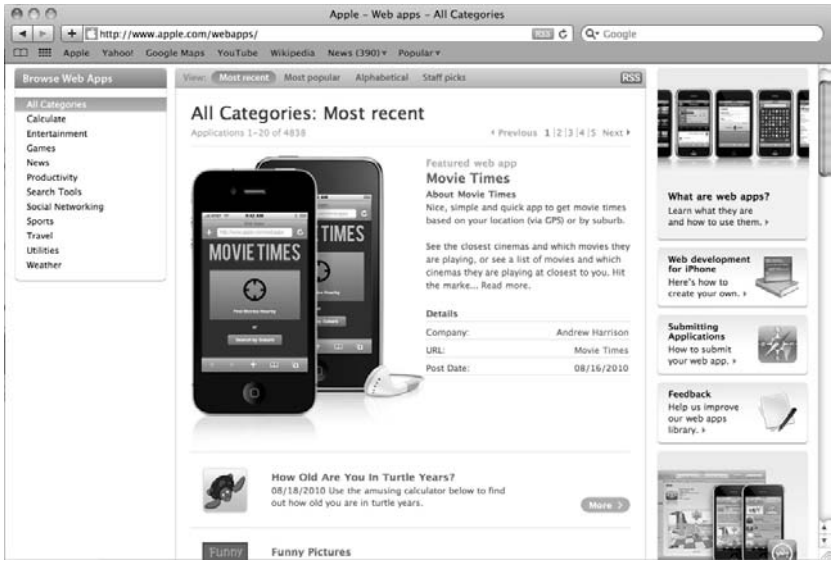


Bild 4.1: Der Apple Web Apps-Katalog.

Der Aufnahmeprozess wird über die Seite *http://developer.apple.com/submission/iphone* durchgeführt. Sie müssen sich als Erstes mit Ihrer Apple ID anmelden und anschließend dem *iPhone Web Application Submission Agreement* – also der Vereinbarung zur Einreichung einer iPhone-Webanwendung – zustimmen. Lesen Sie sich die Vereinbarung in Ruhe ganz durch, dennoch möchten wird gleich hier auf ein paar wichtige Punkte hinweisen:

- Ihre Webanwendung muss sich als iPhone-optimierte Webanwendung anfühlen und auch so funktionieren. Das meint Apple mit der ersten Bedingung, die besagt, dass eine eingereichte Webanwendung den iPhone-Webentwicklungsrichtlinien entsprechen muss. Ihre Chancen, in den Katalog aufgenommen zu werden, stehen sehr schlecht, wenn Sie eine einfache Webseite einreichen, die nicht iOS-optimiert ist. Mit den im Buch vorgestellten Techniken und insbesondere den von uns verwendeten CSS- und JavaScript-Frameworks wie iWebKit 5 und jQTouch sind Sie auf der sicheren Seite.
- Ihre Webanwendung sollte zum Einreichungszeitpunkt fertig programmiert und getestet sein. Kaputte Links, nicht funktionierende Buttons oder fehlerhafte Grafiken sollten nicht mehr Bestandteil Ihrer Web App sein.
- Auch wenn die Webanwendung in den Apple Web Apps-Katalog aufgenommen wird, sind Sie weiterhin für das Betreiben und das Hosten der Webseite verantwortlich. Apple wird nur aus dem Katalog auf Ihre Webseite verlinken.

Sind Sie mit der Vereinbarung einverstanden und haben Sie sie bestätigt, kommen Sie zu der Seite, in der Sie diverse Angaben zu Ihrer Webanwendung machen können. Diese Angaben beinhalten unter anderem die Links zu Ihrer mobilen Web App (*Application Page*), den Namen (*Application Title*), eine Kurzbeschreibung in Englisch (*Product Summary*) und eine ausführlichere Produktbeschreibung (*Product Description*). Die Wahl der richtigen Kategorie ist wichtig, damit interessierte Katalogbesucher sie später

leichter finden. Die Angabe der Lizenzform deutet darauf hin, dass Apple einfach die Seite zum Einstellen von Mac OS X-Anwendungen kopiert hat. Zum Abschluss müssen Sie noch ein Icon in der Größe 128 x 128 und einen Screenshot hochladen.

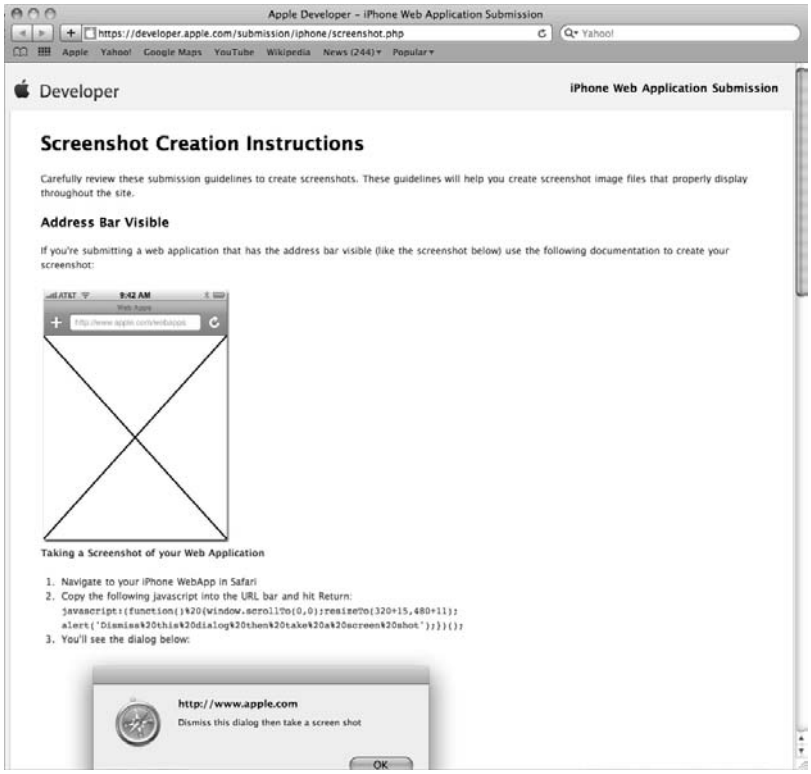


Bild 4.2: Gleichheit hat ihren Preis: Um in den Web-App-Katalog aufgenommen zu werden, müssen klare Regeln bei der Erstellung eines Screenshots befolgt werden.

Sie können leider nicht einfach einen Screenshot von Ihrem iPhone oder vom Simulator machen, da Apple eine Screenshot-Vorlage erstellt hat, die Sie verwenden müssen. Diese Vorlage können Sie von der Seite *http://developer.apple.com/submission/iphone/screenshot.php* als TIFF-Datei herunterladen und in einem Bildbearbeitungsprogramm mit Ihrem erstellten Screenshot verbinden. Somit wird sichergestellt, dass der Web-App-Katalog aufgeräumt aussieht und alle Screenshots im einheitlichen Design dargestellt werden.

Haben Sie alles erledigt, können Sie über *Submit* Ihre Webanwendung an Apple übermitteln.

4.1.1 Andere Vertriebsmöglichkeiten für Web Apps

Mobile Webanwendungen sind technisch nichts anderes als Webseiten, und daher empfiehlt sich auch die Beantragung zur Aufnahme in die großen Suchmaschinen dieser Welt. Falls Sie bereits eine Webseite betreiben und eine Web App als Ergänzung für Ihre

iPhone- und Android-Besucher erstellen, ist der effektivste Weg, Benutzer für die mobile Seite zu finden, eine Umleitung auf Ihrer Webseite einzubauen. Somit erhält jeder, der Ihre normale Webseite mit einem geeigneten mobilen Gerät ansurft, automatisch die optimierte Webanwendung. Solche Umleitungen sind schnell eingerichtet und wurden am Anfang des Buchs beschrieben.

4.2 iTunes App Store

Wenn Sie mit PhoneGap Ihre iPhone-App fertiggestellt haben, wird es Zeit, sie im iTunes App Store zu veröffentlichen und gegebenenfalls dort auch die Anwendung zu verkaufen. Dazu müssen Sie Mitglied im iPhone Developer-Programm sein. Die Mitgliedschaft kostet für Einzelpersonen, die ihre iPhone-App unter ihrem eigenen, persönlichen Namen vertreiben möchten, 99 US-Dollar und für Firmen 299 US-Dollar. Nähere Informationen dazu finden Sie im Abschnitt 3.2.1 »Entwicklungsumgebung für iOS-Entwicklung einrichten«.

Die App mit Xcode für den App Store vorbereiten

Bislang haben Sie Ihre Apps, die wir mit Xcode erstellt und auf den Geräten getestet haben, nur für den eigenen Entwicklungsbedarf kreiert. Die bislang erstellten Dateien können Sie nicht zum Vertrieb über den iTunes App Store verwenden, da sie nur mit einem Development Profile (Entwicklungsprofil) versehen wurden und nicht mit einem Distribution Profile.

1. Um ein Distribution Profile zu erhalten, benötigen Sie ein Vertriebszertifikat. Sparen Sie sich aber die umständliche Erstellung über die Schlüsselbundverwaltung. Gehen Sie in Xcode in das Menü *Window* und rufen Sie dort den *Organizer* auf. Klicken Sie hier auf die bereits bekannte Ansicht *Provisioning Profiles*.

Bild 4.3: Distribution Profile im *Organizer* erstellen.

2. Stellen Sie sicher, dass *Automatic Device Provisioning* markiert ist, und klicken Sie dann auf *Refresh*. Sie werden danach aufgefordert, Ihre Apple ID und Ihr Passwort direkt in Xcode einzugeben. Ist das geschehen und haben Sie *Submit Request* angeklickt, wird Ihnen kurz gemeldet, dass Ihre Anfrage begutachtet wird. In der Regel dauert dieser Prozess nur wenige Minuten, wenn nicht gar Sekunden.
3. Jetzt geht es wieder zurück in den Browser: Ein Distribution Profile erhalten Sie auch über das iPhone Provisioning Portal – *http://developer.apple.com/iphone/manage/overview* – mit einem Klick auf *Provisioning* in der linken Navigationsspalte. Dort sehen Sie unter *Development* Ihre Entwicklungsprofile aufgelistet. Navigieren Sie dann zu *Distribution* und erstellen Sie dort über *New Profile* ein neues Vertriebsprofil.
4. Im darauffolgenden Bildschirm wählen Sie dann die Option *App Store* als Vertriebsmethode aus und geben dem Profil einen verständlichen Namen.

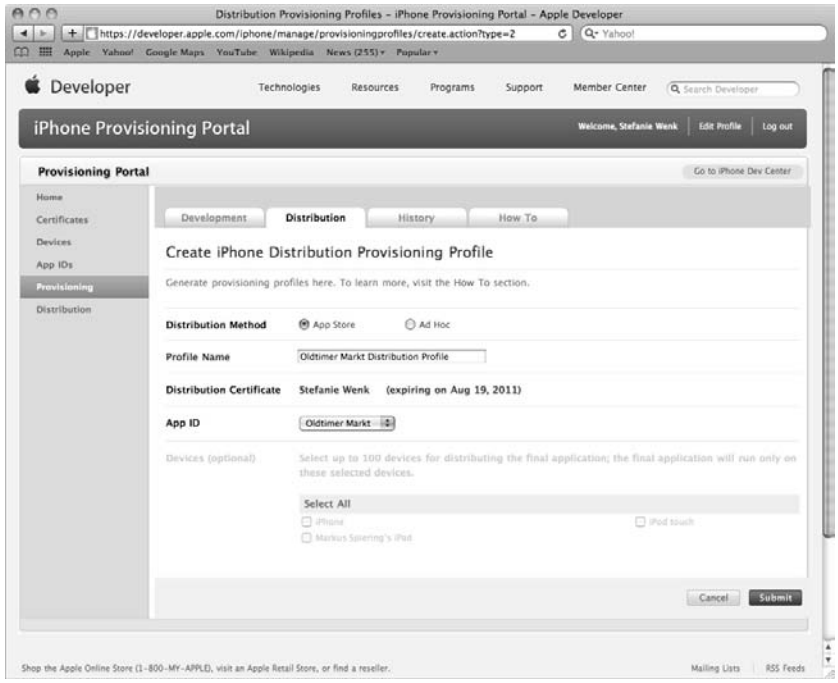


Bild 4.4: Festlegen der Distributionsmethode und Angabe eines Profilnamens.

Wozu ist das Ad-hoc-Zertfikat gut?
Für den Vertrieb im App Store benötigen Sie ein App Store Distribution-Profil. Wir empfehlen jedoch, vorher Ihre Anwendung gegen ein »Ad-hoc-Profil« zu testen. Ein Ad-hoc-Profil erlaubt Ihnen, Ihre App noch einmal auf diversen physischen Geräten als Release-Kandidat zu testen und sicherzustellen, dass darin keine Fehler enthalten sind. Sie können dabei sogar fremde Geräte von Betatestern provisionieren lassen, nachdem diese im Provisioning-Portal unter *Devices* eingetragen wurden und Sie den Betatestern nicht nur die kompilierte App-Datei, sondern auch das generierte Ad-hoc-Profil geschickt haben.

5. Unter *Distribution Certificate* sollte jetzt das von Xcode aus beantragte Zertifikat stehen, das eine Gültigkeitsdauer von einem Jahr besitzt. Falls Sie dort kein Zertifikat sehen, können Sie den Status Ihrer Zertifikatsanfrage durch einen Klick auf *Certificates* in der linken Spalte und einen anschließenden Klick auf *Distribution* überprüfen. Sehen Sie auf der Zertifikatsseite die Meldung *Pending*, wird Ihre Anfrage noch bearbeitet. Sehen Sie dort kein Zertifikat, befolgen Sie bitte die Schritte, die im Abschnitt zum Erstellen eines iPhone Distribution Profile aufgeführt sind.

6. Haben Sie alle Angaben gemacht und auf *Submit* geklickt, werden Sie wieder ein paar Minuten warten müssen, da Ihr Distributionsprofil nicht sofort ausgestellt wird. Laden Sie nach ein paar Augenblicken die Seite neu: Ein *Download*-Button sollte dann neben dem Distribution Provisioning Profile erschienen sein:

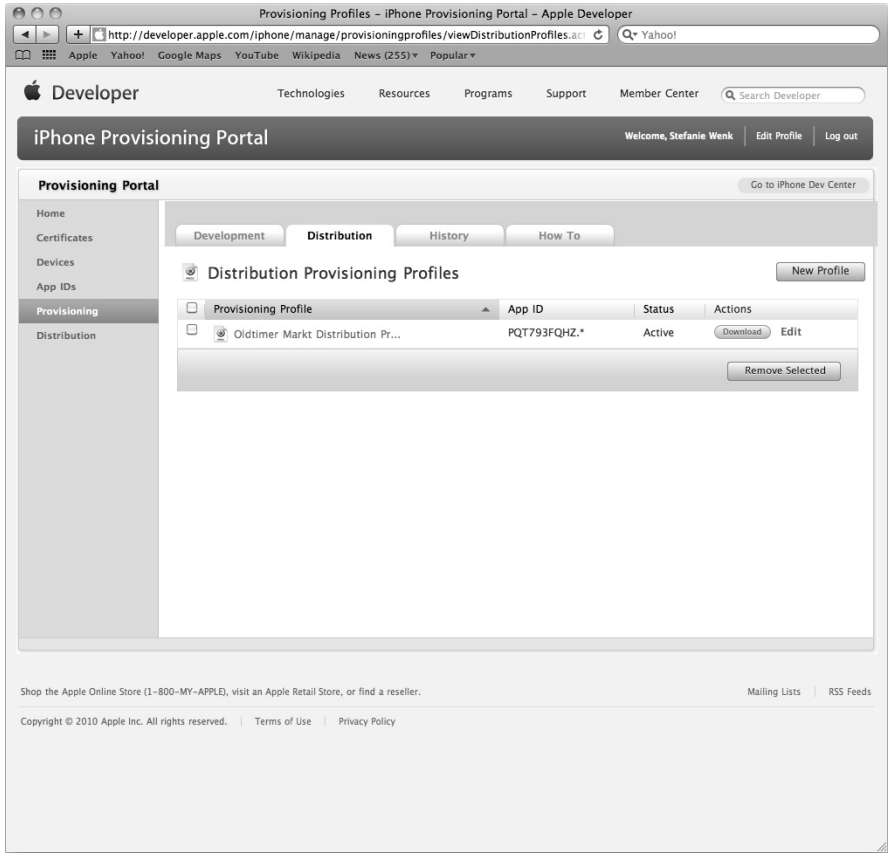


Bild 4.5: Status *Active*: Der *Download*-Button erscheint neben der App.

7. Speichern Sie die Datei mit der Endung *.mobileprovision* auf Ihrem Mac und ziehen Sie das Dateisymbol über das Symbol von Xcode. Jetzt wird wieder das *Organizer*-Fenster in der *Provisioning Profiles*-Ansicht geöffnet, und Sie sollten hier nun die gerade erstellte und heruntergeladene Provisionsdatei aufgelistet sehen.

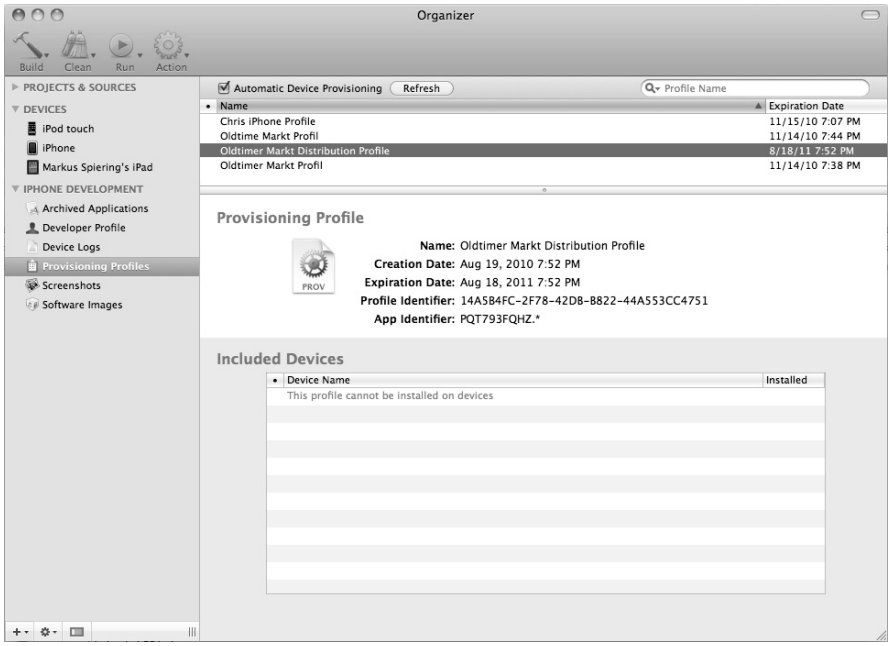


Bild 4.6: Anzeige der eben erstellten Provisionsdatei.

Da sich ab sofort das Profil in Xcode befindet, können Sie nun damit beginnen, die für den Vertrieb gültige Version Ihrer App zu bauen. Vorher sind aber noch ein paar Einstellungen nötig.

8. Um zu den Projekteinstellungen zu gelangen, wählen Sie aus dem Menü *Project* den Unterpunkt *Project Settings*. In dem darauf erscheinenden Fenster klicken Sie in den *Build*-Bereich und wählen aus dem *Configuration*-Drop-down-Menü die Option *Release* aus. Bislang haben Sie im *Debug*-Modus zum Programmieren gearbeitet – jetzt schalten Sie in den *Release*-Modus um.

   Suchen Sie nun in der *Settings*-Liste den Bereich *Code Signing* und wählen Sie dort unter *Code Signing Idendity* Ihr vorhin erstelltes Distributionsprofil aus:

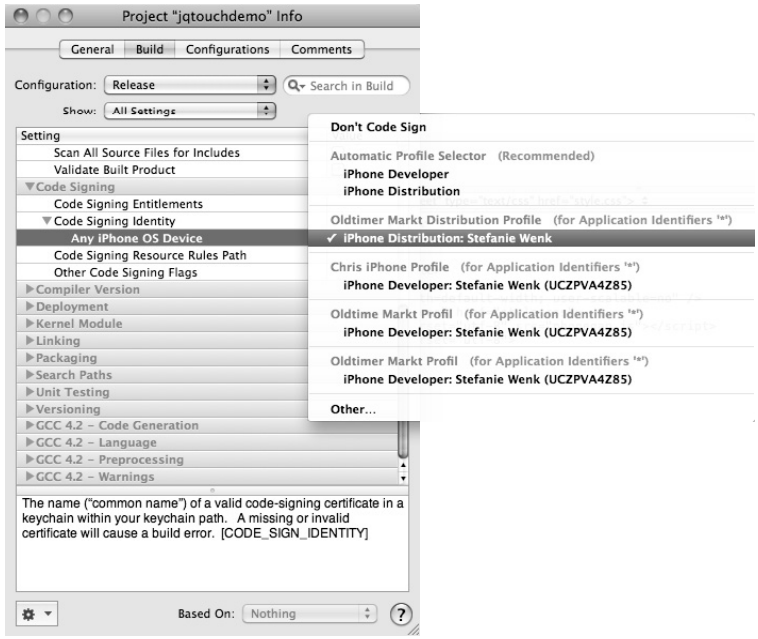


Bild 4.7: Erstelltes Distributionsprofil auswählen.

9. Als nächsten Schritt müssen Sie in der Xcode-Hauptansicht in der Spalte *Groups & Files* den Namen Ihrer App unter *Targets* markieren, über das Kontextmenü den Punkt *Get Info* aufrufen und im erscheinenden Target-Infofenster auf *Properties* klicken.

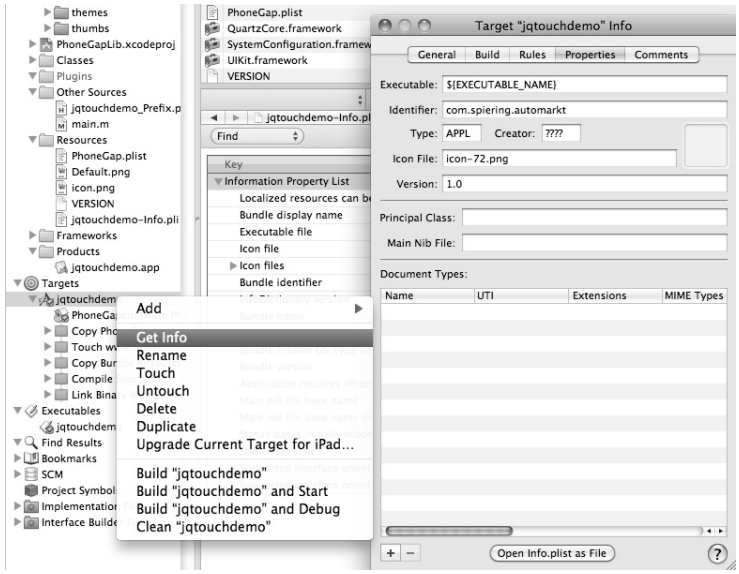


Bild 4.8: Aufruf der *Properties*.

10. Im Provisioning-Portal haben Sie theoretisch zwei Möglichkeiten, eine App ID zu definieren, einmal über den von uns benutzten Assistenten und zum anderen über eine spezielle Unterseite, die durch die Navigationsleiste im Provisioning-Portal über *App IDs* zu erreichen ist. Der Assistent generiert sogenannte Wildcard App IDs, bei denen Sie keine Möglichkeit haben, einen *Bundle Identifier* mit anzugeben. In unserem Beispiel hatten wir von Apple eine ID in folgendem Format bekommen: *PQT793FQHZ.**.

    Hätten wir auf der Provisioning-Portal-Seite für eine App ID einen *Bundle Identifier* definiert, müssten wir diesen jetzt im Target-Infofenster unter *Identifier* eingeben. Bei einer Wildcard Apple ID wie unserer haben wir jetzt die Möglichkeit, diesen Identifier zu wählen, der in folgendem Format angegeben wird:

    ```
    com.firmenname.appname
    ```

    Für das Beispiel verwenden wir:

    ```
    com.spiering.automarkt
    ```

11. Schließen Sie das Einstellungsfenster und geben Sie Xcode ein paar Sekunden Zeit, bis auch die *Info.plist*-Datei mit dieser neuen ID aktualisiert wurde.

12. Zurück in der Xcode-Hauptansicht, erstellen Sie jetzt Ihre finale Version. Dazu wählen Sie aus dem linken oberen Menü *Device* und *Release* aus. Bevor Sie zum vielleicht letzten Mal auf *Build and Run* klicken, empfiehlt es sich, das Projekt noch einmal über *Clean All Targets* zu säubern. Ab dann gilt es: Klicken Sie auf *Build and Run* und bestätigen Sie den Zugriff auf die Schlüsselbundverwaltung. Anschließend finden Sie im Ordner *Products* unter *Groups & Files* auf der linken Seite des Xcode-Fensters eine frische, neue Datei mit der Dateiendung *.app*. Das ist Ihre Vertriebsdatei, die Sie jetzt innerhalb des Finders noch als ZIP-Datei komprimieren sollten.

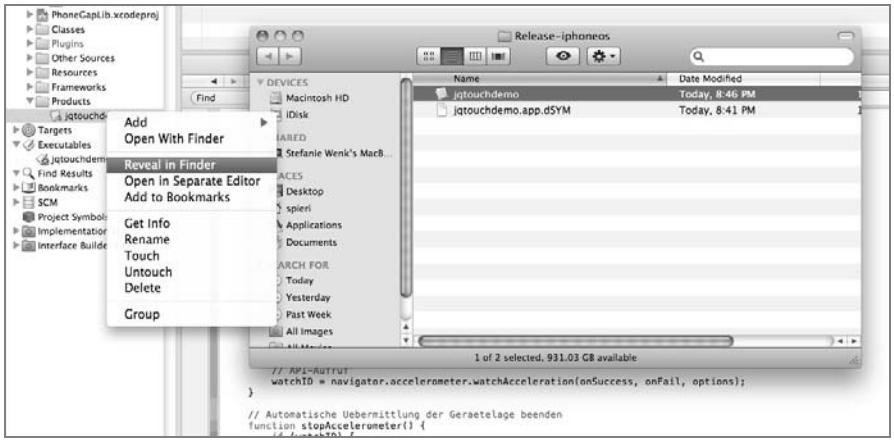


Bild 4.9: Die fertige Vertriebsdatei.

Die fertige ZIP-Datei werden wir in wenigen Augenblicken in den iTunes App Store laden.

iTunes Connect

Das eigentliche Veröffentlichen Ihrer App erfolgt nicht über das iPhone DevCenter, sondern über die Webseite iTunes Connect: *http://itunesconnect.apple.com.*

1. Auch hier identifizieren Sie sich über die Apple ID, die Sie auch für den Zugang zu den Apple Developer-Diensten benutzen. Stimmen Sie beim ersten Aufruf der Seite den iTunes Connect-Geschäftsbedingungen zu.

Bild 4.10: Aufruf der Website *iTunes Connect*.

iTunes Connect ist die zentrale Seite, wenn es um das Verwalten und Verkaufen Ihrer Apps geht. Hier können Sie neue Apps zum Vertrieb bereitstellen, die Beschreibungen und Screenshots für den App Store ändern, monatliche Statistiken der Downloads und Verkäufe sehen oder weitere Benutzer anlegen, die ebenfalls Zugriff auf Ihre iTunes Connect-Seite erhalten.

2. Um eine neue App in den Store zu stellen, klicken Sie von der Startseite aus auf *Manage Your Applications.* Danach sehen Sie eine Übersicht all Ihrer Apps, die entweder in das System gestellt wurden, noch auf die Apple-Freigabe warten oder bereits im App Store zum Verkauf bzw. zur Installation verfügbar sind. Da Sie noch keine Apps in Apples »Laden« haben, ist diese Ansicht für Sie bisher leer. Für die weitere Vorbereitung empfehlen wir Ihnen, die folgenden Beschreibungen und Dateien zur Hand zu haben:

- Eine ausführliche Beschreibung Ihrer App, die im App Store angezeigt wird. Verwenden Sie kurze, knackige Sätze am Anfang und versuchen Sie, die wichtigsten Punkte aufzuzählen. Sie haben dazu maximal 4.000 Zeichen.
- Ein paar Schlüsselwörter, die Ihre App treffend beschreiben, aber zusammen mit Kommata nicht mehr als 100 Zeichen lang sein dürfen. Die Schlüsselwörter werden bei der App Store-Suche benutzt. Wählen Sie daher diese Schlüsselwörter sorgfältig aus, sie könnten darüber entscheiden, ob Ihre App im Store leicht oder schwer gefunden wird.
- Das Programmsymbol Ihrer Anwendung – allerdings im Großformat von 512 x 512 Bildpunkten. Es wird im iTunes App Store auf dem Desktop innerhalb von iTunes und auf den von iTunes generierten Produktwebseiten verwendet.
- iPhone oder iPod touch-Screenshots von Ihrer App. Diese müssen im PNG-, JPEG- oder TIFF-Format vorliegen und dürfen folgende Abmessungen in Bildpunkten haben: 320 x 480, 480 x 320, 480 x 300, 640 x 960 oder 960 x 640.
- Falls Ihre App auch oder sogar nur auf dem iPad läuft, sollten bzw. müssen Sie auch dafür passende Screenshots Ihrer App erstellen. Es werden dabei die gleichen Formate wie für das iPhone akzeptiert, allerdings müssen die Screenshots folgende Größen haben: 768 x 1.024, 1.024 x 768, 748 x 1024 oder 1.004 x 768.

3. Zurück in iTunes Connect, werden Sie beim Anlegen einer neuen Anwendung die primäre Sprache Ihrer Anwendung angeben müssen. Anschließend gelangen Sie zu einem Bildschirm, in dem Sie den Namen Ihrer Anwendung noch einmal angeben müssen. Im zweiten Feld dieser Seite können Sie eine ID eingeben, die einzigartig für Ihre Anwendung ist und auch nicht mehr geändert werden kann. Diese ID ist nicht für den Benutzer sichtbar. Wir verwenden für unser Beispiel einfach *Oldtimer-Markt-001*, da das die erste Version unserer Oldtimer-App ist.

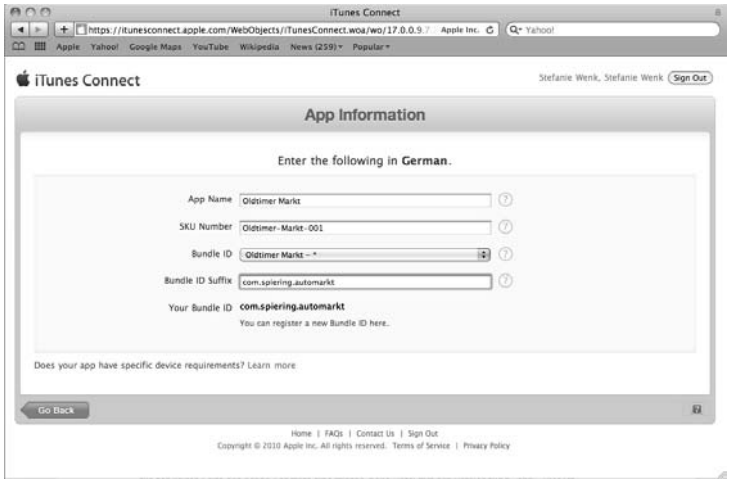


Bild 4.11: Eingabe der *App Information*.

Im nächsten Feld wählen Sie Ihre Wildcard App ID aus und geben im Feld *Bundle ID Suffix* den String ein, den Sie vor dem finalen Erstellen der App definiert haben:

```
com.spiering.automarkt
```

4. Im nächsten Fenster legen Sie fest, ab wann Ihre App im App Store verfügbar sein soll. Machen Sie sich hier keine Illusionen: Auch wenn Sie den aktuellen Tag angeben, wird die App mit Sicherheit nicht binnen Minuten erscheinen. Apple muss die App immer noch überprüfen und dann gegebenenfalls freischalten.

   Falls Sie aber die App an einem bestimmten Tag veröffentlichen wollen, der noch in etwas fernerer Zukunft liegt, können Sie diese Angabe hier vornehmen. Wenn Sie die App so schnell wie möglich im App Store sehen möchten, nehmen Sie einfach das aktuelle Datum.

5. Da Sie Ihre App nicht verkaufen wollen, wählen Sie unter *Price Tier* (Preisstufe) den Punkt *Free*. Sollten Sie planen, für Ihre App Geld zu verlangen, müssen Sie hier die Preisstufe auswählen. Ihnen stehen dafür 85 Stufen zur Verfügung: *Tier 1* bezeichnet die kostengünstigen Apps, die es für 79 Euro-Cent (Ihr Gewinn ist dann 48 Euro-Cent) oder 99 US-Cent zu kaufen gibt. *Tier 85* ist demnach die Preisstufe für die teuersten Apps. Der Verkaufspreis würde im App Store dann bei 799 Euro bzw. 999 US-Dollar liegen. Sobald Sie auf der Seite eine Preisstufe auswählen, können Sie über *See Pricing Matrix* die einzelnen Preisstufen und deren Verkaufspreise und Gewinnausschüttung pro Währungsbereich sehen.

   Falls Sie Ihre App nicht in allen Ländern vertreiben können oder wollen, lässt sich auf dieser Seite der Vertrieb auch auf einzelne Länder beschränken.

6. Auf der nächsten Seite geben Sie neben der Versionsnummer auch die maximal 4.000 Zeichen lange Beschreibung, einige Schlüsselwörter für die App Store-Suche und noch ein paar weitere Informationen an. Weiter unten auf der Seite legen Sie fest, ob Ihre App bestimmte Inhalte enthält oder bestimmter Natur ist. Dadurch wird entschieden, wie alt beispielsweise ein iTunes-Kunde sein muss, um die App herunterladen oder kaufen zu können. Am Ende der Seite finden Sie noch die Uploadlinks, um nicht nur das große 512-x-512-Bild für iTunes hochzuladen, sondern auch um die iPhone-, iPod- und iPad-Screenshots an Apple zu übermitteln.

7. Haben Sie auch diesen Bildschirm hinter sich gebracht, erhalten Sie noch einmal eine Übersicht und müssen dann – wieder einen Bildschirm weiter – angeben, ob Ihre App irgendeine Art von Verschlüsselungstechnologie (*Encryption*) enthält.

8. Jetzt kommen wir zum letzten Punkt: dem Hochladen der als ZIP verpackten APP-Datei. Dazu starten Sie das Programm App Loader. Es wurde bereits zusammen mit dem iOS SDK installiert und befindet sich unter */Developer/Applications/Utilities/Application Loader.app*.

   Nach dem Programmstart melden Sie sich wieder mit Ihrer Apple ID und Ihrem Passwort an und sehen kurze Zeit später in einem Auswahlmenü die gerade in iTunes Connect angelegte App:

Bild 4.12: Auswahl der hochzuladenden Applikation.

9. Anschließend bestätigen Sie, dass Sie Ihre App auch unter iOS 4 getestet haben und nun die ZIP-Datei auswählen und an Apple schicken. Der App Loader wird Ihnen sofortiges Feedback geben, falls etwas mit Ihrem App-Paket nicht stimmt. Anderenfalls wird die App zu Apple gesendet, und Sie erhalten eine Bestätigung.

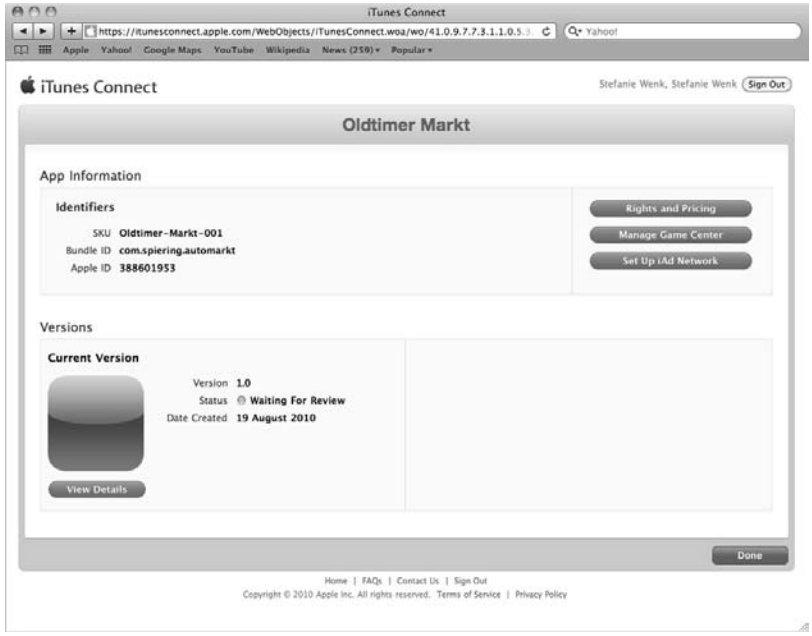


Bild 4.13: Die Bestätigung von Apple.

10. Auch in iTunes Connect können Sie unter *Manage Your Applications* sehen, dass Ihre App jetzt auf das finale Urteil von Apple wartet. In der Zwischenzeit können Sie für Ihre App über *View Details* und *Manage Localizations* wahlweise noch Beschreibungen in mehreren Sprachen hinzufügen, falls Ihre App mehrsprachig ist.

4.3 Android Market

Wenn Sie mit Ihrer Android-App glücklich und zufrieden sind, heißt es hier ebenfalls: raus damit in die Welt und in den Android Market. Im Android Market können sich Android-Benutzer wie im iTunes App Store Programme auf Ihr Gerät installieren und – in zahlreichen Ländern – auch käuflich erwerben.

Bis wir unsere Android-App aber in den Android Market hochladen können, müssen wir erst eine finale Vertriebsversion erstellen. Dazu verwenden wir natürlich Eclipse und bearbeiten die Einstellungen in der Datei *AndroidManifest.xml*.

1. Als Erstes sollten Sie im *Manifest*-Bereich überprüfen, ob Sie die richtige Versionsnummer und den richtigen Versionscode definiert haben. Diese Angaben sind im Android Market sichtbar und können am oberen Rand unter den *Manifest General Attributes* eingestellt werden.

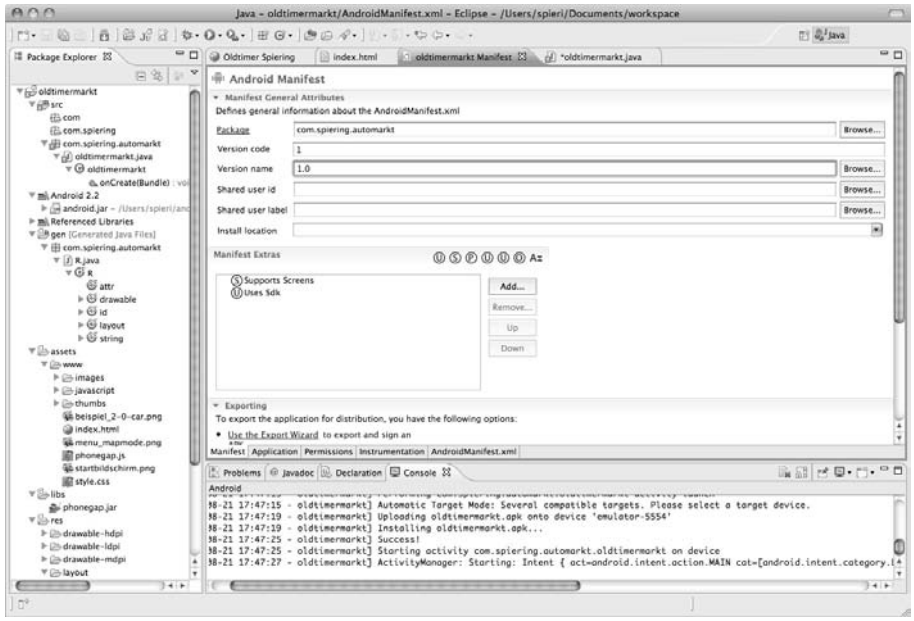


Bild 4.14: Versionscheck durchführen.

2. Als Nächstes navigieren Sie in den *Application*-Bereich. Dort finden Sie auf der rechten Seite die Einstellung, ob Ihre Anwendung *Debuggable* sein soll oder nicht. Momentan müsste das noch auf `true` gesetzt sein, ändern Sie diese Einstellung nun auf `false`. Der zur Fehlersuche und -analyse bestimmte Programmcode, der während des Programmierens nützlich ist, braucht sich in unserer fertigen App nicht zu befinden. Navigieren Sie jetzt zum nächsten Bereich *Permissions*.

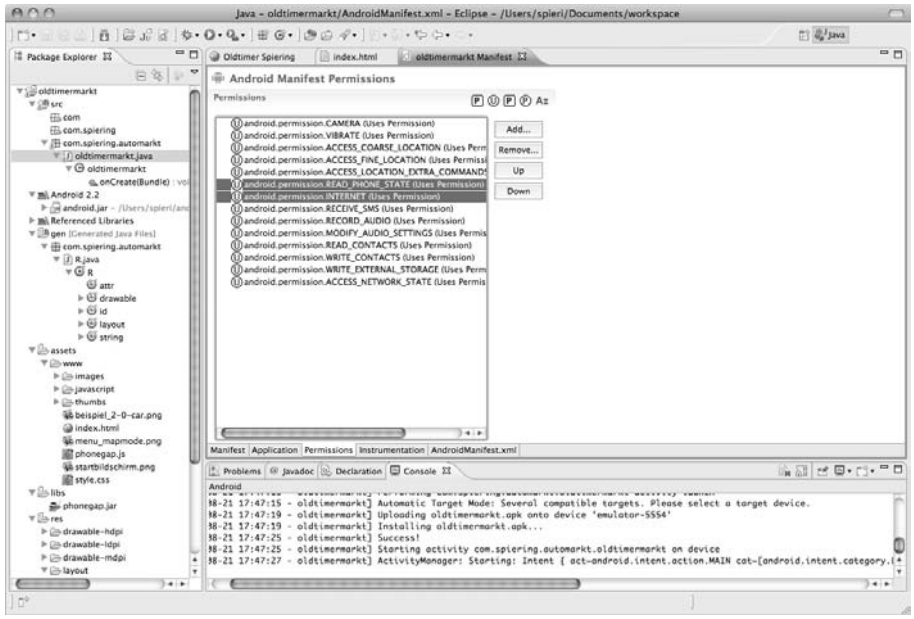


Bild 4.15: Prüfen, ob die Anwendung *Debuggable* sein soll.

Hier sehen Sie eine ganze Menge verschiedener Berechtigungen aufgelistet, die die Anwendung besitzen soll. Falls Sie keine der nativen Systemfunktionen per JavaScript in Ihrer Anwendung verwenden, können Sie alle Berechtigungen löschen bis auf:

```
READ_PHONE_STATE
INTERNET
```

Sprechen Sie die Kamera von Ihrer App aus an, benötigen Sie zusätzlich:

```
CAMERA
```

Falls Sie den Vibrationsalarm benutzen, benötigen Sie:

```
VIBRATE
```

Und wenn Sie Geopositionsfunktionen nutzen, brauchen Sie folgende Berechtigungen:

```
ACCESS_COARSE_LOCATION
ACCESS_FINE_LOCATION
ACCESS_LOCATION_EXTRA_COMMANDS
```

3. Haben Sie bereits selbst Anwendungen vom Android Market heruntergeladen und installiert, werden Sie den Dialogbildschirm kennen, in dem der Benutzer darauf hingewiesen wird, welche Systemfunktionen die Anwendung benutzt. Wenn Sie die überflüssigen Zugriffsberechtigungen hier nicht entfernen, wird der Benutzer vor der Installation Ihrer App an insgesamt 14 Punkten gewarnt. Diese kann er zwar alle auf einmal bestätigen, allerdings sieht es komisch aus, wenn der Benutzer den Kamerazugriff bestätigen muss, obwohl Ihre Anwendung möglicherweise die Kamerafunktion gar nicht eingebaut hat.

Sichern Sie die Datei *AndroidManifest.xml* und öffnen Sie die Datei *strings.xml* im Verzeichnis *res* unter *values*.

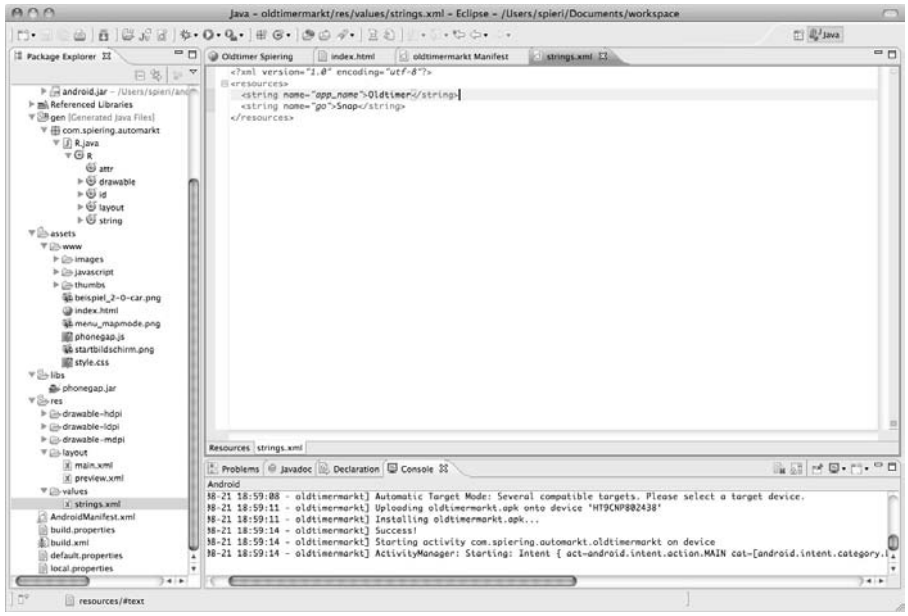


Bild 4.16: Sind Sie mit Ihrem App-Namen noch nicht zufrieden, können Sie ihn hier ändern.

4. Wir ändern den App-Namen von »oldtimermarkt« auf »Oldtimer«, da der alte Name selbst auf Geräten mit hochauflösenden Bildschirmen zu lang war und einen Zeilenumbruch verursacht hatte. Speichern Sie jetzt auch die Datei *strings.xml* ab.

5. Öffnen Sie nun die *AndroidManifest.xml*-Ansicht. Im *Manifest*-Bereich finden Sie relativ weit unten auf der Seite den Link *Use the Export Wizard* unter *Exporting*. Das ist der bequemste und schnellste Weg, eine Android-Anwendungsdatei für die Veröffentlichung zu generieren.

6. Ist der Assistent gestartet, wählen Sie das Projekt aus, das Sie signieren und exportieren möchten. Da es sich um unser aktuelles Android-Projekt handelt, können Sie das vorgeschlagene Projekt einfach mit Klick auf *Next* akzeptieren.

Warnung des Export-Assistenten
Der Export-Assistent teilt Ihnen an dieser Stelle auch mit, ob sich noch offensichtliche Fehler im Code befinden. Nehmen Sie die hier erscheinenden Warnungen ernst und beheben Sie die Fehler, bevor Sie mit dem Exportieren fortfahren.

7. Als Nächstes müssen Sie einen sogenannten *Keystore* anlegen. Hiermit generieren Sie eine Datei, in der ein Schlüssel gespeichert wird, dessen Passwort nur Sie kennen und mit dem die Anwendung signiert wird.

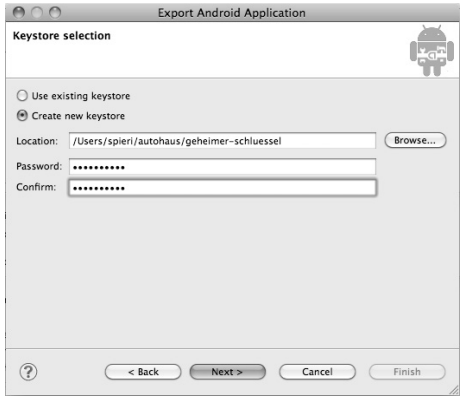


Bild 4.17: Einen *Keystore* anlegen.

8. Vergeben Sie einen frei gewählten Dateinamen und wählen Sie ein Verzeichnis aus, in das die Schlüsseldatei abgelegt wird. Suchen Sie sich dann ein Passwort aus und geben Sie es ein.
9. Jetzt geben Sie Ihrem Schlüssel einen Aliasnamen. Da Sie diesen Schlüssel für all Ihre Android-Anwendungen benutzen können, empfiehlt es sich, ihn unabhängig vom App-Namen zu wählen. Nach der Angabe des Passworts müssen Sie noch festlegen, wie lange der Schlüssel gültig sein soll (*Validity*). Die Angabe erfolgt in Jahren – und seien Sie dabei nicht zögerlich: Ein Zeitraum von 25 Jahren wird empfohlen.

Bild 4.18: Festlegen der Exporteinstellungen.

Wichtig!
Vergessen Sie auf keinen Fall die verwendeten Passwörter, da sonst keine Möglichkeit mehr besteht, die App im Android Market zu aktualisieren oder zu verändern.

10. Im letzten Schritt werden Sie aufgefordert, einen Speicherplatz für Ihre Android-Paketdatei (APK) zu definieren. Diese APK-Datei ist Ihre Android-Programmdatei, die Sie auch in den Android Market hochladen müssen:

Bild 4.19: Den Speicherplatz für die App festlegen.

Jetzt wird die APK-Datei mit einem Klick auf *Finish* in dem von Ihnen angegebenen Verzeichnis erstellt.

4.3.1 App an Betatester schicken oder außerhalb des Android Market vertreiben

Sie müssen Ihre App zum Vertrieb nicht unbedingt in den Android Market stellen. Suchen Sie viele Abnehmer oder Käufer für Ihre Software, ist sicher der Android Market die beste Möglichkeit, doch wenn Sie Ihre Software erst einmal von Betatestern gründlich untersuchen lassen wollen, ist auch das relativ einfach möglich.

Das Einzige, was Sie tun müssen, ist, Ihre signierte APK-Datei auf einen Webserver zu laden und Ihren Kunden oder Benutzern einen Downloadlink anzubieten. Alternativ können Sie die APK-Datei per E-Mail verschicken, dann müssten Ihre Benutzer allerdings die E-Mail direkt auf dem Gerät öffnen oder die APK-Datei von der E-Mail auf die SD-Karte des Android-Geräts schieben und von dort mit einem Installationsprogramm installieren. Ein empfehlenswertes und zudem noch kostenloses Installationsprogramm ist Apk Manager. Sie sehen – der Weg über einen Webserver-Download ist der einfachere.

Mit einer Hürde werden aber alle zukünftigen Benutzer Ihres Programms zu kämpfen haben, sofern sie die App nicht aus dem Android Market beziehen: Wenn die Benutzer ihre Einstellungen nicht schon vorher entsprechend gesetzt haben, werden sie aufgefordert, folgende Einstellungen vorzunehmen:

Der Benutzer muss im Einstellungsprogramm unter dem Punkt *Anwendungen* die Installation von Apps mit *Unbekannter Herkunft* zulassen. Glücklicherweise braucht der Benutzer nicht erst das Einstellungsprogramm selbst aufzurufen, stattdessen wird er direkt vom Installationsdialog dorthin geführt.

Eine hoffentlich nicht zu abschreckende Meldung: Selbst wenn Sie wollten, hätten Sie große Schwierigkeiten, eine PhoneGap Android-App wirklich »gefährlich« für das Telefon zu gestalten.

Bild 4.20: Apps unbekannter Herkunft zulassen.

4.3.2 App im Android Market veröffentlichen

Während Sie das Android SDK und die benötigten Entwicklungsprogramme ohne Registrierung und Kosten herunterladen konnten, müssen Sie sich, um eine App im Android Market zu veröffentlichen, als Android-Entwickler registrieren und eine kleine Aufnahmegebühr von 25 US-Dollar bezahlen.

1. Die Anmeldung erfolgt über die Seite *http://market.android.com/publish/signup*, auf der Sie sich mit Ihrem Google-Account anmelden. Anschließend werden Sie aufgefordert, ein paar Daten von sich zu hinterlegen. Wichtig ist hier das Feld *Developer Name*. In diesem Feld legen Sie fest, unter welchen Namen Ihre Apps im Android Market verkauft werden. Tragen Sie entweder Ihren echten Namen oder Ihren Firmennamen ein.

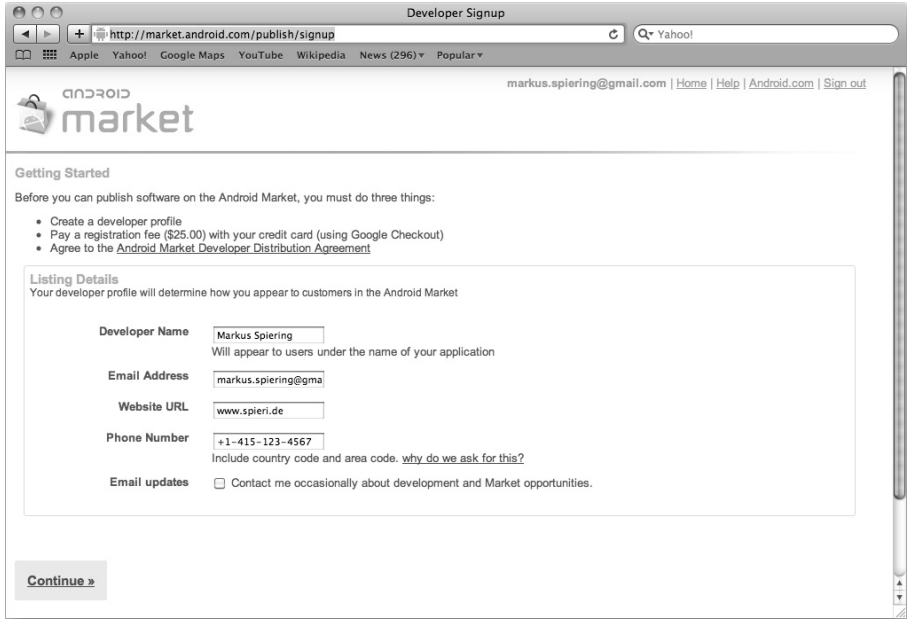


Bild 4.21: Anmeldung im Android Market.

2. Im nächsten Schritt müssen Sie dem Android Market Developer Distribution Agreement – also der Vereinbarung zum Vertrieb – zustimmen und den Betrag von 25 US-Dollar über Google Checkout entrichten. Bei Google Checkout können Sie mit allen gängigen Kreditkarten bezahlen.
3. Haben Sie das erfolgreich hinter sich gebracht, befinden Sie sich jetzt auf Ihrer Android Market-Seite. Von hier aus können Sie über *Upload Application* Ihre erste Android App veröffentlichen.

4.3.3 App im Android Market verkaufen

Wie bei iTunes können Sie Ihre Software im Android Market nicht nur zum Download anbieten, sondern auch verkaufen. Auch Google setzt eine Transaktionsgebühr von 30 % vom Verkaufspreis für einen erfolgreichen Verkauf an. Sie erhalten pro Verkauf also 70 % des Verkaufspreises.

1. Damit das Geld auch bei Ihnen ankommt, müssen Sie von Ihrer Android Market-Seite aus *Setup Merchant Account* auswählen und, je nachdem, in welchem Land Sie sich befinden, bestimmte Bank- oder Steuerinformationen angeben. Es sei darauf hingewiesen, dass der Android Market zwar in fast jedem Land verfügbar ist, jedoch der Verkauf von Software bislang nur in einigen Ländern ermöglicht wird.

⊡ Lesezeichen

http://bit.ly/LUNA2

Eine genaue Übersicht darüber, in welchen Ländern der Verkauf von Software möglich ist und in welchen Ländern nur Freeware angeboten werden kann, finden Sie hier.

2. Nach dem Klick auf *Upload Application* landen Sie auf einer Seite, auf der Sie nicht nur Ihre signierte APK-Datei hochladen, sondern auch diverse Angaben zu Ihrer App machen können, die dann letztendlich im Android Market erscheinen.

   Hier noch einige Tipps:
   Sie können pro Anwendung zwei Screenshots hochladen. Die Screenshots müssen entweder im PNG- oder im JPG-Format vorliegen und dürfen die Abmessungen 320 x 480 Bildpunkte oder 480 x 854 Bildpunkte haben. Screenshots werden im Android Market immer im Hochformat angezeigt. Auch wenn Ihre Anwendung oder Ihr Spiel nur im Querformat läuft: Drehen Sie entweder den Screenshot ins Hochformat oder verwenden Sie zwei querformatige Screenshots übereinander auf einer hochformatigen Grafik.

   Als Nächstes können Sie eine Werbegrafik hochladen. Diese Werbegrafik wird unter Umständen im Android Market verwendet, wenn Google sie gut findet und bewerben möchte. Das Format der Grafik erinnert an einen Banner: 180 x 120 Bildpunkte. Beispiele dieser Werbegrafiken finden Sie beim Aufruf des Android Market von einem Telefon auf der Startseite.

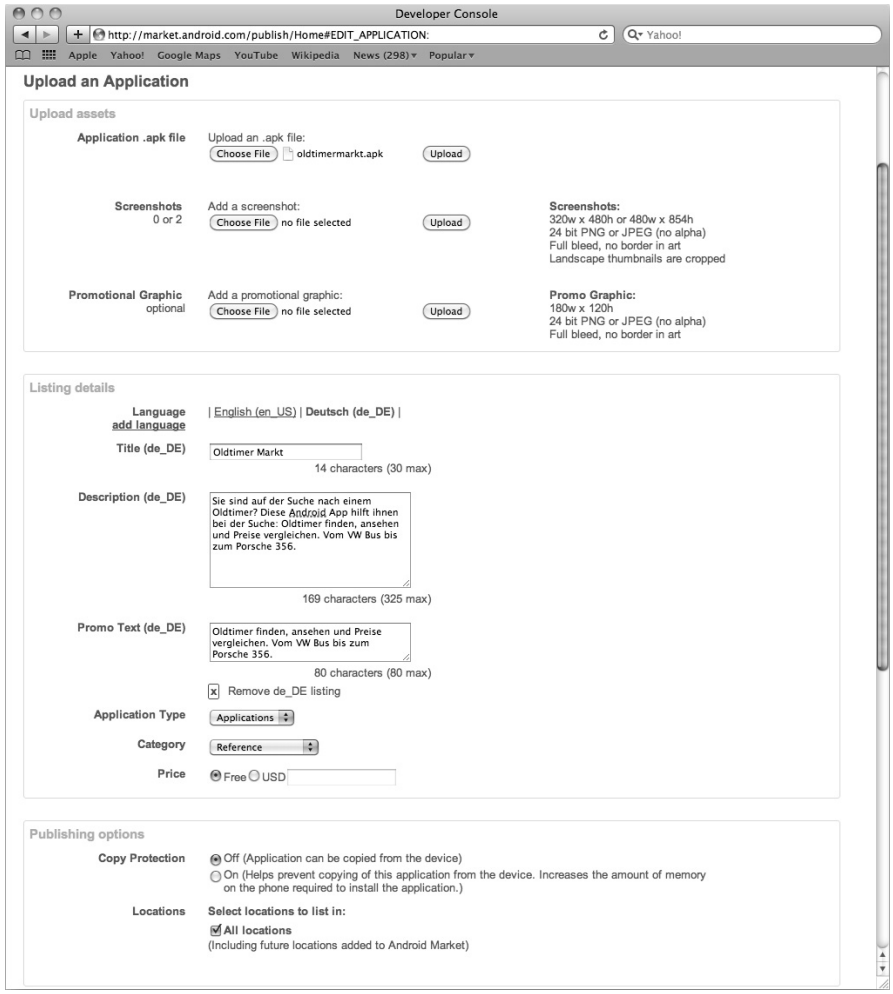


Bild 4.22: Upload einer neuen Applikation.

3. Ihre Anwendung kann im Android Market unterschiedliche Namen pro Sprache besitzen. Um eine weitere Sprache hinzuzufügen, klicken Sie einfach auf *Add Language*.

   Der Titel darf pro Sprache 30 Zeichen besitzen. Wir empfehlen maximal 16 Zeichen, da sonst nicht der volle Name in der Listenansicht innerhalb des Android Market angezeigt wird. Als Nächstes können Sie eine Beschreibung Ihres Produkts angeben. Die App-Beschreibungen im Android Market sind wesentlich kürzer als bei iTunes. Versuchen Sie, Ihr Produkt kurz und präzise mit maximal 325 Zeichen zu beschreiben.

   Falls Google Ihre Anwendung oder Ihr Spiel mag und sich entschließt, es zu bewerben, würde der Promotext neben der oben hochgeladenen Promografik angezeigt. Die Platzierung sehen Sie, wenn Sie im Android Market auf eine der Kategorien klicken und sich die Listenansicht für beispielsweise Spiele ansehen. Der Promotext darf maximal 80 Zeichen lang sein und kann nur angegeben werden, wenn Sie auch eine Promografik erstellt und hochgeladen haben.

4. Geben Sie dann noch an, ob es sich bei Ihrer App um eine Anwendung oder ein Spiel handelt und welcher Kategorie Sie es zuordnen möchten. Geben Sie Ihre Android App nur gegen Bezahlung ab, können Sie jetzt den Preis angeben. Ist ein Gratisangebot eher Ihr Ding, wählen Sie unter *Price* einfach *Free*. Eine als kostenlos eingestellte Anwendung kann später nicht mehr mit einem Preis versehen werden. In diesem Fall müssten Sie ein Update mit einer neuen, signierten APK-Datei durchführen.

5. Unter *Copy Protection* entscheiden Sie, ob Sie Ihre Anwendung mit einem Kopierschutz versehen möchten oder nicht. Nehmen Sie den Kopierschutz, ist es nicht mehr so einfach, die Anwendung nach dem Herunterladen auf eine Speicherkarte zu kopieren und in einem anderen Telefon zum Laufen zu bringen. Wie jeder Kopierschutz ist er nicht 100%ig sicher und daher nur sinnvoll, wenn Sie Ihre Anwendung verkaufen. Kopiergeschützte Programme oder Spiele benötigen auch einen größeren Speicherbedarf bei der Installation.

   Eine weitere Einschränkung ist, dass Apps mit Kopierschutz nicht auf Android-Entwicklertelefonen installiert werden können. Falls Sie Ihr Telefon ganz normal im Handel gekauft haben, sind Sie davon nicht betroffen. Nur Telefone, die direkt über das Android-Entwicklerprogramm gekauft wurden, erlauben keine Installation von kopiergeschützten Programmen.

   Wenn Sie Ihre App in allen Ländern vertreiben wollen, in denen der Android Market verfügbar ist und in Zukunft auch noch verfügbar sein wird, lassen Sie das Häkchen bei *All locations* gesetzt. Sobald Sie es entfernen, sehen Sie eine Länderliste, über die Sie gezielt den Vertrieb in bestimmte Länder festlegen können.

6. Ganz zum Schluss müssen Sie den Android Content-Richtlinien zustimmen und somit garantieren, dass Sie dem Android Market unbedenkliche Inhalte hinzufügen. Abschließend werden Sie noch gefragt, ob Sie Ihre Software ohne Anwendung von nicht zugelassener Verschlüsselungstechnik erstellt haben. Ist dem so, sind Sie fertig und können mit *Publish* Ihre App in den Android Market schicken.

Bild 4.23: Die App ist im Android Market veröffentlicht.

Wenn Sie den iTunes App Store gewohnt sind, werden Sie vom Android Market in einer Hinsicht begeistert sein: Sie warten nicht tagelang auf das finales Urteil darüber, ob Ihre App angenommen wird oder nicht, sondern mit dem Anklicken von *Publish* befindet sich Ihre App bereits für alle Benutzer im Markt.

Stichwortverzeichnis

Markus Spiering / Sven Haiges

2. aktualisierte Auflage

HTML5-Apps
für iPhone und Android

iOS von Apple und Android von Google – wer Apps für das iPhone und andere Smartphones programmieren und damit in allen wichtigen App Stores Geld verdienen will, muss diese beiden Betriebssysteme beherrschen. Den Schlüssel dazu bieten HTML5, CSS und JavaScript, denn damit lassen sich mobile Apps plattformunabhängig entwickeln. Dieses Buch zeigt, wie Sie Anwendungen erstellen, die in den Browsern von iOS, Android und auch Blackberry OS ohne Anpassungen laufen, und wie Sie aus einer Web-App eine native Anwendung für den iTunes App Store und den Android Market bauen.

Ein Programm für die wichtigsten Systeme
Mit einer auf HTML5 basierenden Applikation stellen Sie sicher, dass Ihre Anwendung auf den wichtigsten mobilen Plattformen läuft. Gleichzeitig haben Sie die Option, daraus mit Frameworks wie PhoneGap eine native Anwendung für den iTunes App Store oder den Android Market zu bauen. Im Handumdrehen erstellen Sie Programme, die sich nicht nur in den App Stores verkaufen, sondern auch mit Funktionen bereichern lassen, die im Browser gar nicht zur Verfügung stehen: Die Bewegungssensoren, die Kamera, der Vibrationsalarm, der Kompass, das Adressbuch und vieles mehr lassen sich direkt per HTML-, CSS- und JavaScript-Code über wenige, einfache JavaScript-APIs steuern.

Der richtige iPhone- und Android-Look für Ihre App
Dieses Buch beschäftigt sich auch mit den neuen Funktionen und Möglichkeiten von HTML5 und zeigt diese eindrucksvoll und verständlich anhand vieler Beispiele im echten „iPhone- und Android-Look & Feel". Last, but not least erfahren Sie, wie Sie Ihre frisch gebaute Anwendung dann auch im iTunes App Store und im Android Market verkaufen.

Setzen Sie HTML5 perfekt ein
Die Autoren Markus Spiering und Sven Haiges beschreiben eindrucksvoll, wie Sie HTML5 in der mobilen App-Entwicklung einsetzen und den gesamten Entwicklungsprozess von Anfang bis Ende verstehen. Hier bleiben keine Fragen offen.

Aus dem Inhalt:

Über die Autoren:

Markus Spiering, Jahrgang 1977, ist verantwortlich für die mobile Produktpalette der weltweit größten Fotocommunity Flickr (www.flickr.com) in San Francisco, Kalifornien.

Sven Haiges, Jahrgang 1979, hat Bücher zu den Web-Frameworks Struts, JavaServer Faces (JSF) und Grails geschrieben, ist in München als freier Android-Berater tätig.

Alle Beispieldateien auf www.buch.cd

9 783645 601191

30,– EUR [D]
ISBN 978-3-645-60119-1