Erica Sadun

Das große iPhone Entwicklerbuch

Rezepte für Anwendungsprogrammierung mit dem iPhone SDK

ADDISON-WESLEY

Das große iPhone Entwicklerbuch

Ich widme dieses Buch in Liebe meinem Ehemann Alberto,
der in all den Jahren so viele technische Spielereien und SDKs ertragen musste
und dabei unterm Strich stets freundlich und geduldig blieb.

Erica Sadun

Das große iPhone Entwicklerbuch

Rezepte für Anwendungsprogrammierung mit dem iPhone SDK

ADDISON-WESLEY

An imprint of Pearson Education

München • Boston • San Francisco • Harlow, England
Don Mills, Ontario • Sydney • Mexico City
Madrid • Amsterdam

Bibliografische Information der Deutschen Nationalbibliothek

Die Deutsche Nationalbibliothek verzeichnet diese Publikation in der Deutschen Nationalbibliografie; detaillierte bibliografische Daten sind im Internet über http://dnb.d-nb.de abrufbar.

10 9 8 7 6 5 4 3 2 1

13 12 11 10

ISBN: 978-3-8273-2917-2

© 2010 by Addison-Wesley Verlag,
ein Imprint der Pearson Education Deutschland GmbH,
Martin-Kollar-Straße 10–12, D-81829 München/Germany
Alle Rechte vorbehalten
Lektorat: Boris Karnikowski, bkarnikowski@pearson.de
Fachlektorat: Philipp Homann, www.Page.de
Korrektorat: Frederike Danecke, Zülpich
Covergestaltung: Marco Lindenbeck, mlindenbeck@webwo.de
Herstellung: Philipp Burkart, pburkart@pearson.de
Satz: Cordula Winkler, mediaService (www.media-service.tv)
Druck und Verarbeitung: Kösel Druck, Krugzell (www.KoeselBuch.de)
Printed in Germany

Inhaltsverzeichnis

Danksagung

Diese Buch wäre ohne die Bemühungen von Chuck Toporek (meinem Lektor und Einpeitscher), Chris Zahn (dem fantastisch begabten Development Editor), Romny French (dem treuen und rührigen Redaktionsassistenten, der hinter den Kulissen dafür sorgt, dass alle Räder laufen) und Karen Gettman (Chucks Chefredakteurin) nicht zustande gekommen. Karen hat mich bei diesem immer umfangreicher werdenden Buch (und dabei meine ich »umfangreich« durchaus auch wörtlich – sehen Sie sich nur einmal die Seitenzahl an!) unaufhörlich unterstützt. Ein großes Dankeschön geht auch an das Produktionsteam von Addison-Wesley, vor allem an Kristy Hart, Anne Goebel, Gary Adair, Keith Cline, Geneil Breeze, Cheryl Lenser, Chelsey Marti und Jake McFarland. Außerdem danke ich dem Team von Safari dafür, dass es mein Buch in der Rubrik »Rough Cuts« untergebracht und technische Fehler ausgebügelt hat.

Mein Dank gilt ebenso meinem langjährigen Agenten Neil Salkind und den Fachlektoren, die halfen, dieses Buch realistisch zu gestalten, anstatt in Wunschdenken abzugleiten. Außerdem möchte ich allen Kollegen bei TUAW, Ars Technica und dem Digital Media/Inside iPhone-Blog danken.

Ein besonderer Dank geht an Joachim Bean und Aaron Basil, die nicht nur eine fachliche Begutachtung durchgeführt, sondern auch schon sehr früh Rückmeldung zu den einzelnen Kapiteln gegeben und mir wichtige Erkenntnisse vermittelt und Ratschläge erteilt haben. Mehr als jede andere Person haben sie dazu beigetragen, das Buch zu dem zu machen, was Ihnen jetzt vorliegt. Sie haben mir in einem erstaunlichen Umfang Informationen gegeben, für die ich ihnen sehr, sehr dankbar bin, und sie taten das selbst dann, wenn ich mich zu unchristlichen Tageszeiten an sie wandte. Ich danke auch Tim Isted (dem Co-Autor des bei Addison-Wesley erschienenen Titels *Core Data for iPhone*) für seine wertvollen Hinweise zum Core Data-Kapitel in diesem Buch. Ich würde auch gern denjenigen nennen, der das GameKit-Kapitel mit Argusaugen gelesen hat, aber leider kann ich nur sagen: »Danke, Mr. X!« Ohne die Hilfe meines Fachgutachterteams hätte ich all dies nicht schaffen können, weshalb ich euch allen sehr danke. Besonderer Dank geht auch an die hier noch nicht ausdrücklich genannten Teammitglieder Roberto Gamboni, John Muchow und Scott Mikolaitis.

Zu tiefem Dank bin ich der großen Community der iPhone-Entwickler verpflichtet, darunter Alex Schaefer, Nick Penree, James Cuff, Jay Freeman, Mark Montecalvo, August Joki, Max Weisel, Optimo, Kevin Brosius, Planetbeing, Pytey, Roxfan, MuscleNerd, np101137, UnterPerro,Youssef Francis, Bryan Henry, Daniel Peebles, ChronicProductions, Greg Hartstein, Emanuele Vulcano, Sean Heber, Steven Troughton-Smith, Dick Applebaum, Kevin Ballard, Jay Abbott,Tim Grant Davies, Landon Fuller, Stefan Hafeneger, Scott Elich, chrallelinder, J. Roman, jtbandes, Artissimo, Aaron Alexander, Scott Lawrence, Kenny Chan Ching-Kin, Sjoerd van Geffen, Absentia, Nownot, Matt Brown, Chris Foresman,Aron Trimble, Paul Griffin, Nicolas Haunold, Anatol Ulrich (hypnocode GmbH), Kristian Glass, Yanik Magnan, ashikase, Eric Mock und allen Beteiligten bei den iPhone-Entwicklerkanälen unter *irc.saurik.com* und *irc.freenode.net*, die zu zahlreich sind, um sie hier alle namentlich zu nennen. Ihre technischen Lösungen, Vorschläge und Kommentare haben dieses Buch erst möglich gemacht. Sollte ich irgendjemanden übersehen haben, der seinen Beitrag zu diesem Buch geleistet hat, möge er dieses Versäumnis bitte entschuldigen.

Besonderer Dank gilt meiner Familie und meinen Freunden, die mich während der monatlich neuen Beta-Versionen unterstützt haben und meine unerklärliche Abwesenheit und meine häufigen Verzweiflungsschreie geduldig hingenommen haben. Ich schätze es sehr, dass Ihr mir beigestanden habt. Ich möchte auch meinen Kindern für ihre Treue danken, die sogar dann noch Bestand hatte, als sie erfahren mussten, dass ein krummer Rücken und das Klappern der Tastatur ein schlechter Ersatz für eine richtige Mutter sind. Meine Kinder waren mir in den letzten Monaten beim Testen von Anwendungen eine unschätzbare Hilfe, haben viele Vorschläge eingebracht und sich einfach als wunderbare Menschen erwiesen. Ich bin als Mutter so wahnsinnig glücklich, dass diese Kinder ein Teil meines Lebens sind.

Die Autorin

Erica Sadun hat an die drei Dutzend Fachbücher, besonders in den Bereichen Programmierung, digitales Video und digitale Fotografie, allein oder als Co-Autorin geschrieben oder durch sonstige Beiträge daran mitgewirkt. Als unbelehrbare Computerbesessene ist ihr noch nie eine Funktion begegnet, die sie nicht gebrauchen konnte. Ihre wechselhafte Vergangenheit umfasst Auseinandersetzungen mit NeXT, Newton, iPhone und unzähligen erfolgreichen und nicht erfolgreichen Technologien. Wenn sie nicht gerade schreibt, kümmert sie sich zusammen mit ihrem ebenfalls computerbesessenen Ehemann um ihre drei Kinder (auszubildende Computerbesessene), die ihre Eltern mit zurückhaltender Verwirrung betrachten.

Vorwort

Wenige Plattformen können sich mit den einzigartigen Entwicklungstechnologien des iPhones messen. Es kombiniert ein mobiles Computersystem auf OS X-Grundlage mit einem innovativen Touchscreen, Ortungsdiensten, einem eingebauten Beschleunigungsmesser und mehr. Als Apple Anfang März 2008 die Beta-Version des iPhone SDK vorstellte, reagierten so viele Entwickler, dass die Server von Apple in die Knie gingen. In weniger als einer Woche luden Entwickler das iPhone SDK mehr als 100.000-mal herunter.

Seitdem wurden 90.000 Anwendungen für ein Publikum von über 30 Millionen iPhones und über 20 Millionen iPod touch-Geräte in den App Store eingestellt. Da das iPhone immer größere Kreise zieht, muss sich auch das *iPhone-Entwicklerbuch* als leicht zugängliche Informationsquelle für neue iPhone-Programmierer weiterentwickeln.

V.1 Neuer Stoff in dieser Ausgabe

Wenn Sie bereits die erste Ausgabe dieses Buches besitzen, werden Sie sich vielleicht fragen, wozu Sie eine weitere Ausgabe kaufen sollten. Die Antwort liegt auf der Hand: Vergleichen Sie nur einmal den Umfang der beiden Bücher! Diese Neuausgabe ist mehr als doppelt so umfangreich wie die ursprüngliche. Wir haben fast 500 Seiten an neuem Stoff hinzugefügt, um alle neuen Möglichkeiten des iPhone SDK 3.0 abzudecken, und auch einige der bereits in der ersten Ausgabe behandelten Themen erweitert.

Unter anderem werden folgende neue Gebiete in dieser Ausgabe behandelt werden:

- Xcode und Interface Builder verwenden
- Einführung in Objective-C
- Core Data für das iPhone
- MapKit und Core Location

- GameKit für andere Zwecke als Spiele verwenden, um Chats und Bonjour-Netzwerke zu ermöglichen
- erweiterte Bewegungserkennung mit schüttelgesteuerten Widerrufsmöglichkeiten
- eine neue Klasse für einen Suchanzeigecontroller mit maßgeschneiderten Tabellenheadern und -footern
- die neue Apple-Spezifikation für Gerätefähigkeiten
- Anwendungsinterner Produktverkauf mit StoreKit
- Push-Benachrichtungen von der Client- und von der Serverseite aus
- Medien in der geräteeigenen iPod-Bibliothek suchen und wiedergeben
- Videoaufnahme und -bearbeitung sowie neue Klassen für AV-Player und -Recorder
- das Framework für Bedienungshilfen wie VoiceOver in eigenen Anwendungen nutzen
- und noch sehr viel mehr!

Sie werden auch feststellen, dass wir uns Ihre Kommentare zu Herzen genommen haben. Als die erste Ausgabe herauskam, war nicht ganz klar, an wen sie sich richtete – an erfahrene Entwickler oder an Anfänger? Wir haben uns auch um dieses Problem gekümmert. Zwar ist das Buch für erfahrene iPhone- und Mac-Entwickler gedacht, die bereits mit Objective-C, Xcode und den Cocoa-Frameworks vertraut sind, doch enthält diese neue Ausgabe auch eine Einführung in Objective-C (Kapitel 3) und in Xcode und Interface Builder, damit Entwickler, die Erfahrungen aus anderen Sprachen (oder anderen Plattformen) mitbringen, sich schnell im Mac/iPhone-Umfeld zurechtfinden.

Ein Buch kann zwar nicht jedem Leser alles bieten, doch haben wir mit dieser Neuausgabe einen Schritt in diese Richtung versucht. Wir hoffen, dass die Änderungen, die Sie in diesem umfangreicheren neuen Buch erkennen, zu Ihrer Zufriedenheit ausfallen. Wenn ja, schreiben Sie doch eine Rezension bei Amazon, oder senden Sie mir eine kurze Nachricht (in englischer Sprache an `erica@ericasadun.com`).

V.2 Zielgruppe

Dieses Buch richtet sich an erfahrene Entwickler, die Apps für das iPhone und den iPod touch erstellen möchten. Sie sollten bereits mit Objective-C, den Cocoa-Frameworks und den Werkzeugen von Xcode vertraut sein. Sollte diese Plattform für Sie neu ein, finden Sie in dieser Ausgabe des *iPhone-Entwicklerbuchs* jedoch auch eine Kurzeinführung in Objective-C und in die Werkzeuge von Xcode, damit Sie sich schneller heimisch fühlen.

Keine Erfahrungen mit dem Mac oder dem iPhone?

Für diejenigen, die Erfahrungen in C aufweisen oder bereits mit anderen objektorientierten Sprachen wie C++ oder Java gearbeitet haben, befindet sich in diesem Vorwort der Abschnitt *Der Weg zur Mac/iPhone-Entwicklung*, in dem Sie lesen können, wie Sie zu einem Mac-Entwickler werden können. Übergehen Sie diesen Abschnitt nicht!

Jeder Programmierer hat zwar andere Ziele und Erfahrungen, doch die meisten iPhone-Entwickler müssen bei ihrer Arbeit ähnliche Aufgaben lösen:

- Wie erstelle ich eine Tabelle?
- Wie erstelle ich einen sicheren Schlüsselbundeintrag?
- Wie durchsuche ich das Adressbuch?
- Wie wechsle ich zwischen Ansichten?
- Wie kann ich Core Location und den Kompass des iPhone 3GS verwenden?

Und so weiter. Falls Sie sich diese Fragen schon selbst gestellt haben, ist dieses Buch genau richtig für Sie. Mit seinen klaren, vollständig beschriebenen Beispielen hilft es Ihnen in null Komma nichts, mit dem iPhone SDK arbeiten zu können. Und was das Beste ist: Alle Coderezepte in diesem Buch wurden getestet – und zwar in funktionierenden Anwendungen in der Praxis –, sodass Sie gebrauchsfertige Lösungen für Ihre eigenen Anwendungen erhalten.

V.3 Voraussetzungen

Wenn Sie Anwendungen für das iPhone oder den iPod touch entwickeln möchten, müssen Sie natürlich mindestens eines dieser Geräte haben, um Ihr Programm zu testen. Die folgende Aufstellung gibt an, was Sie alles brauchen, um mit der Programmierung für das iPhone oder den iPod touch zu beginnen:

- **Das iPhone SDK von Apple** Die neueste Version des iPhone SDK können Sie vom iPhone Dev Center von Apple unter *http://developer.apple.com/iphone* herunterladen. Vor dem Download müssen Sie dem (kostenlosen) Entwicklerprogramm von Apple beitreten. Wenn Sie Ihre Anwendungen über den App Store verkaufen möchten, müssen Sie jedoch das gebührenpflichtige Entwicklerprogramm nutzen, das zurzeit 79 € pro Jahr kostet. Registrierte Entwickler erhalten Zertifikate, mit denen sie ihre Anwendungen signieren und sie zum Testen und Debuggen auf ihre iPhone- bzw. iPod touch-Geräte herunterladen können.

Universitäts- und Studentenrabatte

Für Studenten und Dozenten bietet Apple ein Universitätsprogramm an. Wenn Sie ein CS-Student an einer Universität sind, fragen Sie Ihren Professor, ob Ihre Hochschule an diesem Universitätsprogramm teilnimmt. Weitere Informationen darüber finden Sie unter *http://developer.apple.com/support/iphone/university*.

- **Einen Intel-Macintosh mit Leopard oder Snow Leopard** Empfohlen wird Snow Leopard, da dieses Betriebssystem Zugriff auf Xcode 3.2 mit vielen neuen Funktionen wie »Build and Analyze« bietet. Für die Entwicklung brauchen Sie sehr viel Festplattenplatz. Außerdem sollte der Mac über mindestens 1 Gbyte RAM verfügen, am besten aber über 2 oder gar 4 Gbyte, um die Kompilierung zu beschleunigen.

- **Ein iPhone oder einen iPod touch** Das iPhone SDK und Xcode enthalten zwar einen Simulator, auf dem Sie Ihre Anwendungen testen können, doch zur Entwicklung für diese Plattform brauchen Sie auf jeden Fall auch ein physisches iPhone oder einen iPod touch (oder beides). Über ein USB-Kabel können Sie das Gerät an Ihren Computer anschließen und die erstellte Software darauf installieren. Wenn Sie Ihre Programme im App Store anbieten möchten, ist es hilfreich, mehrere Geräte verschiedener Hardwaregenerationen zur Hand zu haben, um Tests auf denselben Plattformen durchzuführen, die auch Ihr Zielpublikum verwendet.
- **Mindestens einen USB 2.0-Anschluss** Damit können Sie das Entwicklungs-iPhone bzw. den iPod touch zur Datenübertragung und zu Testzwecken an Ihren Computer anschließen.
- **Eine Internetverbindung** Diese Verbindung ermöglicht es, Programme zu testen, die eine aktive WiFi-Verbindung oder den EDGE- oder 3G-Dienst nutzen.
- **Erfahrung mit Objective-C** Um für das iPhone zu programmieren, müssen Sie Objective-C 2.0 kennen. Diese Sprache basiert auf ANSI C mit objektorientierten Erweiterungen, weshalb Sie sich auch ein bisschen in C auskennen sollten. Wenn Sie schon in Java oder C++ programmiert haben und mit C vertraut sind, ist der Übergang zu Objective-C recht einfach. Kapitel 3, *Objective-C-Trainingslager*, hilft Ihnen dabei, schnell Fuß zu fassen.

Hinweis

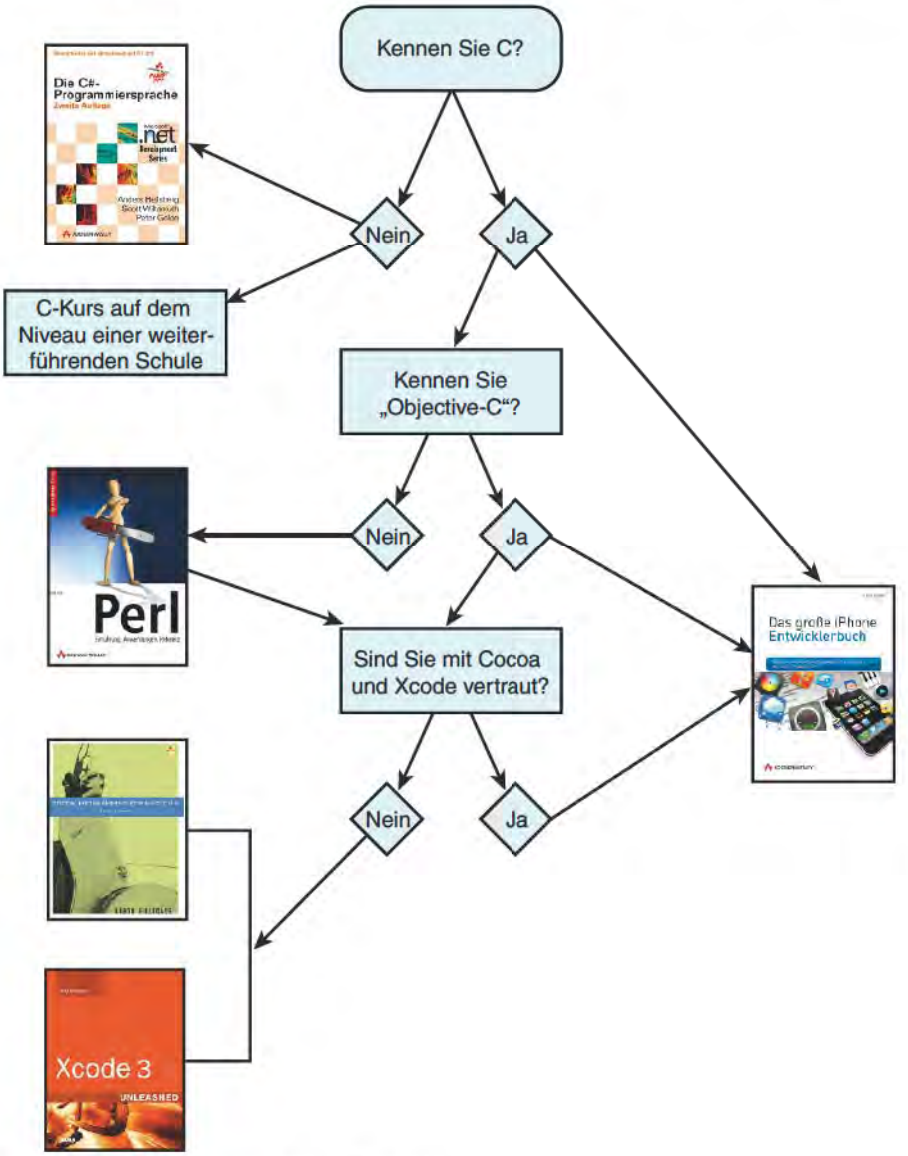
Obwohl das SDK für die Entwicklung für das iPhone, deniPod touch und das iPad sowie für mögliche, noch nicht angekündigte Plattformen vorgesehen ist, wird als Zielplattform in diesem Buch der Einfachheit halber nur das iPhone genannt. Das meiste Material ist auch für den iPod touch verwendbar, wobei natürlich Funktionen und Merkmale wie das Telefonieren und eingebaute Lautsprecher nicht genutzt werden können.

V.4 Der Weg zur Mac/iPhone-Entwicklung

Wie bereits erwähnt, kann ein Buch nicht jedem Leser alles bieten. Wenn wir alles hineinpacken würden, was Sie wissen müssten, könnten Sie den resultierenden Folianten nicht einmal mehr aufheben. Es gibt tatsächlich sehr viel, das Sie wissen müssen, um für den Mac und das iPhone zu programmieren. Wenn Sie gerade erst anfangen und noch keine Programmiererfahrung haben, sollten Sie als Erstes einen Kurs in der Programmiersprache C auf dem Niveau einer weiterführenden Schule belegen. Das Alphabet mag zwar mit A beginnen, der Ausgangspunkt der meisten Programmiersprachen und mit Sicherheit der Startpunkt für Ihre Karriere als Entwickler ist jedoch C.

Wenn Sie C kennen und wissen, wie ein Computer funktioniert (was Sie in einem grundlegenden C-Kurs lernen), ist der Rest leicht. Sie können einfach zu Objective-C übergehen und lernen, wie Sie mithilfe der Cocoa-Frameworks programmieren. Um Ihnen zu helfen, Ihren Weg zu finden, habe ich das Flussdiagramm aus *Abbildung V.1* zusammengestellt, um Sie auf einige wichtige Bücher aufmerksam zu machen.

Wenn Sie C kennen, haben Sie verschiedene Möglichkeiten, um die Programmierung in Objective-C zu erlernen. Eine Kurzeinführung in Objective-C finden Sie im dritten Kapitel dieses Buches. Wollen Sie diese Sprache jedoch ausführlicher kennenlernen, können Sie entweder die Dokumentation von Apple lesen – *Object-Oriented Programming with Objective-C 2.0*[1] – oder ein Buch wie z. B. *Objective-C 2.0: Anwendungen entwickeln für Mac und iPhone* von Stephen Kochan erwerben (Addison-Wesley, 2009).



▸ *Abbildung V.1: Der Weg zum iPhone-Entwickler*

1 Siehe *http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/OOP_ObjC/OOP_ObjC.pdf*.

Wenn Sie die Sprache gemeistert haben, müssen Sie sich mit Cocoa und den Entwicklerwerkzeugen, also Xcode, vertraut machen. Dazu gibt es verschiedene Möglichkeiten. Auch hier können Sie sich die Apple-Dokumentation über Cocoa und Xcode[2] ansehen oder sich die besten Bücher herauspicken. Aaron Hillegass, Gründer der Big Nerd Ranch[3] in Atlanta, ist der Autor von *Cocoa: Programmierung für Mac OS X*, einem Buch, das von Mac-Entwicklern sehr geschätzt wird und in der Mailingliste von CocoaDev am häufigsten empfohlen wird. Wenn Sie mehr über Xcode lernen möchten, greifen Sie einfach zu *Xcode 3* von Fritz Anderson. Die aktuelle Ausgabe deckt zwar die iPhone-spezifischen Funktionen von Xcode nicht ab (die erst mit Xcode 3.1 eingeführt wurden), gibt Ihnen aber eine solide Grundlage für den Einsatz von Xcode als Entwicklungsumgebung.

Hinweis

Es gibt noch viele Bücher von anderen Autoren auf dem Markt, darunter den Bestseller *Beginning iPhone 3 Development* von Dave Marks und Jeff LaMarche (Apress, 2009). Beschränken Sie sich also nicht auf ein einziges Buch oder einen Autor.

Um die Entwicklung für den Mac wirklich zu meistern, brauchen Sie eine breite Palette an Quellen: Bücher, Blogs, Mailinglisten, die Apple-Dokumentation und vor allem Konferenzen. Wenn Sie die Gelegenheit haben, bei der WWDC oder der C4 anwesend zu sein, werden Sie wissen, warum ich das sage. Für ernsthafte Entwickler ist die Zeit, die sie auf solchen Konferenzen im Gespräch mit anderen Entwicklern oder (im Falle der WWDC) mit den Ingenieuren von Apple verbringen, die Ausgabe wert.

V.5 Aufbau des Buchs

Dieses Buch bietet jeweils eigenständige Rezepte für die häufigsten Probleme neuer iPhone-Entwickler an: Gestaltung von Oberflächenelementen, Reaktion auf Benutzeraktionen, Zugriff auf lokale Datenquellen und Verbindung zum Internet. Jedes Kapitel fasst verwandte Aufgaben zusammen, sodass Sie direkt zur gesuchten Lösung springen können, ohne entscheiden zu müssen, welche Klasse oder welches Framework am besten für das Problem geeignet ist.

Das *iPhone-Entwicklerbuch* bietet die bequeme Möglichkeit, Code »auszuschneiden und einzufügen«. Den Quellcode aus den Rezepten in diesem Buch können Sie kostenlos in Ihren eigenen Programmen wiederverwenden und jeweils an die besonderen Bedürfnisse Ihrer Anwendung anpassen.

Die folgende Zusammenfassung zeigt, was Sie in den einzelnen Kapiteln dieses Buchs erwartet:

- **Kapitel 1, »Einführung in das iPhone SDK«** Kapitel 1 gibt eine Einführung in das iPhone SDK, stellt das iPhone als Plattform für die Bereitstellung vor, zeigt die Grenzen auf usw. Es zeigt den Aufbau einer normalen iPhone-Anwendung und hilft Ihnen, mit dem iPhone-Entwicklerportal zurechtzukommen.

2 Siehe *Cocoa Fundamentals Guide* (*http://developer.apple.com/mac/library/documentation/Cocoa/Conceptual/CocoaFundamentals* für Cocoa und *A Tour of Xcode* (*http://developer.apple.com/mac/library/documentation/DeveloperTools/Conceptual/A_Tour_of_Xcode* für Xcode.

3 *http://www.bignerdranch.com*

- **Kapitel 2, »Ein erstes Projekt erstellen«** Kapitel 2 deckt die Grundlagen für den Aufbau einer ersten Hello-World-Anwendung ab. Es gibt eine Einführung in Xcode und Interface Builder und zeigt, wie Sie diese Programme in Ihren Projekten einsetzen. Sie erfahren hier etwas über die grundlegenden Debugging-Werkzeuge, lernen ihre Verwendung kennen und erhalten einige Tipps zu praktischen Compiler-Direktiven. Außerdem lernen Sie, wie Sie Profile einrichten und zur Bereitstellung Ihrer Anwendung auf Ihrem eigenen Gerät, für Beta-Tester und im App Store einsetzen.
- **Kapitel 3, »Objective-C-Trainingslager«** Wenn Ihnen nicht nur das iPhone, sondern auch Objective-C neu sind, werden Sie dieses Grundlagenkapitel zu schätzen wissen. Objective-C ist die übliche Programmiersprache für das iPhone und für Mac OS X. Mit dieser leistungsfähigen objektorientierten Sprache können Sie Anwendungen erstellen, die die Apple-Frameworks Cocoa und Cocoa Touch nutzen. Kapitel 3 gibt eine Einführung in diese Sprache und einen Überblick über die objektorientierten Aspekte. Außerdem werden die Speicherverwaltung angesprochen und die häufigsten Klassen vorgestellt, um Ihnen einen schnellen Einstieg in die Programmierung mit Objective-C zu ermöglichen.
- **Kapitel 4, »Benutzeroberflächen entwerfen«** Kapitel 4 gibt eine Einführung in die Bibliothek der grafischen Klassen für das iPhone und in ihre Geometrie. In diesem Kapitel lernen Sie, diese Klassen einzusetzen und Aufgaben wie die Neuausrichtung des Geräts zu meistern. Außerdem finden Sie hier Lösungen für die Gestaltung und Anpassung von Oberflächen und kombinierte Ansätze, die sowohl mit Interface Builder gestaltete Oberflächen nutzen als auch solche, die mit Objective-C-Code programmiert wurden.
- **Kapitel 5, »Mit Ansichtscontrollern arbeiten«** Kurz gesagt, gilt für das iPhone die Faustregel: kleiner Bildschirm, große virtuelle Welten. In Kapitel 5 lernen Sie die verschiedenen Klassen von Ansichtscontrollern kennen, mit denen Sie den virtuellen Interaktionsraum des Benutzers vergrößern und ordnen können. Sie erfahren, wie Sie die eigentliche Arbeit für den Wechsel zwischen den verschiedenen Bildschirmen einer iPhone-Anwendung diesen leistungsfähigen Objekten überlassen können.
- **Kapitel 6, »Ansichten und Animationen zusammenstellen«** Kapitel 6 stellt iPhone-Ansichten vor, also die Objekte, die auf dem Bildschirm zu sehen sind. Sie lernen, wie Sie Ansichten gestalten, erstellen und ordnen, um Ihren iPhone-Anwendungen ein Rückgrat zu geben. Außerdem erfahren Sie etwas über Ansichtshierarchien, Geometrien und Animationen – Funktionen, die Ihren iPhone-Programmen Leben einhauchen.
- **Kapitel 7, »Mit Bildern arbeiten«** In Kapitel 7 lernen Sie das grundlegende Know-how für den Umgang mit Bildern auf dem iPhone. Dies betrifft vor allem die Klasse `UIImage`. Sie erfahren, wie Sie in Ihren Anwendungen Bilddaten laden, speichern und bearbeiten, wie Sie Bilder zu Ansichten hinzufügen und wie Sie Ansichten in Bilder umwandeln. Außerdem wird gezeigt, wie Sie Bilddaten verarbeiten, um besondere Effekte zu erzielen, wie Sie byteweise auf Bilddaten zugreifen und wie Sie mit der eingebauten Kamera des iPhones Fotos aufnehmen.

- **Kapitel 8, »Gesten und Berührungen«** Auf dem iPhone sind Berührungen die wichtigste Möglichkeit für den Benutzer, der Anwendung seine Absichten mitzuteilen. Diese Berührungen sind aber nicht auf das Antippen von Schaltflächen und die Benutzung einer Tastatur beschränkt. Kapitel 8 stellt Oberflächen mit direkter Bearbeitung, Mehrfachberührungen usw. vor. Sie sehen hier, wie Sie Ansichten erstellen, die die Benutzer auf dem Bildschirm verschieben können, und Sie erfahren, wie Sie Gesten unterscheiden und deuten.
- **Kapitel 9, »Steuerelemente erstellen und verwenden«** Klassen für Steuerelemente sind die Grundlage für viele der interaktiven Elemente des iPhones, z. B. Schaltflächen, Textfelder, Schieberegler und Schalter. In diesem Kapitel werden Steuerelemente und ihre Verwendung vorgestellt. Sie lernen die übliche Interaktion über Steuerelemente kennen und erfahren, wie Sie diese Objekte an die Bedürfnisse Ihrer Anwendung anpassen können. Außerdem lesen Sie, wie Sie eigene Steuerelemente ohne Vorlage selbst erstellen. Als Beispiel dazu wird ein Radsteuerelement angelegt.
- **Kapitel 10, »Benutzer benachrichtigen«** Das iPhone bietet viele Möglichkeiten, um Benutzern eine Rückmeldung zu geben: von Popup-Dialogfeldern und Fortschrittsbalken bis zu Audiosignalen und Aktualisierungen der Statusleiste. Kapitel 10 zeigt, wie Sie in Ihre Anwendungen solche Anzeigen einbauen, und erweitert Ihr Repertoire an Möglichkeiten zur Benachrichtigung von Benutzern. Sie lernen hier den üblichen Umgang mit solchen Popup-Klassen, aber auch Lösungen kennen, mit denen Sie Ihre Programme linearer ohne ausdrückliche Callbacks gestalten können.
- **Kapitel 11, »Tabellenansichten erstellen und verwalten«** Tabellen ermöglichen eine Interaktion durch Scrollen, die besonders auf einem kleinen, unübersichtlichen Gerät sinnvoll ist. Viele, wenn nicht gar die meisten der mit dem iPhone und dem iPod touch ausgelieferten Anwendungen verwenden Tabellen, z. B. Einstellungen, YouTube, Aktien und Wetter. Kapitel 11 zeigt, wie iPhone-Tabellen funktionieren, welche Arten von Tabellen Ihnen als Entwickler zur Verfügung stehen und wie Sie Tabellenfunktionen in Ihren eigenen Programmen nutzen können.
- **Kapitel 12, »Verbindungen mit GameKit und Bonjour«** GameKit ist die neue Lösung von Apple für Ad-hoc-Netzwerke mit Peer-to-Peer-Anbindung und baut auf einer Technologie namens Bonjour auf, die eine einfache, konfigurationsfreie Verbindung zwischen Geräten ermöglicht. Kapitel 12 gibt eine Einführung in das GameKit. Damit können Sie Spiele und Hilfsprogramme erstellen, die Informationen zwischen mehreren iPhones oder zwischen einem iPhone und einem Desktop-System übertragen. Dieses Kapitel deckt das standardmäßige GameKit sowie GameKit Voice für Voice-Chats mit Walkie-Talkie-Kommunikation ab. Außerdem stellt es eine grundlegende Bonjour-Programmierung vor, die die Einschränkungen von GameKit überwindet, damit Sie die iPhone-Kommunikation auch auf Desktop-Computer ausdehnen können.
- **Kapitel 13, »Netzwerke«** Als internetfähiges Gerät eignet sich das iPhone besonders für das Abonnieren von Webdiensten. Apple hat die Plattform mit einem soliden Fundament für alle Arten von Netzwerkdiensten und unterstützenden Technologien ausgestattet. Kapitel

13 stellt die üblichen Techniken für die Datenverarbeitung in Netzwerken vor und bietet Rezepte für alltägliche Aufgaben. Sie lesen hier etwas über die Erreichbarkeit von Netzwerken, über synchrone und asynchrone Downloads, die Verwendung des sicheren Schlüsselbunds zur Authentifizierung usw.

- **Kapitel 14, »Gerätefähigkeiten«** Jedes iPhone weist Eigenschaften auf, die nur für das jeweilige Gerät gelten, andere, die es mit anderen Geräten gemeinsam hat, sowie nur momentan oder auch dauerhaft gültige Eigenschaften. Beispiele dafür sind die physische Ausrichtung, der Modellname, der Ladezustand des Akkus und die eingebaute Hardware. In Kapitel 14 geht es um die Hardwarekonfiguration und die aktiven Sensoren des Geräts. Hier finden Sie Rezepte, mit denen Sie viele Informationen über das jeweilige Gerät zurückgeben können. Sie lesen, wie Sie Hardwarevoraussetzungen zur Laufzeit testen, wie Sie solche Voraussetzungen in der `Info.plist`-Datei des Programms angeben, wie Sie Meldungen der Sensoren anfordern und Benachrichtigungen abonnieren, um Callbacks für Änderungen des Sensorstatus zu erstellen. Dieses Kapitel deckt die Hardware, das Dateisystem und die Sensoren auf dem iPhone ab und hilft Ihnen, im Programmcode auf diese Merkmale zuzugreifen.
- **Kapitel 15, »Audio, Video und MediaKit«** Das iPhone ist ein Medienmeister, denn die eingebauten iPod-Funktionen können perfekt mit Audio- und Videodaten umgehen. Über das iPhone SDK stehen diese Möglichkeiten auch Ihnen als Entwickler zur Verfügung. Eine breite Palette von Klassen vereinfacht die Wiedergabe, Suche und Aufnahme von Medien. Kapitel 15 enthält Rezepte, in denen diese Klassen genutzt werden, um den Benutzern Medien anzuzeigen und diese zu bearbeiten. Sie erfahren, wie Sie Audio- und Video-Player und -Recorder erstellen, wie Sie die iPod-Bibliothek durchsuchen und wie Sie auswählen, welche Elemente wiedergegeben werden sollen.
- **Kapitel 16, »Push-Benachrichtigungen«** Wenn Entwickler direkt mit den Benutzern kommunizieren können, bieten sich dafür Push-Benachrichtigungen an. Damit können Sie über einen besonderen Apple-Dienst Nachrichten direkt auf dem iPhone-Bildschirm zustellen. Push-Benachrichtigungen sorgen dafür, dass das iPhone ein Meldungsfeld anzeigt, einen Klang abspielt oder ein Anwendungs-Badge aktualisiert. Dadurch können Dienste außerhalb des iPhones Verbindung mit einem Client auf dem Gerät aufnehmen und ihn über neue Daten oder Aktualisierungen informieren. In Kapitel 16 werden Push-Benachrichtigungen vorgestellt. Sie lernen, wie diese Benachrichtigungen funktionieren und welche Schritte erforderlich sind, um ein eigenes Push-System einzurichten.
- **Kapitel 17, »Core Location und MapKit«** Core Location übermittelt mithilfe verschiedener Technologien und Quellen auf Anforderung Ortungsdaten an das iPhone. MapKit erweitert Anwendungen um interaktive Landkarten, die die Benutzer einsehen und bearbeiten können. Mit Core Location und MapKit können Sie Anwendungen erstellen, mit deren Hilfe Ihre Benutzer Freunde und lokale Einrichtungen ausfindig machen oder ortsspezifische Informationen empfangen können. Kapitel 17 stellt diese Ortungs-Frameworks vor und zeigt, wie Sie sie in Ihre eigenen iPhone-Anwendungen aufnehmen können.

- **Kapitel 18, »Verbindung mit dem Adressbuch«** Mit dem Framework `AddressBook` können Sie im Programmcode auf die Kontaktdatenbank zugreifen und sie bearbeiten. In Kapitel 18 lernen Sie das Adressbuch kennen und erfahren, wie Sie die entsprechenden Frameworks in Ihren Anwendungen nutzen. Sie lesen, wie Sie auf Informationen über einzelne Kontakte zugreifen, wie Sie Kontaktinformationen bearbeiten und aktualisieren und wie Sie mithilfe von Prädikaten den gewünschten Kontakt finden. Außerdem werden in diesem Kapitel GUI-Klassen für die interaktive Auswahl, Anzeige und Bearbeitung von Kontakten vorgestellt.
- **Kapitel 19, »Core Data«** Core Data ermöglicht die Abfrage und Aktualisierung verwalteter Datenspeicher aus Ihrer Anwendung heraus. Mit einer Objektschnittstelle auf der Grundlage von Cocoa Touch wird die relationale Datenverwaltung aus der Domäne der SQL-Abfragen in das Objective-C-Umfeld der iPhone-Entwicklung übertragen. In Kapitel 19 erhalten Sie eine Einführung in Core Data. Es enthält nur wenige Abschnitte, die Ihnen einen Vorgeschmack dieser Technologie bieten und als Ausgangspunkt für weitere Studien dienen. Sie lernen hier, wie Sie verwaltete Datenbankspeicher entwerfen und wie Sie in Ihrem Code Daten hinzufügen, löschen und abfragen.
- **Kapitel 20, »StoreKit: Anwendungsinterner Produktverkauf«** StoreKit ist neu im SDK 3.0 und bietet Möglichkeiten für den anwendungsinternen Verkauf von Zusatzprodukten. Dieses Kapitel gibt eine Einführung in StoreKit und zeigt, wie Sie die StoreKit-API einsetzen, um den Benutzern Kaufmöglichkeiten zu bieten. Sie lernen die ersten Schritte mit StoreKit kennen und erfahren, wie Sie Produkte in iTunes Connect einrichten und ihre Beschreibung übersetzen, wie Sie Testbenutzer anlegen und wie Sie die verschiedenen Hürden der Entwicklung und Bereitstellung überwinden. Außerdem lernen Sie, wie Sie Kaufanforderungen von den Benutzern einholen und wie Sie diese Anforderungen zur Zahlungsabwicklung an den Store übermitteln. Von der Produktentwicklung bis zum Verkauf deckt dieses Kapitel den gesamten StoreKit-Vorgang ab.
- **Kapitel 21, »Bedienungshilfen und andere iPhone OS-Dienste«** Anwendungen arbeiten auf verschiedene Weisen mit den Standarddiensten des iPhones zusammen. In diesem Kapitel lernen Sie einige dieser Möglichkeiten kennen. Ein Programm kann seine Oberfläche gegenüber dem Handler der iPhone-Bedienungshilfe VoiceOver definieren, um Beschreibungen der Oberflächenelemente zu erstellen. Es ist möglich, Bundles für die mitgelieferte Anwendung *Einstellungen* einzurichten, damit die Benutzer über diese Schnittstelle auf die Einstellungen eines Programms zugreifen können. Darüber hinaus können Anwendungen auch öffentliche URL-Schemas deklarieren, über die andere iPhone-Anwendungen Kontakt mit ihnen aufnehmen und Dienste anfordern können. In diesem Kapitel lernen Sie die Interaktion zwischen Anwendungen und Diensten kennen. Sie sehen, wie Sie diese Merkmale in Ihre Programme einbauen und wie Sie im Code, in Interface Builder und in unterstützenden Dateien »Brücken« für diese Dienste ertellen.
- **Anhang A, »Schlüssel in Info.plist«** Der Anhang gibt einen Überblick über viele der Schlüssel, die in der Datei `Info.plist` verfügbar sind. Diese Datei beschreibt Ihre Anwendung dem Betriebssystem des iPhones.

V.6 Der Beispielcode

Aus didaktischen Gründen ist der Beispielcode zu diesem Buch gewöhnlich in einer einzigen `main.m`-Datei untergebracht. Das ist nicht die Art und Weise, wie iPhone- oder Cocoa-Anwendungen normalerweise geschrieben werden oder geschrieben werden sollten, eignet sich aber hervorragend dafür, einen bestimmten Gedanken deutlich zu machen. Es ist schwer, etwas zusammenhängend zu erklären, wenn die Leser dazu fünf, sieben oder neun verschiedene Dateien auf einmal durchsuchen müssen. In einer einzigen Datei dargestellt, kann man sich besser auf den Kernpunkt konzentrieren.

Die Beispiele sind nicht als eigenständige Anwendungen gedacht, sondern sollen ein einzelnes Rezept oder ein bestimmtes Prinzip verdeutlichen. Eine `main.m`-Datei mit einer zentralen Darstellung veranschaulicht die Implementierung in einem Stück. Die Leser können dieses Konzentrat an Prinzipien studieren und in den normalen Anwendungsaufbau mit den üblichen Dateistrukturen und -layouts übersetzen. Die Darstellung in diesem Code zeigt keinen Code, wie er für die tägliche Arbeit empfohlen wird, sondern spiegelt einen didaktischen Ansatz mit konzentrierten Lösungen wider, die Sie nach Bedarf in Ihre eigene Arbeit einfügen können.

Im Gegensatz dazu steht der Beispielcode von Apple, bei dem Sie viele Dateien durchkämmen müssen, um sich im Geiste ein Bild von den dort vorgestellten Prinzipien zu machen. Diese Beispiele sind als vollwertige Anwendungen ausgeführt und erledigen häufig Aufgaben, die mit dem zu tun haben, was Sie selbst vorhaben, aber nicht unbedingt dazu erforderlich sind. Die für Sie wichtigen Teile herauszufinden, ist keine leichte Aufgabe, und der Aufwand kann den möglichen Gewinn übersteigen. In diesem Buch gibt es zwei Ausnahmen von der Eine-Datei-Regel:

- Erstens wird im Überblick über das Erstellen von Anwendungen die vollständige Dateistruktur verwendet, die Xcode anlegt. Anhand dieser Datei will ich Ihnen zeigen, womit Sie in der Praxis zu rechnen haben. Die Ordner zu diesem Überblick können daher ein Dutzend oder mehr Dateien enthalten.
- Zweitens werden standardmäßige Klassen- und Headerdateien bereitgestellt, wenn die Klasse selbst das Rezept *ist* oder eine vorgefertigte Hilfsklasse bildet. Anstatt eine bestimmte Technik vorzustellen, zeigen einige der Rezepte solche vorgefertigten Klassenimplementierungen und Kategorien (also Erweiterungen einer bereits vorhandenen Klasse). Bei solchen Rezepten finden Sie neben `main.m` mit dem Grundgerüst zusätzliche `.m`- und `.h`-Dateien, die das Gesamtbild vervollständigen.

Für die meisten Beispiele in diesem Buch wird ein einziger Anwendungsbezeichner verwendet, nämlich `com.sadun.helloworld`, den Sie durch denjenigen aus Ihrem eigenen Profil ersetzen müssen. Die Beschränkung auf diesen einen Bezeichner erfolgt, um Ihr iPhone nicht mit Dutzenden von Beispielen auf einmal vollzustopfen. Jedes Beispiel ersetzt das vorherige, sodass SpringBoard übersichtlich bleibt. Wenn Sie mehrere Beispiele nebeneinander installieren möchten, ändern Sie einfach den Bezeichner, indem Sie ein einmaliges Suffix anhängen, z. B. `com.sadun.helloworld.tableedits`.

V.6.1 Den Beispielcode abrufen

Den Quellcode zu diesem Buch finden auf der Open-Source-Hostingsite GitHub unter *http://github.com/erica/iphone-3.0-cookbook-/tree*. Dort sehen Sie eine nach Kapiteln gegliederte Sammlung von Quellcode mit funktionierenden Beispielen zu den in diesem Buch behandelten Themen.

Beispielcode ist niemals endgültig fertig, sondern entwickelt sich weiter, wenn Apple das SDK und die Cocoa Touch-Bibliotheken erweitert. Beteiligen Sie sich daran, indem Sie z. B. Lösungen für Bugs und andere Korrekturen vorschlagen oder den angebotenen Code erweitern. Auf GitHub können Sie Repositorys verzweigen und durch Ihre eigenen Verbesserungen und Funktionen erweitern, die Sie dann wieder im Hauptrepository zur Verfügung stellen können. Wenn Sie eine neue Idee oder einen neuen Ansatz haben, lassen Sie es uns wissen. Wir freuen uns, großartige Vorschläge sowohl in das Repository als auch in die nächste Ausgabe dieses Buches aufzunehmen.

V.6.2 Git beziehen

Sie können den Quellcode zu diesem Buch mit dem Versionssteuerungssystem git herunterladen. Eine Mac OS X-Implementierung von git finden Sie unter *http://code.google.com/p/git-osx-installer*. Es gibt sowohl Kommandozeilen- als auch GUI-Versionen. Suchen Sie also nach der Implementierung, die sich für Sie am besten eignet.

V.6.3 GitHub nutzen

GitHub (*http://github.com*) ist die größte git-Hostingwebsite mit mehr als 150.000 öffentlichen Repositorys. Sie bietet kostenloses Hosting für öffentliche Projekte sowie gebührenpflichtige Optionen für private. Mit einer eigenen Webschnittstelle, die Wiki-Hosting und Problemnachverfolgung bietet und großes Gewicht auf die Vernetzung der Projektentwickler legt, ist dies ein hervorragender Ort, um neuen Code zu finden oder an bestehenden Bibliotheken mitzuarbeiten. Auf der Website von GitHub können Sie sich für einen kostenlosen Account registrieren, um das Repository zu diesem Entwicklerbuch zu kopieren und zu verändern oder um Ihre eigenen Open-Source-Projekte für das iPhone zu erstellen und anderen zugänglich zu machen.

V.7 Kontakt mit der Autorin

Sollten Sie Fragen oder Kommentare zu diesem Buch haben, senden Sie mir eine E-Mail (in englischer Sprache) an *erica@ericasadun.com*, oder besuchen Sie *www.ericasadun.com*. Diese Website stellt viele der Programme dieses Buchs bereit. Scheuen Sie sich nicht, Software herunterzuladen, Dokumentationen zu lesen und Ihre Kommentare zu hinterlassen.

1

Einführung in das iPhone SDK

Das iPhone und der iPod touch sind mobile Plattformen, für die es Spaß macht, zu programmieren. Es sind die ersten Mitglieder der neuen Apple-Produktreihe von Taschencomputern. Trotz ihrer geringen Abmessungen läuft auf ihnen eine OS X-Version erster Klasse, für die es ein reichhaltiges und abwechslungsreiches SDK gibt, mit dem Sie eine breite Palette von Programmen entwerfen, implementieren und realisieren können. Die Vorteile der Multitouch-Oberfläche des iPhones und der leistungsfähigen Onboard-Funktionen können Sie mithilfe von Xcode, der integrierten Entwicklungsumgebung von Apple, für Ihre Projekte nutzen. In diesem Kapitel lernen Sie die Bestandteile des SDK kennen und erfahren mehr über das, was Sie damit erstellen – nämlich iPhone-Programme. Des Weiteren geht es um die verschiedenen iPhone-Entwicklerprogramme von Apple und darum, wie Sie ihnen beitreten können. Sie lernen die Entwurfsprinzipien für iPhone-Programme kennen und sehen, wie Anwendungen aufgebaut werden. Schließlich lesen Sie auch, wie Sie die Anmeldeinformationen für das Entwicklerprogramm erhalten, damit Sie die Prinzipien umsetzen und mit dem Programmieren beginnen können.

1.1 iPhone-Entwicklerprogramme

Sind Sie bereit, mit der Programmierung für das iPhone zu beginnen? Sind Sie bereit zu erfahren, worum der ganze Rummel geht? Das iPhone SDK (Software Development Kit) steht für die Mitglieder der vier iPhone-Entwicklerprogramme von Apple zur Verfügung – dem kostenlosen Onlineprogramm, dem kostenpflichtigen Unternehmensprogramm für die interne Entwicklung, dem ebenfalls kostenpflichtigen Standardprogramm, das es den Entwicklern erlaubt, ihre Produkte im App Store einzureichen, und einem besonderen Programm für Universitäten (siehe *Tabelle 1.1*).

Programm	Kosten	Mögliche Teilnehmer
Online Developer Program	Kostenlos	Alle, die das iPhone SDK unverbindlich kennenlernen möchten
Standard iPhone Developer Program	79 EUR pro Jahr	Entwickler, die ihre Programme über den App Store verkaufen möchten
Enterprise iPhone Developer Program	279 EUR pro Jahr	Große Unternehmen, die firmeneigene Software für ihre Angestellten entwickeln
University iPhone Developer Program	Kostenlos	Kostenloses Programm für höhere Bildungseinrichtungen, bei denen die iPhone-Entwicklung auf dem Lehrplan steht

▸ *Tabelle 1.1: iPhone-Entwicklerprogramme*

Jedes Programm bietet Zugriff auf das iPhone SDK, mit dem Sie die Möglichkeit bekommen, Programme zu schreiben und bereitzustellen. Jedes Programm ist für eine bestimmte Art von Teilnehmern gedacht.

1.1.1 Online Developer Program

Dieses kostenlose Programm ist für alle gedacht, die die Programmierumgebung des iPhone SDK kennenlernen, aber nicht für weitergehende Rechte bezahlen möchten. Hierbei sind Sie auf die reine Programmierung auf dem Mac beschränkt. Sie können Ihre Anwendungen zwar im Simulator ausführen, aber nicht auf einem physischen Gerät bereitstellen oder im App Store verkaufen.

Der Simulator nähert sich zwar mit jeder weiteren Version immer stärker der Leistung auf einem tatsächlichen Gerät an, doch sollten Sie sich zum Testen einer Anwendung nicht darauf verlassen. Ein Programm, das zuverlässig auf dem Simulator läuft, kann auf dem physischen Gerät zu reagieren aufhören oder sogar abstürzen. Beispielsweise kann der Simulator weder Vibrationen noch den Beschleunigungsmesser verwenden. Solche und andere Funktionen des Geräts stehen auf dem Simulator nicht zur Verfügung. Mehr über die Einschränkungen des Simulators erfahren Sie weiter hinten in diesem Kapitel in Abschnitt *Einschränkungen des Simulators*.

1.1.2 Standard Developer Program

Um Rechte für die Bereitstellung auf Geräten und für den Verkauf zu erlangen, müssen Sie eine Jahresgebühr von 79 EUR für das standardmäßige iPhone-Entwicklerprogramm bezahlen. Nach der ersten Zahlung erhalten Sie die Möglichkeit, Ihre Software über den App Store zu verkaufen, und können Ihre Software auf echter iPhone-Hardware testen. Dieses Programm erlaubt auch die

Ad-hoc-Verteilung, womit Sie Vorabversionen Ihrer Anwendung auf bis zu 100 registrierte Geräte übertragen können. Das Standardprogramm ist für die Mehrheit der iPhone-Programmierer da, die den App Store nutzen möchten. Wenn Sie mit dem Verkauf von Anwendungen geschäftlich tätig werden wollen, ist dies das Programm, für das Sie sich registrieren müssen.

1.1.3 Enterprise Developer Program

Das Unternehmensprogramm für jährlich 279 EUR ist für die innerbetriebliche Verteilung von Anwendungen gedacht und zielt auf Unternehmen mit mehr als 500 Angestellten ab. Mit der Mitgliedschaft in diesem Programm haben Sie keinen Zugriff auf den iPhone App Store, können aber Ihre firmeneigenen Anwendungen erstellen und über einen privaten Shop an die Geräte Ihrer Mitarbeiter verteilen. Dieses Programm ist für große Unternehmen da, die eigene Anwendungen (z. B. Bestellsysteme) an ihre Mitarbeiter verteilen möchten.

1.1.4 University Developer Program

Das University Developer Program ist nur für höhere Bildungseinrichtungen verfügbar und soll Hochschulen dazu ermutigen, einen Lehrplan für die iPhone-Entwicklung aufzustellen. Dieses Programm erlaubt Professoren und Dozenten, Teams aus bis zu 200 Studenten zu bilden und ihnen Zugriff auf das vollständige iPhone SDK zu geben. Die Studenten können ihre Anwendungen untereinander und mit den Dozenten austauschen, und die Hochschule selbst kann Anwendungen im App Store einreichen.

1.1.5 Registrierung

Für das kostenlose Programm können Sie sich auf der Hauptwebsite für iPhone-Entwickler unter *http://developer.apple.com/iphone* registrieren. Die Einschreibung für die kostenpflichtigen Programme (Standard und Enterprise) erfolgt auf *http://developer.apple.com/programs/iphone/*.

1.2 Erste Schritte

Unabhängig davon, für welches Programm Sie sich entscheiden, müssen Sie Zugriff auf einen Intel-Mac haben, auf dem eine aktuelle Version von Mac OS X läuft. Es ist auch hilfreich, zu Testzwecken mindestens ein iPhone- oder iPod touch-Gerät zur Verfügung zu haben (nach Möglichkeit jedoch mehrere), um sicherzustellen, dass Ihre Anwendungen auf jeder Plattform korrekt funktionieren, auch auf älteren Geräten wie der ersten Generation des iPhones oder iPod touch.

Bei der Registrierung für die kostenpflichtigen Programme können häufig Verzögerungen auftreten. Bis zur Einrichtung des Accounts und der Rechnungsstellung kann es einige Wochen dauern. Sobald Sie gezahlt haben, können noch einmal ein bis drei Tage vergehen, bis Sie Zugriff auf erweiterte Funktionen des Portals haben.

Die Registrierung für iTunes Connect, die Sie brauchen, um Ihre Anwendungen über den App Store verkaufen zu können, stellt eine weitere Hürde dar. Zum Glück können Sie diesen Vorgang zurück-

stellen, bis Sie die Registrierung für eines der kostenpflichtigen Programme abgeschlossen haben. Für iTunes Connect müssen Sie Ihre Bankverbindung und Gewerbeangaben einreichen, bevor Sie einen App Store-Account einrichten können. Außerdem müssen Sie die Absatzverträge von Apple lesen und ihnen zustimmen. Apple führt sämtliche Einzelheiten unter *itunesconnect.apple.com* auf.

1.2.1 Das SDK herunterladen

Das iPhone SDK können Sie von der Hauptwebsite für iPhone-Entwickler unter *http://developer.apple.com/iphone* herunterladen. Geben Sie Ihre Anmeldeinformationen für das Entwicklerprogramm ein, um Zugang zur Downloadseite zu erhalten. Bevor Sie also den Dowload versuchen, müssen Sie sich für eines der Programme registriert haben. Beim kostenlosen Programm haben Sie nur Zugriff auf die vollständig veröffentlichten SDKs, bei den kostenpflichtigen sehen Sie auch schon neue Betaversionen, sodass Sie bereits Anwendungen für kommende iPhone-OS-Versionen schreiben können.

Das Kit, das gewöhnlich wenige Gigabyte groß ist, installiert die komplette Suite der interaktiven Entwurfswerkzeuge auf Ihrem Macintosh. Die Bestandteile dieser Suite bilden die Grundlage der iPhone-Entwicklungsumgebung. Dazu gehört die folgende Software:

- **Xcode** Xcode ist das wichtigste Werkzeug für die iPhone-Entwicklung. Es bildet eine umfassende Umgebung für die Entwicklung und das Management von Projekten samt Quellcodebearbeitung, ausführlicher Dokumentation und grafischem Debugger. Xcode baut auf verschiedenen Open-Source-GNU-Programmen auf, vor allem auf gcc (Compiler) und gdb (Debugger).
- **Interface Builder** Interface Builder (IB) ist ein Werkzeug für die schnelle Prototypentwicklung, mit dem Sie das Layout der Benutzeroberfläche grafisch erstellen und mit vorgefertigten Interfaces aus dem Xcode-Quellcode verlinken können. In IB gestalten Sie die Benutzeroberfläche mit grafischen Entwurfswerkzeugen und verbinden diese Bildschirmelemente mit den Objekten und Methodenaufrufen in Ihrer Anwendung.
- **Simulator** Der iPhone-Simulator wird auf dem Macintosh ausgeführt und ermöglicht es Ihnen, Anwendungen auf Ihrem Desktop zu erstellen und zu testen. Dazu müssen Sie keine Verbindung zu einem iPhone oder iPod touch herstellen. Der Simulator stellt dieselbe API zur Verfügung, die auf dem iPhone verwendet wird, und bietet eine Vorschau darauf, wie Ihre Entwürfe aussehen. Bei der Arbeit mit dem Simulator kompiliert Xcode Intel-x86-Code, der auf dem Macintosh selbst ausgeführt wird, im Gegensatz zu dem auf dem iPhone verwendeten ARM-Code.
- **Instruments** Instruments erstellt ein Profil davon, wie iPhone-Anwendungen unter der Motorhaube funktionieren. Es prüft die Speicherauslastung und überwacht die Leistung. Auf diese Weise können Sie Problembereiche in Ihrer Anwendung erkennen und ihre Funktionsfähigkeit verbessern. Instruments bietet grafische Darstellungen zur Leistungsfähigkeit, die anzeigen, wo Ihre Anwendungen die meisten Ressourcen benötigen. Es basiert auf dem von Sun Microsystems entwickelten Open-Source-Paket DTrace und spielt eine wichtige Rolle dabei, Speicherlecks ausfindig zu machen und sicherzustellen, dass Ihre Anwendung auf der iPhone-Plattform effizient ausgeführt wird.

- **Shark** Shark analysiert, womit eine Anwendung die meiste Zeit verbringt, und dient somit zur Leistungsoptimierung. Diese Software erkennt Engpässe und zeigt sie Ihnen an, sodass Sie die Leistung der Anwendung steigern können.

Mit den Komponenten dieses iPhone SDK sind Sie in der Lage, sowohl herkömmliche als auch webbasierte Anwendungen zu entwickeln. Aus der Sicht eines Entwicklers nativer Anwendungen sind die wichtigsten Komponenten Xcode, Interface Builder und Simulator, wobei Instruments ein wichtiges Optimierungswerkzeug darstellt. Zusätzlich zu diesen Tools gibt es noch einen wichtigen Bestandteil, der in dieser Liste nicht aufgeführt ist. Er ist im SDK enthalten, wird aber leicht übersehen – ich meine Cocoa Touch.

Cocoa Touch ist die Bibliothek der Klassen, die Apple für die schnelle Entwicklung von iPhone-Anwendungen bereitstellt. Diese Bibliothek, die die Form einer Vielzahl von Framework-Bibliotheken annimmt, ermöglicht das Erstellen grafischer ereignisgesteuerter Anwendungen unter Verwendung von Elementen der Benutzerschnittstelle wie Fenstern, Texten oder Tabellen. Cocoa Touch auf dem iPhone entspricht dem AppKit auf Mac OS X und unterstützt den Aufbau leistungsfähiger, wiederverwendbarer Schnittstellen auf dem iPhone.

Viele Entwickler sind von der Größe der iPhone-Anwendungen überrascht – sie sind winzig. Der Hauptgrund hierfür liegt in der Bibliotheksunterstützung durch Cocoa Touch. Dadurch, dass Cocoa Touch die aufwendigen Bewegungen der Benutzeroberfläche abwickelt, können sich Ihre Anwendungen darauf konzentrieren, ihre eigenen Aufgaben zu erledigen. Das Ergebnis ist ein kompakter, konzentrierter Code, der jeweils eine Aufgabe auf einmal ausführt.

Mithilfe von Cocoa Touch können Sie Anwendungen mit einem professionellen Erscheinungsbild erstellen, das mit dem der von Apple entwickelten Anwendungen übereinstimmt. Denken Sie daran, dass Apple Ihre Software genehmigen muss. Dabei bewertet Apple das Erscheinungsbild, den Betrieb und sogar den Inhalt. Wenn Sie Cocoa Touch einsetzen, können Sie eher den hohen Designstandards genügen, die durch die Apple-eigenen Anwendungen festgelegt wurden.

1.2.2 Entwicklungsgeräte

Ein physisches iPhone- oder iPod touch-Gerät ist ein wichtiges Element für die Softwareentwicklung. Tests auf dem iPhone sind unabdingbar. So einfach und bequem der Simulator im SDK auch sein mag, so muss er doch hinter Tests auf einem richtigen iPhone zurückstecken. Da das iPhone die Zielplattform darstellt, ist es wichtig, dass Ihre Software optimal auf dem tatsächlichen System läuft und nicht nur auf dem Simulator. Das iPhone selbst ist die vollständige, unverwässerte Testplattform, die Sie brauchen.

Apple empfiehlt immer wieder, das Entwicklungsgerät ausschließlich für die Entwicklung einzusetzen. In der Praxis ist diese Frage jedoch nicht so eindeutig zu klären. Wenn Sie Ihr iPhone zum ersten Mal über ein USB-Standardkabel an den Computer anschließen, wird es von Xcode erkannt. Wollen Sie das Gerät für die Entwicklung verwenden, bestätigen Sie das; anderenfalls klicken Sie auf **Ignorieren**.

Wenn Sie ein Gerät für die Entwicklung nutzen, unterliegt es Onboard-Datenänderungen, sodass es im Normalbetrieb möglicherweise nicht mehr zuverlässig funktioniert. Wenn Sie ein SDK verwenden, das die ersten frühen Betastadien schon hinter sich hat, halten die Geräte dies aber erfahrungsgemäß aus und können nach wie vor für den regelmäßigen, alltäglichen Gebrauch eingesetzt werden. Es ist immer noch am besten, einige Extrageräte zu verwenden, die ausschließlich für Entwicklungszwecke dienen, aber wenn Sie nicht so viele Geräte haben, können Sie wahrscheinlich auch Ihr eigenes iPhone für die Entwicklung einsetzen. Seien Sie sich aber der Gefahren bewusst!

Bei der Entwicklung ist es wichtig, die Anwendungen auf so vielen iPhone-Plattformen wie möglich zu testen. Seien Sie sich bewusst, dass es echte Plattformunterschiede zwischen den einzelnen Modellen des iPhones und des iPod touch gibt. So hat die zweite Generation des iPod z. B. eingebaute Lautsprecher, die erste nicht. Außerdem verwendet sie einen schnelleren Prozessor. iPhones verfügen über Kameras, die keines der derzeitigen iPod touch-Modelle enthält. Eine Aufstellung der Unterschiede zwischen den Modellen finden Sie weiter hinten in diesem Kapitel.

1.2.3 Einschränkungen des Simulators

Jedes Release des iPhone-Simulators für den Macintosh ist besser als das vorhergehende. Dennoch gibt es deutliche Einschränkungen, die Sie bedenken müssen. Von der Softwarekompatibilität bis zur Hardware nähert sich der Simulator zwar dem tatsächlichen Gerät an, er verhält sich aber nicht identisch.

Der Simulator verwendet viele Macintosh-Frameworks und Bibliotheken und bietet Funktionen, die auf dem iPhone nicht vorhanden sind. Anwendungen, die auf dem Simulator betriebsfähig und fehlerfrei erscheinen, können auf dem Gerät selbst plötzlich aussetzen oder abstürzen. Es ist nicht möglich, ein Programm ausschließlich im Simulator vollständig zu debuggen und dann sicher zu sein, dass die Software fehlerfrei auf dem iPhone läuft.

Außerdem fehlen dem Simulator einige Hardwarefunktionen. So können Sie ihn z. B. nicht einsetzen, um die eingebaute Kamera oder den Beschleunigungsmesser zu testen. Zwar kann der Simulator Beschleunigungsdaten eines Bewegungsmessers verarbeiten, falls der Macintosh damit ausgestattet ist (was bei Laptops gewöhnlich der Fall ist), doch unterscheiden sich diese Messwerte von denen auf dem iPhone und eignen sich nicht für die Entwicklung und für Tests. Der Simulator vibriert nicht und bietet keine Möglichkeit für die Multitouch-Eingabe (zumindest keine, die über die einfache »Kneifgeste« hinausgeht). Core Location ist auf die Koordinaten von 1 Infinite Loop, Cupertino, in Kalifornien festgelegt, den Firmensitz von Apple.

Was die Software angeht, so steht das grundlegende Schlüsselbund-Sicherheitssystem auf dem Simulator nicht zur Verfügung. Sie können auch keine Anwendung registrieren, um Push-Benachrichtigungen zu erhalten. Das Fehlen dieser Elemente bedeutet, dass manche Arten von Programmen nur dann richtig nutzbar sind, wenn sie auf einem iPhone bereitgestellt wurden.

Ein weiterer Unterschied zwischen dem Simulator und dem Gerät ist das Audiosystem. Die Struktur für Audiositzungen ist auf dem Simulator nicht implementiert, was verschleiert, wie kompliziert die Abläufe auf dem Gerät sind. Selbst in den Gebieten, in denen der Simulator die iPhone-APIs tatsächlich emuliert, können Unterschiede im Verhalten auftreten, da der Simulator auf den Cocoa-Frameworks von Mac OS X aufbaut.

Das heißt nicht, dass der Simulator keine wichtige Rolle beim Testen spielt! Sie können ein Programm auf dem Simulator schnell und einfach ausprobieren, was gewöhnlich sehr viel schneller geht, als eine kompilierte Anwendung auf ein iPhone zu übertragen. Auf dem Simulator können Sie das virtuelle Gerät drehen, um die Neuausrichtung zu testen, simulierte Speicherwarnungen ausgeben und Ihre Benutzeroberfläche so ausprobieren, als wären Sie ein Benutzer, der einen Telefonanruf empfängt. Es ist sehr viel einfacher, Textverarbeitung auf dem Simulator zu testen, da Sie die Tastatur verwenden können. Dies erleichtert die wiederholte Texteingabe z. B. von Account-Namen und Kennwörtern für Anwendungen, die Verbindung mit dem Internet aufnehmen.

Unter dem Strich ist der Simulator ein Kompromiss. Sie gewinnen eine Menge Komfort für Ihre Tests, aber das reicht nicht aus, um auf Tests auf dem eigentlichen Gerät zu verzichten.

1.2.4 Anbindung

Alle interaktiven Tests müssen über ein USB-Kabel erfolgen. Zurzeit bietet Apple keine Möglichkeit an, um Anwendungen drahtlos zu übertragen, zu debuggen oder zu überwachen. Das bedeutet, dass Sie fast die gesamte Arbeit über eine Anbindung mithilfe eines standardmäßigen iPhone-USB-Kabels durchführen müssen. Die physischen Umstände des Debuggings via Kabel können jedoch Probleme hervorrufen, z. B. aus den folgenden Gründen:

- Wenn Sie das Kabel abziehen, trennen Sie alle interaktiven Funktionen für Debugging, Konsolenzugriff und Screenshots. Das Kabel muss also die ganze Zeit über eingesteckt sein.
- Sie können das iPhone nicht sinnvoll mit einem Dock verwenden. Das Dock ist zwar stabil, aber wenn Sie zum Test der Benutzeroberfläche den Bildschirm berühren müssen, ist das äußerst unbequem, da das iPhone im Winkel von 75° steht.
- Die Anbindung erfolgt unten, nicht oben am Gerät, was leicht dazu führen kann, dass man sich im Kabel verheddert und das iPhone zu Boden reißt.

Tests ohne Kabelanbindung würden viele dieser Schwierigkeiten deutlich mildern. Leider bietet Apple zurzeit noch keine Möglichkeit dafür. Wenn Sie wollen, können Sie Ihr iPhone auch mit kühnen Anbauten versehen, um solche Probleme zu lösen. Eine Möglichkeit besteht darin, an der Rückseite einer iPhone-Tasche – die den unteren Anschluss freilässt – Klettband anzubringen und das Gerät damit auf dem Tisch zu fixieren. Das sieht hässlich aus, hilft aber zu vermeiden, dass das iPhone dauernd auf dem Fußboden landet. Sie können auch Halterungen von Drittherstellern für das iPhone erwerben, um die Entwicklungsarbeit zu erleichtern. In diesen Aufhängungen liegt das iPhone einige Zentimeter oberhalb des Schreibtischs, wobei das Kabel nach hinten geführt wird.

Versuchen Sie nach Möglichkeit, das Gerät direkt mit einem Anschluss auf Ihrem Mac zu verbinden, um die besten Ergebnisse zu erzielen. Wenn Sie einen Hub verwenden müssen, schließen Sie ihn an ein System an, das USB 2.0 mit Stromversorgung unterstützt. Bei den meisten älteren Tastaturen und Monitoren sind nur USB 1.0-Verbindungen ohne Stromversorgung möglich. Beim Testen ist es hilfreich, wenn Sie sich auf einen USB 2.0-Anschluss mit Stromversorgung verlassen können.

1.3 Unterschiede zwischen den Modellen

Bei der Entwicklung von iPhone-Anwendungen müssen Sie sich meistens keine Gedanken über die Plattform machen, auf der Ihre Programme ausgeführt werden. Die meisten Anwendungen nutzen lediglich die Anzeige und die Berührungseingabe und können problemlos auf allen vier Geräten bereitgestellt werden, aus denen die iPhone-Produktreihe zurzeit besteht. Weder besondere Programmierung noch Überlegungen zur Zielplattform sind erforderlich.

Es gibt jedoch Unterschiede zwischen den Plattformen, die sowohl bedeutend als auch beachtenswert sind. Sie spielen eine Rolle bei der Entscheidung, wie der App Store Ihre Software vermarkten soll und wie Sie sie überhaupt entwerfen. Wollen Sie die Software nur für das iPhone bereitstellen? Für das iPhone und die zweite Generation des iPod touch (oder höher)? Oder soll sie für alle Plattformen geeignet sein? Dabei müssen Sie die Punkte beachten, die in den folgenden Abschnitten erläutert werden.

1.3.1 Kamera

Jedes iPhone ist mit einer Kamera ausgestattet, der iPod touch dagegen nicht. Diese Kameras sind nützlich: Sie können damit Aufnahmen machen und an Flickr oder Twitter senden, Sie können Bilder für die direkte Bearbeitung abrufen usw. Das iPhone SDK hat einen integrierten Auswahlcontroller, über den Ihre Benutzer Zugriff auf die Kamera bekommen, aber nur auf Plattformen mit Kameras. Videodienste gibt es nur für das Modell 3G S und höher.

Wenn Sie Anwendungen mit Kameranutzung erstellen, können Sie sie nicht auf iPods bereitstellen – Kameradienste sind auf die iPhone-Produktreihe beschränkt. Mit der eingebauten 2-Megapixel-Kamera der iPhones der ersten und zweiten Generation können Sie keinen Blumentopf gewinnen, aber die Kamera der dritten Generation ist stark verbessert, bietet Autofokus und Makrofotografie, Videoaufzeichnung sowie eine erhöhte Lichtempfindlichkeit bei Dunkelheit.

1.3.2 Lautsprecher und Mikrofon

Dem iPod touch der ersten Generation (1G) fehlt der eingebaute Lautsprecher, den es im iPhone und im iPod touch der zweiten Generation gibt. Zwar können Sie am iPod touch 1G Lautsprecher von Drittherstellern betreiben, die Sie über die untere Buchse anschließen, doch betrachtet Apple dies als unzulässiges Zubehör. Solche Lautsprecher werden auch selten verwendet.

Gehen Sie nicht davon aus, dass die Endbenutzer Kopfhörer tragen, wenn sie Ihre Anwendungen einsetzen. Bei der Entwicklung für den iPod der ersten Generation müssen Sie die Rolle akustischer Hinweise sorgfältig bedenken. Wenn sie für das Programm von entscheidender Bedeutung sind, sollten Sie die Verwendung von Kopfhörern empfehlen oder auf den iPod 1G als Absatzplattform verzichten.

An die zweite Generation des iPod touch können Sie ein externes Headset-Mikrofon anschließen, bei der ersten Generation geht das nicht. Wenn Sie eine Anwendung zur Tonaufzeichnung bereitstellen, müssen Sie ausdrücklich darauf hinweisen, dass ein iPod Zubehör benötigt, um diese Funktionen nutzen zu können.

Das iPhone 3G S der dritten Generation bietet eine Reihe von Zugriffsfunktionen wie Sprachsteuerung. Während ich diese Zeilen schreibe, ist es noch nicht klar, ob die APIs für die Sprachsteuerung für iPhone-Entwickler zugänglich gemacht werden.

1.3.3 Telefonie

Es mag offensichtlich erscheinen, aber das Telefoniesystem des iPhones, das sowohl Telefonanrufe als auch SMS-Nachrichten handhabt, unterbricht Anwendungen, wenn das Gerät einen Telefonanruf empfängt. Sowohl auf dem iPod als auch auf dem iPhone können Benutzer eine Anwendung jederzeit benden, aber nur auf dem iPhone müssen Sie sich um die Art der Beendigung kümmern, die nicht der Benutzer wählt, sondern das System erzwingt.

Bedenken Sie, wie sich die verschiedenen Arten von Unterbrechungen auf Ihre Anwendung auswirken können. Beim Entwickeln der Software ist es wichtig, stets alle möglichen Arten von Beendigungen zu berücksichtigen. Seien Sie sich bewusst, dass nicht immer der Benutzer die Entscheidung trifft, die Anwendung zu verlassen, vor allem nicht auf dem iPhone.

Eine weitere Nebenwirkung des Telefoniebetriebs besteht darin, dass auf einem iPhone sehr viel mehr Dinge im Hintergrund ausgeführt werden als auf einem iPod touch. Das bedeutet, dass die Menge des freien Speichers auf dem iPhone wahrscheinlich geringer ist als auf dem iPod touch. Dies ist einer der Gründe dafür, warum Sie das iPhone als Hauptentwicklungswerkzeug gegenüber dem iPod touch bevorzugen sollten. Wenn Sie mit den stärkeren Einschränkungen des iPhones arbeiten, können Sie Software erstellen, die sowohl auf dem iPhone als auch auf dem iPod touch stabil läuft.

1.3.4 Unterschiede bei Core Location

Core Location stützt sich auf drei verschiedene Ansätze, deren Vorhandensein von der jeweiligen Plattform abhängt und die durch die eingebauten Fähigkeiten des jeweiligen Geräts beschränkt sind. Die Wi-Fi-Ortsbestimmung, die lokale Router wahrnimmt und eine zentrale Positionsdatenbank nach deren MAC-Adressen durchsucht, ist auf allen iPhone- und iPod touch-Plattformen frei verfügbar.

Für die Mobilfunkortung jedoch ist eine Antenne erforderlich, die es nur auf dem iPhone gibt. Hierbei wird eine Dreieckspeilung über örtliche Mobilfunkmasten durchgeführt, deren Positionen den Mobilfunkanbietern bekannt sind. Die letzte und genaueste Möglichkeit, die GPS-Ortung, ist nur auf iPhones der zweiten Generation und höher verfügbar. GPS ist nicht auf iPhones der ersten Generation erhältlich und steht auf iPod touch-Geräten zurzeit ebenfalls nicht zur Verfügung.

Das iPhone 3G S der dritten Generation hat erstmalig einen eingebauten Kompass (der über ein Magnetometer funktioniert) und die Core Location-APIs zu dessen Unterstützung.

1.3.5 Vibration und Annäherungssensor

Vibrationen, die eine fühlbare Rückmeldung für viele Spiele ermöglichen, sind auf das iPhone beschränkt. Der iPod touch gibt keine Vibrationssignale und hat auch keinen Annäherungssensor, der wie beim iPhone den Bildschirm ausschalten würde, wenn Sie das Gerät während eines Telefonats ans

Ohr halten. Vor SDK 3.0 war die Verwendung des Annäherungssensors in Ihren eigenen Anwendungen theoretisch nicht möglich, obwohl er in einer Reihe von App Store-Produkten genutzt wurde, vor allem in der mobilen Google-Anwendung (*http://itunes.com/apps/googlemobileapp*). Ab Version 3.0 bietet die Klasse `UIDevice` direkten Zugriff auf den aktuellen Status des Annäherungssensors.

1.3.6 Prozessorgeschwindigkeiten

Der iPod touch der zweiten Generation verfügt über einen Prozessor mit 532 MHz. Damit wies er die höchste Rechenleistung der Produktreihe auf, bis er vom iPhone 3G S mit 600 MHz überrundet wurde. Testen Sie Ihre Software sowohl auf älteren, langsameren Geräten als auch auf jüngeren. Die Reaktionszeit einer Anwendung hängt von dem Gerät ab, auf dem Sie sie ausführen.

Wenn Ihre Anwendung auf einer älteren Plattform zu langsam reagiert, sollten Sie noch etwas an ihrer Leistungsfähigkeit feilen. Es gibt im App Store zurzeit keine Möglichkeit, die erste Generation des iPhones vom potenziellen Absatzmarkt auszuschließen.

1.3.7 OpenGL ES

OpenGL ES ist eine lizenzfreie, plattformübergreifende API zur Entwicklung von zwei- und dreidimensionalen Grafiken und wird als Teil des iPhone SDK mitgeliefert. Nicht alle iPhone-Modelle bieten die gleiche Unterstützung für OpenGL ES. Das iPhone 3G S und jüngere Modelle führen sowohl OpenGL ES 2.0 als auch 1.1 aus, ältere Modelle wie das iPhone 2G und 3G sowie der iPod touch der ersten und zweiten Generation nur OpenGL ES 1.1. Die Version 2.0 der API bietet bessere Möglichkeiten für Schattierungen und Text, womit qualitativ hochwertigere Grafiken möglich sind.

Bei der Entwicklung für alle iPhones dürfen Sie Ihre Grafiken nur mit Version 1.1 erstellen. Anwendungen, die die Version 2.0 der API verwenden, sind auf das iPhone 3G S und zukünftige Modelle beschränkt.

1.4 Einschränkungen der Plattform

Wenn man über mobile Plattformen wie das iPhone spricht, tauchen immer einige Bedenken auf, z. B. in Hinblick auf Speicher, Interaktionsbeschränkungen und Batterielebensdauer. Mobile Plattformen können nicht dieselbe Festplattengröße anbieten wie ihre Gegenstücke für den Schreibtisch. Neben den Speicherbeschränkungen legen auch beschränkte Anzeigemöglichkeiten und der Energieverbrauch Ihnen als Entwickler spürbare Beschränkungen auf.

Bei der Entwicklung für das iPhone können Sie nicht für einen großen Bildschirm, eine Maus, eine physische Tastatur (noch nicht) und nicht einmal für eine stets eingeschaltete Wechselstromversorgung programmieren. Stattdessen legen die Einschränkungen der Plattform Ihre Entwicklung fest und leiten sie. Glücklicherweise hat Apple gute Arbeit geleistet und eine neue Plattform entworfen, die trotz ihres begrenzten Speichers, der begrenzten Interaktionssteuerung und der begrenzten Batterielebensdauer eine gewisse Flexibilität bietet.

1.4.1 Grenzen der Speicherkapazität

Das iPhone führt eine leistungsfähige, aber dennoch kompakte OS X-Installation aus. Obwohl das gesamte Betriebssystem des iPhones nur einige Hundert Megabyte benötigt – fast nichts im Vergleich zu den heutigen großen Betriebssysteminstallationen –, bietet es eine umfangreiche Framework-Bibliothek. Dieses Framework aus vorkompilierten Routinen ermöglicht den Benutzern des iPhones, eine mannigfache Sammlung kompakter Anwendungen auszuführen, von der Telefonie zur Audiowiedergabe, von E-Mail zum Webbrowsing. Das iPhone bietet gerade genügend Programmierunterstützung zum Erstellen flexibler Schnittstellen, während Systemdateien so beschnitten werden, dass sie problemlos in die engen Speicherkapazitäten passen.

Hinweis

Anwendungen sind jeweils auf eine Maximalgröße von 2 Gbyte beschränkt. Meines Wissens ist keine Anwendung dieser Grenze jemals nahegekommen. Viele Benutzer beschweren sich schon, wenn eine Anwendung 10 Mbyte übersteigt.

1.4.2 Grenzen des Datenzugriffs

Jede iPhone-Anwendung wird in einer Sandbox ausgeführt. Das bedeutet, dass sie in einem streng geregelten Bereich des Betriebssystems aktiv ist. Ihr Programm kann nicht auf andere Anwendungen und nicht auf bestimmte Ordner zugreifen. Diese Einschränkungen verhindern zurzeit z. B. die Arbeit mit der iTunes-Bibliothek und dem Kalender. Ihr Programm kann jedoch alle Daten erreichen, die frei im Internet verfügbar sind, wenn das iPhone mit einem Netzwerk verbunden ist. Neu in Version 3.0 ist auch die Möglichkeit, auf eine gemeinsam genutzte, systemweite Zwischenablage zuzugreifen. Seit dem iPhone SDK 3.2, das im Zusammenhang mit dem iPad veröffentlicht wurde, ist es möglich Daten der Sandbox mit einem Desktop-PC auszutauschen.

1.4.3 Grenzen des Arbeitsspeichers

Die Speicherverwaltung auf dem iPhone ist kritisch. Es kann keinen virtuellen Speicher auf einer Festplatte nutzen. Wenn der Speicher aufgebraucht ist, führt das iPhone einen Neustart durch – wie Apple es ausdrückt. Willkürliche Neustarts sind sicherlich nicht die Benutzererfahrung, auf die wir gewartet haben. Ohne Auslagerungsdatei müssen Sie Ihre Speicheranforderungen sorgfältig verwalten und darauf vorbereitet sein, dass das iPhone-Betriebssystem Ihre Anwendung beendet, wenn sie zu viel Arbeitsspeicher auf einmal verbraucht. Sie müssen auch darauf achten, welche Ressourcen Ihre Anwendung nutzt. Zu viele hochaufgelöste Bilder oder Audiodateien bringen Ihre Anwendungen in den Bereich, in dem sie automatisch beendet werden.

Hinweis

Xcode optimiert Ihre PNG-Bilder automatisch mithilfe des im SDK enthaltenen Hilfsprogramms *pngcrush*. (Sie finden es in den `iPhoneOS`-Plattformordnern in `/Developer`). Führen Sie es auf der Kommandozeile mit dem Schalter `-iphone` aus, um Standard-PNG-Dateien in für das iPhone formatierte Dateien umzuwandeln. Setzen Sie aus diesem Grund in Ihren iPhone-Anwendungen nach Möglichkeit PNG-Dateien als bevorzugtes Bildformat ein.

Das Open-Source-Hilfsprogramm *fixpng*, das Sie unter *http://www.cyberhq.nl* finden, geht genau umgekehrt vor. Es überträgt die komprimierten Bilder zurück in ein Mac-freundliches Format. Damit ist es ein wertvolles Werkzeug für die iPhone-Entwicklung. Die verdienstvolle Anwendung Graphics Convert (*http://lemkesoft.com*, 35 EUR) bietet auch PNG-Unterstützung für das iPhone.

1.4.4 Grenzen der Interaktion

Der Verlust der physischen Eingabegeräte und die Arbeit mit einem kleinen Bildschirm bedeuten nicht, dass Sie die Flexibilität der Interaktion verlieren. Mit Multitouch können Sie Benutzerschnittstellen erstellen, die sich über die Regeln hinwegsetzen. Die Berührtechnologie des iPhones bedeutet, dass Sie Anwendungen mit Texteingaben und Zeigersteuerung unter Verwendung eines virtuellen Bildschirms entwerfen können, der viel größer ist als derjenige, den Sie physisch in Ihrer Hand halten.

Eine elegante, selbstkorrigierende Bildschirmtastatur, ein eingebautes Mikrofon (für alle Geräte außer der ersten Generation des iPod touch) und ein Beschleunigungsmesser, der die Orientierung erkennt, sind nur zwei der Schlüsseltechnologien, die das iPhone vom Rest der mobilen Rechnerwelt unterscheiden. Das bedeutet jedoch, dass Sie Dinge wie Texteingabe und scrollende Fenster einschränken müssen.

Richten Sie Ihr Design auf einfach einzutippende Schnittstellen aus, anstatt den Desktop nachzuahmen. Denken Sie daran, dass Sie nur ein Fenster zur selben Zeit nutzen können – im Gegensatz zu Desktopanwendungen, die mehrere Fenster gleichzeitig öffnen können.

Hinweis

Der Bildschirm des iPhones unterstützt bis zu fünf Berührungen gleichzeitig, obwohl es schwierig ist, eine Anwendung zu finden, die mehr als zwei gleichzeitig nutzt.

1.4.5 Energieeinschränkungen

Bei mobilen Plattformen können Sie die Energieeinschränkungen nicht ignorieren. Andererseits helfen Ihnen die Funktionen des SDKs von Apple dabei, Anwendungen so zu entwerfen, dass sie die CPU-Verwendung einschränken und einen schnellen Batterieverbrauch vermeiden. Ein geschickter Einsatz dieser Technologie (beispielsweise das saubere Beenden von Programmen) sorgt dafür, dass Ihre Anwendung brav auf dem iPhone mitspielt und keine Löcher in die Taschen der Benutzer brennt (manchmal fast im wahrsten Sinne des Wortes). Einige Programme verursachen bei längerer Ausführung eine derart hohe Abwärme, dass das Telefon beim Berühren heiß ist und die Batterie schnell verbraucht wird. Die Kameraanwendung ist ein bemerkenswertes Beispiel.

1.4.6 Einschränkungen für Anwendungen

Apple hat die strenge Richtlinie »nur jeweils eine Anwendung auf einmal« herausgegeben. Das bedeutet, dass Sie als Fremdentwickler keine Anwendungen entwickeln können, die wie die Dienstprogramme Mail und Telefon von Apple im Hintergrund ausgeführt werden. Jedes Mal, wenn Ihre Anwendung ausgeführt wird, muss sie aufräumen und bildlich gesehen aus dem Weg gehen, bevor sie die Kontrolle an die nächste vom Benutzer ausgewählte Anwendung übergibt. Sie können keinen Daemon dauerhaft ausführen, der nach neuen Nachrichten sucht oder regelmäßige Aktualisierungen aussendet.

Andererseits unterstützt Apple mit der Firmware 3.0 Push-Daten von Webdiensten. Registrierte Dienste können Benutzern Nachrichten übermitteln und dadurch Badge-Nummern oder Sound- bzw. Textnotifizierung dem Anwender mitteilen. Dadurch wissen Sie z. B., dass Daten auf diesen Servern warten. In Kapitel 16, *Push-Benachrichtigungen*, erfahren Sie mehr über Push-Benachrichtigungen und lernen, wie Sie solche Meldungen an Benutzer senden.

Hinweis

Gemäß den Nutzungsrichtlinien des iPhones dürfen Sie die Plug-In-Architektur von Cocoa Touch nicht für Anwendungen einsetzen, die Sie im App Store einreichen. Sie können statische Bibliotheken erstellen, die Sie zur Kompilierungszeit einschließen, aber keine Programmiertechniken verwenden, die zur Laufzeit zu willkürlichem Code verlinken.

1.4.7 Einschränkungen aufgrund des Benutzerverhaltens

Obwohl es keine physische gerätebasierte Einschränkung ist, sollten Sie sich daran gewöhnen, dass iPhone-Benutzer Anwendungen auf Telefonen sporadisch nutzen. Sie laden ein Programm, nutzen es kurz und verlassen es dann schnell. Die Natur der Handheld-Geräte bedeutet, dass Sie Ihre Anwendung für kurze Interaktionszeiträume entwerfen und darauf vorbereitet sein müssen, dass sie abgeschaltet wird, wenn der Benutzer das Telefon zurück in die Tasche steckt. Speichern Sie den Zustand Ihrer Anwendung zwischen den Sitzungen, und starten Sie schnell neu mit ungefähr derselben Aufgabe, die der Benutzer beim letzten Mal ausgeführt hat. Dies erfordert Sorgfalt vom Programmierer, aber die Zeitinvestition zahlt sich in Form erhöhter Benutzerzufriedenheit aus.

1.5 Einschränkungen des SDK

Wie Sie vermutlich erwartet haben, erfolgt das Erstellen von Anwendungen für das iPhone ähnlich wie für den Macintosh: Auf beiden Plattformen läuft eine Version von OS X. Sie verwenden Objective-C 2.0 und kompilieren durch Verlinkung mit einer Auswahl von Frameworks. Mit anderen Worten, das iPhone SDK ist beschränkt. Die folgenden Schlüsseleigenschaften sollten Sie im Hinterkopf behalten:

- Die Garbage Collection fehlt, und dabei wird es vermutlich immer bleiben. Auf dem iPhone sind Sie dafür verantwortlich, Objekte im Speicher beizubehalten und freizugeben. Das Fehlen der Garbage Collection lässt sich auf zwei Gründe zurückführen. Erstens erfordert eine eingeschränkte mobile Plattform wie das iPhone genaue Leistungsmerkmale, vor allem für prozessorintensive Anwendungen wie Spiele. Eine Garbage Collection führt ein unvorhersagbares Element in die Leistung ein, denn sie muss Threads einfrieren, wenn sie den Speicher aufräumt. Zweitens erlaubt der eingeschränkte Arbeitsspeicher keine sinnvolle und nützliche Implementierung einer Garbage Collection. Anwendungen mit Garbage Collection setzen die Messlatte für die Speichernutzung viel höher an. Anwendungen erfahren dadurch ein häufigeres Herunterfahren des Betriebssystems.
- Viele Bibliotheken sind nur teilweise implementiert. Die Kernanimation ist durch das Quartz Core-Framework teilweise verfügbar, aber viele Klassen und Methoden fehlen. Das ist der Preis dafür, dass Sie mit der frühen Version einer Software arbeiten. Umgehen Sie die fehlenden Teile, und reichen Sie Ihre Fehlerberichte bei Apple ein, damit man dort (hoffentlich) die benötigten Elemente ergänzt. Beachten Sie, dass Apple den Zugriff auf einige proprietäre Klassen absichtlich verhindert. Beispielsweise können Sie die EXIF-Ausrichtung aus Bildern auslesen, können diese Daten aber nicht hinzufügen, da die Methode dazu nicht veröffentlicht ist.

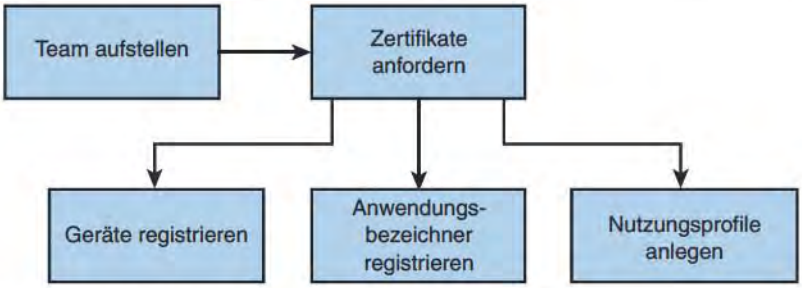
Hinweis

Der Compiler von Xcode erlaubt es, in einem Projekt sowohl C++- als auch Objective-C-Code zu verwenden. In dem resultierenden Objective-C++-Hybridprojekt können Sie vorhandene C++-Bibliotheken in Objective-C-Anwendungen wiederverwenden. Einzelheiten entnehmen Sie der Dokumentation von Apple.

1.6 Das Entwicklerportal nutzen

Das iPhone-Entwicklerportal (iPhone Provisioning Portal) beherbergt alle Werkzeuge, die Sie brauchen, um Ihr System zur iPhone-Entwicklung einzurichten. Sie finden es unter *http://developer.apple.com/iphone/manage/overview/index.action*, haben aber keinen Zugriff darauf, solange Sie sich nicht für eines der kostenpflichtigen iPhone-Entwicklerprogramme angemeldet haben. Hier können Sie Ihr Entwicklungsteam aufstellen, Zertifikate erwerben, Entwicklungsgeräte und Anwendungsbezeichner erwerben und Nutzungsprofile (Provisioning-Profile) anlegen, um Ihre Anwendungen korrekt zu signieren.

Da sich die Einzelheiten ändern können, gebe ich Ihnen hier nur einen Überblick. Sollte Apple einzelne Verfahren ändern, kennen Sie immer noch den allgemeinen Verlauf, sodass Sie sich entsprechend umorientieren können. *Abbildung 1.1* zeigt die Schlüsselelemente des Vorgangs.



► *Abbildung 1.1: Grundfunktionen des iPhone-Entwicklerportals*

1.6.1 Das Team aufstellen

Ein iPhone-Entwicklungsteam muss mindestens ein Mitglied haben. Das Hauptmitglied, der sogenannte »Agent«, ist die Person, die sich für das iPhone-Entwicklungsprogramm registriert hat. Er hat administrative Rechte für den Account und kann andere Mitglieder in das Team aufnehmen (falls es sich nicht um einen Account für eine Einzelperson handelt), Zertifikate anfordern usw. Außerdem kann der Agent administrative Rechte an andere Mitglieder verleihen, die dann, was nicht überraschen dürfte, »Admins« genannt werden. Mitglieder ohne Administratorrechte können neue Nutzungsprofile anfordern und herunterladen, aber das ist auch schon so gut wie alles.

Administratoren können mithilfe des Teambildschirms neue Mitglieder zum Portal einladen. Dies ist auch der Bildschirm, auf dem Sie E-Mails aktualisieren, Zertifikate prüfen und Mitglieder hinzufügen und entfernen können. Auf den weiteren Registerkarten dieses Bildschirms können Sie Ihre Inanspruchnahme des technischen Supports und Ihre Entwicklerverträge mit Apple einsehen.

1.6.2 Zertifikate anfordern

Zertifikate spielen eine bedeutende Rolle bei der iPhone-Entwicklung. Ohne gültiges Entwicklungszertifikat können Sie keine Anwendungen auf iPhones bereitstellen, nicht einmal zum Testen. Außerdem brauchen Sie ein Verteilungszertifikat, um Anwendungen über den App Store verkaufen zu können. Diese Zertifikate können Sie im Portal anfordern und herunterladen.

Als Erstes erstellen Sie in der Schlüsselbundverwaltung auf dem Macintosh eine Zertifikatanforderung:

1. Starten Sie das Programm im Ordner `/Programme/Dienstprogramme`.
2. Wählen Sie aus dem Hauptmenü **Schlüsselbundverwaltung ► Zertifikatsassistent ► Zertifikat einer Zertifizierungsinstanz anfordern**. Überprüfen Sie die E-Mail-Adresse. Es sollte sich dabei um die E-Mail-Adresse handeln, die Sie zu Ihrer Registrierung beim iPhone-

Entwicklerprogramm benutzt haben. Wählen Sie **Auf der Festplatte sichern**, und klicken Sie auf **Fortfahren**.

3. Wählen Sie, wo das Zertifikat gespeichert werden soll (der Schreibtisch eignet sich gut dafür), und klicken Sie auf **Sichern**. Warten Sie darauf, dass das Zertifikat generiert wird, und klicken Sie auf **Fertig**.

Anschließend laden Sie die Anforderung zum Portal hoch, um entweder ein Entwicklungs- oder ein Verteilungszertifikat zu erstellen. Das Portal führt Sie durch den Vorgang. Jedes Zertifikat muss vom Teamagenten genehmigt werden, bevor es ausgestellt wird. Sobald es genehmigt ist, können Sie es vom Zertifikatfenster der Portalwebsite herunterladen.

Installieren Sie das neue Zertifikat in Ihrem Schlüsselbund, indem Sie darauf doppelklicken. Zertifikate sind zurzeit ein Jahr lang gültig. Entfernen Sie abgelaufene Zertifikate aus dem Schlüsselbund, da Xcode sie nicht von den gültigen unterscheiden kann und Sie sonst Probleme bei der Kompilierung bekommen. Wählen Sie das abgelaufene Zertifikat in der Schlüsselbundverwaltung auf dem Macintosh aus (`/Programme/Dienstprogramme/KeychainAccess.app`) und löschen Sie es.

Neben diesen beiden Zertifikaten müssen Sie auch das WWDR-Zwischenzertifikat ausgeben, dass von Apple Worldwide Developer Relations ausgegeben wird. Sie können es über das Portal oder direkt unter *http://developer.apple.com/certificationauthority/AppleWWDRCA.cer* herunterladen. Fügen Sie auch dieses Zertifikat Ihrem Schlüsselbund hinzu.

Sollten Sie Anwendungen auf mehr als einem Computer auf einmal entwickeln, können Sie das Entwickler- und Verteilungszertifikat mithilfe der Schlüsselbundverwaltung exportieren. Rechtsklicken Sie auf ein Zertifikat, und wählen Sie die Exportoption. Wählen Sie als Format **Personal Information Exchange (.p12)**, und klicken Sie auf **Sichern**. Geben Sie ein Kennwort ein, das Sie sich merken können, und bestätigen Sie es. Klicken Sie auf **OK**, um fortzufahren. OS X fordert Sie auf, ein Administratorkennwort für den Macintosh einzugeben. Tun Sie dies, und klicken Sie auf **Erlauben**. Die Schlüsselbundverwaltung erstellt die verschlüsselte `.p12`-Datei, die Sie dann auf einen anderen Macintosh übertragen und mit einem Doppelklick installieren können. Der lokale Schlüsselbund fragt Sie nach dem Kennwort.

1.6.3 Geräte registrieren

Sie müssen alle Entwicklungs-iPhones im Portal des Programms registrieren. Dazu geben Sie den Gerätenamen und die eindeutige Gerätekennung (Unique Device Identifier, UDID) an. Sie können jederzeit bis zu 100 Geräte registrieren. Ist ein Gerät registriert, können Sie es für die Entwicklung und für Ad-hoc-Nutzungsprofile verwenden. Wechseln Sie zum Bildschirm **Devices** des Portals, und klicken Sie auf **Add Device**. Geben Sie einen Namen und eine UDID ein, und klicken Sie auf **Submit**.

Die UDID herauszufinden, ist nicht schwierig: Dies können Sie in iTunes erledigen. Wählen Sie den Namen des angedockten Geräts aus der Quellenliste (die linke iTunes-Spalte), und wechseln Sie zum Titel **Übersicht**. Klicken Sie auf **Seriennummer**, um die Anzeige von **Seriennummer** in **Identifizierung** (UDID) zu ändern. Wählen Sie **Bearbeiten ▸ Kopieren** (⌘ + C), um die UDID in die Zwischenablage des Systems zu übertragen. Von dort aus können Sie sie in eine Datei einfügen.

Alternativ können Ihre Benutzer Ad Hoc Helper (*http://itunes.com/apps/adhochelper*) auf ihre iPhones herunterladen. Dabei handelt es sich um ein kostenloses Hilfsprogramm, das ich entwickelt habe, um Gerätekennungen direkt per E-Mail an einen Entwickler zu senden. Wenn Sie das Programm starten, erstellt es automatisch eine neue E-Mail, in die es die UDID des Geräts einträgt. Die Benutzer müssen nur noch Ihre Adresse als Empfänger angeben und auf **Send** tippen.

Apple bietet Ihnen verschiedene Möglichkeiten, um mehrere Geräte auf einmal zu registrieren. Am sichersten geht es, wenn Sie mehrere Einträge auf dem Bildschirm **Add Devices** angeben, bevor Sie auf die Schaltfläche **Add Device** klicken. Sie können auch das iPhone Configuration Utility von Apple verwenden, um UDIDs zu verwalten. Dieses Programm steht zum Download auf der Portalwebsite zur Verfügung, weist aber, was die Stabilität angeht, Höhen und Tiefen auf.

Beachten Sie, dass Sie bei der Option **Unregister** nicht sofort einen Platz in der auf 100 Geräte beschränkten Liste freigeben. Da einige Entwickler das System missbraucht haben, gibt es eine einjährige Sperrfrist, bevor ein Listenplatz erneut verwendet werden kann. Wenn Sie die Listenplätze vor Ablauf eines Jahres aus einem guten Grund wiederverwenden müssen, sollten Sie Kontakt mit Apple aufnehmen und darum bitten, diese Einstellung zu umgehen.

1.6.4 Anwendungsbezeichner registrieren

Jede Anwendung, die Sie erstellen, muss einen eindeutigen Bezeichner haben. Dieser String erlaubt der Anwendung, sich gegenüber SpringBoard eindeutig zu identifizieren, und garantiert, dass es keine Konflikte mit anderen Anwendungen gibt. Gewöhnlich erstellen Sie die Bezeichnung mit der umgekehrten Domänenschreibweise von Apple, also z. B. in der Form `com.sadun.Anwendungsname`, `uk.co.sadun.Anwendungsname`, `org.sadun.Anwendungsname` usw. Verwenden Sie in Anwendungsbezeichnern keine Sonderzeichen.

Sie müssen nicht jede Anwendung beim Portal registrieren, sollten aber zumindest einen Platzhalterbezeichner eintragen, also einen Bezeichner, in dem ein Sternchen als Platzhalter vorkommt, z. B. `com.sadun.*`. Mit diesem einzelnen Bezeichner können Sie Nutzungsprofile für alle Ihre Anwendungen erstellen, und zwar unabhängig davon, ob sie nur für die Entwicklung verwendet werden oder für den App Store bestimmt sind. Ein Platzhalter-Nutzungsprofil signiert alle Anwendungen, deren Bezeichner mit seinem Muster übereinstimmen.

Die einzige Ausnahme von dieser Regel bilden Anwendungsbezeichner für Push-Benachrichtigungen. In *Kapitel 16* lernen Sie den Unterschied kennen und erfahren, warum Sie solche Anwendungen einzeln registrieren müssen und wie Sie das tun. Wenn wir Push-Anwendungen außer Acht lassen, kommen die meisten Entwickler damit aus, einen einzigen Platzhalter-Anwendungsbezeichner im Portal zu registrieren.

Für die meisten Beispiele in diesem Buch wird der Anwendungsbezeichner `com.sadun.helloworld` verwendet. Diesen Bezeichner müssen Sie jeweils durch den ersetzen, der Ihrem Nutzungsprofil entspricht. Meistens verwende ich nur einen einzigen Bezeichner, um Ihr iPhone nicht mit Dutzenden von Beispielen auf einmal zuzuschütten. Jedes Beispiel ersetzt das vorhergehende, sodass SpringBoard relativ übersichtlich bleibt.

Hinweis

Falls Sie sich fragen, was es mit den Zufallszeichen auf sich hat, die Ihren registrierten Bezeichnern vorausgehen: Es handelt sich dabei um *Bundle Seed IDs*. Diese sind für Anwendungen gedacht, die Schlüsselbunddaten nutzen. Einzelheiten zur Verwendung von Seed-IDs finden Sie im Apple-Portal.

1.6.5 Nutzungsprofile (Provisioning)

Nutzungs- oder Provisioning-Profile verknüpfen registrierte Entwickler und registrierte Geräte mit einem iPhone-Entwicklungsteam. Sie werden in Xcode eingesetzt, um Code zu signieren und Software für die Ausführung auf dem Gerät oder für die Zulassung im App Store zu autorisieren. Die meisten Entwickler verwenden zwei Hauptarten von Nutzungsprofilen: ein Platzhalter-Nutzungsprofil für die Entwicklung und eines für die Verteilung. Außerdem erstellen viele Entwickler im Laufe der Zeit mindestens ein Ad-hoc-Nutzungsprofil, das es ihnen erlaubt, ihre Anwendungen außerhalb des App Stores auf Geräte zu verteilen, die sie im Portal registriert haben.

Profile erstellen Sie auf dem Bildschirm **Provisioning** des Programmportals. Wählen Sie den Titel **Development** oder **Distribution**, klicken Sie auf **New Profile**, überprüfen Sie das Feld mit den Zertifikatnamen, und wählen Sie einen Platzhalter-Anwendungsbezeichner. Für Entwicklungs- und Ad-hoc-Nutzungsprofile müssen Sie die eingeschlossenen Geräte auswählen. Klicken Sie auf **Submit**, und aktualisieren Sie den Bildschirm mehrfach. Gewöhnlich dauert es weniger als eine Minute, bis das Nutzungsprofil erstellt ist und zum Download bereitsteht.

Wenn Sie zu einem späteren Zeitpunkt weitere Geräte hinzufügen müssen, können Sie das auf einfache Weise tun. Erweitern Sie die Menge der Benutzergeräte, indem Sie die bereits angelegten Nutzungsprofile bearbeiten. Wählen Sie dazu **Edit ▸ Modify**, markieren Sie die neuen Geräte, und klicken Sie auf **Submit**. Laden Sie das aktualisierte Nutzungsprofil herunter, indem Sie auf **Download** klicken.

Um Nutzungsprofile zu installieren, ziehen Sie sie auf das Xcode-Symbol (dies gilt nur für Entwicklungs- und Ad-hoc-Profile) und lassen sie in Xcode im Organizer-Fenster für das Gerät los. Xcode liest sie automatisch ein und installiert sie in Ihrem Benutzerordner in `~/Library/MobileDevice/Provisioning Profiles`. Um ein Profil zu entfernen, verwenden Sie das Fenster **Provisioning Profiles** des Organizers.

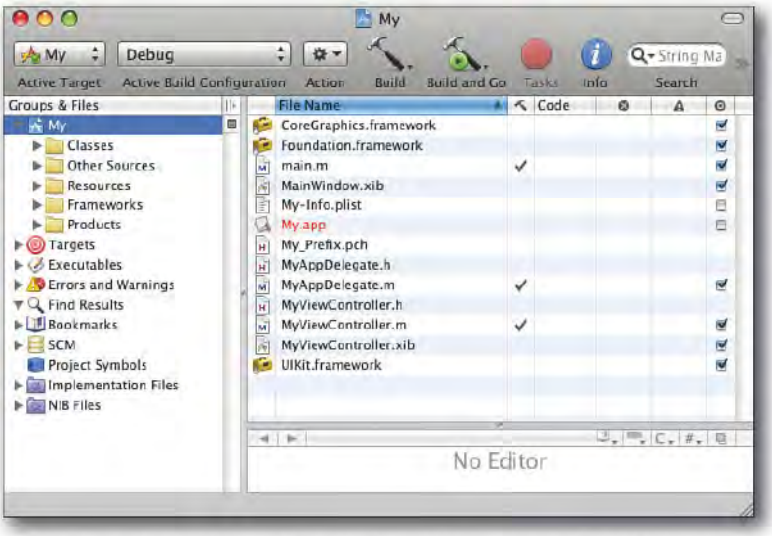
Hinweis

Wenn Sie Ihre Profile lieber von der Kommandozeile aus verwalten möchten, beenden Sie Xcode und löschen sie im Profile-Ordner. Die Profile behalten ihre ursprünglichen Namen nicht bei. Um sicherzustellen, dass Sie die richtige Datei löschen, setzen Sie zunächst das Kommandozeilenprogramm grep ein (z. B. `grep -i firstpush *`) oder schauen sich die Dateien in einer Textverarbeitung an, um die richtige herauszufinden.

Xcode installiert Nutzungsprofile automatisch auf Geräte, um sicherzustellen, dass die mit diesen Profilen kompilierten Anwendungen korrekt ausgeführt werden. Um ein Nutzungsprofil von einem Gerät zu entfernen, öffnen Sie auf dem betreffenden iPhone oder iPod touch **Einstellungen ▸ Allgemein ▸ Profile**, wählen ein Profil aus und tippen auf die rote Löschschaltfläche. Wenn Sie ein Nutzungsprofil auf einem Gerät entfernen, können Sie keine Anwendungen mehr ausführen, die mit diesem Profil signiert wurden.

1.7 iPhone-Projekte zusammenstellen

iPhone-Xcode-Projekte enthalten unterschiedliche Standard- und benutzerdefinierte Komponenten. *Abbildung 1.2* zeigt ein typisches Projekt. Zu seinen Elementen gehören der Quellcode, verlinkte Frameworks sowie Medien wie Bild- und Audiodateien. Xcode kompiliert den Quelltext, verlinkt ihn mit den Frameworks und erstellt ein Anwendungspaket, das für eine Installation auf dem iPhone geeignet ist. Es fügt die Medien zu diesem Paket hinzu, damit das Programm auf sie zugreifen kann, wenn die Anwendung auf dem iPhone ausgeführt wird.



▸ *Abbildung 1.2: Xcode-Projekte führen Quellcode, Frameworks und Medien zusammen und bilden so die Basis für iPhone-Anwendungen.*

Quelltext für das iPhone wird in der Regel in Objective-C 2.0 geschrieben. Das ist eine objektorientierte Obermenge von ANSI C, die aus einer Mischung von C und Smalltalk entwickelt wurde. Kapitel 3, *Objective-C-Trainingslager*, führt praxisorientiert in die Sprache ein. Wenn Sie weitere Informationen über diese Sprache wünschen, finden Sie auf der iPhone-Entwicklerseite von Apple mehrere hervorragende Online-Anleitungen. Darunter befinden sich eine Einführung in die objektorientierte Programmierung mit Objective-C und eine Referenz zu Objective-C 2.0 (*http://developer.apple.com/iphone/library/documentation/Cocoa/Conceptual/ObjectiveC*).

Frameworks sind von Apple zur Verfügung gestellte Softwarebibliotheken, die die wiederverwendbaren Klassendefinitionen für Cocoa Touch bereitstellen. Um Frameworks zu Xcode hinzuzufügen, ziehen Sie sie in den `Frameworks`-Ordner. Nach dem Hinzufügen der geeigneten Headerdateien (wie `UIKit/UIKit.h` oder `QuartzCore/QuartzCore.h`) rufen Sie ihre Routinen aus Ihrem Programm heraus auf.

Mit dem Paket können Medien wie Audio-, Bild- und Videodateien sowie Textdateien gebündelt werden, die Ihre Anwendungen dem iPhone-Betriebssystem gegenüber definieren. Ziehen Sie Mediendateien in Ihr Projekt, und verweisen Sie aus dem Quelltext heraus auf sie.

Das in *Abbildung 1.2* gezeigte Projekt ist sowohl einfach als auch – trotz seines ziemlich überfüllten Erscheinungsbilds – typisch. Es besteht aus fünf Quelldateien (`main.m`, `MyAppDelegate.h`, `MyAppDelegate.m`, `MyViewController.h` und `MyViewController.m`) und zwei Schnittstellendateien (`MyViewcontroller.xib`, `MyWindows.xib`) sowie aus den standardmäßigen iPhone-Projektframeworks (UIKit, Foundation und Core Graphics) und einigen Unterstützungsdateien (`Default.png`, `icon.png` und `My-Info.plist`). Diese Elemente sind alles, was Sie brauchen, um eine sehr grundlegende Anwendung zu erstellen. Wie Sie in Kapitel 2, *Ein erstes Projekt erstellen*, erfahren werden, kann Xcode die meisten dieser Elemente automatisch für Sie anlegen. Anschließend können Sie sie nach Bedarf bearbeiten, um ihnen Funktionalität zu geben.

Hinweis

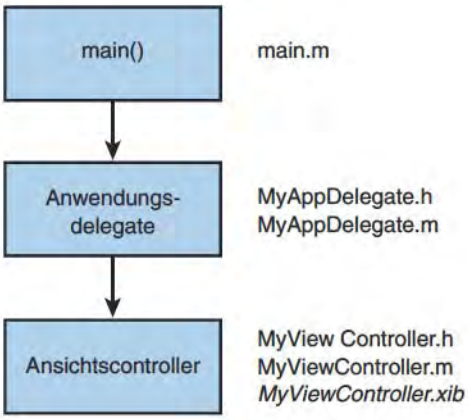
Die Datei `My_Prefix.pch` wird automatisch von Xcode erstellt und enthält vorkompilierte Headerdateien.

1.7.1 Das Grundgerüst einer iPhone-Anwendung

Nahezu jede iPhone-Anwendung, die Sie erstellen, enthält ähnliche Kernquelldateien. *Abbildung 1.3* zeigt das gebräuchlichste Quellcodemuster: die Datei `main.m`, einen Anwendungsdelegate und einen Ansichtscontroller. Diese fünf Dateien (oder mehr, wenn Sie noch `.xib`-Dateien von Interface Builder verwenden) sind alle Komponenten, die erforderlich sind, um eine einfache Hello-World-Anwendung zu erstellen, die auf dem Bildschirm eine Ansicht anzeigt.

Einige dieser Elemente sind Ihnen vielleicht vertraut, andere unter Umständen nicht. Sehen wir uns die Dateitypen also kurz an:

- Die Implementierungsdateien weisen die Endung `.m` statt `.c` auf. Diese `.m`-Dateien enthalten neben Funktionen im C-Stil auch Implementierungen von Objective-C-Methoden. Das Projekt in *Abbildung 1.3* umfasst drei `.m`-Dateien.
- iPhone-Quelldateien tragen die standardmäßige C-Erweitung `.h` für Headerdateien. Solche Dateien enthalten öffentliche Deklarationen von Klassenschnittstellen, Konstanten und Protokollen. Gewöhnlich geben Sie jeder Klassenimplementierungsdatei (in diesem Fall den `.m`-Dateien für den Anwendungsdelegate und den Ansichtscontroller) eine Headerdatei bei, wie Sie in *Abbildung 1.3* sehen.



► *Abbildung 1.3: Diese Dateien zeigen das übliche Muster für Quellcodedateien, die mindestens für eine iPhone-Anwendung erforderlich sind. Ob Sie eine* `.xib`*-Datei zur Definition der Schnittstelle verwenden, ist Ihnen freigestellt.*

> `.xib`-Dateien werden in Interface Builder erstellt. Diese XML-Dateien zur Definition der Benutzeroberfläche werden mit Ihrer Anwendung verlinkt und von ihr zur Laufzeit im kompilierten `.nib`-Format aufgerufen. Das Projekt in *Abbildung 1.3* enthält eine einzige `.xib`-Datei, die den Inhalt der Hauptansicht definiert. In einem standardmäßigen Xcode-Projekt kann eine Datei namens `MainWindow.nib` hinzukommen, die nicht viel mehr tut, als ein neues, leeres Fenster zu erstellen.

Im Folgenden erfahren Sie, was dies für Dateien sind und welche Rolle sie bei einer Anwendung spielen.

1.7.2 main.m

Die Datei `main.m` hat zwei Aufgaben: Erstens erstellt sie den primären Autoreleasepool für die Anwendung, zweitens ruft sie die Anwendungs-Ereignisschleife auf. Diese beiden Elemente sind von entscheidender Bedeutung dafür, dass Ihre Anwendung starten und laufen kann. Sie sehen wie folgt aus:

```
int main(int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil, @"MyAppDelegate");
    [pool release];
    return retVal;
}
```

Hinweis

Die an `main()` übergebenen Variablen `argc` und `argv` beziehen sich auf Kommandozeilenargumente. Da auf dem iPhone keine Kommandozeile zum Start der Programme verwendet wird (alle Anwendungen laufen in einer gemeinsamen grafischen Benutzerschnittstelle), werden diese Elemente nicht genutzt. Sie sind aber zur Übereinstimmung mit den Standardpraktiken von ANSI C enthalten.

Autoreleasepools

Autoreleasepools sind Objekte, die das Speicherverwaltungssystem des iPhones unterstützen. Dieses Speichersystem stützt sich normalerweise auf die Beobachtung von Referenzzählern, d. h., es zählt, wie viele Objekte auf einen zugewiesenen Teil des Arbeitsspeichers verweisen. Normalerweise sind Sie dafür verantwortlich, diese Objekte wieder freizugeben. Hier aber kommt der Autoreleasepool ins Spiel: Er sendet am Ende jedes Ereignisschleifenzyklus automatisch Freigabenachrichten an alle Objekte, die sich in seinem Besitz befinden, sodass Sie dies nicht tun müssen.

Autoreleaseobjekte werden gewöhnlich mit einer Syntax wie der folgenden erstellt:

```
[[[Irgendeine_Klasse alloc] init] autorelease]
```

Sobald solche Objekte dem Autoreleasepool hinzugefügt wurden, übergeben Sie die Verantwortung für ihre Freigabe dem Pool. Am Ende der Ereignisschleife wird der Pool geleert und sendet die Freigabenachrichten.

Das iPhone erwartet, dass stets ein Autoreleasepool verfügbar ist, sodass der Speicher für diese Objekte am Ende von deren Lebensdauer wiedergewonnen werden kann. Wenn Sie in einer Anwendung einen zweiten Thread erstellen, müssen Sie ihn mit seinem eigenen Autoreleasepool versorgen. Autoreleasepools und die in ihnen enthaltenen Objekte besprechen wir genauer in *Kapitel 3*.

Die Funktion UIApplicationMain

Die Funktion `UIApplicationMain` stellt den Haupteintrittspunkt zum Erstellen eines neuen Anwendungsobjekts dar. Sie legt die neue Anwendungsinstanz und deren Delegate an. Stellen Sie sich einen Delegate (Deligierten) als eine Person vor, die wichtige Informationen empfängt. Wenn z. B. Ihr Abteilungsleiter Sie über eine Zustandsänderung an einem Projekt informiert, wodurch Sie entsprechend reagieren können, sind Sie der Deligierte. Ein Delegate ist also für die Handhabung von Änderungen des Anwendungsstatus und für die Bereitstellung programmspezifischer Reaktionen auf diese Änderungen verantwortlich.

Das dritte und vierte Argument der Funktion `UIApplicationMain` gibt den Namen der Hauptanwendungsklasse und deren Delegate an. Wird das dritte Argument ausgelassen (auf `nil` gesetzt), verwendet das iPhone die Standardklasse `UIApplication`.

`UIApplicationMain` richtet auch die Ereignisschleife der Anwendung ein. Eine Ereignisschleife hält wiederholt nach Low-Level-Benutzerinteraktionen wie Bildschirmberührungen oder dem Auslösen von Sensoren Ausschau. Diese Ereignisse werden vom Kernel des iPhones erfasst und an eine Ereigniswarteschlange übergeben, die zur Verarbeitung an die Anwendung übergeben wird.

Mithilfe von Ereignisschleifen können Sie Ihr Programm auf Callbacks stützen. Mit einem Callback geben Sie an, wie die Anwendung auf Ereignisse reagieren soll. In Objective-C entspricht dies einem Methodenaufruf. Sie können z. B. Methoden erstellen, um zu bestimmen, wie sich die Anwendung ausrichten soll, wenn der Benutzer den Bildschirm vom Hoch- ins Querformat dreht, oder wie Ansichten aktualisiert werden, wenn der Benutzer den Finger über den Bildschirm zieht. Diese Art der Programmierung stützt sich auf die zugrunde liegende Ereignisschleife, die in `main.m` eingerichtet ist.

1.7.3 Anwendungsdelegate

Der Anwendungsdelegate legt fest, wie das Programm an kritischen Stellen in seinem Lebenszyklus reagieren soll. Er ist für die Initialisierung des Fenstersystems beim Start und für das Aufräumen beim Beenden verantwortlich. Außerdem übernimmt er die Schlüsselrolle bei der Handhabung von Speicherwarnungen. Zu den wichtigsten Delegatemethoden, die eine Anwendung implementieren muss, gehören:

- `applicationDidFinishLaunching:` Diese Methode ist das Erste, was nach der Instanziierung des Anwendungsobjekts in Ihrem Programm ausgelöst wird. Hier werden beim Start das grundlegende Fenster erstellt und sein Inhalt festgelegt. Außerdem wird es zum Hauptreaktionselement der Anwendung bestimmt.
- `applicationWillTerminate:` Mit dieser Methode können Sie den Status finalisieren, bevor Sie die Steuerung an SpringBoard zurückgeben. Damit speichern Sie Standardwerte, aktualisieren Daten und schließen Dateien.
- `applicationDidReceiveMemoryWarning` Wenn diese Methode aufgerufen wird, muss Ihre Anwendung so viel Speicher wie möglich freigeben. Diese Methode arbeitet Hand in Hand mit der Methode `didReceiveMemoryWarning:` von `UIViewController`. Wenn Ihre Anwendung nicht genügend Speicher freigeben kann, wird sie vom iPhone beendet, sodass der Benutzer unvermittelt wieder bei SpringBoard landet. Dies ist das Hauptmenü des iPhones, über dessen Symbole Benutzer Programme starten können.

Der Anwendungsdelegate ist auch dafür verantwortlich, dass Ihre Anwendung angehalten oder wiederaufgenommen wird, z. B. wenn der Benutzer den Bildschirm sperrt.

Nach dem Starten und Laden des Fensters zieht sich der Anwendungsdelegate in den Hintergrund zurück. Fast die gesamte Anwendungssemantik geht an ein Kindobjekt der Klasse `UIViewController` über. Gewöhnlich greift der Anwendungsdelegate nicht mehr ein, bis die Anwendung beendet werden soll oder bis ein Speicherproblem auftritt.

1.7.4 Ansichtscontroller

Im Programmiermodell für das iPhone sind Ansichtscontroller das Kernelement, um zu bestimmen, wie eine Anwendung ausgeführt wird. Hier legen Sie normalerweise fest, wie eine Anwendung auf eine Auswahl, das Antippen einer Schaltfläche und auf das Auslösen von Sensoren reagiert. Wenn Sie nicht Interface Builder verwenden, um eine fertige Darstellung zu erstellen, ist der Ansichtscontroller das Element, in dem Sie die Ansichten laden und gestalten. Während `main.m` und die Datei für den Anwendungsdelegate gewöhnlich klein sind, ist der Quellcode für den Ansichtscontroller normalerweise sehr umfangreich, da er alle Möglichkeiten festlegt, mit denen die Anwendung auf Ressourcen zugreift und auf die Benutzer reagiert. Im Ansichtscontroller verwenden Sie u. a. folgende wichtige Methoden:

- **`loadView` und `viewDidLoad`** Wenn Sie für das Layout Ihrer Ansichten keine `.xib`-Dateien verwenden, muss die Methode `loadView` den Bildschirm einrichten und die Unteransichten anordnen. Sie müssen entweder `[super loadView]` aufrufen oder alternativ `viewDidLoad` implementieren, wenn Sie von einer spezialisierten Unterklasse wie `UITableViewController` oder `UITabBarController` erben. Dadurch kann die Elternklasse den Bildschirm korrekt einrichten, bevor Sie Ihre Anpassungen hinzufügen. Die Dokumentation von Apple und der Beispielcode ermuntern dazu, den Ansatz mit `viewDidLoad` zu verwenden, wenn Sie Ihren Code auf spezialisierten Unterklassen aufbauen.
- **`shouldAutorotateToInterfaceOrientation:`** Sofern Sie keine zwingenden Gründe dafür haben, den Benutzer auf das Hochformat einzuschränken, sollten Sie die Methode `should-autorotate` hinzufügen, damit die `UIViewController`-Methode den Bildschirm automatisch an die Ausrichtung des iPhones anpassen kann. Dabei müssen Sie festlegen, wie die Bildschirmelemente aktualisiert werden sollen.
- **`viewWillAppear:` und `viewDidAppear:`** Diese Methoden werden immer dann aufgerufen, wenn eine Ansicht auf dem Bildschirm erscheinen soll bzw. wenn sie vollständig angezeigt wird. Die Methode `viewWillAppear:` sollte die Informationen über die Ansichten aktualisieren, die in Kürze auf dem Bildschirm erscheinen werden. Wenn sie aufgerufen wird, ist die Ansicht möglicherweise noch nicht geladen. Wenn Sie sich auf den Zugriff auf IBOutlets verlassen, die mit Unteransichten verbunden sind, rufen Sie `self.view` ab, um sicherzustellen, dass die Ansichtshierarchie geladen wird. Mit `viewDidAppear:` lösen Sie das Verhalten aus, das auftreten soll, wenn sich die Ansicht vollständig auf dem Bildschirm befindet, z. B. Animationen.

Anzahl und Art der `.xib`-Dateien hängen vom Entwurf Ihres Projekts ab. In *Abbildung 1.3* wurde eine einzige `.xib`-Datei für den Ansichtscontroller erstellt. Sie können mit Interface Builder zusätzliche Komponenten anlegen, dieses Programm aber auch umgehen und die Schnittstellen im Programm erstellen.

Hinweis

Nur Instanzen von `UIView` können Aufrufe durch Antippen direkt empfangen, `UIView Controller`-Objekte dagegen nicht. In Kapitel 8, *Gesten und Berührungen*, erfahren Sie mehr darüber, wie Sie Berührungen und Gesten in Ihrer Anwendung direkt handhaben und interpretieren.

1.7.5 Hinweise zum Beispielcode in diesem Buch

Aus didaktischen Gründen stehen die Codebeispiele in diesem Buch gewöhnlich jeweils in einer einzigen `main.m`-Datei. Es ist schwierig, einen einzelnen Punkt zu erklären, wenn Sie als Leser erst fünf, sieben oder neun einzelne Dateien auf einmal betrachten müssen. Die Darbietung in einer einzelnen Datei vereinfacht die Betrachtung.

Die Beispiele sind nicht als eigenständige Anwendungen gedacht, sondern dienen dazu, ein einzelnes Rezept und einen einzelnen Gedanken zu veranschaulichen. Eine `main.m`-Datei mit einer zentralen Darstellung zeigt beispielhaft die Implementierung. Diese ohne viel Beiwerk vorgestellte Grundidee können Sie dann mithilfe des standardmäßigen Dateisystems und Layouts in die normalen Strukturen einer Anwendung überführen.

Von dieser Eine-Datei-Regel gibt es zwei Ausnahmen. Erstens zeigen Anleitungen zur Anwendungserstellung die gesamte Dateistruktur, die Xcode erstellt, um das widerzuspiegeln, was Sie sehen, wenn Sie eigene Programme schreiben. Die Arbeitsordner können daher ein Dutzend oder noch mehr Dateien enthalten.

Zweitens werden die standardmäßigen Klassen- und Headerdateien angegeben, wenn die Klasse selbst das Rezept ist oder eine »vorgekochte« Hilfsklasse bereitstellt. Anstatt eine bestimmte Technik hervorzuheben, bieten einige Rezepte solche vorgefertigten Klassenimplementierungen und Kategorien (also Erweiterungen einer vorhandenen Klasse im Gegensatz zu einer neuen Klasse). Bei solchen Rezepten finden Sie neben der grundlegenden Datei `main.m` noch zusätzliche `.m`- und `.h`-Dateien, die den Rest von dem enthalten, was dieses Rezept Ihnen zeigen soll.

1.8 Komponenten von iPhone-Anwendungen

iPhone-Anwendungen sind in Anwendungsbundles zu Hause. Wie bei ihren Macintosh-Schwestern sind das einfache Ordner mit der Erweiterung `.app`. In einem solchen Ordner befinden sich der Inhalt und die Ressourcen Ihres Programms einschließlich der kompilierten ausführbaren Datei, der unterstützenden Medien (wie Bild- und Audiodateien) und einiger besonderer Dateien, die die Anwendung gegenüber dem Betriebssystem beschreiben. Dieser Ordner wird vom Betriebssystem als einzelnes Bundle behandelt.

1.8.1 Ordnerhierarchie

iPhone-Bundles sind einfach. Im Unterschied zum Mac verwenden sie zum Speichern von Daten nicht die Ordner `Contents` und `Resources` und keinen MacOS-Ordner für die ausführbare Datei. Alle notwendigen Elemente befinden sich in der obersten Ebene des Ordners. Xcode stellt beispielsweise einen Ordner `.lproj` zur Sprachunterstützung nicht in `/Contents/Resources/`, sondern direkt in den Ordner `.app`. Sie können weiterhin Unterordner verwenden, um Ihr Projekt zu gliedern, doch das sind eigens zu diesem Zweck angelegte benutzerdefinierte Ordner, die keine Standards befolgen.

Die Klasse `NSBundle` bildet die grundlegende Betriebssystemunterstützung des iPhone SDK. Diese Klasse bietet Zugriff auf die Dateien, die im Anwendungsbundle gespeichert sind. Mit ihrer Hilfe ist es einfach, den Wurzelordner Ihrer Anwendung zu finden und in die benutzerdefinierten Unterordner zu wechseln, um auf integrierte Ressourcen wie Sounds, Bilder und Datendateien zu verweisen und sie zu laden.

Hinweis

Wie auf dem Macintosh spiegeln Benutzerdomänen Systemdomänen wider. Offiziell von Apple herausgegebene Anwendungen befinden sich im primären Ordner `/Applications`, Anwendungen von Drittanbietern hingegen in `/var/mobile`. Das zugrunde liegende Unix-Dateisystem wird größtenteils von der iPhone-Sandbox verborgen, die weiter hinten in diesem Kapitel behandelt wird.

1.8.2 Die ausführbare Datei

Die ausführbare Datei Ihrer Anwendung befindet sich auf der obersten Ordnerebene des Anwendungsbundles. Sie hat Ausführungsberechtigungen, sodass sie ausgeführt werden kann, und wird als Teil des Anwendungsbundles während der Kompilierung signiert. Nur Anwendungen, die mit einem offiziellen Entwicklerzertifikat signiert sind, können geladen und ausgeführt werden. Diese Zertifikate werden von Apple über das Portal des iPhone-Entwicklerprogramms auf der offiziellen Entwicklerwebsite ausgestellt.

Apple bietet verschiedene Arten von Signaturprofilen an. Sie werden als mobile Provisioning- oder Nutzungsprofile bezeichnet und unterscheiden sich je nachdem, wie die Anwendung bereitgestellt wird. Für Anwendungen, die Sie bei der Entwicklung auf einem lokalen Gerät testen, für Anwendungen, die Sie zum Testen an registrierte Geräte senden, und für solche, die Sie über den App Store verkaufen, brauchen Sie jeweils unterschiedliche Nutzungsprofile. Weiter vorn in diesem Kapitel haben Sie bereits erfahren, wie Sie solche Profile erstellen. Der Signiervorgang für Anwendungen wird ausführlicher in *Kapitel 2* besprochen.

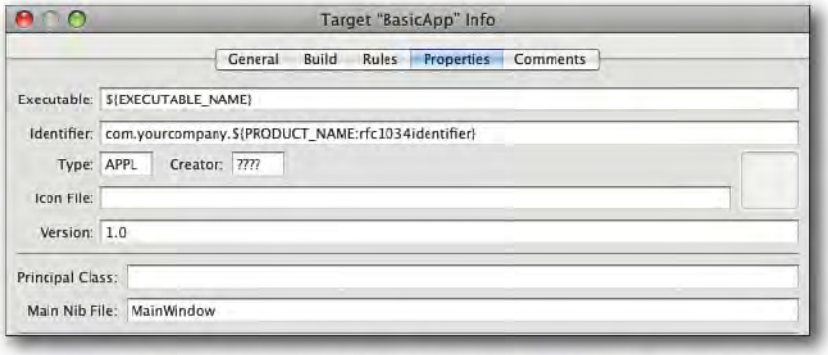
1.8.3 Die Datei Info.plist

Wie auf dem Macintosh enthält der Anwendungsordner die äußerst wichtige Datei `Info.plist`. Es handelt sich um eine Eigenschaftsliste im XML-Format, die die Anwendung gegenüber dem Betriebssystem beschreibt. Eigenschaftslisten speichern Schlüssel-Wert-Paare für viele verschiedene Zwecke und können im lesbaren Text- oder komprimierten Binärformat gespeichert werden. In einer `Info.plist`-Datei geben Sie an, wo die ausführbare Datei (`CFBundleExecutable`) zu finden ist, welcher Text unter dem Anwendungssymbol erscheinen soll (`CFBundleDisplayName`, der Anzeigename des Bundles) und wie der Bezeichner (`CFBundleIdentifier`, der Bundlebezeichner) der Anwendung lautet.

Gehen Sie bei der Wahl des Anzeigenamens sorgfältig vor. Titel, die zu lang für die Anzeige sind, werden abgeschnitten, wobei noch drei Auslassungspunkte angezeigt werden. Wenn Sie also ein Programm »Meine große Schulungsanwendung« nennen, kann es als »Meine große Sch...« angezeigt werden, was dem Endbenutzer weniger Informationen bietet als ein einfacher Titel wie »Schulung«.

Der Anwendungsbezeichner steht gewöhnlich im umgekehrten Domänennamenformat von Apple (z. B. `com.sadun.anwendungsname`) und spielt eine entscheidende Rolle für das richtige Verhalten und die korrekte Ausführung. Er darf nicht mit einem anderen Bezeichner im App Store identisch sein. Im Einsatz auf einem Handy wird Ihre Anwendung mithilfe des Produktbezeichners korrekt bei SpringBoard registriert, dem »Finder« des iPhones. SpringBoard zeigt das Hauptmenü an, auf dem Sie Ihre Anwendungen starten. Der Produktbezeichner bildet außerdem die Grundlage für das eingebaute Voreinstellungssystem, die sogenannten Standardeinstellungen.

Beim Bezeichner wird zwischen Groß- und Kleinschreibung unterschieden. Außerdem muss er mit den Nutzungsprofilen übereinstimmen, die Sie im Entwicklerportal angelegt haben. Probleme aufgrund von falsch benannten Bundlebezeichnern haben manche Entwickler schon Stunden an verschwendeter Zeit gekostet. Geben Sie den Bezeichner an, indem Sie die Projekteinstellungen in Xcode bearbeiten (siehe *Abbildung 1.4*).



▸ *Abbildung 1.4: Den Bundlebezeichner Ihrer Anwendung können Sie anpassen, indem Sie die Zieleigenschaften bearbeiten. Die Änderungen sind in der Datei* `Info.plist` *Ihrer Anwendung zu sehen. In den Projekteinstellungen ist der Bezeichner* `PRODUCT_NAME` *angegeben.*

Hinweis

Um Bezeichner zu ändern, öffnen Sie die Liste **Targets** in der linken Spalte des Xcode-Projekts. Doppelklicken Sie auf **Targets** ▸ *Anwendungsname*. Dadurch wird das Fenster **Target Info** geöffnet. Klicken Sie auf den Titel **Properties**, und ändern Sie den Bezeichner `com.yourcompany` in einen umgekehrten Domänennamen, der Ihren tatsächlichen Firmennamen angibt. Geben Sie Ihre persönliche Domäne ein, und lassen Sie Xcode den Produktnamen der Anwendung anhängen.

Die Voreinstellungen der Anwendung werden automatisch in ihrer Sandbox gesichert. Die Sandbox ahmt die Domänen und Ordner nach, die gewöhnlich im Kernbetriebssystem zu finden sind. Auf dem iPhone erscheinen die Voreinstellungen im lokalen `Library`-Ordner und sind mit dem Anwendungsbezeichner benannt. An den Bezeichner ist die Erweiterung `.plist` angehängt (z. B. `com.sadun.appname.plist`), und zur Speicherung der Voreinstellungen wird das binäre `.plist`-Format verwendet. Eine binäre `.plist`-Datei können Sie lesen, indem Sie sie über den Organizer von Xcode auf einen Macintosh übertragen.

Hinweis

Um Anwendungsdaten vom iPhone auf den Macintosh zu übertragen, öffnen Sie ein Organizer-Fenster (**Window** ▸ **Organizer**), wählen das Gerät und dann einen Eintrag der Anwendungsliste aus. Klicken Sie auf den Pfeil neben dem Namen, um das Bundle der Anwendungsdaten anzuzeigen, und ziehen Sie es auf den Schreibtisch. Es wird zu einem Standardordner erweitert, der als Name den Anwendungsbezeichner sowie Datum und Uhrzeit des Datenabrufs trägt.

Dateien mit Eigenschaftenlisten (`plist`) können Sie direkt in Xcode bearbeiten, aber auch mit dem mitgelieferten Xcode-Hilfsprogramm *Property List Editor*, das eine komfortable grafische Benutzeroberfläche aufweist und unter `/Developer/Applications/Utilities` zu finden ist. Mit dem Hilfsprogramm *plutil* von Apple können Sie Eigenschaftenlisten aus dem binären in ein XML-Textformat konvertieren: `plutil -convert xml1 plistfile`. Apple verwendet binäre `plist`-Dateien, um die Speicheranforderungen zu senken und die Systemleistung zu steigern.

Wie auf dem Macintosh bieten `Info.plist`-Dateien ein Mehr an Flexibilität und lassen sich weitgehend anpassen. Damit können Sie anwendungsspezifische Variablen setzen (`UIRequiresPersistentWiFi`) oder angeben, wie ein Symbol angezeigt werden soll (`UIPrerenderedIcon`). Diese Variablen sind sehr leistungsstark, denn damit lassen sich mehrere Rollen für eine einzelne Anwendung definieren. Allerdings steht diese Funktion für die Entwicklung durch Dritte nicht zur Verfügung. Beispielsweise sind die Programme *Fotos* und *Kamera* im Grunde dieselbe Anwendung, nämlich MobileSlideShow, die verschiedene Rollen spielt. In Anhang A, *Info.plist-Schlüssel*, sind diese Schlüssel im Einzelnen aufgeführt.

Weitere Standardschlüssel von `Info.plist` sind `UIStatusBarStyle` zur Angabe des Aussehens und der Farbe der Statusleiste sowie `UIStatusBarHidden`, um sie vollständig zu verbergen. Mit `UIInterfaceOrientation` können Sie den Beschleunigungsmesser überschreiben, um eine Darstellung im reinen Querformat zu erreichen. (`UIInterfaceOrientationLandscapeRight`). Sie registrieren die URL-Schemas Ihrer benutzerdefinierten Anwendungen (beispielsweise *myCustomApp://*), indem Sie `CFBundleURLTypes` einrichten. Weitere Informationen zu URL-Schemas erhalten Sie in Kapitel 21, *Bedienungshilfen und andere iPhone OS-Dienste*.

1.8.4 Symbol- und Standardbilder

`Icon.png` und `Default.png` sind zwei wichtige Bilddateien. `Icon.png` ist das Symbol Ihrer Anwendung, das zu ihrer Darstellung im Hauptmenü von SpringBoard verwendet wird, und `Default.png` ist das beim Starten der Anwendung gezeigte Begrüßungsbild (offiziell »Startbild« genannt).

Im Gegensatz zu `Default.png` ist der Dateiname des Symbols beliebig. Wenn Sie nicht `Icon.png` verwenden möchten, setzen Sie den Schlüssel `CFBundleIconFile` in Ihrer `Info.plist`-Datei auf den gewünschten Dateinamen. Beachten Sie aber, dass dies zu Problemen führen kann, wenn Sie Ihre Anwendung beim App Store einreichen, denn iTunes Connects erfordert, dass die Anwendung `icon.png` (oder `Icon.png`) verwendet, selbst wenn in `Info.plist` ein anderer Name angegeben ist. Der Schlüssel ist standardmäßig nicht gesetzt, weshalb Sie unabhängig von dem verwendeten Bild sicherstellen müssen, dass Sie einen Wert angegeben haben.

Apple empfiehlt, `Default.png` an den Hintergrund Ihrer Anwendung anzupassen. Viele Entwickler verwenden dieses Startbild für eine Logoeinblendung oder die Nachricht »Bitte warten«. Das widerspricht den Richtlinien von Apple für Benutzerschnittstellen (Startbilder sollen eine optische Kontinuität bieten und keine Werbung oder Entschuldigungen für Verzögerungen), diese Einsatzzwecke sind aber sehr gut nachvollziehbar. Im Organizer-Fenster von Xcode (**Window ▸ Organizer**) können Sie einen Screenshot der Anwendung in Aktion erstellen. Hier haben Sie auch die Wahl, einen dieser Screenshots als Bild für `Default.png` zu verwenden.

Die offizielle Größe des Anwendungssymbols beträgt 57 × 57 Pixel. SpringBoard passt größere Bilder automatisch an. Verwenden Sie matte (nicht glänzende), quadratische Bilder. SpringBoard glättet und rundet die Ecken und fügt einen automatischen Glanzeffekt hinzu. Wenn Sie aus einem zwingenden Grund bereits gerenderte Bilder verwenden müssen, setzen Sie den Schlüssel `UIPrerenderedIcon` in `Info.plist` auf `<true/>`.

Wie bei allen Ein/Aus-Einträgen in `Info.plist` müssen Sie darauf achten, `UIPrerenderedIcon` auf den booleschen Wert `true` (`<true/>`, das markierte Feld in der grafischen Benutzeroberfläche von Xcode) zu setzen. Die Verwendung eines Strings (`<string>true</string>`) mag auf dem Simulator funktionieren, zeigt auf dem iPhone aber keine Wirkung. Beachten Sie auch, dass der Eigenschaftenlisteneditor der Version 3.0 von Xcode den eigentlichen Schlüsselnamen verbirgt. Fügen Sie ein Feld für den Schlüssel **Icon already includes gloss and bevel effects** hinzu, und markieren Sie das Feld, das in der Wertspalte erscheint.

Wenn Sie Ihre Anwendung in App Store einreichen möchten, benötigen Sie eine hochaufgelöste Version Ihres Symbols (512 × 512 Pixel). Zwar können Sie auch Ihre 57 × 57 Pixel große Datei `Icon.png` vergrößern, doch sieht das Ergebnis nicht gut aus. Wenn Sie andersherum vorgehen, erhalten Sie qualitativ hochwertige Bilder, die Sie bei Bedarf auf Symbolgröße verkleinern können. Halten Sie die Bilder einfach und komprimierbar. Ein mit vielen Einzelheiten überfrachtetes Symbol, das bei 512 × 512 beeindruckend aussieht, wirkt bei 57 × 57 wirr und schlampig.

Hinweis

Sie können in Ihr Projekt auch ein Bild im Format 29 × 29 Pixel mit dem Namen `Icon-settings.png` aufnehmen, das in der Anwendung *Einstellungen* für Ihr Programm verwendet wird. Die meisten Entwickler nutzen diese Möglichkeit jedoch nicht. Wenn Sie dieses Bild nicht bereitstellen, verkleinert *Einstellungen* einfach das `icon.png`-Bild.

1.8.5 NIB-Dateien

Interface Builder erstellt `.xib`-Dateien, die vorgefertigte, adressierbare Benutzerschnittstellenlayouts im XML-Format enthalten. (Wenn Sie neugierig sind, können Sie diese Dateien in einer Textverarbeitung öffnen und sich den XML-Code ansehen.) Die meisten mithilfe von Interface Builder erstellen Anwendungen enthalten mehrere `.xib`-Dteien, die die verschiedenen Ansichtskomponenten definieren. Typische `.xib`-Inhalte sind Fensterlayouts, benutzerdefinierte Tabellenzellen, Popup-Dialogfelder usw.

Beim Erstellen des Anwendungsbundles kompiliert Xcode die XML-Daten in ein NIB-Paket, das neben die ausführbare Datei und die anderen Bestandteile der Anwendung gestellt wird. (Die altertümliche Bezeichnung NIB steht für *NeXT Interface Builder*, den Vorgänger des OS X Interface Builders, den Sie zum Erstellen von iPhone-Anwendungen verwenden.) Die `.nib`-Dateien erscheinen auf der obersten Ebene Ihres Anwendungsbundles und werden beim Laden von Bildschirmen direkt von Ihrem Programm aufgerufen.

Hinweis

Wenn Sie Programme entwickeln, die keine XIB-Bundles von Interface Builder verwenden, entfernen Sie den Schlüssel `NSMainNibFile` aus `Info.plist` und entfernen die automatisch erstellte Datei `MainWindow.xib` aus Ihrem Projekt. Dadurch wird das Programm übersichtlicher, und außerdem halten Sie dadurch die Anwendung davon ab, eine Schnittstellendatei zu laden, die nicht vollständig definiert ist. Setzen Sie das vierte Argument von `UIApplicationMain()` in `main()` auf den Klassennamen des Anwendungsdelegates.

1.8.6 Dateien außerhalb des Anwendungsbundles

Wie beim Macintosh befinden sich Dateien für Voreinstellungen (in der Regel in der Anwendungssandbox in `Library/Preferences`), Plug-Ins von Anwendungen (zurzeit in `/System/Library`, aber nicht für die allgemeine Entwicklung verfügbar) und Dokumente (in der Sandbox in `Documents`) außerhalb des Anwendungsbundles.

Zurzeit lässt das iPhone SDK es nicht zu, diese Ordner vorab zu füllen. Da Ihr Programm keine Dateien im Anwendungsbundle bearbeiten oder überschreiben kann, müssen Sie alle Dateien, die einer Änderung bedürfen (z. B. Datenbankdateien), bei der ersten Ausführung des Programms in einen anderen Ordner (`Documents` oder `Library`) speichern.

Des Weiteren scheinen Ordner zur Unterstützung der Anwendung zu fehlen (zumindest aus der Sicht eines Macintosh-Programmierers). Kopieren Sie Unterstützungsdateien, die eher in eine Struktur zur Unterstützung der Anwendung gehören, beim ersten Start Ihrer Anwendung aus dem Anwendungsbundle in den Ordner `Documents` oder `Library`. Prüfen Sie anschließend, ob die Daten dort angekommen sind, und wiederholen Sie den Kopiervorgang gegebenenfalls.

1.8.7 IPA-Archive

Wenn Benutzer Ihre Anwendung erwerben, laden sie eine `.ipa`-Datei von iTunes herunter. Dabei handelt es sich um ein Zip-Archiv mit komprimiertem Inhalt, nämlich dem Anwendungsbundle, das Sie aus den zuvor beschriebenen Komponenten erstellt haben. iTunes speichert `.ipa`-Archive im Ordner `Mobile Applications` von iTunes Library. Wenn Sie eine `.ipa`-Datei umbenennen und ihr die Endung `.zip` geben, können Sie sie ganz einfach mit normaler Komprimierungssoftware öffnen.

Jede Anwendung wird beim Download angepasst, um sicherzustellen, dass sie nur auf den in Ihrem iTunes-Account autorisierten iPhone-Geräten installiert und ausgeführt werden kann. Dadurch wird verhindert, dass die Anwendung uneingeschränkt über das Internet verteilt wird. Softwarepiraten haben zwar Cracks entwickelt, doch werden diese in der Praxis nicht häufig eingesetzt. Die grundlegenden Schutzmechanismen von Apple sorgen dafür, dass in den meisten Fällen nur diejenigen Ihre Software verwenden können, die sie erworben und von iTunes heruntergeladen haben.

1.8.8 Sandboxes

Das iPhone-Betriebssystem beschränkt aus Sicherheitsgründen alle SDK-Entwicklungen auf »Sandboxes« für Anwendungen. Die iPhone-Sandbox beschränkt den Zugriff Ihrer Anwendung auf das Dateisystem auf eine minimale Menge von Ordnern, Netzwerkressourcen und Hardware. Das ist so, als ob Sie eine überaus strenge Schule mit einem paranoiden Direktor besuchen:

- Ihre Anwendung kann in ihrer eigenen Sandbox spielen, aber nicht die Sandbox einer anderen aufsuchen.
- Sie können kein Spielzeug, d. h. keine Daten gemeinsam benutzen (außer über die vom Benutzer gesteuerte Zwischenablage des Systems) und keine Unordnung in den Verwaltungsbüros

anrichten. Ihre Dateien müssen in den Ordnern bleiben, die Sie von der Sandbox erhalten, und Sie können keine Dateien in die Ordner bzw. aus den Ordnern anderer Anwendungen kopieren.

- Sie können nicht über den Zaun schauen. Wenn Ihre Anwendung versucht, Dateien außerhalb ihrer Sandbox zu lesen oder darin zu schreiben, ist das ein Grund dafür, dass sie im App Store abgelehnt wird. Das iPhone-Betriebssystem hindert Ihre Anwendung daran, in den meisten Ordnern außerhalb der Sandbox zu schreiben.
- Ihre Anwendung hat ihre eigenen `Library`-, `Documents`- und `/tmp`-Ordner. Sie imitieren die Standardordner, die Sie auf einer weniger restriktiven Plattform verwenden würden, beschränken aber insbesondere Ihre Möglichkeiten, diese Daten zu schreiben und auf sie zuzugreifen.

Zusätzlich zu diesen Beschränkungen muss Ihre Anwendung digital signiert sein und sich selbst dem Betriebssystem gegenüber mit einem codierten Anwendungsbezeichner authentifizieren, den Sie auf der Website für das Entwicklerprogramm von Apple erstellen müssen. Einzelheiten darüber, wie das geht, haben Sie weiter oben in diesem Kapitel erfahren.

Positiv betrachtet, stellen Sandboxes sicher, dass alle Programmdaten synchronisiert werden, wenn Sie Ihr Gerät mit seinem Standardcomputer verbinden. Negativ betrachtet, können Sie bis zum iPhone OS 3.2 für das iPad keine Daten mit einer Windows- oder Macintosh-Desktop-Anwendung synchronisieren.

Hinweis

Spezifikationsdateien für Sandboxes (mit der Endung `.sb`) werden zusammen mit den tatsächlichen Sandbox-Ordnern in `/var/mobile/Applications` gespeichert. Diese Dateien steuern Berechtigungen wie den Lese- und Schreibzugriff auf verschiedene Teile des Dateisystems. Als Entwickler können Sie diese Dateien nicht sehen oder bearbeiten, aber sie sind vorhanden und legen fest, wie Ihre Anwendung mit dem Betriebssystem interagiert.

1.9 Programmiermodelle

Die Programmierung für das iPhone konzentriert sich auf zwei wichtige Paradigmen: objektorientierte Programmierung und das MVC-Entwurfsmuster (Model-View-Controller). Das iPhone SDK ist dazu entworfen, diese Konzepte in den von Ihnen erstellten Programmen zu unterstützen. Dazu führt es Delegation (*Controller/Steuerung*), Datenquellenmethoden (*Modell*) sowie Klassen für benutzerdefinierte Ansichten (*View/Präsentation*) ein. In den folgenden Abschnitten finden Sie einen kurzen Abriss einiger wichtiger Entwurfsvokabeln von iPhone und Cocoa Touch, die in diesem Buch verwendet werden.

1.9.1 Objektorientierte Programmierung

Objective-C basiert stark auf Smalltalk, einer der historisch wichtigsten objektorientierten Sprachen. Die objektorientierte Programmierung nutzt die Konzepte der Kapselung und Vererbung, um wiederverwendbare Klassen mit veröffentlichten externen Schnittstellen und privater interner Umsetzung zu erstellen. Sie bauen Ihre Anwendungen aus konkreten Klassen auf, die wie Legosteine zusammengesteckt werden können, da aufgrund der Klassendeklarationen stets eindeutig ist, welche Teile zusammenpassen.

Eine wichtige Eigenschaft des objektorientierten Ansatzes von Objective-C ist die Pseudo-Mehrfachvererbung (über Aufrufweiterleitung und Protokolle). iPhone-Klassen können Verhalten und Datentypen von mehr als einem Elternteil erben. Nehmen Sie beispielsweise die Klasse `UITextView`. Sie ist sowohl Text als auch Ansicht. Wie andere Ansichtsklassen kann sie auf dem Bildschirm angezeigt werden. Sie hat feste Begrenzungen, eine gegebene Opazität und erbt gleichzeitig textspezifisches Verhalten, sodass Sie die Schriftart, die Farbe oder die Textgröße leicht ändern können. Objective-C und Cocoa Touch vereinen diese Verhaltensweisen in einer einzelnen leicht zu benutzenden Klasse.

1.9.2 MVC

MVC (Model-View-Controller) trennt das Aussehen von Bildschirmobjekten von ihrem Verhalten. Eine Schaltfläche (die Präsentation) hat keine eigene Bedeutung, sondern ist nur eine Schaltfläche, die der Benutzer betätigen kann. Die Steuerung dieser Ansicht fungiert als Vermittler. Sie verbindet Benutzereingaben wie das Betätigen der Schaltfläche mit Zielmethoden Ihrer Anwendung, die das Modell ist. Die Anwendung stellt sinnvolle Daten zur Verfügung, speichert sie und reagiert mit nützlichen Ergebnissen auf Aktionen. MVC wird am besten in dem wegweisenden Artikel von Glenn Krasner und Stephen Pope von 1988 beschrieben, der online zur Verfügung steht.

Jedes MVC-Element arbeitet für sich. Sie können eine Schaltfläche durch beispielsweise einen Umschalter ersetzen, ohne Ihr Modell oder Ihre Steuerung zu ändern. Das Programm funktioniert wie zuvor, aber die grafische Oberfläche sieht anders aus. Alternativ können Sie die Schnittstelle so lassen, wie sie ist, und Ihre Anwendung ändern, sodass eine Schaltfläche in Ihrem Modell eine andere Art von Reaktion auslöst. Die Trennung dieser Elemente ermöglicht es, wartungsfreundliche Programmkomponenten zu erstellen, die unabhängig voneinander aktualisiert werden können.

Das MVC-Paradigma des iPhones teilt sich in die folgenden Kategorien auf:

- **Modell** Modellmethoden liefern Daten durch Protokolle wie Datenquellen und -bedeutung, indem sie von der Steuerung ausgelöste Callbackmethoden implementieren.
- **Präsentation** (View) Komponenten der Präsentation werden von Kindern der Klasse `UIView` und der (etwas ungünstig benannten) zugehörigen Klasse `UIViewController` bereitgestellt.
- **Steuerung** (Controller) Das Verhalten der Steuerung wird durch drei Schlüsseltechnologien implementiert: Delegation, Target-Action und Benachrichtigungen.

Diese drei Elemente des iPhone-Entwurfsmusters bilden zusammen das Rückgrat des MVC-Programmierparadigmas. In den folgenden Abschnitten schauen wir uns diese Elemente und die unterstützenden Klassen genauer an.

Präsentationsklassen

Das iPhone gründet seine Präsentation auf zwei wichtige Klassen: `UIView` und `UIViewController`. Diese beiden sind für die Definition und Platzierung aller Elemente auf dem Bildschirm verantwortlich.

Wenn Ansichten Elemente auf den Bildschirm zeichnen, stellt `UIView` die abstrakteste Präsentationsklasse dar. Fast alle Klassen für Benutzerschnittstellen stammen von `UIView` und seiner Elternklasse `UIResponder` ab. Ansichten stellen alle sichtbaren Anwendungselemente bereit, die Ihre Anwendung ausmachen. Zu den wichtigen `UIView`-Klassen gehören `UITextViews`, `UIImageViews`, `UIAlertViews` usw. Die Klasse `UIWindow`, eine Art von `UIView`, stellt Ihrer Anwendung ein Darstellungsfeld zur Verfügung und bildet die Wurzel der Anzeige.

Da sie auf dem Bildschirm angezeigt werden, haben alle Ansichten eine bestimmte Art von Rahmen. Es handelt sich um ein Rechteck, das den Bereich definiert, den jede Ansicht einnimmt. Das Rechteck wird durch den Ursprung und die Ausdehnung der Ansicht gebildet.

Ansichten sind hierarchisch gegliedert und mit Bäumen von Unteransichten aufgebaut. Sie können eine Ansicht darstellen, indem Sie sie als die Inhaltsansicht Ihres Hauptfensters festlegen, Sie können sie aber auch einer anderen Ansicht hinzufügen, indem Sie die Methode `addSubview` verwenden, die ein Kind einem Elternteil zuweist. Stellen Sie sich Ansichten so vor, dass Sie transparente Folien mit jeweils einer Art von Zeichnung auf einem Bildschirm anbringen. Die zuletzt hinzugefügten Ansichten sind diejenigen, die Sie sofort sehen. Früher hinzugefügte Ansichten können durch andere verborgen sein, die über ihnen liegen.

Trotz ihres Namens fungiert die Klasse `UIViewController` nicht streng als Steuerung im Sinne von MVC. Sie ist dafür verantwortlich, Elemente auf dem Bildschirm anzuordnen und viele der spezifischen Einzelheiten des Layouts zu verbergen. Die Terminologie von Apple stimmt nicht immer mit dem in Informatikseminaren gelehrten MVC-Paradigma überein.

Ansichtscontroller dienen dazu, Ihnen das Leben zu erleichtern. Sie übernehmen die Verantwortung für das Drehen der Anzeige, wenn ein Benutzer sein iPhone dreht. Sie ändern die Größe von Ansichten, damit sie in die Grenzen passen, wenn eine Navigations- oder eine Werkzeugleiste verwendet wird. Sie handhaben alle heiklen Dinge der Schnittstelle und verbergen die Komplexität der direkten Verwaltung von Interaktionselementen. Zwar können Sie iPhone-Anwendungen entwerfen und erstellen, ohne jemals einen `UIViewController` oder eine Unterklasse davon zu verwenden, doch warum sollten Sie sich das antun? Die Klasse bietet so viel Bequemlichkeit, dass es sich kaum lohnt, eine Anwendung ohne sie zu schreiben.

Neben der Unterstützung des Basiscontrollers für die Orientierung und die Größenänderung erledigen zwei besondere Controller, `UINavigationController` und `UITabBarController`, wie von Zauberhand das Verschieben. Die Navigationsversion ermöglicht Ihnen den Wechsel zwischen Ansichten, indem Ihre Anzeige weich von einer Ansicht in die nächste gleitet. Navigationscontroller erinnern sich daran, welche Ansicht zuerst da war, und bieten eine vollständige Breadcrumb-

Navigationszeile (*http://de.wikipedia.org/wiki/Brotkrümelnavigation*) von **Zurück**-Schaltflächen, um ohne zusätzliche Programmierung zu vorherigen Ansichten zurückzukehren.

Der `UITabBarController` lässt Sie auf einfache Weise zwischen Instanzen von Ansichtscontrollern umschalten. Wenn Ihre Anwendung eine Top-Ten-Liste, ein Fenster für Spiele und eine Hilfeseite hat, können Sie eine Button-Leiste mit drei Schaltflächen hinzufügen, die sofort zwischen diesen Ansichten wechselt, ohne dass eine zusätzliche Programmierung erforderlich ist.

Jede Unterklasse von `UIViewController` hat ihre eigene Methode zum Laden einer Ansicht, entweder durch die Implementierung einer prozeduralen `loadView`-Methode oder durch den Abruf einer vorgefertigten Schnittstelle aus einer `.xib`-Datei. Das ist die Methode, die die Unteransichten des Controllers anordnet und auch alle Trigger, Callbacks und Delegates einrichten kann, falls diese noch nicht in Interface Builder erstellt wurden.

In diesem Sinne fungiert der `UIViewController` als Steuerung, indem er die Verknüpfungen zwischen dem Aussehen der Elemente und der Interaktion interpretiert. Da Sie die Callbacks fast immer zum `UIViewController` selbst senden, fungiert er oftmals als Ihr Modell, zusätzlich zu seiner primären Rolle als Controller für alle Ansichten, die Sie erstellen und zur Anzeige bringen möchten. Es ist nicht strenges MVC, aber bequem und einfach zu programmieren.

Steuerung

Wenn die Entwickler von Apple interaktive Elemente wie Schieberegler und Tabellen entwerfen, wissen sie nicht, wie Sie sie verwenden werden. Die Klassen sind absichtlich allgemein. Bei MVC ist mit der Auswahl einer Zeile oder dem Betätigen einer Schaltfläche keine Bedeutung für das Programm verbunden. Es liegt an Ihnen als Entwickler, das Modell zu liefern, das die Bedeutung hinzufügt. Das iPhone bietet mehrere Möglichkeiten, wie sich vorgefertigte Klassen von Cocoa Touch mit den von Ihnen erstellten Klassen unterhalten können. Im Folgenden sehen Sie die drei wichtigsten: Delegation, Target-Action und Benachrichtigungen.

Delegation

Viele `UIKit`-Klassen verwenden *Delegation*, um die Verantwortung für die Reaktion auf Benutzeraktionen abzugeben. Wenn Sie die Delegation eines Objekts einrichten, teilen Sie ihm mit, dass es alle Interaktionsnachrichten weitergeben soll und der so definierten Delegation die Verantwortung dafür überlässt.

`UITableView` ist ein gutes Beispiel dafür. Wenn ein Benutzer auf eine Tabellenzeile tippt, enthält `UITableView` keine eingebaute Möglichkeit, darauf zu reagieren. Stattdessen befragt es seine Delegation – in der Regel eine Ansichtscontrollerklasse oder die Delegation für die Hauptanwendung – und übergibt die Auswahländerung durch eine Delegationsmethode. Dadurch geben Sie der Berührung ihre Bedeutung erst zu einem späteren Zeitpunkt als dem der ersten Implementierung der Tabellenklasse. Durch Delegation können Sie Klassen ohne Bedeutung erstellen und gleichzeitig dafür sorgen, dass sich später anwendungsspezifische Handler hinzufügen lassen.

Die Delegationsmethode `tableView: didSelectRowAtIndexPath:` von `UITableView` ist ein typisches Beispiel. Ihr Modell übernimmt die Steuerung dieser Methode und implementiert, wie sie auf die Zeilenänderung reagieren soll. Sie können ein Menü anzeigen, zu einer Unteransicht wech-

seln oder eine Prüfmarkierung in der Nähe der aktuellen Auswahl platzieren. Die Reaktion hängt vollständig davon ab, wie Sie die delegierte Auswahländerungsmethode implementieren.

Um die Delegation eines Objekts einzurichten, sollten Sie eine Änderung an der Methode `setDelegate:` vornehmen. Das weist Ihre Anwendung an, die Callbacks für die Interaktion an die Delegation umzulenken. Sie teilen Xcode mit, dass Ihr Objekt Delegationsaufrufe implementiert, indem Sie das implementierte Delegationsprotokoll in der Klassendeklaration benennen. Es erscheint rechts von der Klassenvererbung in spitzen Klammern. *Listing 1.1* zeigt eine Art von `UIViewConroller`, die Delegationsmethoden für `UITableView`-Ansichten implementiert. Daher ist die Klasse `MergedTableController` dafür verantwortlich, alle benötigten Methoden zur Delegation von Tabellen zu implementieren.

Die Delegation ist nicht auf die Klassen von Apple beschränkt. Es ist einfach, Ihren Klassen eigene Protokolldeklarierungen hinzuzufügen und sie zur Definition einer Callback-Vokabelliste zu verwenden. *Listing 1.1* erstellt das Protokoll `FTPHostDelegate`, das die Instanzenvariable `ftpHost` deklariert. Wenn es verwendet wird, muss dieses Objekt alle drei im Objekt deklarierten Methoden implementieren. Protokolle sind ein spannendes und leistungsfähiges Element der Objective-C-Programmierung, denn damit können Sie Clientklassen erstellen, die garantiert den gesamten für die Hauptklasse erforderlichen Funktionsumfang unterstützen.

Hinweis

Wenn Ihre Anwendung auf einer zentralen Tabellenansicht beruht, verwenden Sie `UITableViewController`-Instanzen, um den Aufbau und die Verwendung der Tabelle zu vereinfachen.

```
@protocol FTPHostDelegate <NSObject>
- (void) percentDone: (NSString *) percent;
- (void) downloadDone: (id) sender;
- (void) uploadDone: (id) sender;
@end

@interface MergedTableController : UIViewController <UITableViewDelegate,
                                   UITableViewDataSource>
{
   UIView                *contentView;
   UITableView           *subView;
   UIButton              *button;
   id <FTPHostDelegate>  *ftpHost;
   SEL                   finishedAction;
}
@end
```

▸ *Listing 1.1: Deklarationen für Delegationsprotokolle definieren und zu einer Klassendefinition hinzufügen*

Target-Action

Target-Actions stellen eine Möglichkeit dar, Benutzerinteraktionen auf niedrigerer Ebene umzuleiten. Sie werden Ihnen fast ausschließlich für Kinder der Klasse `UIControl` begegnen. Mit Target-Action weisen Sie die Steuerung an, ein gegebenes Objekt zu kontaktieren, wenn ein bestimmtes Benutzerereignis eintritt. Beispielsweise geben Sie an, mit welchem Objekt Kontakt aufzunehmen ist, wenn der Benutzer eine Schaltfläche betätigt.

Im Folgenden sehen Sie ein typisches Beispiel. Der Auszug definiert eine Instanz von `UIBarButtonItem`, einem verbreiteten schaltflächenartigen Steuerelement, das in iPhone-Symbolleisten verwendet wird. Es setzt das Ziel des Elements auf `self` und die Aktion auf `@selector(setHelvetica:)`. Wird es betätigt, löst es einen Aufruf an das definierende Objekt aus, indem es die Nachricht `setHelvetica:` sendet.

```
UIBarButtonItem *helvItem = [[[UIBarButtonItem alloc]
   initWithTitle:@"Helvetica" style:UIBarButtonItemStyleBordered
   target:self action:@selector(setHelvetica:)] autorelease];
```

Wie Sie sehen, ist der Name der Methode (`setHelvetica:`) völlig willkürlich. Target-Actions verlassen sich nicht wie Delegationen auf ein feststehendes Methodenvokabular. Sie arbeiten jedoch auf genau dieselbe Weise. Der Benutzer führt etwas aus (in diesem Fall betätigt er eine Schaltfläche), und das Ziel implementiert den Selektor, um eine sinnvolle Reaktion zu geben.

Alle Objekte, die diese `UIBarButtonItem`-Instanz definieren, müssen eine Methode namens `setHelvetica:` implementieren. Ist das nicht der Fall, stürzt das Programm zur Laufzeit mit einem Fehler aufgrund eines undefinierten Methodenaufrufs ab. Anders als bei Delegationen und ihren erforderlichen Protokollen gibt es keine Garantie dafür, dass `setHelvetica:` zur Kompilierungszeit implementiert ist. Der Programmierer muss dafür sorgen, dass der Callback auf eine vorhandene Methode verweist.

Standardmäßige Target-Action-Paare übergeben entweder gar kein Argument, eines oder zwei. Bei den Argumenten handelt es sich um das Interaktionsobjekt und ein `UIEvent`-Objekt, das für die Benutzereingabe steht. Der Selektor kann eines dieser Objekte oder beide übergeben. In diesem Fall verwendet der Selektor ein Argument, nämlich die `UIBarButtonItem`-Instanz für die Schaltfläche, die betätigt wurde. Die Selbstreferenz, bei der das ausgelöste Objekt in dem Aufruf enthalten ist, ermöglicht es, allgemeineren Action-Code zu schreiben. Anstatt einzelne Methoden für `setHelvetica:`, `setGeneva:` und `setCourier:` zu erstellen, können Sie die Methode `setFontFace:` aufsetzen, um eine Schriftart auf der Grundlage einer vom Benutzer betätigten Schaltfläche zu aktualisieren.

Hinweis

Um Target-Action in eigene Klassen der `UIControl`-Form einzubauen, fügen Sie eine Target-Variable vom Typ `id` (eine beliebige Objektklasse) und eine Action-Variable vom Typ `SEL` (den Methodenselektor) hinzu. Zur Laufzeit senden Sie den Methodenselektor mit `performSelector: withObject:` an das Objekt. Um den Selektor ohne Parameter zu verwenden, z. B. `@selector(action)`, übergeben Sie `nil` als Objekt.

Benachrichtigungen (Notifications)

Neben Delegationen und Target-Actions nutzt das iPhone noch einen anderen Weg, um Benutzeraktionen – und auch andere Ereignisse – zwischen Ihrem Modell und Ihrer Ansicht zu übermitteln. Mithilfe von *Benachrichtigungen* können Objekte aus Ihrer Anwendung untereinander und auch mit anderen Anwendungen auf Ihrem System kommunizieren. Durch den Broadcast (Rundruf) von Informationen können Objekte mit Benachrichtigungen Zustandsmeldungen wie »Ich habe mich verändert«, »Ich habe mit etwas begonnen« oder »Ich bin fertig« aussenden.

Andere Objekte können auf diese Broadcasts lauschen (müssen es aber nicht). Damit Ihre Objekte eine Benachrichtigung »hören«, müssen Sie sich in einem Benachrichtigungszentrum registrieren und damit beginnen, nach Nachrichten zu horchen. Das iPhone implementiert mindestens vier Arten von Benachrichtigungszentren. Für die App Store-Entwicklung ist nur `NSNotificationCenter` üblich.

`NSNotificationCenter` ist der empfohlene Standard für die Benachrichtigung innerhalb von Anwendungen. Mit diesem Benachrichtigungszentrum können Sie sich für jede oder alle Benachrichtigungen registrieren und dabei zuhören, wie Ihre Objekte sich unterhalten. Die Benachrichtigungen sind vollständig implementiert und können sowohl Daten als auch den Benachrichtigungsnamen übertragen. Die Namens- und Datenimplementierung bietet eine große Flexibilität, sodass Sie dieses Zentrum für komplexe Benachrichtigungen verwenden können.

Es ist einfach, ein Benachrichtigungszentrum zu abonnieren. Fügen Sie den Delegationen Ihrer Anwendung oder – was üblicher ist – Ihren `UIViewController` als registrierten Beobachter hinzu. Sie geben einen beliebigen Selektor an, der aufgerufen wird, wenn eine Benachrichtigung eingeht, in diesem Fall `trackNotifications:`. Die Methode übernimmt ein Argument, eine `NSNotification`. Stellen Sie sicher, dass Ihre Callback-Methode alle Anwendungsbenachrichtigungen hört, indem Sie die Argumente `name` und `object` auf `nil` setzen.

```
[[NSNotificationCenter defaultCenter] addObserver:self
   selector:@selector(trackNotifications:) name:nil object:nil];
```

Alle Benachrichtigungen enthalten drei Datenelemente: den Benachrichtigungsnamen, ein dazugehöriges Objekt und ein `NSDictionary`-Objekt für Benutzerinformationen. Wenn Sie sich nicht sicher sind, welche Benachrichtigungen `UIKit`-Objekte in Ihrer Anwendung erstellen, lassen Sie Ihren Callback die Namen aller erhaltenen Benachrichtigungen ausgeben – beispielsweise `NSLog(@"%@", [notification name])`. Apple dokumentiert Benachrichtigungsprotokolle nicht mit derselben Sorgfalt wie Delegationen und Datenquellenprotokolle.

Die Arten der Benachrichtigungen hängen von der Aufgabe ab, die Sie ausführen. Beispielsweise gehören zu den Benachrichtigungen über das Drehen einer Anwendung `UIApplicationWillChangeStatusBarOrientationNotification` und `UIDeviceOrientationDidChangeNotification`.

Stellen Sie sicher, dass Sie die Methode `trackNotifications:` implementieren (oder eine andere Callback-Methode, deren Selektor Sie bereitgestellt haben), die in diesem Fall für alle Programmbenachrichtigungen unabhängig vom Namen oder Objekt aufgerufen wird. Die Angabe von `nil` beim Abhören dient als Platzhalter.

```
- (void) trackNotifications: (NSNotification *) theNotification
{
   CFShow([theNotification name]);
   CFShow([theNotification object]);
   CFShow([[theNotification userInfo] description]);
}
```

Hinweis

Die Rezepte in diesem Buch verwenden sowohl `printf` und `CFShow` als auch `NSLog`. Jede Methode für eine Rückmeldung beim Debuggen hat ihre eigenen Vor- und Nachteile. Die beiden ersten Methoden schließen zum Beispiel Datum und Uhrzeit nicht ein, was zu einer deutlicheren Ausgabe führt. Wie Sie Informationen aufzeichnen, ist letztendlich Geschmackssache. Für die Aufnahme von Druckanweisungen in ein Programm gibt es kein »Richtig oder Falsch«. Weitere Einzelheiten über die Protokollierung von Informationen finden Sie in *Kapitel 3*.

Modell

Sie sind für das Erstellen der gesamten Semantik der Anwendung verantwortlich – den Modellanteil jeder MVC-Anwendung. Dazu erstellen Sie die Callback-Methoden, die der Controller Ihrer Anwendung anstößt, und liefern die benötigten Implementierungen aller Delegationsprotokolle. Für relativ einfache Programme fügen Sie Modellinformatonen einfach zu einer Unterklasse von `UIViewController` hinzu. Bei komplizierterem Code sollten Sie es vermeiden, die Implementierung in einen `UIViewController` zu zwängen. Mit selbst erstellten Klassen können Sie die zur Unterstützung des Anwendungsmodells erforderlichen semantischen Einzelheiten besser umsetzen.

Es gibt eine Stelle, an der Ihnen das iPhone SDK bei der Bedeutung behilflich ist, und zwar bei Datenquellen. Mithilfe von Datenquellen können Sie `UIKit`-Objekte mit benutzerdefiniertem Inhalt füllen.

Datenquellen

Datenquellen sind Objekte, die andere Objekte auf Anforderung mit Daten versorgen. Einige Objekte der Benutzerschnittstelle (UI-Objekte) sind Container ohne eigenen Inhalt. Wenn Sie ein anderes Objekt als Datenquelle eines UI-Objekts bestimmen, indem Sie ihm die Eigenschaft `dataSource` zuweisen (die bevorzugte Vorgehensweise) oder einem Aufruf wie `[uiobject setDataSource:applicationobject]` verwenden, ermöglichen Sie dem UI-Objekt (der Ansicht), die Datenquelle (das Modell) nach Daten wie Tabelleneinträgen für eine gegebene `UITableView` abzufragen. In der Regel erhält die Datenquelle ihre Daten aus einer Datei wie einer lokalen Datenbank, einem Webdienst wie einem XML-Feed oder einer gescannten Quelle. `UITableView` und `UIPickerView` sind zwei der wenigen Cocoa Touch-Klassen, die Datenquellen unterstützen oder benötigen.

Datenquellen ähneln Delegationen darin, dass Sie ihre Methoden in einem anderen Objekt implementieren müssen, üblicherweise in dem `UITableViewController`, dem die Tabelle gehört. Sie unterscheiden sich darin, dass Sie Objekte erstellen bzw. zur Verfügung stellen, anstatt auf Benutzeraktionen zu reagieren.

Listing 1.2 zeigt eine typische Datenquellenmethode, die ein Tabellenelement (eine Zelle) für eine gegebene Zeile zurückgibt. Wie andere Datenquellenmethoden ermöglicht sie das Loslösen der Implementierungssemantik, die eine gegebene Ansicht füllt, von der von Apple gelieferten Funktionalität, die den Ansichtscontainer erstellt.

Objekte, die Datenquellenprotokolle implementieren, müssen sich selbst deklarieren, so wie sie es mit Delegationsprotokollen tun würden. *Listing 1.1* zeigte die Deklaration einer Klasse, die sowohl Delegationen als auch Datenquellenprotokolle für `UITableViews` unterstützt. Apple dokumentiert Datenquellenprotokolle sorgfältig. Diese Dokumentation finden Sie im Fenster **Documentation** von Xcode (**Help ▸ Documentation**).

```
// Rückgabe einer Zelle für die i-te Zeile, die mit ihrer Nummer
// bezeichnet ist
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView
        dequeueReusableCellWithIdentifier:@"any-cell"];
    if (!cell) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
            reuseIdentifier:@"any-cell"] autorelease];
    }
    // Zelle einrichten
    cell.textLabel.text = [tableTitles objectAtIndex:[indexPath row]];
    return cell;
}
```

▸ *Listing 1.2: Datenquellenmethoden füllen Ansichten mit sinnvollem Inhalt.*

Das Objekt UIApplication

In der Theorie konzentriert sich die »Modell«-Komponente des iPhones auf die Klasse `UIApplication`. In der Praxis tut sie es nicht, zumindest nicht im MVC-Sinne des Worts *Modell*. In der Welt des Apple-SDKs enthält jedes Programm genau eine `UIApplication`-Instanz, die Sie über `[UIApplication sharedInstanz]` referenzieren können.

Sie können `UIApplication` größtenteils vollständig ignorieren, es sei denn, Sie müssen einen URL in Safari öffnen, das Hauptfenster wiederherstellen oder das Aussehen der Statusleiste einstellen. Erstellen Sie Ihr Programm auf der Grundlage einer benutzerdefinierten Anwendungsdelegationsklasse, die dafür zuständig ist, Dinge einzurichten, wenn die Anwendung gestartet wird, und Dinge

zu schließen, wenn sie wieder beendet wird. Andernfalls übergeben Sie die übrigen Modellpflichten an Methoden in Ihren benutzerdefinierten `UIViewController`-Klassen oder an benutzerdefinierte Modellklassen.

Hinweis

Verwenden Sie `[[UIApplication sharedInstance] keyWindow]`, um das Hauptfensterobjekt Ihrer Anwendung zu lokalisieren.

Datenquellen und Delegationsmethoden offenlegen

Neben den Überwachungsbenachrichtigungen kann die Nachrichtenverfolgung ein unbezahlbares Werkzeug sein. Fügen Sie Ihren Klassendefinitionen das folgende Fragment hinzu, um alle Methoden aufzudecken, auf die Ihre Klasse antwortet – dokumentierte und undokumentierte, Datenquellen und Delegationen:

```
-(BOOL) respondsToSelector:(SEL)aSelector {
   printf("SELECTOR: %s\n", [NSStringFromSelector(aSelector)
UTF8String]);
   return [super respondsToSelector:aSelector];

}
```

1.10 Zusammenfassung

In diesem Kapitel wurden das iPhone SDK, das Entwicklerportal und iPhone-Anwendungen eingeführt. Sie haben gelernt, wie Sie ein Entwicklerprogramm auswählen und wie Sie Nutzungsprofile erstellen. Sie haben sich typische iPhone-Anwendungen von den Projekt- und Quelldateien bis zum Endprodukt angeschaut und etwas über die Entwurfseinschränkungen erfahren, die Sie bei der Entwicklung beachten müssen. Aus diesem Kapitel können Sie die folgenden Punkte mitnehmen:

- Die meisten Entwickler entscheiden sich für das iPhone Standard Developer Program für 79 EUR im Jahr. Dies ist das beste und umfassendste Programm, für das Sie sich registrieren können, da es Tests auf echten Geräten ermöglicht und Ihnen Zugang zum App Store gewährt.
- Es gibt bedeutende Unterschiede zwischen den einzelnen iPhone-, iPod touch- und iPad-Plattformen. Um die bestmögliche Anwendung für den Benutzer zu entwickeln, müssen Sie diese Unterschiede genau kennen.
- Die Entwicklung für mobile Systeme unterscheidet sich von der für Desktop-Computer. Beachten Sie die Grundregel: große Finger, kleine Bildschirme, kurze Verweildauer.

- Das iPhone-Anwendungsbundle ist viel einfacher und weniger strukturiert als sein Bruder für den Macintosh, obwohl beide viele Funktionen gemeinsam haben, darunter `Info.plist` und den Ordner `lproj`.
- Wenn Sie bereits mit Cocoa gearbeitet haben, sind Sie schon zum Erstellen von iPhone-Anwendungen qualifiziert, wenn nicht sogar überqualifiziert. Kenntnisse in Objective-C und den empfohlenen Vorgehensweisen von Cocoa geben Ihnen eine gute Grundlage für die Entwicklung.
- Wenn Sie sich in C++ sicherer fühlen als in Objective-C, können Sie hybride Projekte erstellen und darin Ihre C++-Kenntnisse mit einem Minimum an zusätzlichem Objective-C einbringen.

2 Ein erstes Projekt erstellen

Xcode und Interface Builder helfen Ihnen dabei, Anwendungen für das iPhone zu erstellen. In diesem Kapitel werden Sie in die Verwendung dieser Werkzeuge eingeführt. Sie erfahren, wie Sie ein einfaches Hello-World-Projekt erstellen, es für den Simulator kompilieren und dort testen, Sie lernen aber auch, wie Sie es für ein physisches Gerät kompilieren und dort bereitstellen. Außerdem lernen Sie einige grundlegende Werkzeuge für das Debugging kennen, erhalten eine Anleitung zu deren Verwendung und bekommen einige Tipps zu praktischen Compiler-Direktiven. In diesem Kapitel sehen wir uns auch an, wie Sie Anwendungen im App Store einreichen und über die Ad-hoc-Verteilung verbreiten. Nachdem Sie dieses Kapitel durchgearbeitet haben, kennen Sie den Vorgang der Anwendungserstellung vom Anfang bis zum Ende und haben dabei auch wertvolle Tricks erfahren.

2.1 Neue Projekte erstellen

Sich ohne Netz und doppelten Boden kopfüber in die SDK-Programmierung zu stürzen, mag gewagt erscheinen, dennoch können Sie beruhigt aufatmen: Xcode macht die ersten Schritte so einfach wie möglich. Das Programm bietet sechs vorkonfigurierte Projekte an, die Sie auf einfache Weise anpassen können, während Sie sich das SDK genauer ansehen. Da jedes dieser Projekte ein voll funktionsfähiges Grundgerüst ist, müssen Sie lediglich ein wenig eigene Funktionalität hinzufügen, um daraus ein eigenes Projekt zu machen.

Um zu beginnen, starten Sie Xcode (`/Developer/Applications`) und wählen **File ▸ New Project** ⌘ + ⇧ + N. Daraufhin wird das Template-Fenster **New Project** geöffnet (siehe *Abbildung 2.1*), in dem Sie einen Anwendungstyp auswählen können, mit dem Sie beginnen möchten.

Die sechs möglichen Projekttypen entsprechen den häufigsten Entwicklungsmustern für das iPhone. Folgende Möglichkeiten stehen zur Wahl:

- **Navigation-based Application** Navigationsanwendungen stützen sich gewöhnlich auf Listen und Tabellen und bieten eine Reihe von Auswahlmöglichkeiten, die jeweils zu einem neuen Bildschirm führen. Die Leiste am oberen Rand des Bildschirms enthält eine **Zurück**-Schaltfläche, über die Sie zum vorherigen Bildschirm zurückkehren können. Die Apple-Anwendung *Kontakte* gehört in diese Kategorie.
- **OpenGL ES Application** Wenn Sie mit OpenGL ES programmieren, brauchen Sie nur eine Ansicht, in der Sie zeichnen, und einen Timer als Taktvorgabe für Animationen. Diese Elemente enthält das OpenGL ES-Template, sodass Sie darauf OpenGL ES-Grafiken erstellen können.
- **Tab Bar Applications** Der iPod von Apple und YouTube sind typische Beispiele für Tab-Bar-Anwendungen. In einem solchen Programm können die Benutzer aus einer Reihe von Bildschirmen auswählen, indem sie auf Schaltflächen in einer Leiste am unteren Bildrand tippen. In der YouTube-Anwendung können Sie z. B. zwischen **Highlights**, **Topvideos**, **Favoriten** und dem Suchbereich wählen. Das Template **Tab Bar Application** umfasst ein Grundgerüst, das Sie um weitere Bereiche und deren Inhalte erweitern können.

▸ *Abbildung 2.1: Das Xcode-Fenster* **New Project** *zur Auswahl eines Templates*

- **Utility Application** Dieses Template ist für den einfachsten Typ von Anwendung da und erstellt eine Präsentation mit einer einzigen zweiseitigen Ansicht wie die in den Anwendungen *Aktien* und *Wetter*. Es enthält eine Hauptansicht und eine Rückseitenansicht, die Sie auf einfache Weise anpassen können.

- **View-based Application** Dieses Template enthält das Grundgerüst für eine einzige Ansicht mit einem Ansichtscontroller zu deren Handhabung und einer leeren `.xib`-Datei, um die Ansicht mit eigenen GUI-Elementen zu füllen. Dieses Template verwenden Sie in diesem Kapitel, um eine erste Hello-World-Anwendung zu erstellen.
- **Windows-based Application** Dieses Template entspricht **View-Based Application**, enthält aber keinen Ansichtscontroller und keine Ansicht, sondern lediglich einen Anwendungs-Delegate und ein Fenster, und das war's. Ein Vorteil dieses Templates besteht darin, dass Sie die Interface Builder-Elemente relativ einfach entfernen können, falls Sie Ihre iPhone-Anwendung lieber komplett selbst gestalten möchten.

Hinweis

In der iPhone Reference Library finden Sie Beispielcode und Tutorials von Apple. Die Bibliothek finden Sie unter *http://developer.apple.com/iphone/library/navigation/index.html*. Für den Zugriff auf den Inhalt müssen Sie Ihre Anmeldeinformationen als Entwickler angeben. Neben Beispielcode finden Sie hier auch Release Notes, technische Hinweise, Einführungen, Programmieranleitungen usw.

2.2 Hello World mit einem Template erstellen

Die vorgefertigten Templates von Xcode sind die einfachste Möglichkeit, um eine Hello-World-Beispielanwendung zu erstellen. Anhand der folgenden Anleitung legen Sie ein neues Projekt an, bearbeiten es so, dass es »Hello World« ausgibt, und führen es auf dem iPhone-Simulator aus. Bei der Arbeit an Ihrem ersten Xcode-Projekt lernen Sie einige der wichtigsten Entwicklungsmöglichkeiten kennen.

2.2.1 Ein neues Projekt anlegen

Starten Sie Xcode mit installiertem iPhone SDK. Es muss mindestens in der Version 3.0 vorliegen. Schließen Sie die Begrüßungsseite von Xcode. Das ist das Fenster, das **Welcome to Xcode 3.x** und Optionen wie **Create Your First Cocoa Application** anzeigt. Dieses Fenster erscheint bei jedem Start des Programms, sofern Sie nicht die Markierung von **Show at Launch** vor dem Beenden entfernen.

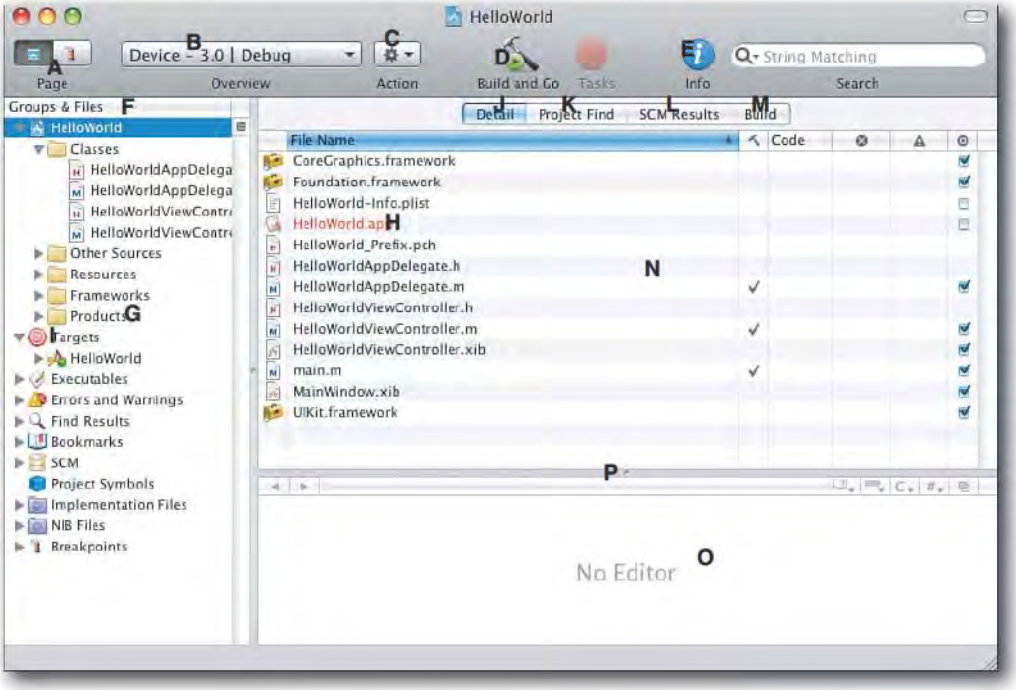
Hinweis

Wenn Sie das Fenster doch wieder anzeigen lassen möchten, finden Sie es unter **Help ▸ Xcode News**.

Um ein neues Projekt zu erstellen, wählen Sie **File ▸ New Project** ⌘ + ⇧ + N. Dadurch wird das Fenster zur Template-Auswahl geöffnet, das Sie schon aus *Abbildung 2.1* kennen. Wie Sie sehen, gibt es zurzeit nur zwei Sätze von iPhone-Templates (**Application** und **Library** in der linken Spalte) gegenüber einem Dutzend möglicher Projekttypen für den Macintosh. Sie können Xcode jedoch neue iPhone-Templates hinzufügen. Wie Sie das machen, lernen Sie weiter hinten in diesem Kapitel. Jetzt aber wählen Sie **Application**, falls dies nicht vorausgewählt sein sollte.

Wählen Sie **View-based Application**, und klicken Sie auf **Choose**. Wenn Xcode Sie zum Sichern auffordert, nennen Sie das Projekt »Hello World« und speichern es mit **Save** auf dem Schreibtisch. Ein neues Xcode-Projektfenster namens **Hello World** erscheint (siehe *Abbildung 2.2*). Dieses Projekt enthält alle Dateien, die Sie brauchen, um eine Anwendung zu erstellen, die sich auf eine Hauptansicht stützt.

Wie das Projektfenster aussieht, hängt von den Xcode-Einstellungen ab. Wählen Sie **Xcode ▸ Preferences** ⌘ + ,, rufen Sie den Titel **General** auf, und wählen Sie das gewünschte Layout aus dem Einblendmenü. Bereits geöffnete Projekte müssen Sie vor Änderung dieser Einstellung jedoch geschlossen werden. Für die Beispiele in diesem Kapitel habe ich das Layout **All-In-One** verwendet, das alle Elemente in einem einzigen Fenster zeigt (siehe *Abbildung 2.2*). Andere Möglichkeiten sind **Condensed** mit einem eigenen Fenster für die meisten Aufgaben, und **Default** mit einem Hauptprojektfenster und eigenen Fenstern für die einzelnen Werkzeuge.



▸ *Abbildung 2.2: Dieses brandneue Hello-Word-Projekt wurde durch die Auswahl des Templates* **View-based Application** *erstellt.*

Hinweis

Beim Erstellen neuer iPhone-Projekte finden Sie in manchen Templates das Markierungsfeld **Use Core Date for Storage**. Solche Projekte enthalten ein Grundgerüst, das einen Core-Data-Stack für die dauerhafte Speicherung erstellt. In Kapitel 19, »*Core Data*«, erfahren Sie mehr darüber.

2.2.2 Das Projektfenster

Das standardmäßige Xcode-Projektfenster ist normalerweise in drei Teile gegliedert, wie Sie in *Abbildung 2.2* sehen können: in die graue Symbolleiste am oberen Rand, in die linke Spalte und in den Hauptbereich des Fensters. Jedes dieser Bestandteile erfüllt seine eigene Aufgabe bei der Handhabung eines Projekts.

Die obere, graue Symbolleiste enthält eine Reihe nützlicher Werkzeuge. Mit dem Umschalter ganz links (A) wechseln Sie zwischen der Projektübersicht und dem grafischen Debugger. Rechts daneben finden Sie ein Einblendmenü (B), um Ziele und Konfigurationen festzulegen. Damit steuern Sie, was für eine Anwendung Sie erstellen möchten und wie Sie das tun. Die normalen iPhone-Templates enthalten zwei Konfigurationen, die Sie in diesem Einblendmenü auswählen können, nämlich **Debug** und **Release**. Leider bilden diese Konfigurationen die Realität der iPhone-Entwicklung nur ungenügend ab. Bessere Auswahlmöglichkeiten wären **Debug** (für die innerbetriebliche Entwiclung), **Ad hoc** (für die Ad-hoc-Verteilung) und **Distribution** (für die Veröffentlichung im App Store). Zum Glück lassen sich diese Konfigurationen bearbeiten. Weiter hinten in diesem Kapitel erfahren Sie, wie Sie eine bessere Auswahl erstellen.

Rechts neben dem Konfigurations-Einblendmenü befindet sich das Aktionsmenü (C), das typische Aufgaben für Projekte wie das Hinzufügen neuer Dateien und die Anzeige im Finder enthält, um diese Dateien auf dem Macintosh zu finden. Rechts daneben finden Sie die Schaltfläche **Build and Go** (D) bzw. **Build and Run** unter Xcode 3.2. Wenn Sie darauf klicken, wird Ihre Anwendung kompiliert und ausgeführt, entweder auf dem Gerät oder auf dem Simulator. Über die Schaltfläche **Info** öffnen Sie ein Fenster, in dem Sie Teile des Projekts anpassen können.

Links neben dem Projektfenster befindet sich die Spalte **Groups & Files** (F). Sie beginnt mit den Quelldaten für das Projekt und enthält sämtliche Dateien, die Sie zum Erstellen der Anwendung verwendet oder dem Projekt hinzugefügt haben. Das dargestellte Ordnersystem ist völlig willkürlich. Sie können das Material mit Ordnern gliedern, aber auch ganz auf Ordner verzichten. Gruppen bieten eine elegante Möglichkeit, um den Code und die Ressourcen in Xcode zu ordnen, ohne das Dateisystem anfassen zu müssen.

Der Ordner **Products** (G), standardmäßig der letzte Eintrag in der Ordnerliste, enthält das Element, das Sie zu erstellen beabsichtigen, in diesem Fall die Anwendung `HelloWorld.app` (H). Sie wird rot angezeigt, da sie noch nicht mit **Build** erstellt wurde. Sobald sie erstellt ist, erscheint sie in Schwarz.

Unter der ersten Gruppe von Dateien befindet sich der Eintrag **Targets** (I). Wenn Sie auf das graue Einblenddreieck klicken, wird der Eintrag **Hello World** unter **Targets** angezeigt. Es ist sehr wichtig zu wissen, wo sich das Ziel (*target*) befindet. Wählen Sie es aus, und klicken Sie auf die blaue **Info**-Schaltfläche am oberen Rand des Projektfensters, um das Fenster **Target „Hello World“ Info** zu öffnen. Dieses Fenster ist eines Ihrer wichtigsten Werkzeuge. Merken Sie sich den Ablauf: Einblenddreieck **Targets** öffnen, Ziel auswählen, auf **Info** klicken.

Rechts neben der Spalte **Groups & Files** befindet sich ein Bereich mit verschiedenen Titeln. Oben finden Sie vier Optionen: **Detail** (J), **Project Find** (K), **SCM Results** (L) und **Build** (M). Diese vier Optionen stehen zwar nebeneinander, erfüllen aber jeweils unterschiedliche Aufgaben und bieten sämtliche wichtigen Projektinformationen:

- **Detail** listet alle Dateien in Ihrem Projekt auf und zeigt eine Vorschau. Dies hilft Ihnen dabei, Dateien zu finden und zu bearbeiten.
- **Project Find** durchsucht das gesamte Projekt, wobei Sie nach Strings in Dateien und Frameworks suchen können. Diese Funktion ähnelt dem Fenster **Finden** zur Suche in einer einzigen Datei, das Sie über ⌘ + F erreichen, und erweitert dessen Möglichkeiten.
- **SCM Results** zeigt den Status der Dateien im Source Code Management System. SCM-Systeme verfolgen Änderungen an Projekten nach, und zwar sowohl für einzelne als auch für mehrere Programmierer. Xcode arbeitet mit SVN, Perforce und CVS zusammen und zeigt die Ergebnisse Ihrer Synchronisierungsvorgänge an. Leider wird das Versionssteuerungssystem *git* zurzeit nicht unterstützt.
- **Build** zeigt ein Ergebnisfenster für den Erstellungsvorgang des Projekts an und zeigt jegliche Fehler und genauere Einzelheiten dazu an.

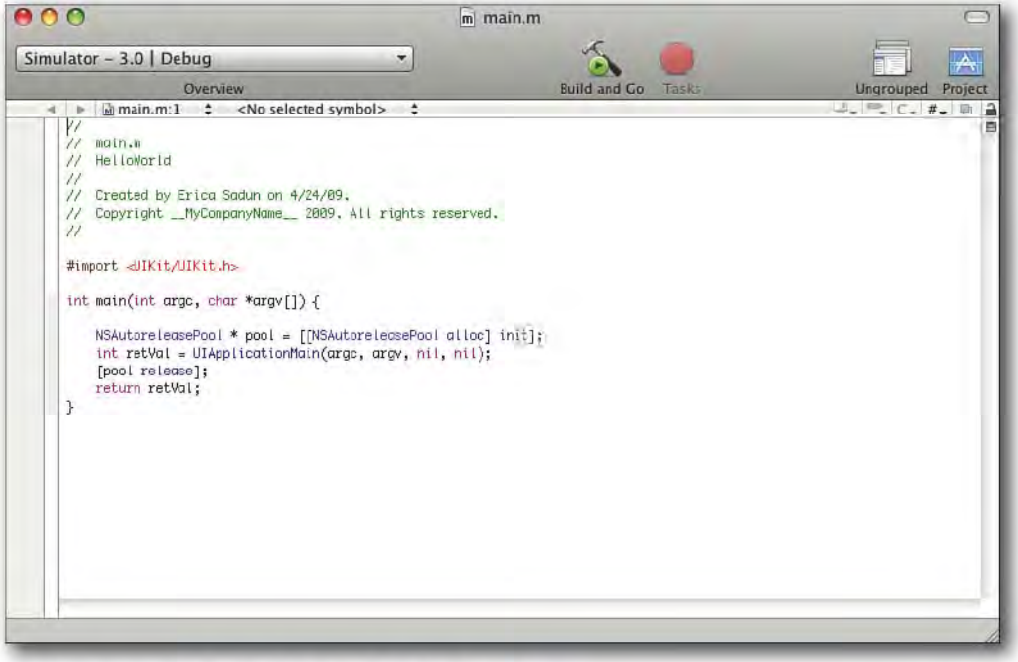
2.2.3 Der Detailbereich

Wahrscheinlich werden Sie viel Zeit im Detailbereich (N) verbringen, der Standardauswahl für das Projektfenster. Hier sind alle Dateien des Projekts aufgeführt, und hier finden Sie die einzelnen Elemente nach ihrem Namen geordnet, wobei die Spalte **Groups & Files** bestimmt, was hier zu sehen ist. Wenn Sie ganz oben in der Spalte auf die Gruppe **Hello World** klicken, sehen Sie alle Dateien auf einmal. Durch die Auswahl einer Gruppe wird die Liste der Dateien auf diejenigen in der betreffenden Gruppe eingeschränkt.

Der untere Bereich (O) zeigt eine Vorschau des Elements, das Sie im Bereich darüber ausgewählt haben. Gleichzeitig ist dies ein Editor: Jegliche Änderungen, die Sie im unteren Bereich vornehmen, sorgen für eine Aktualisierung der betreffenden Datei. Wenn Sie auf `main.m` klicken, zeigt der untere Bereich sofort die Inhalte dieser Datei an. Zwischen der Dateiliste oben und dem Editor- und Vorschaufenster befindet sich eine Leiste zur Größenänderung (P), mit der Sie das Größenverhältnis zwischen den beiden Elementen einstellen können. Wenn Sie keine Vorschau wünschen, ziehen Sie die Leiste einfach bis ganz nach unten.

2.2.4 Editorfenster

Sie müssen Ihre Dateien nicht in dem kleinen Vorschaubereich des Projektfensters bearbeiten. Wenn Sie auf eine der Quelldateien im oberen Bereich doppelklicken, z. B. auf `main.m`, wird ein eigenständiges Editorfenster geöffnet (siehe *Abbildung 2.3*). Dieses Quellcodefenster weist einige der Steuerelemente auf, die Sie auch in der oberen, grauen Leiste des Projektfensters finden. Unter dieser Leiste liegt ein Standardfenster zur Codebearbeitung. Alle gewünschten Änderungen können Sie vornehmen, indem Sie den Text im Fenster ändern. Speichern Sie dabei Ihre Arbeit regelmäßig mit **File ▸ Save** ⌘ + S.



▸ *Abbildung 2.3: Ein Quellcode-Bearbeitungsfenster in Xcode*

In Xcode können Sie Arbeitsschritte innerhalb einer Sitzung unbeschränkt rückgängig machen. Es ist sogar möglich, Schritte vor dem letzten Speicherungsvorgang zu widerrufen, sofern sie in derselben Sitzung ausgeführt wurden. Sie können also nicht ein Projekt schließen, erneut öffnen und dann Änderungen rückgängig machen, die Sie vor dem Schließen des Projekts vorgenommen hatten.

Hinweis

Um in den Quellcode-Bearbeitungsfenstern Zeilennummern einzublenden, öffnen Sie die Voreinstellungen (**Xcode ▸ Preferences** oder ⌘ + ,, wechseln zum Bereich **Text Editing** und markieren **Show Line Numbers** (links unter **Display Options**). Klicken Sie auf **Apply** und dann auf **OK**.

2.2.5 Das Projekt im Überblick

Xcode füllt das neue Projekt beim Anlegen mit allen Grundelementen und Frameworks, die Sie brauchen, um eine iPhone-Anwendung zu erstellen. In dem Projekt sehen Sie die folgenden Elemente:

- **Foundation- und Core Graphics-Framework** Mit diesen notwendigen Frameworks können Sie Ihre iPhone-Anwendungen mit denselben grundlegenden Klassen und Aufrufen erstellen, die Sie vom Macintosh kennen.
- **UIKit-Framework** Dieses Framework bietet iPhone-spezifische Elemente für die Benutzerschnittstelle und ist der Schlüssel zur Entwicklung von Anwendungen, die interaktiv auf dem iPhone-Bildschirm ablaufen.
- **HelloWorld.app** Dieser in Rot dargestellte Platzhalter wird verwendet, um Ihre fertige Anwendung zu speichern. Wie auf dem Macintosh sind iPhone-Anwendungen Bundles und bestehen aus vielen Elementen, die in einem zentralen Ordner gespeichert werden.
- **HelloWorld-Info.plist** Diese Datei beschreibt Ihre Anwendung gegenüber dem iPhone-System. Hier geben Sie die zugehörige ausführbare Datei, den Anwendungsbezeichner und andere Schlüsseleigenschaften an. Sie hat dieselbe Funktion wie auf dem Mac.
- **MainWindow.xib** Diese Datei von Interface Builder erstellt ein leeres Fenster, das Sie beim ersten Durchgang nicht benötigen.
- **HelloWorldViewController.xib** Diese Interface Builder-Datei erstellt die Ansicht, die in Ihrer ersten Anwendung angezeigt wird. Sie müssen sie anpassen, um ihr Erscheinungsbild zu ändern.
- **main.m, HelloWorldAppDelegate.h, HelloWorldAppDelegate.m usw.** Diese Dateien enthalten ein grobes Gerüst, das Sie anpassen und erweitern können, um Ihre Anwendung zu erstellen. Sie können sich den Code gern genauer ansehen, werden ihn in dieser Übung aber nicht ändern. Stattdessen verwenden Sie diese Dateien so, wie Xcode sie angelegt hat, und beschränken Ihre Änderungen auf die `.xib`-Datei des Ansichtscontrollers.

2.2.6 Die .xib-Datei des Ansichtscontrollers öffnen

Suchen Sie im Projektfenster von Hello World nach der Datei `HelloWorldViewController.xib`. Wie Sie in Kapitel 1, *Einführung in das iPhone SDK*, gelesen haben, befinden sich in `.xib`-Dateien Interface Builder-Layouts. Doppelklicken Sie auf die `.xib`-Datei, um Interface Builder zu starten und die Datei bearbeiten zu können. Das kann einige Sekunden dauern, da das Programm geöffnet wird und Daten lädt. Sobald es läuft, wechseln Sie in das Hauptfenster für die `.xib`-Datei in Interface Builder, die Sie in *Abbildung 2.4* sehen.

Die drei Symbole in diesem Fenster stehen für die drei Elemente der Schnittstelle, die Sie bearbeiten. Ganz rechts finden Sie **View**, also die Ansicht, die standardmäßig ein Member der Klasse `UIView` ist. Sie enthält die Bildschirmelemente, die Sie in Ihrer Anwendung anzeigen möchten.

Links steht `File's Owner`, was den Ansichtscontroller darstellt. Dies ist eine abstrakte Klasse, und ihr Symbol wird als *Proxy* (Stellvertreter) bezeichnet, da sie zwar eine Rolle in Interface Builder (IB) spielt, das Objekt selbst aber nicht in das `.xib`-Archiv eingebettet ist.

▸ *Abbildung 2.4: Das Interface Builder-Fenster für die* `.xib`*-Datei eines Ansichtscontrollers*

Ansichtscontroller haben keine grafische Darstellung. Sie handhaben Ansichten, zeigen aber selbst nichts an. Jeder Ansichtscontroller verfügt über eine Instanzvariable, die als »Ansicht« bezeichnet und auf irgendeine **UIView** (in diesem Fall die auf der rechten Seite) gesetzt wird. Diese `UIView` ist dann für die eigentliche Darstellung auf dem Bildschirm verantwortlich. Bei einem Ansichtscontroller steht der Proxy `File's Owner` also für das Objekt, das die `.xib`-Datei lädt und besitzt.

Dies können Sie sich ansehen, indem Sie ein Informationsfenster öffnen. Wählen Sie **Tools ▸ Identity Inspector** ⌘ + 4, klicken Sie auf das Objekt `File's Owner`, und schauen Sie sich dessen Klasse im Informationsfenster an. Sie ist auf `HelloWorldViewController` gesetzt. Klicken Sie dann auf das `View`-Objekt. Seine Klasse ist `UIView`.

Um die Verbindung dazwischen zu sehen, klicken Sie im Fenster der `.xib`-Datei auf `File's Owner`, und wählen Sie dann **Tools ▸ Connections Inspector** ⌘ + 2. Sie können erkennen, dass ein Outlet aufgelistet ist. Outlet ist der IB-Begriff für eine Instanzvariable. Wenn Sie mit dem Mauszeiger über den Listeneintrag **view–View** des Informationsfensters **Connection Inspector** fahren, wird das **View**-Objekt im Fenster der `.xib`-Datei hervorgehoben, da das Ansichtsoutlet für den Ansichtscontroller bereits mit dieser Ansicht verbunden ist. Xcode hat die Datei so vorbereitet, dass sie mit dieser Ansicht korrekt funktioniert.

Das letzte Symbol, in der Mitte von *Abbildung 2.4*, heißt `First Responder` und ist wie `File's Owner` ein Proxy. Es steht für das Objekt auf dem Bildschirm, das zurzeit auf Berührungen durch den Benutzer reagiert. Während der Lebensdauer der Anwendung wird im Laufe der Benutzeraktion immer wieder ein anderes Objekt zum »first responder«. Stellen Sie sich z. B. ein Formular vor: Wenn der Benutzer ein Textfeld darin antippt, wird dieses Feld aktiv und übernimmt die First-Responder-Rolle.

2.2.7 Die Ansicht bearbeiten

Um mit der Anpassung der Ansicht zu beginnen, doppelklicken Sie im Fenster der `.xib`-Datei auf das Objekt `View`. Dadurch wird ein neues Editorfenster geöffnet (siehe *Abbildung 2.5* links). Standardmäßig ist die Ansicht leer. Es ist Ihre Aufgabe, sie anzupassen und mit Inhalten zu füllen. Dazu verwenden Sie zwei Werkzeuge, nämlich die Interface Builder-Bibliothek und das Informationsfenster.

Wechseln Sie zum Ansichtseditor, indem Sie darauf klicken, und wählen Sie **Tools ▸ Attributes Inspector** ⌘ + 1. Im Informationsfenster finden Sie das Auswahlfeld **Background**. Klicken Sie darauf, und wählen Sie eine neue Farbe aus der Palette. Die Hintergrundfarbe der Sicht wird automatisch aktualisiert. Wie Sie sehen, können Sie im Attributfenster die Eigenschaften des zurzeit ausgewählten Objekts ändern, in diesem Fall die Eigenschaft der Ansicht, die Sie bearbeiten.

▸ *Abbildung 2.5: Ein leeres Ansichtseditor-Fenster (links) und die Interface Builder-Bibliothek*

Öffnen Sie als Nächstes die Bibliothek über **Tools ▸ Library** ⌘ + ⇧ + L. Sie zeigt eine Liste aller vorgefertigten Cocoa Touch-Elemente, die Sie in IB-Projekten verwenden können (siehe *Abbildung 2.5* rechts). Dazu gehören sowohl abstrakte Elemente wie Ansichtscontroller als auch grafische Bestandteile wie Schaltflächen und Schieberegler. Geben Sie in das Suchfeld am unteren Rand des Bibliotheksfensters `UILabel` ein. Ziehen Sie das Label aus dem mittleren Bereich, der in *Abbildung 2.5* (rechts) hervorgehoben ist, und lassen Sie es über der Ansicht los. Alternativ können Sie auch auf das Label im mittleren Bereich doppelklicken, wodurch es automatisch zur Ansicht hinzugefügt wird. Im unteren Bereich finden Sie eine Erklärung der ausgewählten Klasse. Von dort können Sie nichts in die Ansicht ziehen.

Nachdem Sie das Label in die Ansicht gezogen haben, doppelklicken Sie darauf und ändern die Bezeichnung »Label« in »Hello World«. Sie können das Label auch nach Ihrem Geschmack im Fenster verschieben oder seine Position im Größeninformationsfeld ⌘ + 3 festlegen. Sobald Sie mit dem Ergebnis zufrieden sind, sichern Sie das Projekt mit **File ▸ Save** ⌘ + S. Sie haben die Ansicht jetzt mit diesem Inhalt angepasst.

2.2.8 Die Anwendung ausführen

Kehren Sie zu Xcode und zum Projektfenster zurück. Wählen Sie **Project ▸ Set Active SDK ▸ iPhone Simulator (3.0)**. Dadurch weisen Sie Xcode an, das Projekt für den iPhone-Simulator auf dem Macintosh zu kompilieren. Sollten Sie bereits eine neuere Version des iPhone SDK installiert haben, z. B. 3.1, so können Sie natürlich auch diese bei der Festlegung des zu verwendenden Ziels auswählen. Klicken Sie im Hauptprojektfenster auf **Build and Go**, und warten Sie, während Xcode die Arbeit erledigt. Es dauert einige Sekunden, bis die Kompilierung abgeschlossen ist. Danach startet Xcode automatisch den Simulator, installiert das Projekt und führt es aus. *Abbildung 2.6* zeigt das Ergebnis: die Hello-World-Anwendung, wie sie auf dem Simulator läuft.

▸ *Abbildung 2.6: Die angepasste Hello-World-Anwendung wird auf dem Simulator ausgeführt.*

Hinweis

Soll Ihre iPhone-Anwendung auch frühere Geräte, der ersten und zweiten Generation von iPhone oder iPod touch unterstützen, müssen Sie die entsprechenden SDKs installiert haben und Ihr Projekt dementsprechend anpassen. Das minimal zu unterstützende SDK legen Sie innerhalb der Projekteinstellungen, und zwar im Bereich **Build** mit dem Eintrag **Base SDK** fest.

2.3 Den Simulator verwenden

Der Simulator aus dem iPhone SDK ermöglicht es, Anwendungen auf dem Macintosh zu testen und dabei viele der Aktionen auszuprobieren, die Benutzer auf physischen Geräten durchführen. Da der Macintosh jedoch kein berührungsempfindliches Handheld-Gerät ist, müssen Sie Menüs, Tastaturkurzbefehle und die Maus verwenden, um die Interaktion zu simulieren. *Tabelle 2.1* zeigt, wie Sie die einzelnen Tätigkeiten im Simulator vornehmen.

Aktion	Entsprechung im Simulator
Gerät drehen	**Hardware ▸ Links drehen** ⌘ + ← oder **Hardware ▸ Rechts drehen** ⌘ + →
Gerät schütteln	**Hardware ▸ Schüttelgeste** ⌘ + ⇧ + Z. Hierdurch wird mithilfe eines Bewegungsereignisses ein Stoß simuliert, aber keine anderen Aktionen des Beschleunigungsmessers.
Taste Home drücken	Klicken Sie auf die Schaltfläche **Home** auf dem Simulatorbildschirm, oder wählen Sie **Hardware ▸ Home** ⌘ + ⇧ + H.
Gerät sperren	**Hardware ▸ Sperren** ⌘ + L
Tippen und doppelt tippen	Einfacher Klick bzw. Doppelklick mit der Maus
Auf die virtuelle Tastatur tippen	Klicken Sie auf die virtuelle Tastatur, oder verwenden Sie die physische Tastatur des Macs.
Ziehen, wegstreichen und blättern	Klicken Sie, ziehen Sie, und lassen Sie die Maustaste wieder los. Die Geschwindigkeit des Ziehvorgangs bestimmt die Aktion. Zum Blättern müssen Sie sehr schnell ziehen.
Auf- und zuziehen	Drücken Sie die Taste Alt auf der Tastatur, und halten Sie sie gedrückt. Wenn die beiden Punkte erscheinen, ziehen Sie sie voneinander weg oder aufeinander zu.
Speichermangel	**Hardware ▸ Speicherwarnhinweis simulieren**
Telefonanruf (nur optische Anzeige)	**Hardware ▸ Status für eingeh. Anrufe ein/aus**. Auf dem iPhone können Sie eine Anwendung ausführen, während Sie telefonieren. Während der Dauer des Anrufs erscheint am oberen Rand des Bildschirms die Anrufleiste.

▸ *Tabelle 2.1: Simulatorbefehle für iPhone-Aktionen*

2.3.1 Hinter den Kulissen des Simulators

Da der Simulator auf einem Macintosh läuft, kompiliert Xcode simulierte Anwendungen für den Intel-Chip. Innerhalb des Simulators auf dem Macintosh läuft Ihre Anwendung mithilfe eines Satzes von Intel-Frameworks, die den Frameworks entsprechen, die mit dem iPhone-Betriebssystem auf physischen Geräten installiert werden. Die Simulatorversionen dieser Frameworks befinden sich im Xcode-Entwicklerverzeichnis unter `/Developer/Platforms/iPhoneSimulator.platform/Developer/SDKs/iPhoneSimulator3.x.sdk/System/Library`.

Ihre Anwendungen finden Sie in Ihrem `~/Library/Application Support`-Ordner unter `iPhone Simulator/User/Applications/` bzw. bei der Verwendung von Mac OS Snow Leopard und Xcode 3.2 unter iPhone `Simulator/3.x/Applications/`. Es ist hilfreich, diesen Ordner aufzusuchen und einen Blick unter die Haube zu werfen, um zu erkennen, wie Anwendungen auf dem iPhone bereitgestellt werden.

Jede Anwendung wird in einer eigenen Sandbox gespeichert. Der Name der Sandbox ist zufällig und besteht aus einem eindeutigen Code (erstellt von `CFUUIDCreateString()`). Bis zu OS 3.0 wurde jeder Sandbox-Ordner von einer Sandbox-Datei begleitet, die denselben Namen wie der Ordner trug, aber die Endung `.sb` aufwies. Sie diente zur Speicherung der Berechtigungen. Seit Version 3.0 wird diese Sandbox-Berechtigungsdatei nicht mehr verwendet. Früher mussten Sie sowohl den Ordner als auch die `.sb`-Datei komprimieren, um kompilierte Simulatoranwendungen an andere weiterzugeben. Jetzt reicht es dafür aus, nur den Ordner zu komprimieren.

Der Name der Sandbox sagt nichts über die Anwendung aus, die sich darin befindet, weshalb Sie hineinschauen müssen, um zu sehen, was sich dort befindet. Im Inneren der Sandbox finden Sie das Anwendungsbundle (z. B. `HelloWorld.app`) und die Ordner `Documents`, `Library` und `/tmp`. Wenn eine Anwendung ausgeführt wird, ist ihr Zugriff auf diese lokalen Ordner beschränkt – im Gegensatz zu einer Macintosh-Anwendung, die den `Library`-Hauptordner des Benutzers nicht verwendet.

Wenn Sie den Anwendungsordner aufräumen möchten, können Sie Dateien direkt löschen, sofern der Simulator nicht läuft. Alternativ können Sie auch im Simulator die Methode »Tippen und bis zum Zittern halten« verwenden, die Sie auch auf dem iPhone einsetzen. Wenn Sie auf ein Anwendungssymbol tippen (bzw. im Simulator darauf klicken) und es einige Sekunden lang festhalten, fängt es an zu zittern. Dadurch gerät es in den Bearbeitungsmodus, in dem Sie die Symbole verschieben oder auf das **X**-Symbol in der Ecke tippen können, um die Anwendung mitsamt ihren Daten zu löschen. Über die Home-Taste verlassen Sie den Bearbeitungsmodus wieder. Sie können auch sämtliche Simulatordaten löschen, indem Sie **iPhone Simulator ▸ Inhalte und Einstellungen zurücksetzen** auswählen.

Anwendungen können zwar nicht auf den Bibliotheksordner des Benutzers zugreifen, aber Sie können es. Wenn Sie die Bibliothek des Simulators ändern möchten, finden Sie die Dateien im Ordner `iPhone Simulator/User/Library` im Ordner `Application Support` Ihres Benutzerordners. Durch die Bearbeitung der Bibliothek können Sie z. B. Anwendungen testen, die auf das Adressbuch zurückgreifen. Um Ihren Quellcode zu überprüfen, laden Sie eigene `sqlitedb`-Adressbuchdateien mit nur wenigen Kontakten in `Library/AddresBook`.

Hinweis

Der iPhone-Simulator und Mac OS X verwenden getrennte Zwischenablagen. Der Simulator speichert seine eigenen Zwischenablagendaten, die er über die neuen Kopier- und Einfügefunktionen der Firmware von Version 3.0 gewinnt. Sie können zwar **Edit ▸ Paste** ⌘ + V verwenden, um Text vom Macintosh in Simulatoranwendungen zu kopieren, aber dies berührt nicht die Zwischenablage des Simulators.

2.4 Die Sparversion von Hello World

Wir sehen uns das iPhone SDK weiter an und bleiben bei »Hello World« als Beispiel, aber jetzt geht es um die nützliche Kunst, sparsame Anwendungen zu erstellen, d. h., Sie erfahren, wie Sie Anwendungen von Grund auf neu erstellen, ohne fünf Quell- und zwei Schnittstellendateien. Mit der folgenden Anleitung machen Sie genau das: Sie erstellen eine sehr einfache Hello-World-Anwendung, die zwar den `UIViewController`-Ansatz aus dem vorherigen Beispiel widerspiegelt, dabei aber mit einer einzigen Datei und ohne `.xib`-Dateien auskommt.

Sie beginnen damit, dass Sie in Xcode ein neues Projekt anlegen (**File ▸ New Project**, ⌘ + ⇧ + N). Wählen Sie diesmal das Template **Window-based Application**, und speichern Sie das Projekt als `HelloWorld2` auf Ihrem Schreibtisch. Im Projektfenster wählen Sie in der linken Spalte den Ordner **Classes** und drücken die ⟵-Taste, um ihn zu löschen. Wählen Sie **Also Move to Trash**, wenn Ihnen diese Option angezeigt wird, und löschen Sie danach `MainWindow.xib`.

Doppelklicken Sie auf die Datei `HelloWorld2-Info.plist` (im Ordner `Resources`), um sie im Editor zu öffnen. In der letzten Zeile sollte `Main nib file base name` stehen. Wählen Sie diese Zeile aus, und löschen Sie sie. Speichern und schließen Sie die Datei.

Öffnen Sie `main.m`, und ersetzen Sie den Inhalt durch den Code aus *Listing 2.1*. Den Quellcode finden Sie auch im Beispielcode zu diesem Buch (Einzelheiten erfahren Sie im Vorwort), sodass Sie ihn nicht per Hand eingeben müssen.

```
#import <UIKit/UIKit.h>

@interface HelloWorldViewController : UIViewController
@end

@implementation HelloWorldViewController
- (void)loadView
{
   UIView *contentView = [[UIView alloc] initWithFrame:
      [[UIScreen mainScreen] applicationFrame]];

   contentView.backgroundColor = [UIColor lightGrayColor];
```

```
    UILabel *label = [[UILabel alloc] initWithFrame:
        CGRectMake(0.0f, 0.0f, 320.0f, 30.0f)];
    label.text = @"Hello World";
    label.center = contentView.center;
    label.backgroundColor = [UIColor clearColor];
    label.textAlignment = UITextAlignmentCenter;

    [contentView addSubview:label];
    [label release];

    self.view = contentView;
    [contentView release];
}
@end

@interface HelloWorldAppDelegate : NSObject <UIApplicationDelegate>
{

}
@end

@implementation HelloWorldAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    UIWindow *window = [[UIWindow alloc] initWithFrame:
        [[UIScreen mainScreen] bounds]];

    HelloWorldViewController *hwvc;
    hwvc = [[HelloWorldViewController alloc] init];
    [window addSubview:hwvc.view];
    [window makeKeyAndVisible];
}
@end

int main(int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
        @"HelloWorldAppDelegate");
    [pool release];
    return retVal;
}
```

▶ *Listing 2.1: Einfache* `main.m`

Was macht diese Anwendung? Sie erstellt eine Ansicht, färbt den Hintergrund ein und fügt ein Label mit dem Text »Hello World« hinzu. Mit anderen Worten, sie tut genau dasselbe wie das erste Beispiel, aber manuell, ohne Interface Builder.

Alles beginnt in `main.m` damit, dass der Autorelease-Pool eingerichtet und `UIApplicationMain` aufgerufen wird. Anschließend geht die Steuerung an den Anwendungs-Delegate über, der im letzten Argument des Aufrufs angegeben ist. Dies ist ein entscheidender Punkt für Projekte ohne Interface Builder – und eine Hürde, über die viele neue iPhone-Entwickler gestolpert sind.

Der Delegate, der die Nachricht über den Start der Anwendung empfängt, erstellt ein neues Fenster und eine neue Instanz des benutzerdefinierten Ansichtscontrollers und fügt die Ansicht des Controllers dem Fenster hinzu. Der Ansichtscontroller wartet auf die Anforderung, seine Sicht zu laden. Wenn diese eintrifft, führt er `loadView` aus, womit er die Ansicht erstellt und den Text »Hello World« hinzufügt.

Ansichten manuell zu erstellen bedeutet, die Hauptansicht und deren Kindobjekte mit der Methode `loadView` einzurichten. In diesem Beispiel beginnt der Vorgang damit, dass eine neue Ansicht erstellt und angewiesen wird, den gesamten verfügbaren Platz der Anwendung auszufüllen. Anschließend wird die Hintergrundfarbe festgelegt, in diesem Fall auf Hellgrau. Danach erstellt der Beispielcode eine neue Instanz der Klasse `UILabel`. Alle Labeleigenschaften werden manuell festgelegt.

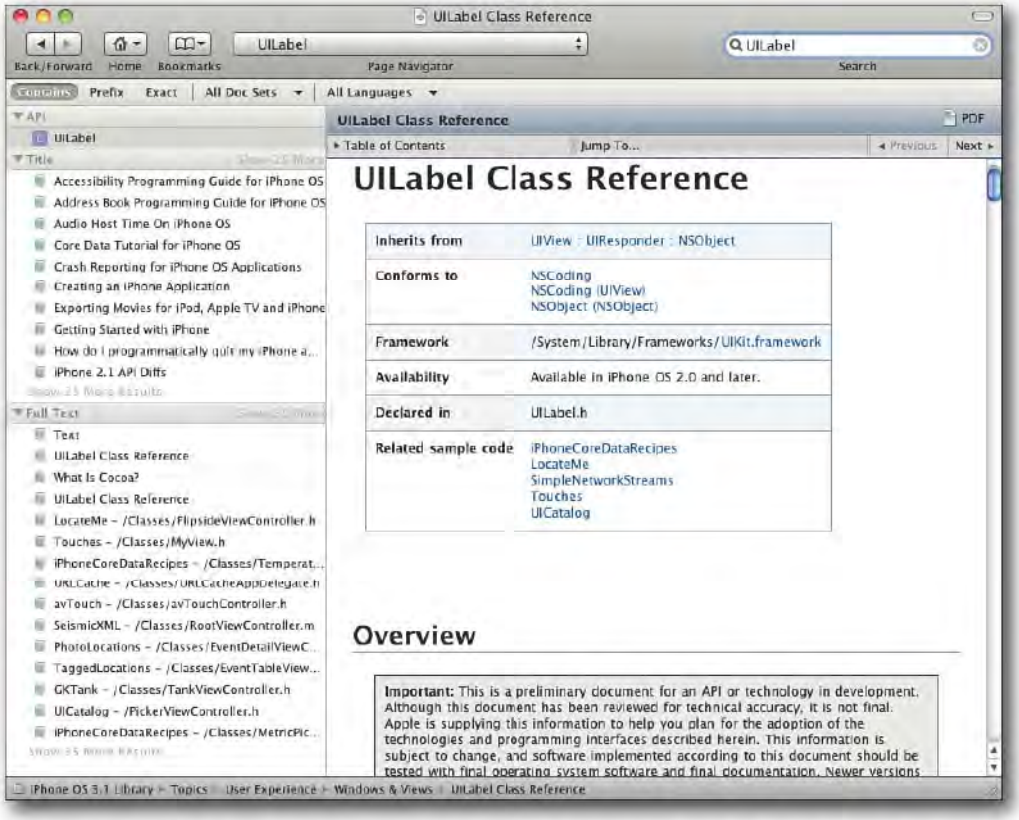
In Interface Builder erfüllt das Attributfenster diese Funktion. Es zeigt die Labeleigenschaften und enthält interaktive Steuerelemente, um Eigenschaften wie die linksbündige, zentrierte oder rechtsbündige Ausrichtung zu wählen. Hier wird die Ausrichtung im Programm auf die Konstante `UITextAlignmentCenter` gesetzt, die Hintergrundfarbe auf transparent. Ebenfalls programmgesteuert, wird das Label über seine Eigenschaft `center` in die richtige Position verschoben. Letzten Endes führen das manuelle Vorgehen und die Verwendung von Interface Builder zum gleichen Ergebnis, doch hier wendet der Programmierer Kenntnisse über die SDK-APIs an, um die gleichwertigen Befehle zu erreichen.

Wie bei den anderen Beispielen in diesem Buch enthält der Code keine `dealloc`-Methode, da der Anwendungs-Delegate niemals aufgerufen wird. Das iPhone-Betriebssystem gewinnt den gesamten Anwendungsspeicher zurück, wenn es die Anwendung beendet. Technisch gesehen weist der Ansichtscontroller ein Speicherleck auf, doch in der Praxis ist das kein Problem.

2.4.1 Die SDK-APIs durchsuchen

Die iPhone SDK-APIs sind vollständig dokumentiert, und diese Dokumentation ist in Xcode zugänglich. Wählen Sie **Help ▸ Documentation** ⌘ + Alt + ⇧ + ?, um die Xcode Developer Documentation zu öffnen. Wenn Sie in der linken Spalte **Apple iPhone OS 3.0** wählen (unter Xcode 3.2 nutzen Sie bitte die **Home**-Schaltfläche in der oberen Menüleiste) und oben rechts im Suchfeld `UILabel` eingeben, wird die gesamte Referenz der Klasse `UILabel` angezeigt (siehe *Abbildung 2.7*), in der Sie alle Klassenmethoden, Eigenschaften und Instanzmethoden sowie einen allgemeinen Überblick über die Klasse finden.

Die Dokumentation in Xcode ist gründlich und verständlich. Damit haben Sie unmittelbaren Zugriff auf die gesamte SDK-Referenz. Sie können alles nachschlagen, was Sie brauchen, ohne Xcode verlassen zu müssen. Wenn Inhalte veralten, können Sie Aktualisierungen über ein Abonnementsystem direkt in Xcode herunterladen.



▸ *Abbildung 2.7: Apple bietet eine vollständige Entwicklerdokumentation an, die direkt in Xcode zugänglich ist.*

Interface Builder ist für alle Entwickler ungeachtet ihrer Erfahrung äußerst nützlich. Doch bei vielen Entwicklungsaufgaben, z. B. bei der Einrichtung von Instanzvariablen und Callbacks, kann es Sie stark einschränken. Im Code können Sie viele Dinge tun, die in Interface Builder nicht möglich sind. Die Entwicklerdokumentation von Xcode hilft Ihnen, diese Einschränkungen zu überwinden. Setzen Sie Interface Builder hauptsächlich für die Gestaltung der Schnittstellen ein, denn das kann dieses Werkzeug am besten. Wenn Sie tiefergehende Kenntnisse des SDK haben, können Sie raffiniertere und leistungsfähigere Anwendungen erstellen.

2.4.2 Interface Builder-Dateien in Objective-C-Code umwandeln

Ein praktisches Open-Source-Hilfsprogramm von Adrian Kosmaczewski erlaubt es Ihnen, Interface Builder-Dateien in Objective-C-Code umzuwandlen. Damit können Sie alle Layoutinformationen und Eigenschaften des grafischen Designs herausziehen und erkennen, wie sie handkodiert aussehen würden. Das Programm *nib2objc* macht genau das, was sein Name andeutet. Damit können Sie aus der Umwandlung Code gewinnen, der die Klassenkonstruktoren, Methodenaufrufe usw. berücksichtigt.

Listing 2.2 zeigt das Ergebnis, das Sie bekommen, wenn Sie *nib2objc* für die `.xib`-Datei ausführen, die wir im ersten Beispiel verwendet haben. Vergleichen Sie diesen Code mit der viel einfacheren (und weniger gründlichen) manuellen Version aus *Listing 2.1*. Dieser Code führt im Grunde genommen dieselben Aufgaben aus: Er erstellt eine neue Sicht und ein neues Label und fügt das Label zu der Sicht hinzu. Das Konvertierungsprogramm zeigt jedoch alle zugrunde liegenden Eigenschaften an, nicht nur die wenigen, die wir in *Listing 2.1* bearbeitet haben.

Um sich den ursprünglichen XML-Text von Interface Builder anzusehen, öffnen Sie die `.xib`-Datei in TextEdit. Dies können Sie tun, indem Sie in der Terminal-Befehlszeile den Befehl `open -e` verwenden, während Sie sich im Projektordner für HelloWorld befinden:

```
open -e HelloWorldViewController.xib
```

Hinweis

Das Programm *nib2objc* finden Sie unter *http://github.com/akosma/nib2objc*. Es ist unter einer allgemeinen Lizenz erhältlich, die von Ihnen verlangt, es nicht für unlautere Zwecke einzusetzen.

```
UIView *view6 = [[UIView alloc] initWithFrame:CGRectMake(0.0, 0.0,
   320.0, 460.0)];
view6.frame = CGRectMake(0.0, 0.0, 320.0, 460.0);
view6.alpha = 1.000;
view6.autoresizingMask = UIViewAutoresizingFlexibleWidth |
UIViewAutoresizingFlexibleHeight;
view6.backgroundColor = [UIColor colorWithRed:0.740 green:0.750
   blue:0.638 alpha:1.000];
view6.clearsContextBeforeDrawing = NO;
view6.clipsToBounds = NO;
view6.contentMode = UIViewContentModeScaleToFill;
view6.hidden = NO;
view6.multipleTouchEnabled = NO;
view6.opaque = YES;
view6.tag = 0;
view6.userInteractionEnabled = YES;
```

```
UILabel *view8 = [[UILabel alloc] initWithFrame:CGRectMake(100.0, 188.0,
  89.0, 21.0)];
view8.frame = CGRectMake(100.0, 188.0, 89.0, 21.0);
view8.adjustsFontSizeToFitWidth = YES;
view8.alpha = 1.000;
view8.autoresizingMask = UIViewAutoresizingFlexibleRightMargin |
   UIViewAutoresizingFlexibleBottomMargin;
view8.baselineAdjustment = UIBaselineAdjustmentAlignCenters;
view8.clearsContextBeforeDrawing = YES;
view8.clipsToBounds = YES;
view8.contentMode = UIViewContentModeScaleToFill;
view8.enabled = YES;
view8.font = [UIFont fontWithName:@"Helvetica" size:17.000];
view8.hidden = NO;
view8.lineBreakMode = UILineBreakModeTailTruncation;
view8.minimumFontSize = 10.000;
view8.multipleTouchEnabled = NO;
view8.numberOfLines = 1;
view8.opaque = NO;
view8.shadowOffset = CGSizeMake(0.0, -1.0);
view8.tag = 0;
view8.text = @"Hello World";
view8.textAlignment = UITextAlignmentLeft;
view8.textColor = [UIColor colorWithRed:0.000 green:0.000 blue:0.000
   alpha:1.000];
view8.userInteractionEnabled = NO;

[view6 addSubview:view8];
```

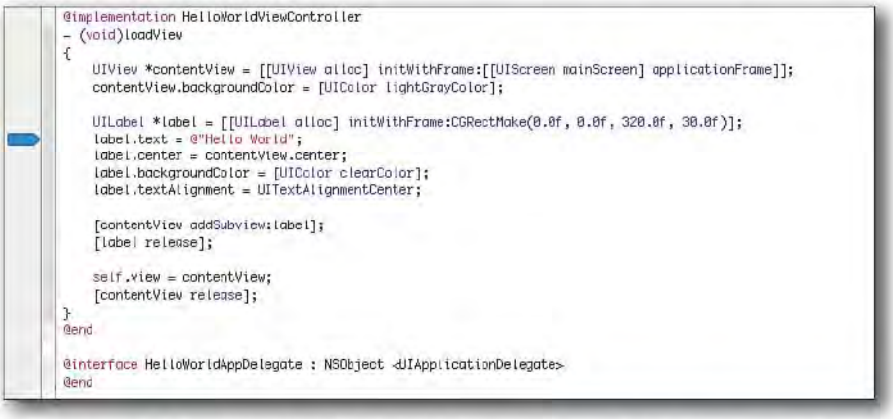
▸ *Listing 2.2:* `HelloWorldViewController.xib` *nach der Konvertierung in Objective-C*

2.5 Den Debugger verwenden

Der integrierte Debugger von Xcode ist ein wertvolles Werkzeug für die Entwicklung von iPhone-Anwendungen. Die folgenden Abschnitte zeigen Ihnen, wo der Debugger zu finden ist, und stellen einige grundlegende Möglichkeiten vor, ihn in Ihrem Programm einzusetzen. In diesen Schritten lernen Sie, wie Sie Haltepunkte setzen und die Debuggerkonsole verwenden, um Einzelheiten des Programms zu untersuchen. Voraussetzung ist, dass Sie das zweite, einfachere Hello-World-Beispiel verwenden, dass das Projektfenster geöffnet ist und die Datei `main.m` angezeigt wird.

2.5.1 Einen Haltepunkt setzen

Suchen Sie in Ihrem Hello-World-Programm die Methode `loadView`. Klicken Sie in die äußerste linke Spalte des Xcode-Fensters, unmittelbar links neben der Zeile mit der Zuweisung `label.text`. Es erscheint eine blaue Anzeige für den Haltepunkt (siehe *Abbildung 2.8*). Die dunkelblaue Farbe besagt, dass ein Haltepunkt aktiv ist. Klicken Sie einmal, um ihn zu deaktivieren – der Haltepunkt wird hellblau –, und noch einmal, um ihn wieder zu aktivieren. Sie können Haltepunkte entfernen, indem Sie sie vom Bildschirm ziehen, und hinzufügen, indem Sie in die Spalte links vom Code klicken.



```
@implementation HelloWorldViewController
- (void)loadView
{
    UIView *contentView = [[UIView alloc] initWithFrame:[[UIScreen mainScreen] applicationFrame]];
    contentView.backgroundColor = [UIColor lightGrayColor];

    UILabel *label = [[UILabel alloc] initWithFrame:CGRectMake(0.0f, 0.0f, 320.0f, 30.0f)];
    label.text = @"Hello World";
    label.center = contentView.center;
    label.backgroundColor = [UIColor clearColor];
    label.textAlignment = UITextAlignmentCenter;

    [contentView addSubview:label];
    [label release];

    self.view = contentView;
    [contentView release];
}
@end

@interface HelloWorldAppDelegate : NSObject <UIApplicationDelegate>
@end
```

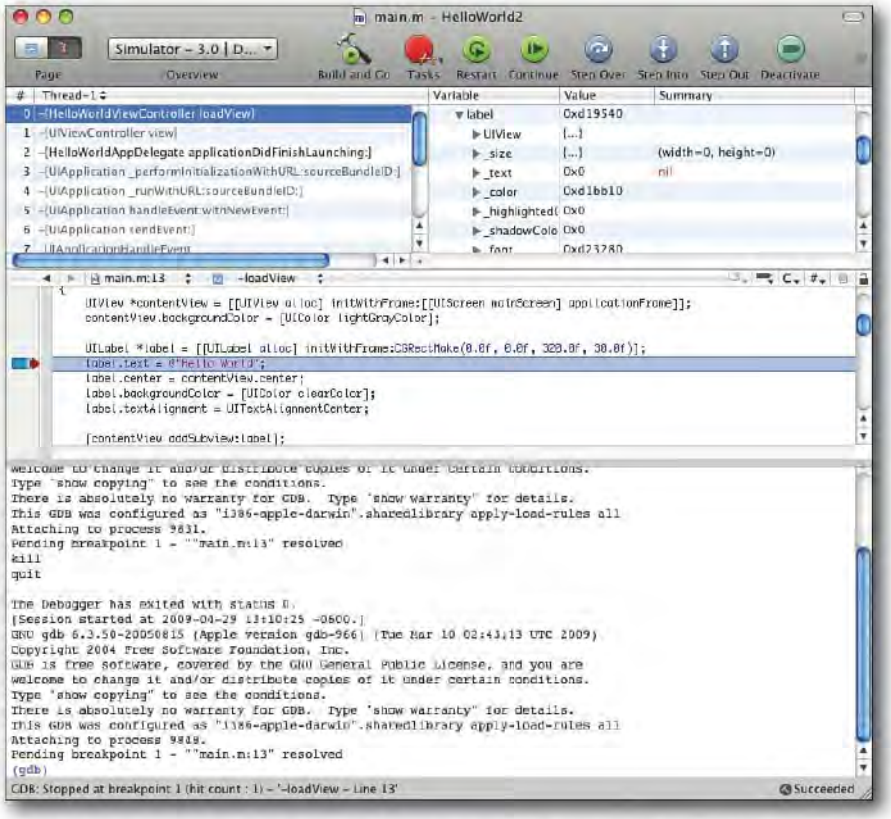
► *Abbildung 2.8: In der linken Spalte des Xcode-Fensters erscheinen blaue Markierungen für Haltepunkte.*

2.5.2 Den Debugger öffnen

Klicken Sie im Projektfenster auf den Umschalter **Project/Debug**, um den Debugger anzuzeigen. Der Debugger enthält ein grafisches Front-End für die Untersuchung von Programmobjekten, ein Quellcodefenster und einen Protokollbereich mit interaktiver *gdb*-Shell. Oben rechts im Debugger finden Sie die Schaltfläche **Activate/Deactivate**. Stellen Sie sicher, dass der Debugger aktiviert ist, dass die Schaltfläche also die Aufschrift **Deactivate** zeigt.

Das Programm ausführen

Stellen Sie sicher, dass der Haltepunkt dunkelblau ist und dass die Schaltfläche am oberen Rand des Xcode-Bildschirms mit **Deactivate** bezeichnet ist (was bedeutet, dass der Haltepunkt aktiv ist), und klicken Sie auf **Build and Go**, um das Programm im Simulator auszuführen. Das Programm hält automatisch an, wenn es den Haltepunkt erreicht. Das Simulatorfenster bleibt schwarz, und das Debuggerfenster wird aktualisiert, um die interaktive Schnittstelle aus *Abbildung 2.9* zu zeigen.



▶ *Abbildung 2.9: Mit dem grafischen Debugger von Xcode können Sie den Zustand des Programms interaktiv untersuchen. Im Konsolenfenster wird gleichzeitig eine Befehlszeilenversion von gdb ausgeführt, wie Sie an der Eingabeaufforderung (gdb) erkennen können. Am aktiven Haltepunkt erscheint ein roter Pfeil.*

Das Label inspizieren

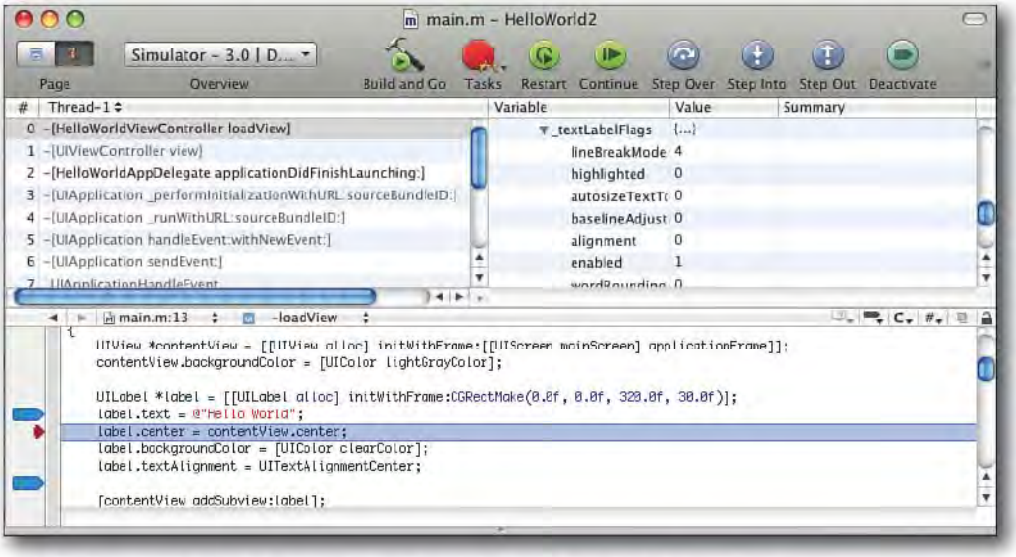
Wenn die Ausführung am Haltepunkt angehalten wurde, können Sie den interaktiven Debugger oder die gdb-Kommandozeile verwenden, um Objekte in Ihrem Programm zu inspizieren. Gehen Sie in diesem Beispiel die Variablenkette durch – das Informationsfenster für Variablen befindet sich oben rechts im Debuggerfenster. Suchen Sie die Variablenliste, und scrollen Sie bis knapp unter die Liste **Arguments** nach unten. Klicken Sie darin auf das Einblenddreieck links neben dem Label, um dessen Eigenschaften einzusehen. Beachten Sie, dass der Text entweder `nil` oder `Invalid` lautet.

In der oberen Symbolleiste des Fensters sehen Sie die graue Schaltfläche **Step Into**. Klicken Sie einmal darauf. Dadurch wird die Textzuweisung ausgeführt, und der rote Pfeil wandert eine Zeile tiefer. Die Übersicht von `label.text` wird aktualisiert und zeigt jetzt den Text `Hello World` an.

Weitere Haltepunkte setzen

In einer Debuggersitzung können Sie weitere Haltepunkte setzen. Fügen Sie z. B. einen zweiten Haltepunkt unmittelbar unter der Zeile hinzu, die den Text zentriert. Dies können Sie im mittleren Bereich erledigen – es gibt keinen Grund dafür, das ursprüngliche Quellcodefenster erneut zu öffnen. Klicken Sie abermals auf die äußerste linke Spalte neben der Zeile, in der Sie einen Haltepunkt setzen möchten.

Überprüfen Sie, ob die Ausrichtung auf den Standardwert 0 gesetzt ist, indem Sie sich die Eigenschaft `textLabelFlags` des Labels ansehen. Unter Umständen müssen Sie dazu ein wenig nach unten scrollen und die Größe der Variablenspalte anpassen. *Abbildung 2.10* zeigt die beiden Haltepunkte, den roten Pfeil hinter der Zuweisung und den Ausrichtungswert mit dem Standard 0.



► *Abbildung 2.10: In einer Debuggersitzung können Sie auch weitere Haltepunkte setzen.*

Wenn der neue Haltepunkt gesetzt ist, klicken Sie auf die Schaltfläche **Continue**. Hello World wird bis zum nächsten Haltepunkt wieder aufgenommen, wo die Anwendung erneut anhält. Der rote Pfeil zeigt jetzt auf die Zeile `addSubview`, und das Ausrichtungsflag wird von 0 auf 1 aktualisiert, da jetzt der Code ausgeführt worden ist, der den Wert für diese Variable ändert.

Hinweis

Sie entfernen Haltepunkte, indem Sie sie aus der linken Spalte herausziehen.

2.5.3 Textausgabe

Der untere Bereich des Debuggerfensters zeigt die Textausgabe des GNU-Debuggers gdb, die den Ergebnissen und Daten der beiden oberen Bereiche entspricht. Wenn Sie z. B. `backtrace` in der gdb-Eingabeaufforderung eingeben, sollten Sie dieselbe Ablaufverfolgung wie im oberen linken Bereich sehen. Nach dem Stopp beim zweiten Haltepunkt zeigt die Ablaufverfolgung, dass Sie sich bei Zeile 19 des Quelltextes von `main.m` befinden.

Dieser untere Abschnitt wird auch als *Konsole* bezeichnet. Wählen Sie in Xcode **Run ▸ Console** ⌘ + ⇧ + R, um in die Xcode-Konsole zu wechseln. Ist der Debugger bereits geöffnet, springt der Cursor in den unteren Bereich. Dorthin werden im normalen Debugmodus mit Anbindung und bei der Verwendung des Simulators auch standardmäßig die `printf`-, `NSLog`- und `CFShow`-Nachrichten gesendet. Die Größe der Konsole können Sie durch Verschieben ihrer oberen Leiste anpassen. Wenn Sie wollen, können Sie sie auch ganz bis nach oben ziehen. Dadurch erhalten Sie eine fensterfüllende Textkonsole.

Um die Protokollierung in der Konsole zu testen, fügen Sie in den Code die Zeile `NSLog(@"Hello World!")` ein, und zwar hinter der Freigabe von `contentView`. Kompilieren Sie die Anwendung, und führen Sie sie im Simulator aus. Die Protokollnachricht erscheint im Konsolenbereich. Die Konsole führt ein laufendes Protokoll der Nachrichten, und zwar unabhängig davon, wie oft Sie die Anwendung getestet haben. Bei Bedarf können Sie dieses Protokoll manuell leeren.

Um Protokollnachrichten zu sehen, müssen Sie nicht gdb und die Debuggerkonsole verwenden. Angebundene iPhones senden ihre `NSLog`-Ausgaben an den Xcode-Organizer (**Window ▸ Organizer ▸** *Gerätename* **> Console**). Die Organizer-Konsole zeigt die von `NSLog` hervorgerufene Ausgabe. Wenn Sie den vorstehenden `NSLog`-Befehl auf einem iPhone ausführen, wird er wie folgt mit Datum und Uhrzeit, dem Programmnamen und der `NSLog`-Ausgabe (in diesem Fall »`Hello World!`« angezeigt):

```
Sun May 3 09:08:11 unknown HelloWorld2[2198] <Warning>: Hello World!
```

2.5.4 Die Schaltfläche »Clear Log« verschieben

In der aktuellen Version des iPhone SDK liegt die Schaltfläche **Clear Log** standardmäßig ganz rechts in der Symbolleiste und ist daher nicht zu sehen, wenn das Fenster wie in *Abbildung 2.10* zu schmal ist. Sie können trotzdem noch darauf zugreifen, indem Sie auf die doppelten Winkel oben rechts im Fenster klicken. Allerdings empfinde ich die Standardposition der Schaltfläche als unbequem, da ich sie ständig verwende.

Zum Glück ist wie bei den meisten OS X-Symbolleisten auch hier eine Anpassung möglich. Klicken Sie dazu bei gedrückter Ctrl-Taste oder mit der rechten Maustaste auf die Symbolleiste, und wählen Sie **Customize Toolbar** aus dem Kontextmenü. Jetzt können Sie **Clear Log** an eine bessere Stelle ziehen, wobei weniger wichtige Schaltflächen in das Doppelwinkel-Untermenü wandern, während **Clear Log** ständig sichtbar bleibt. Um die Konsole über die Tastatur zu leeren, müssen Sie den äußerst unbequemen Kurzbefehl Ctrl + Alt + ⌘ + R verwenden. Alternativ können Sie hier die Tastenbelegung in Xcode ändern. Rufen Sie dazu das Fenster **Key Bindings** in den Xcode-Voreinstellungen auf ⌘ + ,.

Es ist auch möglich, die Konsole automatisch leeren zu lassen, wobei Sie dabei manchmal aber auch Inhalte entfernen würden, die Sie noch überprüfen wollen. Öffnen Sie die Xcode-Voreinstellungen (**Xcode ▸ Preferences**, ⌘ + ,, und markieren Sie **Debugging ▸ Auto Clear Debug Console**. Dadurch wird die Konsole jedes Mal gelöscht, wenn Sie die Anwendung ausführen.

2.5.5 Zombiemodus

In Horrorfilmen sind Zombies wandelnde Tote. In der Xcode-Terminologie handelt es sich um Objekte, die zerstört oder freigegeben worden sind, denen Sie aber immer noch Nachrichten zu senden versuchen. Beim Debuggen können Sie den Sondermodus `NSZombieEnabled` aktivieren, um Informationen über Nachrichten zu sammeln, die Sie an ungültige Objekte senden. Nehmen wir z. B. an, dass Sie die Instanzvariable `array` erstellt haben und in der Methode `loadView` der Anwendung einrichten und freigeben:

```
// Array erstellen und dann freigeben
array = [[NSArray alloc] init];
[array release];
```

Sollten Sie versuchen, an anderer Stelle im Programm auf dieses Objekt zuzugreifen, stürzt die Anwendung ab. Auch der Debugger schlägt mit `objc_msgSend` fehl. Die beste Chance, die Sie haben, besteht noch darin, in einer Ablaufverfolgung den Fehler zu finden. Solche Ablaufverfolgungen zeigen den Systemstack und die Kette der Nachrichten, die zu dem Fehler geführt haben.

```
- (void) accessArray
{
   CFShow([array self]);
}
```

Mit `NSZombieEnabled` können Sie das Problem genau lokalisieren. Wählen Sie in dem Projekt die Projektansicht (also nicht die Debugansicht), suchen Sie in der Projektliste nach **Executables**, und öffnen Sie das Einblenddreieck. Wählen Sie die Anwendung aus, und klicken Sie auf die blaue **Info**-Schaltfläche in der oberen Symbolleiste. Klicken Sie auf den Titel **Arguments**, und suchen Sie den Abschnitt **Variables** unten im Bereich **Arguments**. Klicken Sie auf **+**, fügen Sie den Namen `NSZombieEnabled` hinzu (Zombie, nicht Zombies!), und geben Sie als Wert `YES` an. Schließen Sie das Fenster **Executable Info**.

Wenn Sie das Programm jetzt ausführen, erhalten Sie eine weit hilfreichere Meldung:

```
2009-05-03 13:20:31.014 HelloWorld[16603:20b] *** -[CFArray self]: message
sent to deallocated instance 0xd32590
```

Jetzt können Sie das interaktive Debuggerfenster verwenden, um die Identität des Objekts über den in der Meldung angegebenen Instanzwert herauszufinden – Sie wissen jetzt genau, welches Objekt sich als Zombie verhält. Um den Zombiemodus wieder zu deaktivieren, löschen Sie `NSZombieEnabled` aus dem Variablenabschnitt im Fenster **Executables Info**. Vergewissern Sie sich, dass Sie das getan haben, bevor Sie die Anwendung verteilen!

Hinweis

In Xcode 3.2 und höher können Sie auch Run ▸ Run with Performance Tool ▸ Instruments ▸ Zombies verwenden.

2.6 Speicherverwaltung

Für das iPhone gibt es keine Garbage Collection. Stattdessen verlässt sich das Speicherverwaltungssystem auf Referenzzählung. Für Sie als Entwickler bedeutet das, dass Sie steuern müssen, wann Objekte erstellt und beibehalten werden und wann ihr Speicher freigegeben wird. Wenn Sie zu viel Speicher verwenden, warnt das iPhone den Anwendungs-Delegate und die `UIViewController`. Der Delegate empfängt `applicationDidReceiveMemoryWarning:`-Callbacks, der Ansichtscontroller `didReceiveMemoryWarning`. Verwenden Sie weiterhin zu viel Speicher, beendet das iPhone die Anwendung, sodass der Benutzer wieder in der SpringBoard-Oberfläche landet. Wie Apple schon wiederholt betont hat, ist dies sicher keine Erfahrung, die Sie Ihren Benutzern zumuten möchten. Ein solches Verhalten sorgt dafür, dass Ihre Anwendung nicht im App Store angenommen wird.

Sie müssen den Arbeitsspeicher in Ihren Programmen sorgfältig verwalten und unter Mangelbedingungen Speicher freigeben. Für Speichermangel gibt es zwei Hauptursachen: zum einen Lecks, bei denen Speicherblöcke zugewiesen werden, auf die kein Zugriff besteht, die aber auch nicht wiederverwendet werden können, und zum anderen zu große Datenmengen, die auf einmal im Arbeitsspeicher gehalten werden.

Hinweis

In Objective-C können Sie Objekte nicht nur beibehalten und freigeben, sondern auch die automatische Freigabe des Speichers nutzen. Wenn Sie einem Objekt, gewöhnlich während es erstellt wird, eine Autorelease-Nachricht senden, heißt das, dass es irgendwann in der Zukunft automatisch verworfen werden soll. Eine Methode, die ein solches Objekt anfordert, kann es direkt verwenden und es am Ende der Ausführungsschleife verwerfen oder es für die weitere Nutzung beibehalten. In Kapitel 3, *Objective-C-Trainigslager*, erfahren Sie genauere Einzelheiten über die Speicherverwaltung.

2.6.1 Lecks

Jedes Objekt in Objective-C wird mit einem ganzzahligen Beibehaltungswert erstellt. Solange dieser Wert 1 oder mehr beträgt, wird die Zuweisung des Objekts nicht aufgehoben. Sie als Entwickler sind dafür verantwortlich, Techniken dafür umzusetzen, dass das Objekt dann freigegeben wird, wenn Sie es nicht mehr brauchen.

Jedes mit `alloc`, `new` oder `copy` erstellte Objekt hat zu Anfang einen Beibehaltungswert von 1. Wenn Sie dem Objekt eine `retain`-Nachricht (»beibehalten«) senden, wird der Zähler um 1 erhöht, bei einer `release`-Nachricht (»freigeben«) dagegen verringert. (Auch die Zuweisung des Objekts zu

einer beibehaltenen Eigenschaft erhöht den Zähler.) Ein Leck liegt vor, wenn Sie keinen Zugriff mehr auf ein Objekt haben, dessen Zählerwert aber nicht auf 0 reduziert worden ist, denn der Speicher dafür ist zugewiesen, kann aber nicht mehr wiedergewonnen werden. Im folgenden Code verursacht das Array ein Leck:

```
NSArray *leakyarray = [[NSMutableArray alloc] init];
leakyarray = nil;
```

2.6.2 Zwischenspeicherung (Caching)

Auch wenn Sie zu viele Daten auf einmal laden, kann Ihnen der Arbeitsspeicher ausgehen. Bei der Nutzung speicherintensiver Ressourcen (wie von Bildern, Audiodaten und PDFs) kann es zu Problemen führen, wenn Sie alles im Programm geladen lassen. Mit der sogenannten *Zwischenspeicherung* (*Caching*) können Sie Ladevorgänge verzögern, bis die Ressourcen tatsächlich gebraucht werden, und den Speicher freigeben, wenn das System ihn benötigt.

Beim einfachsten Ansatz erstellen Sie einen Cache aus einem `NSMutableDictionary`-Objekt. Ein grundlegender Objekt-Cache funktioniert wie folgt: Wenn der Cache abgefragt wird, prüft er, ob das angeforderte Objekt bereits geladen ist. Wenn nicht, sendet der Cache eine Ladeanforderung mit dem Objektnamen. Die Objektlademethode kann Daten lokal oder aus dem Web abrufen. Sobald sie geladen sind, werden die neuen Informationen für den schnellen Abruf im Arbeitsspeicher abgelegt.

Der folgende Code erledigt die erste Hälfte der Pflichten eines Caches. Er verzögert das Laden neuer Daten in den Arbeitsspeicher, bis diese Daten eigens angefordert werden. (In der Praxis verwenden Sie nicht den generischen Typ `id`, sondern geben Ihre Daten- und Rückgabeobjekte mit ihrer Klasse an.)

```
- (id) retrieveObjectNamed: (NSString *) someKey
{
   id object = [self.myCache objectForKey:someKey];
   if (!object)
   {
      object = [self loadObjectNamed:someKey];
      [self.myCache setObject:object forKey:someKey];
   }
   return object;
}
```

Die zweite Aufgabe des Caches besteht darin, sich selbst zu leeren, wenn bei der Anwendung Speichermangel auftritt. Bei einem Dictionary-Cache müssen Sie dazu nur die Objekte entfernen. Wenn die nächste Abrufanforderung eintrifft, kann der Cache die angeforderten Objekte erneut laden.

```
- (void) respondToMemoryWarning
{
   [self.myCache removeAllObjects];
}
```

Durch die Kombination aus verzögertem Laden und durch Speicherwarnungen ausgelöster Leerung kann der Cache speicherfreundlich arbeiten. Sobald Objekte in den Speicher geladen werden, können sie ohne Ladeverzögerung verwendet und wiederverwendet werden. Ist der Speicher jedoch knapp, leistet der Cache seinen Beitrag, um Ressourcen freizugeben, damit die Anwendung weiter ausgeführt werden kann.

2.7 Rezept: Lecks mit Instruments erkennen

Instruments spielt eine wichtige Rolle bei der Optimierung von Anwendungen. Dieses Programm enthält eine Reihe von Werkzeugen, mit denen Sie die Leistung überwachen können. Mithilfe der Leckerkennung können Sie Speicherlecks innerhalb Ihres Programms nachverfolgen, identifizieren und aufheben. *Rezept 2.3* zeigt eine Anwendung, die auf Anforderung zwei Arten von Lecks hervorruft: einen mit `malloc()` erstellten String, für den es kein zugehöriges `free()` gibt, und das `NSArray`-Beispiel, das Sie schon kennen.

Um Instruments im Einsatz zu erleben, laden Sie zunächst das einfache Projekt aus *Rezept 2.3*. Wählen Sie in Xcode **Run ▸ Start with Performance Tool ▸ Leaks**. Dadurch werden sowohl Instruments als auch der Simulator gestartet. Die Anwendung wird im Simulator ausgeführt, wobei Instruments den Fortschritt überwacht.

Klicken Sie auf eine der beiden Schaltflächen, um ein Speicherleck hervorzurufen. Die Schaltfläche **Leak String** führt zu einem Leck aufgrund eines mit `malloc` erstellten Blocks, **Leak Array** zu einem Leck durch ein `NSArray` von 32 Byte. Die Speicherlecks erscheinen in Instruments als orangefarbene Dreiecke, wobei deren Größe der Größe des Lecks entspricht.

Klicken Sie auf **Leaks**, um wie in *Abbildung 2.11* eine Liste der einzelnen Lecks zu sehen. Standardmäßig ist **ObjectAlloc** ausgewählt. Bei jedem Leck ist der Betrag an betroffenem Speicher angegeben, die Adresse, an dem das Leck beginnt, und die Art des Verursacherobjekts.

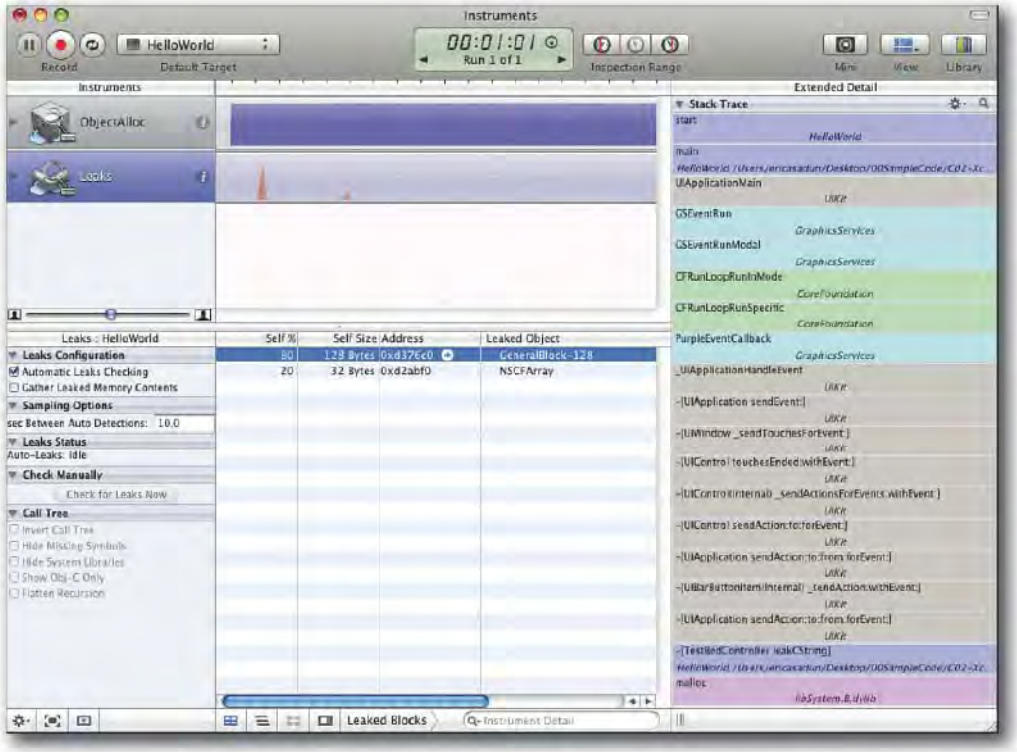
▸ *Abbildung 2.11: Instruments verfolgt Lecks nach, also Speicher, der nicht wiedergewonnen werden kann.*

Um genauere Angaben darüber zu erhalten, wo das Leck auftritt, öffnen Sie den Bereich **Extended Detail** (**View ▸ Extended Detail**, ⌘ + E). Alternativ können Sie auch auf die Schaltfläche **Detail** klicken, die sich links neben den Worten **Leaked Blocks** am unteren Rand des Instruments-Fensters befindet. Wenn Sie auf einen beliebigen Eintrag in der Liste der Lecks klicken, wird in der erweiterten Detailansicht eine Stack-Ablaufverfolgung für das betreffende Leck geöffnet, wie Sie in *Abbildung 2.12* sehen.

Die Stack-Ablaufverfolgung zeigt die Verbindung zwischen dem Leck und seiner Entstehung. Wie Sie an dem Screenshot erkennen können, wurde das fragliche Speicherleck in `leakString` nach dem Erstellen mit `malloc` zugewiesen. Der Einblick in die Entstehung des Objekts kann dabei helfen, herauszufinden, an welchem Punkt in seiner Lebensdauer das Leck auftritt. Wenn Sie das Leck erkannt haben, werden Sie es wahrscheinlich stopfen und Ihre Anwendung von diesem Speicherproblem befreien können.

```
@implementation TestBedViewController
- (void) leakString
{
   char *leakystring = malloc(sizeof(char)*128);
   leakystring = NULL;
}

- (void) leakArray
{
   NSArray *leakyarray = [[NSMutableArray alloc] init];
   leakyarray = nil;
}

- (void) viewDidLoad
{
   // Schaltflächen anlegen
   self.navigationController.navigationBar.tintColor =
      COOKBOOK_PURPLE_COLOR;
   self.navigationItem.rightBarButtonItem = BARBUTTON(@"Leak Array",
      @selector(leakArray));
   self.navigationItem.leftBarButtonItem = BARBUTTON(@"Leak String",
      @selector(leakString));
}
@end
```

▸ *Rezept 2.1: Programmgesteuert Speicherlecks hervorrufen*



► *Abbildung 2.12: Die Stack-Ablaufverfolgung in der Ansicht* **Extended Detail** *zeigt, wo das Leck auftritt.*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 2 und öffnen das Projekt zu diesem Rezept.

2.8 Rezept: Zuweisung von zwischengespeicherten Objekten mit Instruments überwachen

Eine Funktion des Simulators erlaubt Ihnen zu testen, wie Ihre Anwendung auf Speichermangel reagiert. Wenn Sie **Hardware ► Speicherhinweis simulieren** wählen, wird an den Anwendungs-Delegate und den Ansichtscontroller der Aufruf gesendet, nicht benötigten Speicher freizugeben. In Instruments können Sie Speicherzuweisungen in Echtzeit beobachten und auch diese Freigaben überwachen. Dadurch können Sie sicherstellen, dass Ihre Anwendung richtig reagiert, wenn Warnungen auftreten. Mit Instruments können Sie Speicherstrategien wie die zuvor besprochenen Caches prüfen.

Rezept 2.4 erstellt einen grundlegenden Bild-Cache. Anstatt Daten aus dem Web abzurufen, baut dieser Cache leere `UIImage`-Objekte auf, um die tatsächliche Nutzung zu simulieren. Beim Eintreffen von Warnungen wie in *Abbildung 2.13* reagiert der Cache mit der Freigabe seiner Daten.

▶ *Abbildung 2.13: Mit Instruments können Sie Objektzuweisungen überwachen und das Verhalten Ihrer Freigabetechniken bei Speicherwarnungen prüfen.*

Das hier gezeigte Treppenstufenprofil stellt die drei Speicherzuweisungen dar, die nach dem Klick auf die Schaltfläche **Consume** erfolgen. Daraufhin gibt der Simulator eine Speicherwarnung aus. Als Reaktion darauf hat der Speicher pflichtschuldig die in ihm gespeicherten Bilder freigegeben. Der Speicherverbrauch ging dann auf das vorherige Niveau zurück. In Instruments können Sie Ablaufverfolgungsdaten speichern und damit ein Bild von der Leistung einer Anwendung im Laufe der Zeit gewinnen. Wählen Sie **File ▸ Save**, um eine neue Ablaufverfolgungsdatei zu erstellen. Durch einen Vergleich der einzelnen Ausführungen können Sie Änderungen in der Leistung und der Speicherverwaltung zwischen den verschiedenen Versionen Ihrer Anwendung untersuchen.

Einige SDK-Objekte werden bei Bedarf automatisch zwischengespeichert und freigegeben. Die Methode `UIImage imageNamed:` ruft auf diese Weise Bilder ab und speichert sie zwischen. Allerdings ist diese automatische Freigabe zu Recht in den Ruf geraten, sich nicht so sauber zu verhalten, wie sie sollte, und Speicher beizubehalten, den sie eigentlich freigeben müsste. `.nib`-Dateien zum Aufbau von Ansichtscontrollern werden ebenfalls zwischengespeichert und bei Bedarf neu geladen, wenn die Controller wieder erscheinen müssen.

Hinweis

Als Faustregel für die ersten beiden Generationen von iPhones gilt, dass eine Anwendung bis zu 20 Mbyte Speicher verwenden kann, bevor Speicherwarnungen auftreten, und bis zu 30 Mbyte, bevor das iPhone-Betriebssystem die Anwendung beendet.

```
#import "ImageCache.h"

// MyCreateBitmapContext: basierent auf original Apple Code
CGContextRef MyCreateBitmapContext (int pixelsWide, int pixelsHigh)
{
    CGContextRef    context = NULL;
    CGColorSpaceRef colorSpace;
    void *          bitmapData;
    int             bitmapByteCount;
    int             bitmapBytesPerRow;

    bitmapBytesPerRow   = (pixelsWide * 4);
    bitmapByteCount     = (bitmapBytesPerRow * pixelsHigh);

    colorSpace = CGColorSpaceCreateDeviceRGB();
    bitmapData = malloc( bitmapByteCount );
    if (bitmapData == NULL)
    {
        fprintf (stderr, "Memory not allocated!");
        return NULL;
    }
    context = CGBitmapContextCreate (bitmapData,
pixelsWide,
pixelsHigh,
                                                          8,
bitmapBytesPerRow,
colorSpace,
kCGImageAlphaPremultipliedLast);
    if (context== NULL)
    {
        free (bitmapData);
        fprintf (stderr, "Context not created!");
        return NULL;
    }
    CGColorSpaceRelease( colorSpace );

    return context;
}

// Leeres Bild erstellen
UIImage *buildImage(int imgsize)
{
      CGContextRef context  = MyCreateBitmapContext(imgsize, imgsize);
      CGImageRef myRef = CGBitmapContextCreateImage(context);
      free(CGBitmapContextGetData(context));
```

```
        CGContextRelease(context);
        UIImage *img = [UIImage imageWithCGImage:myRef];
        CFRelease(myRef);
        return img;
}

@implementation ImageCache
@synthesize myCache;
+ (ImageCache *) cache
{
        return [[[ImageCache alloc] init] autorelease];
}

- (id) init
{
        if (!(self = [super init])) return self;
        self.myCache = [NSMutableDictionary dictionary];
        return self;
}

- (UIImage *) loadObjectNamed: (NSString *) someKey
{
    // Dieses Beispiel verwendet nicht den Schlüssel, um Daten aus dem Web
    // oder lokal abzurufen, sondern gibt einfach ein weiteres
    // Bild zurück, um den Speicher zu füllen
       return buildImage(320);
}

- (UIImage *) retrieveObjectNamed: (NSString *) someKey
{
        UIImage *object = [self.myCache objectForKey:someKey];
        if (!object)
        {
                object = [self loadObjectNamed:someKey];
                [self.myCache setObject:object forKey:someKey];
        }
        return object;
}

// Cache bei Speicherwarnungen löschen
- (void) respondToMemoryWarning
{
        [self.myCache removeAllObjects];
}
```

```
- (void) dealloc
{
      [self.myCache removeAllObjects];
      self.myCache = nil;
      [super dealloc];
}
@end
```

▸ *Rezept 2.2: Veranschaulichung des Bild-Caches*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 2 und öffnen das Projekt zu diesem Rezept.

2.9 Den statischen Analysator Clang verwenden

Der statische Analysator *LLVM/Clang* erkennt automatisch Bugs in Objective-C-Programmen. Er ist ein hervorragendes Werkzeug, um Speicherlecks und andere Probleme zu finden. Ab Version 3.2 können Sie den Analysator direkt aus Xcode heraus ausführen. Wählen Sie **Build ▸ Build and Analyze** ⌘ + ⇧ + A. Der interaktive Bildschirm aus *Abbildung 2.14* zeigt Ihnen alle vermuteten Lecks und sonstigen möglichen Probleme.

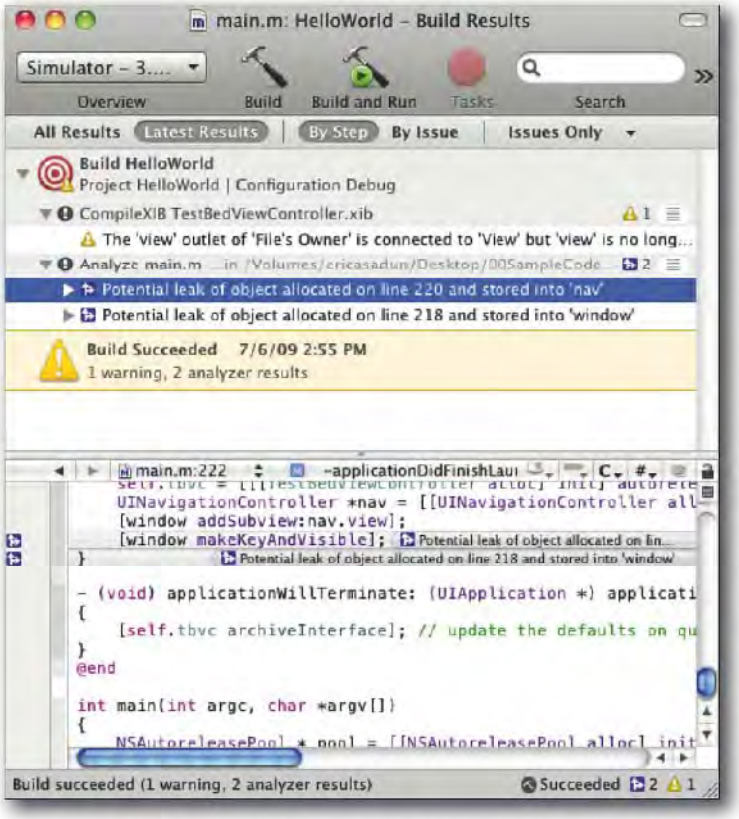
Die Stellen, die der statische Analysator moniert, müssen nicht notwendigerweise Bugs sein. Es ist durchaus möglich, dass Clang gültigen Code als inkorrekt ansieht. Untersuchen Sie alle gemeldeten Probleme sorgfältig, bevor Sie irgendwelche Änderungen an Ihrem Code vornehmen.

Für ältere Versionen von Xcode können Sie eine eigenständige Version von Clang verwenden. Um den statischen Analysator herunterzuladen, zu installieren und in Ihren eigenen Projekten zu verwenden, sind folgende Schritte nötig:

1. Laden Sie den Analysator von *http://clang-analyzer.llvm.org/* herunter. Entkomprimieren Sie den Ordner, und benennen Sie ihn um. Ich habe den Namen »analyzer« verwendet. Passen Sie das Script in Schritt 3 so an, dass es den von Ihnen gewählten Namen widerspiegelt.
2. Verschieben Sie den Ordner an seinen Bestimmungsort, normalerweise in Ihr Benutzerverzeichnis. Auf meinem Rechner habe ich ihn in `~/bin` platziert, weshalb auch das folgende kleine Shellscript diesen Pfad enthält.
3. Fügen Sie das folgende Script namens `clangit` zum Zielordner (bei mir `~/bin`) hinzu, wobei Sie Platzierung und Namen natürlich an die Verhältnisse auf Ihrem Computer anpassen müssen.

```
rm -rf /tmp/scan-build*
rm -rf build
~/bin/analyzer/scan-build –view xcodebuild
```

4. Öffnen Sie ein Xcode-Projekt, wählen Sie die Konfiguration **Simulator | Debug**, und schließen Sie Xcode.
5. Wechseln Sie an der Kommandozeile in den Projektordner. Führen Sie das Script `clangit` von dort aus. Nach der Analyse wird der Bericht automatisch in Ihrem Webbrowser geöffnet.



▸ *Abbildung 2.14: Der statische Analysator Clang erstellt Fehlerberichte über den Quellcode und zeigt diese in einem Xcode-Fenster an.*

2.10 Anwendungen für das iPhone erstellen

Builds und Tests im Simulator bringen Sie nur ein Stück voran. Das Endziel der iPhone-Entwicklung besteht darin, Anwendungen zu schreiben, die auf physischen Geräten laufen. Es gibt drei Wege, um dorthin zu kommen: Erstellung für die Entwicklung, für die Verteilung und für die Ad-hoc-Verteilung. Die einzelnen Verfahrensweisen erlauben Ihnen, die Anwendung lokal auf Ihrem Gerät zu testen, für den App Store zu erstellen oder testweise für bis zu 100 registrierte Geräte zu erstellen und dort zu prüfen. In Kapitel 1 haben Sie Nutzungsprofile kennengelernt und erfahren, wie Sie sie innerhalb des Portals für das Apple iPhone-Entwicklerprogramm erstellen. Jetzt verwenden Sie diese Profile, um ein Programm auf dem iPhone selbst bereitzustellen.

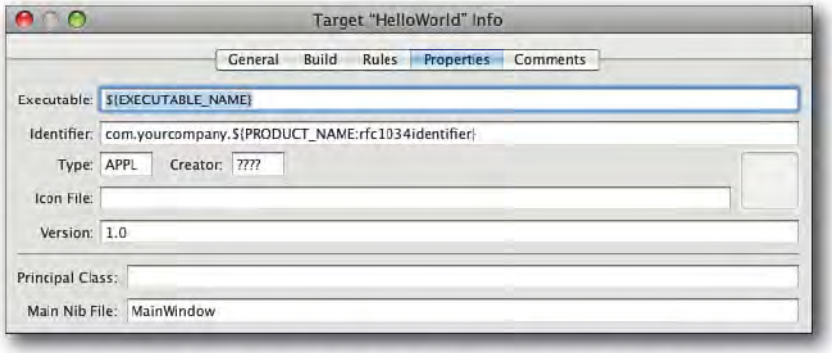
2.10.1 Ein Entwicklungsprofil installieren

Die Mindestvoraussetzung für die iPhone-Entwicklung ist ein Entwicklungsprofil. Bevor Sie fortfahren, müssen Sie also ein Platzhalter-Entwicklungsprofil erstellt und in Xcode installiert haben. Um Letzteres zu tun, ziehen Sie die `mobileprovision`-Datei auf das Anwendungsymbol von Xcode. (Alternativ können Sie das Profil auch über iTunes loslassen.) Schließen Sie danach Xcode, und starten Sie das Programm neu, um sicherzustellen, dass das Profil geladen wird und bereitsteht.

Überprüfen Sie auch Ihren Schlüsselbund, um sicherzustellen, dass das WWDR-(Worldwide Developer Relations-) und Ihr Entwicklerzertifikat für die Verwendung bereit sind. Bei der Kompilierung vergleicht Xcode das Nutzungsprofil mit der Identität im Schlüsselbund. Wenn sie nicht übereinstimmen, kann Xcode die Kompilierung nicht abschließen und Ihre Anwendung signieren. Um die Zertifikate zu prüfen, öffnen Sie die Schlüsselbundverwaltung (in `/Programme/Dienstprogramme`) und geben in das Suchfeld oben rechts »developer« ein. Es sollten mindestens ein Zertifikat der Zertifizierungsinstanz *Apple Worldwide Developer Relations* und ein Zertifikat mit dem Namen *iPhone Developer*, gefolgt von Ihrem (Firmen-)Namen vorhanden sein.

2.10.2 Anwendungsbezeichner bearbeiten

Den Anwendungsbezeichner für das Projekt können Sie im Fenster **Target Info** unter dem Titel **Properties** setzen. Öffnen Sie dazu in der linken Spalte des Projektfensters das Einblenddreieck neben **Targets**, und wählen Sie das darin befindliche Element aus (der Name sollte mit dem des Projekts übereinstimmen). Klicken Sie auf die große, blaue **Info**-Schaltfläche oben im Projektfenster. Dadurch wird das Fenster **Target Info** mit seinen fünf Titeln geöffnet. Klicken Sie auf **Properties**, den vierten Titel von links (siehe *Abbildung 2.15*).



▶ *Abbildung 2.15: Der Titel* **Properties** *zeigt die derzeitige Einstellung für den Anwendungsbezeichner.*

Ihr Platzhalter-Entwicklungsprofil muss mit dem tatsächlich verwendeten Anwendungsbezeichner übereinstimmen. Wenn Sie also einen Platzhalter-Anwendungsbezeichner wie `com.sadun.*` registriert und zum Erstellen Ihres Nutzungsprofils verwendet haben, muss der Anwendungsbezeichner Ihres Projekts mit diesem registrierten Bezeichner übereinstimmen. Beispielsweise können Sie `com.sadun.helloworld` oder `com.sadun.testing` verwenden, nicht aber `helloworld` oder `com.mycompany.helloworld`.

Standardmäßig setzt Xcode den Anwendungsbezeichner auf `com.yourcompany.produktname`, wobei als Produktname automatisch der Name eingefügt wird, den Sie beim Erstellen des Projekts angegeben haben. Ändern Sie `com.yourcompany` in den Namen, den Sie in Ihrem Platzhalter-Bezeichner verwendet haben, ohne die Xcode-Variable zu beschädigen, die mit dem Dollarzeichen beginnt.

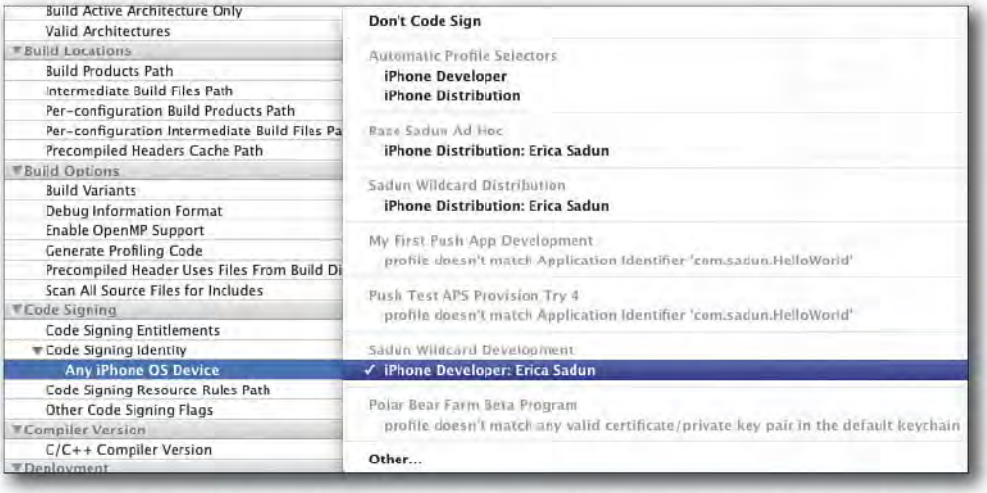
Hinweis

Den Standardfirmennamen können Sie ändern, indem Sie die Templates unter /Developer/Platforms/iPhoneOS.platform/Developer/Library/Xcode/Project Templates/Application bearbeiten oder, was noch besser ist, kopieren und in eigene Templates umarbeiten. Wie Sie dies tun, erfahren Sie weiter hinten in diesem Kapitel.

2.10.3 Identität für die Codesignatur festlegen

Nachdem Sie den Bezeichner festgelegt haben, klicken Sie auf den Titel **Build**, um sich zu vergewissern, dass in der Dropdown-Liste **Configuration** oben links auf dem Bildschirm der Konfigurationstyp ausgewählt ist, den Sie ändern möchten (**Debug** oder **Release**). Blättern Sie bis zum Eintrag **Code Signing Identity** nach unten. Klicken Sie auf das Einblenddreieck, um **Any iPhone OS Device** anzuzeigen, und klicken Sie auf das Einblendmenü rechts daneben. Hier wählen Sie die Identität des Nutzungsprofils aus, mit dem Sie Ihre Anwendungen signieren.

Wenn Sie mehr und mehr Nutzungsprofile und Identitäten erwerben, kann die Liste der Optionen ziemlich lang werden. Das Beispiel in *Abbildung 2.16* wurde aus didaktischen Gründen gekürzt. Normalerweise ist es dreimal so lang, vor allem aufgrund von Ad-hoc-Profilen von Drittanbietern wie *Polar Bear Farm Beta Program One*.



▶ *Abbildung 2.16: Wählen Sie ein Nutzungsprofil als Identität für die Codesignierung aus. Damit ein Profil verwendet werden kann, muss es mit dem Anwendungsbezeichner übereinstimmen.*

Wie Sie sehen, erscheinen einige der Einträge in Schwarz, andere in Grau. Grau angezeigte Profile stimmen nicht mit dem Anwendungsbezeichner des Projekts überein und können nicht zum Signieren verwendet werden. In diesem Beispiel gehören dazu einige Profile für Push-Benachrichtigungen, die an besondere Anwendungsbezeichner gebunden sind und nicht mit dem derzeitigen Bezeichner `com.sadun.HelloWorld` übereinstimmen.

Bei den schwarzen Einträgen handelt es sich um meine drei passenden Nutzungsprofile: mein normales Ad-hoc-Profil, mein Platzhalter-Verteilungsprofil und mein Platzhalter-Entwicklungsprofil, das in dem Bild ausgewählt ist. Jedes dieser drei Profile ist mit einer Zertifikatidentität aufgeführt, nämlich `iPhone Developer` bzw. `iPhone Distribution`, jeweils gefolgt von einem Doppelpunkt und meinem Namen. Beide entsprechen den im Schlüsselbund gespeicherten Identitäten und den im Portal zum Erstellen der Nutzungsprofile verwendeten Zertifikaten.

Die beiden Einträge unter **Automatic Profile Selector** wählen automatisch jeweils das erste passende Profil aus. Das funktioniert sehr gut für die Entwickleridentität, da ich nur eine habe, aber schlecht für die Verteileridentität, da hierbei zuerst mein Ad-hoc-Profil ausgewählt wird, das ich nur selten verwende. Bei der täglichen Arbeit sollten Sie die automatische Profilauswahl nicht nutzen, sondern denjenigen Eintrag auswählen, den Sie auch wirklich verwenden möchten. Schauen Sie sich sowohl den Zertifikatnamen als auch die Profilidentität an, die darüber steht, bevor Sie ein Profil auswählen.

2.10.4 Die Anwendung »Hello World« kompilieren und ausführen

Nun ist es an der Zeit, Hello World auf einem physischen iPhone oder iPod touch zu testen. Schließen Sie ein Gerät an, das Sie für die Entwicklung verwenden möchten. Wenn Sie dies zum ersten Mal tun, fordert Xcode Sie auf, zu bestätigen, dass Sie dieses Gerät zur Entwicklung verwenden möchten. Fahren Sie fort, und geben Sie die Bestätigung; Apple warnt stets vor möglichen schweren Folgen. Anfänger befürchten manchmal, dass ihr Gerät in einen »Entwicklungsmodus« versetzt wird, aber in der Praxis habe ich noch nie von langfristigen Problemen gehört. Machen Sie aber trotzdem Ihre Hausaufgaben, bevor Sie Ihr Gerät zur Entwicklungseinheit bestimmen, und lesen Sie die jüngsten Release Notes zum SDK, um Genaueres zu erfahren.

Vor der Kompilierung müssen Sie Xcode anweisen, die Anwendung für die ARM-Architektur des iPhones zu erstellen statt für die Intel-Architektur des Macintosh. Wählen Sie im Projektfenster **iPhone Device** unter **Active SDK** aus (siehe *Abbildung 2.17*). Markieren Sie dann die Einstellung **Active Executable**. Wenn Sie an den Macintosh mehr als ein Entwicklungsgerät angeschlossen haben, wählen Sie das aus, auf dem Sie den Test durchführen möchten. Neben dem Namen des verwendeten Geräts erscheint ein Häkchen.

Klicken Sie im Projektfenster auf die Schaltfläche **Build and Go**. Wenn Sie die zuvor in diesem Kapitel gegebenen Anleitungen befolgt haben, wird das Projekt »Hello World« fehlerfrei kompiliert, auf das iPhone kopiert und gestartet.

▶ *Abbildung 2.17: Bei der Auswahl unter* **Active Executable** *geben Sie an, welches Gerät verwendet werden soll. In diesem Beispiel sind an den Mac zwei Entwicklungseinheiten angeschlossen, wobei das Gerät mit dem Namen »Bologna« ausgewählt ist.*

Wenn Sie die Warnung erhalten, dass kein Gerät angeschlossen ist, öffnen Sie das Organizer-Fenster in Xcode und überprüfen, ob der Punkt neben dem Gerätenamen grün ist. Ist das nicht der Fall, müssen Sie das Gerät oder den Computer neu starten.

2.10.5 Kompilierte Anwendungen signieren

Sie können bereits kompilierte Anwendungen mithilfe eines einfachen Shellscripts auf der Kommandozeile signieren. Dies funktioniert bei Anwendungen, die für die Entwicklung erstellt wurden. Mithilfe der direkten Signierung von Anwendungen können Entwickler Anwendungen außerhalb der Ad-hoc-Verbreitungskanäle untereinander austauschen.

```
#! /bin/bash

export CODESIGN_ALLOCATE=/Developer/Platforms/iPhoneOS.platform/Developer/
usr/bin/codesign_allocate

codesign -f -s "iPhone Developer" $1.app
```

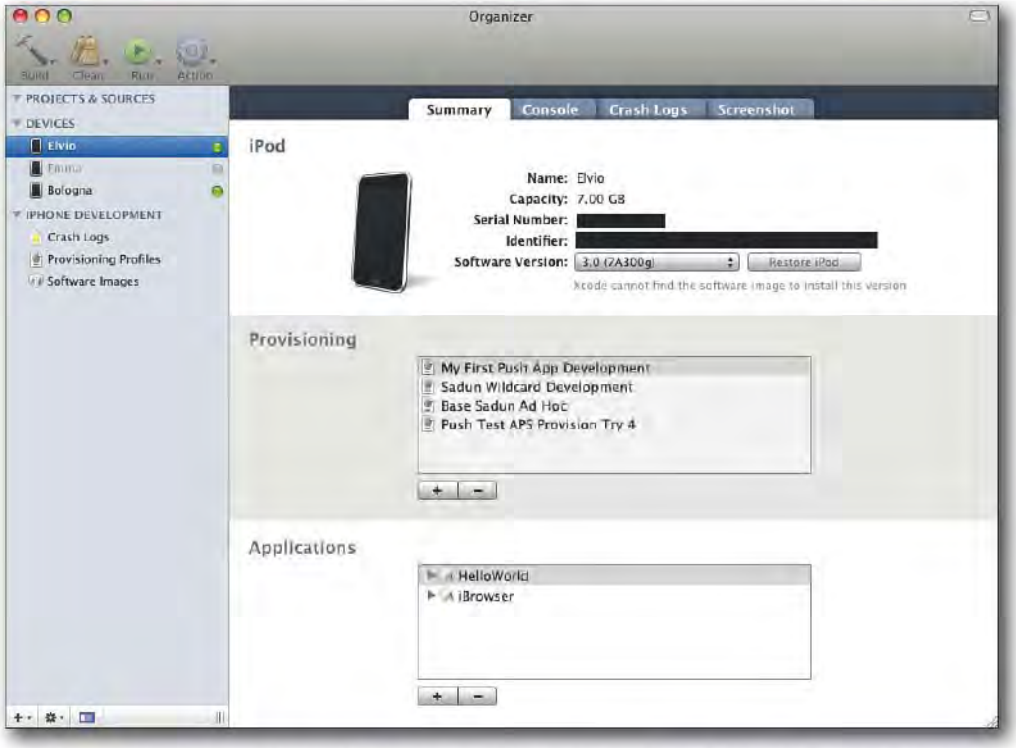
Wenn Sie in Ihrem Schlüsselbund mehrere `iPhone Developer`-Profile haben, müssen Sie dieses Script unter Umständen anpassen, sodass es nur zu einem davon passt. Anderenfalls beschwert sich `codesign` über eine nicht eindeutige Zuordnung.

Ich persönlich verwende diesen Ansatz, um die Testversionen des Beispielcodes zu diesem Buch zu verbreiten. Durch die Entwickler-Codesignierung kann ich die Mühen der Ad-hoc-Verteilung umgehen und die Anwendungen schnell einem beliebigen Publikum zukommen lassen.

2.11 Von Xcode auf das iPhone: Die Organizer-Oberfläche

Ihre Entwicklungseinheiten verwalten Sie mit dem Werkzeug *Organizer* von Xcode (**Window ▸ Organizer**; Ctrl + ⌘ + O). Dieses Fenster (siehe *Abbildung 2.18*) ist das Steuerungszentrum für die Verbindung zwischen Ihrem Entwicklungscomputer und Ihrer iPhone- bzw. iPod-Testumgebung. Hier

können Sie Anwendungen hinzufügen und entfernen, Konsolenergebnisse während der Tests einsehen, Crashlogs untersuchen und Screenshots Ihrer Geräte während der Tests erstellen. Im Folgenden finden Sie eine kurze Zusammenfassung der in der Organizer-Konsole verfügbaren Hauptfunktionen.



▶ *Abbildung 2.18: Das Xcode-Fenster für den iPhone-Organizer (**Window ▸ Organizer**) bietet eine zentrale Steuerungsstelle für die meisten Bedürfnisse beim Testen von Anwendungen. Hier können Sie Firmware laden, Anwendungen installieren und entfernen, Crashlogs lesen, Screenshots erstellen und vieles mehr.*

2.11.1 Die Projekt- und die Quellliste

Sorgen Sie für eine leichte Erreichbarkeit Ihrer aktuellen Projekte, indem Sie sie in die **Projects & Sources**-Liste des Organizers ziehen. Doppelklicken Sie dann auf den Namen eines Projekts, um es zu öffnen. Sie können sowohl einzelne Quelldateien als auch vollständige Projekte hinzufügen. Verwenden Sie die Optionen **Build**, **Clean**, **Run** und **Action** am oberen Rand des Fensters, um noch mehr Entwicklungsaufgaben direkt aus dem Organizer heraus auszuführen.

Die Projekt- und Quellliste dient nicht nur zum Speichern von Dateien, sondern auch zur Anzeige des Inhalts von Sandboxes. Wenn Sie über den Titel **Summary** Sandbox-Daten von einem Gerät herunterladen, fügt Xcode der Liste automatisch einen Ordner hinzu, in dem Sie den Dateiinhalt durchsuchen können.

Um Einträge von der Liste zu entfernen, vor allem solche, die Sie nicht aufnehmen wollten, sondern die automatisch hinzugefügt wurden, öffnen Sie das Kontextmenü. Klicken Sie mit der rechten Maustaste oder bei gedrückter Ctrl-Taste auf einen Eintrag, und wählen Sie **Remove from Organizer**. Klicken Sie dann auf **OK**. Dies hat keinen Einfluss auf die Dateien auf der Festplatte. Sie löschen hierdurch keine Dateien, sondern unterdrücken nur ihre Erwähnung in der Projekt- und Quellliste.

2.11.2 Die Geräteliste

Die Liste **Devices** zeigt den Namen und den Status derjenigen Geräte an, die Sie als Entwicklungsplattformen autorisiert haben. Die Indikatoren rechts neben den Namen geben an, ob das Gerät verbunden ist (grün) oder nicht. Ein grauer Indikator zeigt an, dass eine Einheit noch nicht für die Entwicklung eingerichtet wurde bzw. dass kein gültiges Entwicklerzertifikat gefunden wurde und das Gerät daher ignoriert, also von der Liste der aktiven Geräte entfernt wird. Eine bernsteinfarbene Markierung sehen Sie, wenn ein Gerät gerade angeschlossen wurde. Ändert sich die Farbe dieses Indikators nicht, liegt ein Verbindungsproblem vor. Dies kann daran liegen, dass iTunes eine Synchronisierung durchführt, dass die Einheit noch nicht verfügbar ist oder dass ein Problem mit der Verbindung zu einem Onboard-Dienst besteht. Letzteres können Sie gewöhnlich durch einen Neustart des iPhones beheben.

2.11.3 iPhone-Entwicklungswerkzeuge

Die Einträge in der Liste **iPhone Development** zeigen die Entwicklungsressourcen auf dem Mac an. Dazu gehören archivierte Crashlogs (die also nicht an ein bestimmtes Gerät gebunden sind, sondern auf Ihr System ausgelagert wurden), ein Manager für Nutzungsprofile und eine Liste der Software-Images, die die zurzeit auf Ihrem System verfügbaren Firmware-Bundles aufführt. Vor allem der Profilmanager ist nützlich, da er zeigt, auf welchen Geräten jeweils welches Profil installiert ist. Außerdem gibt er einen Profilbezeichner an (sodass Sie herausfinden können, welche Datei in `~/Library/MobileDevice/Provisioning Profiles` zu welchem Profil gehört) und erlaubt praktische Überprüfungen der Ablaufdaten.

2.11.4 Der Titel »Summary«

Der Titel **Summary** gibt den Namen, die Kapazität, die Seriennummer und den Bezeichner Ihres iPhones oder iPod touch an. Hier können Sie Ihr Gerät bereitstellen (d. h. zur Zusammenarbeit mit den Projekten autorisieren, die Sie in Xcode erstellen), Anwendungen hinzufügen und entfernen sowie die neueste Firmware laden.

Mit jeder Entwicklerlizenz können Sie bis zu fünf iPhones bzw. iPod Touch-Geräte gleichzeitig zum Testen verwenden. Die Liste **Provisioning** führt alle Nutzungsprofile auf, die für Ihre Geräte verfügbar sind. Das Profil bestimmt, welche Anwendungen auf dem Gerät laufen dürfen und welche nicht. Im Regelfall werden hier nur Entwicklungs- und Ad-hoc-Profile aufgeführt, was sinnvoll ist, denn Verteilungsprofile werden zum Signieren von Anwendungen für den App Store verwendet und nicht für bestimmte Geräte.

Am unteren Rand des Titels **Summary** erscheint eine Liste der installierten Anwendungen. Verwenden Sie die Schaltfläche **-**, um Anwendungen zu entfernen. Wollen Sie eine Anwendung installieren, ziehen Sie sie auf die Liste oder klicken auf die Schaltfläche **+**, um nach ihr zu suchen. Stellen Sie sicher, dass die Anwendung für das iPhone-Betriebssystem kompiliert ist und dass das Gerät über ein Profil für diese Anwendung verfügt. Die Anwendung wird unmittelbar überspielt. Aus dem App Store installierte Anwendungen sind in der Anwendungsliste grau dargestellt.

Öffnen Sie das Einblenddreieck, das neben dem Namen der Anwendung steht, um die Daten anzuzeigen, die mit dieser Anwendung verbunden sind. Um die Anwendungsdaten herunterzuladen, klicken Sie auf den nach unten weisenden Pfeil und klicken auf **Save**. Xcode erstellt einen Ordner mit Datumsangabe und füllt ihn mit den Inhalten der Sandbox, also den Ordnern `Documents`, `Library` und `tmp`. Außerdem fügt Xcode die Ordner der Projekt- und Quellliste hinzu, sodass Sie den Inhalt direkt im Organizer durchsuchen können.

Sie können diesen Vorgang auch umkehren und die bearbeitete Sandbox auf das Gerät zurückspielen. Suchen Sie den von Ihnen erstellten Ordner (verwenden Sie den Befehl **Reveal in Finder** aus dem Kontextmenü der Projekt- und Quellliste), bringen Sie neue Elemente in die Unterordner ein, und ziehen Sie anschließend den gesamten Ordner zurück auf den Anwendungsnamen am unteren Rand des **Summary**-Bereichs. Xcode liest die neuen Elemente und überträgt sie sofort auf das Gerät. Dies ist eine hervorragende Möglichkeit, um den Ordner `Documents` vorab mit Material für Tests zu füllen.

2.11.5 Der Titel »Console«

Verwenden Sie die Konsole, um Systemnachrichten von den verbundenen Geräten zu sehen. Dieser Bildschirm zeigt Aufrufe von `NSLog()`, während Sie Software auf dem angeschlossenen iPhone ausführen. Dazu müssen Sie nicht den Debugger von Xcode verwenden. Die Konsole hört die zurzeit auf dem Gerät laufende Anwendung ab.

Neben den Debugmeldungen, die Sie zu Ihren iPhone-Anwendungen hinzufügen, sehen Sie auch Systembenachrichtigungen, Geräteinformationen und Debugaufrufe der Systemsoftware von Apple. Es ist im Wesentlichen ein Durcheinander in Textform. Protokollierte Daten erscheinen auch auf der Debugkonsole von Xcode (**Run ▸ Console**) zusammen mit allen Ausgaben von `printf()`. Klicken Sie auf **Save Log As**, um den Inhalt der Konsole auf die Festplatte zu schreiben.

2.11.6 Der Titel »Crash Log«

Sie erhalten direkten Zugriff auf Ihre Crashlogs, indem Sie in der Scrollliste einen einzelnen Crash auswählen (bezeichnet mit dem Namen der iPhone-Anwendung sowie dem Datum und der Uhrzeit des Absturzes). Im unteren Bereich erscheinen die Einzelheiten zu diesem Crash, darunter eine Stack-Ablaufverfolgung, Informationen über Threads, Ausnahmetypen usw.

Neben den von Ihnen selbst erstellen Crashlogs können Sie auch Absturzberichte von den Computern der Benutzer und von iTunes Connect abrufen. Das iPhone überträgt automatisch Absturzberichte auf Computer, wenn es eine Sicherung (Synchronisierung) auf iTunes vornimmt. Je nach

der Plattform, die zur Synchronisierung des Geräts verwendet wird, befinden sich diese Berichte an unterschiedlichen Speicherorten:

- **Mac OS X** `~/Library/Logs/CrashReporter/MobileDevice/`*`Gerätename`*
- **Windows XP** `C:\Dokumente und Einstellungen\`*`Benutzername`*`\Anwendungsdaten\Apple Computer\Logs\CrashReporter\MobileDevice\`*`Gerätename`*
- **Windows Vista** `C:\Benutzer\`*`Benutzername`*`\AppData\Roaming\Apple Computer\Logs\CrashReporter\MobileDevice\`*`Gerätename`*

iTunes Connect sammelt Daten aus Absturzprotokollen von Ihren App Store-Benutzern und stellt Sie Ihnen zur Verfügung. Um diese Berichte herunterzuladen, wählen Sie für eine Anwendung **Manage Your Applications ▸ App Details ▸ View Crash Reports**. Dadurch erhalten Sie eine Liste der häufigsten Absturztypen und für jeden Typ eine Schaltfläche zum Herunterladen des Berichts.

Wenn Sie die Berichte in den Ordner `CrashReporter` von Mac OS X kopieren, werden sie unmittelbar in den Organizer geladen. Achten Sie darauf, sie in den Geräteordner für das zurzeit ausgewählte Gerät zu laden. Die Berichte erscheinen unter **iPhone Development ▸ Crash Logs**.

Sobald die Berichte im Organizer sind, ersetzt Xcode mithilfe der Binärdatei der Anwendung und der `.dSYM`-Datei die normalerweise im Bericht angegebenen Hexadezimaladressen durch Funktions- und Methodennamen. Dieser Vorgang wird *Symbolifizierung* genannt. Diese Elemente müssen Sie nicht manuell suchen. Xcode findet die ursprüngliche Binärdatei und die `.dSYM`-Datei über Spotlight und den eindeutigen Bezeichner (UUID) der Anwendung, vorausgesetzt, dass sich die beiden Dateien irgendwo in Ihrem Benutzerordner befinden.

Wie die Crashlogs im Organizer enthalten auch die Berichte der Benutzer eine Stack-Ablaufverfolgung, die Sie in Xcode laden können, um herauszufinden, wo der Fehler aufgetreten ist. Die Ablaufverfolgung erscheint immer in umgekehrter chronologischer Reihenfolge: Die ersten Einträge in der Liste sind also diejenigen, die zuletzt ausgeführt wurden.

Crashberichte zeigen Ihnen nicht nur, wo die Anwendung abgestürzt ist, sondern auch warum. Der häufigste Grund ist `EXC_BAD_ACCESS`, der durch den Zugriff auf nicht zugeordneten Speicher (`KERN_INVALID_ADDRESS`) oder den Versuch hervorgerufen werden kann, in schreibgeschützten Speicher zu schreiben (`KERN_PROTECTION_FAILURE`).

Andere wichtige Elemente im Absturzbericht sind die Betriebssystemversion des betroffenen Geräts und die Version der abgestürzten Anwendung. Benutzer aktualisieren ihre Software nicht immer auf das jüngste Release, sodass es sehr wichtig ist, zu erkennen, welche Abstürze bei älteren, inzwischen möglicherweise korrigierten Versionen auftraten.

Hinweis

Weitere Einzelheiten über die Fehlerberichterstattung im iPhone-Betriebssystem finden Sie in der Apple Technical Note TN2151.

2.11.7 Der Titel »Screenshot«

Kopieren Sie den Bildschirm Ihres angeschlossenen iPhone, indem Sie auf die Schaltfläche **Capture** auf dem Titel **Screenshot** klicken. Diese Funktion nimmt ein Bild davon auf, was auf dem iPhone ausgeführt wird, wenn Ihre Anwendungen geöffnet sind. Auf diese Weise haben Sie Zugriff auf Aufnahmen der eingebauten Software von Apple und aller anderen Anwendungen auf dem iPhone.

Sobald sie aufgenommen sind, können Sie die Bilder auf den Desktop ziehen oder als neues *Default .png* eines offenen Projekts speichern. Archivbilder erscheinen in der Bibliothek auf der linken Seite des Fensters. Wählen Sie eins aus und betätigen Sie die Taste Entf, um es dauerhaft zu löschen.

Hinweis

Screenshots werden im Ordner `~/Library/Application support/Developer/Shared/Xcode/Screenshots` Ihres Benutzerordners gespeichert.

2.12 Compiler-Direktiven verwenden

Xcode-Direktiven geben dem Compiler Anweisungen, um die Plattform und die Firmware zu erkennen, für die Sie Ihre Anwendung erstellen. Dadurch können Sie die Anwendung so anpassen, dass sie gefahrlos Funktionen nutzt, die nur bei bestimmten Plattformen oder Firmware-Versionen zur Verfügung stehen. Wenn Sie dem Code `#if`-Anweisungen hinzufügen, können Sie einzelne Funktionen aufgrund dieser Möglichkeiten blockieren oder freischalten. Um zu bestimmen, ob der Code für den Simulator oder für das iPhone kompiliert ist, verwenden Sie die Zieldefinitionen `TARGET_IPHONE_SIMULATOR` und `TARGET_OS_IPHONE`.

```
#if TARGET_IPHONE_SIMULATOR
   Simulatorspezifischer Code
#else
   iPhone-spezifischer Code
#endif
```

Mit der einfachen Versionsüberprüfung »OS 3 oder höher« können Sie betriebsspezifische Blöcke abteilen. In solche Blöcke können Sie beispielsweise Code aufnehmen, der nur für das MapKit 3.0 gilt, sodass das Programm auch auf Geräten mit Version 2.2.x noch kompiliert und ausgeführt wird. Mit diesem Ansatz erstellen Sie versionsspezifische Builds. Das Programm passt sich wie bei einer Plattformdirektive nicht an geänderte Gerätebedingungen an, da diese Prüfung nur zur Kompilierungszeit durchgeführt wird.

```
#ifdef _USE_OS_3_OR_LATER
   #import <MapKit/MapKit.h>
#endif
```

Eine weitere Möglichkeit besteht darin, zu prüfen, ob die mindestens erforderliche Betriebssystemversion für die Anwendung vorhanden ist. Dazu können Sie jede der OS-Voreinstellungen verwenden. Die folgende Direktive stellt sicher, dass für die Version 3.1 kompilierte Anwendungen auch nur auf Code der Version 3.1 ausgeführt werden:

```
#if __IPHONE_OS_VERSION_MIN_REQUIRED < 30100
   Code für Versionen vor 3.1
#else
   Code für Version 3.1
#endif
```

Die Werte für die Betriebssystemversionen folgen dem im Anschluss gezeigten grundlegenden Benennungsmuster, das wahrscheinlich mit Version 3.2 fortgeführt wird. Diese Definitionen wurden dem globalen Satz von iPhone-Definitionen entnommen. Im nächsten Abschnitt erfahren Sie, wie Sie solche Daten selbst abrufen.

```
#define __IPHONE_2_0 20000
#define __IPHONE_2_1 20100
#define __IPHONE_2_2 20200
#define __IPHONE_3_0 30000
#define __IPHONE_3_1 30100
```

2.12.1 iPhone-spezifische Definitionen abrufen

Die Definitionen für Direktiven sind zwar nicht geheim, aber auch nicht gerade allgemein bekannt. Um die aktuelle Liste der iPhone-spezifischen Definitionen einzusehen, gehen Sie nach der folgenden Anweisung vor. Damit geben Sie während der Kompilierung eine Liste aus Xcode aus, die Sie zum Nachschlagen verwenden können.

1. Öffnen Sie das **Target Info**-Fenster für das iPhone-Projekt »Hello World« vom Anfang dieses Kapitels.
2. Fügen Sie unter dem Titel **Build** die folgenden Schalter zu `OTHER_CFLAGS` hinzu: `-g3 -save-temps -dD`. Erstellen Sie gegebenenfalls `OTHER_CFLAGS`, oder stellen Sie `Base SDK` auf ein `Simulator SDK` um, Sie finden dann `OTHER_CFLAGS` unterhalb der `GCC 4.2 - Language`-Einstellungen.
3. Erstellen Sie das Projekt. Bei der Kompilierung treten Fehler auf, die Sie aber ignorieren können.
4. Öffnen Sie eine Terminal-Shell, und wechseln Sie zu Ihrem Projektordner. Darin finden Sie die neue Datei `main.mi`.
5. Geben Sie den Befehl `grep -i iPhone main.mi | open -f`. Damit durchsuchen Sie `main.mi` nach allen iPhone-Verweisen und fügen diese zu einem neuen TextEdit-Dokument hinzu. Diese Liste enthält alle zurzeit definierten Makroelemente. Speichern Sie die Liste an einem gut erreichbaren Ort.
6. Entfernen Sie die benutzerdefinierten Schalter von Ihrem Projekt, und speichern Sie es. Sie können es jetzt erneut erstellen, ohne dass dabei Fehler auftreten.

Hinweis

Um plattformspezifische Bedingungen wie den Zugriff auf integrierte Kameras und Mikrofone sollten Sie sich ebenfalls im Code kümmern. Mehr über die Programmierung bei solchen möglichen Hindernissen lesen Sie in Kapitel 14, »*Gerätefähigkeiten*«.

2.12.2 Überprüfungen zur Laufzeit

Mit Compiler-Direktiven können Sie 2.x- und 3.x-spezifische Versionen Ihrer Anwendungen erstellen. Es ist damit aber nicht möglich, Code auszuführen, der sich jeweils an die aktuelle Firmware anpasst.

Um Ihre Anwendungen an möglichst viele Kunden verkaufen zu können, müssen Sie sie für ein SDK mit einer Versionsnummer erstellen, die nicht höher als die der niedrigsten Versionsnummer des gewünschten Kundenkreises ist. An Anwender mit älteren Firmware-Versionen auf ihren Geräten können Sie dann immer noch Software verkaufen. Allerdings müssen Sie sorgfältig prüfen, ob die Anwendung auch auf neuerer Firmware läuft. Verkaufsstatistiken und Crashlog-Analysen verschiedener Entwickler zeigen jedoch, dass Firmware-Versionen vor 3.0 nur noch äußerst selten anzutreffen sind. Dies liegt sicherlich einerseits daran, dass mittlerweile Upgrades für den iPod kostenlos angeboten werden, aber auch daran, dass bei der Installation von Xcode 3.2 für Snow Leopard ältere SDKs der Versionen 2.x deinstalliert werden und somit dem Entwickler nicht mehr zur Verfügung stehen.

Wenn Sie moderne Klassen und Aufrufe einsetzen möchten, müssen Sie entweder die Benutzer älterer Firmware ganz ausschließen oder eine Anwendung entwickeln, die zwar diese neuen Möglichkeiten bietet, aber für ältere Firmware kompiliert wird. Das bedeutet, die Kompatibilität zur Laufzeit statt zur Kompilierungszeit zu prüfen.

Dies können Sie auf verschiedene Weise erreichen. Erstens können Sie das System untersuchen, das auf dem Gerät läuft, und die für die Firmware geeigneten Methoden aufrufen. Das folgende Beispiel zeigt genau diesen Ansatz. Der Code ruft bei der Kompilierung Warnungen für einen 2.x-Build hervor, damit Sie wissen, dass Tabellenzellen möglicherweise nicht auf `textLabel` reagieren. Dies ist jedoch nicht die bevorzugte Vorgehensweise. Apple empfiehlt, den Funktionsumfang und die Verfügbarkeit zu prüfen und nicht die Firmware-Version.

```
NSString *celltext = [[UIFont familyNames] objectAtIndex:
   [indexPath row]];
if ([[[UIDevice currentDevice] systemVersion] hasPrefix:@"2."])
   [cell setText:celltext];
else if ([[[UIDevice currentDevice] systemVersion] hasPrefix:@"3."])
   [[cell textLabel] setText:celltext];
return cell;
```

Sie können auch Objekte testen, um herauszufinden, ob sie auf bestimmte Selektoren reagieren. Wenn das Framework ab Version 3.0 zur Verfügung steht, melden die Objekte, dass sie auf diese Selektoren reagieren, sodass Sie sie aufrufen können, ohne dass das Programm abstürzt. Wie beim vorhergehenden Ansatz treten auch hier bei der Kompilierung Warnungen über nicht implementierte Selektoren auf.

```
NSString *celltext = [[UIFont familyNames] objectAtIndex:
   [indexPath row]];
if (![cell respondsToSelector:@selector(textLabel)])
   [cell setText:celltext];
else
   [[cell textLabel] setText:celltext];
return cell;
```

Um diese Warnungen zur Kompilierungszeit zu vermeiden, können Sie dem 2.x-Quellcode 3.x-spezifische Schnittstellendeklarationen hinzufügen:

```
@interface UITableViewCell (SDK3)
- (UILabel *) textLabel;
@end
```

Ein besserer Ansatz ist es jedoch, die Ziele »Base SDK« und »Deployment« für das Projekt festzulegen. Setzen Sie **Base SDK** unter **Target Info ▸ Build Settings** auf die höchste Betriebssystemversion, die Sie erreichen möchten, also eine 3.x-Version, und das Ziel **iPhone OS Deployment** auf die niedrigste Version, für die Sie die Anwendung erstellen wollen.

Sie können auch verschiedene andere Hilfskonstruktionen nutzen, z. B. das Label indirekt herausziehen. Der folgende Code ruft das Label ab und legt dessen Text fest:

```
UILabel *label = (UILabel *)[cell valueForKey:@"textLabel"];
if (label) [label setText:celltext];
```

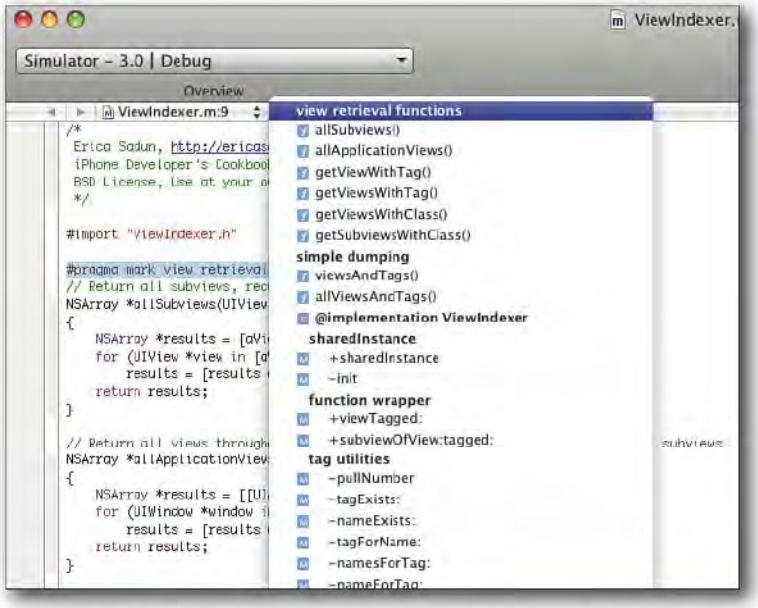
Klassen der Version 3.x können Sie von einem 2.x-Build aus mithilfe von `NSClassFromString()` verwenden. Prüfen Sie, ob die Klasse `nil` zurückgibt. Ist dies nicht der Fall, steht die Klasse für die Verwendung in der aktuellen Firmware zur Verfügung. Verlinken Sie die Anwendung mit einem Framework Ihrer Wahl, unabhängig davon, ob es für den 2.x-Build verfügbar ist oder nicht.

```
Class MFMCVC = NSClassFromString(@"MFMailComposeViewController");
If (MFMVC) myMFMCViewController = [[MFMCVC alloc] init];
```

Und wenn Sie es wirklich auf die harte Tour probieren möchten, können Sie direkt `NSInvocation`-Instanzen erstellen.

2.12.3 Pragma-Markierungen

Pragma-Markierungen gliedern Ihren Quellcode, indem Sie Lesezeichen in das Einblendmenü der Methodenliste am oberen Rand aller Xcode-Fenster einfügen. Diese Listen zeigen alle Methoden und Funktionen, die im aktuellen Dokument verfügbar sind. Mit Pragma-Markierungen können Sie verwandte Elemente gruppieren, wie Sie in *Abbildung 2.19* sehen. Wenn Sie in der Dropdown-Liste auf eine dieser Markierungen klicken, können Sie zu einem Abschnitt der Datei springen (z. B. zu `tag utilities`), aber auch zu einer bestimmten Methode (z. B. `-tagExists:`).



▶ *Abbildung 2.19: Methoden- und Funktionslisten mit Pragma-Markierungen ordnen*

Um ein neues Lesezeichen zu erstellen, fügen Sie in den Code eine einfache Definition für eine Pragma-Markierung ein. Für die erste Gruppe in *Abbildung 2.19* lautet sie:

```
#pragma mark view retrieval functions
```

Mit einem besonderen Pragma-Aufruf können Sie auch eine Trennlinie einfügen. Geben Sie hinter dem Bindestrich keinen Text ein, da Xcode sonst ein normales Lesezeichen erstellt.

```
#pragma mark -
```

Die Markierungen üben keine Funktion aus und haben keine Auswirkungen auf den Code, sondern sind einfach Gliederungsmöglichkeiten, die Sie nach Belieben einsetzen können.

2.12.4 Methoden ausblenden

Wenn Sie einen größeren Überblick über Ihren Code brauchen, können Sie in Xcode Methodengruppen schließen und öffnen. Platzieren Sie den Mauszeiger direkt in der Leiste links neben einer Methode. Dabei erscheint ein Paar von Einblenddreiecken. Wenn Sie auf eines dieser Dreiecke klicken, blendet Xcode den Code dieser Methode aus, wie Sie in *Abbildung 2.20* sehen. Die drei Auslassungspunkte stehen für die ausgeblendete Methode. Klicken Sie erneut auf das Einblenddreieck, damit Xcode den ausgeblendeten Code wieder anzeigt.

```
@implementation TestViewController
- (void) performAction: (id) sender
{...}

- (void) viewDidLoad
{
    [[self.view viewWithTag:101] registerName:@"my label"];
    [[self.view viewWithTag:102] registerName:@"my switch"];
}
@end

@interface TestBedAppDelegate : NSObject <UIApplicationDelegate>
@end

@implementation TestBedAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    UIWindow *window = [[UIWindow alloc] initWithFrame:[[UIScreen mainScreen] bounds]];
    TestViewController *tvc = [[TestViewController alloc] init];
    [window addSubview:tvc.view];
    [window makeKeyAndVisible];
}
@end

int main(int argc, char *argv[])
```

► *Abbildung 2.20: In Xcode können Sie einzelne Methoden und Funktionen ausblenden, um Teile des Programms zu sehen, die normalerweise nicht zusammen auf den Bildschirm passen.*

2.13 Anwendungen zur Verteilung erstellen

Eine Anwendung zur Verteilung zu erstellen bedeutet, eine Version von ihr anzulegen, die Sie für den Verkauf über den App Store an Apple übermitteln können. Bevor Sie die Erstellung (Build) in Angriff nehmen können, müssen Sie wissen, wie Sie einen Build aufräumen, wie Sie eine Verteilungskonfiguration anlegen und wo Sie das erstellte Produkt finden. Die Kompilierung für den App Store muss mit Genauigkeit erfolgen. Räumen Sie zunächst auf, und kompilieren Sie den Code dann mit einer voreingestellten Verteilungskonfiguration, damit die Anwendung sauber hochgeladen werden kann. Sie müssen wissen, wo sich die erstellte Anwendung befindet, damit Sie die richtige Datei komprimieren und übermitteln können. In den folgenden Abschnitten erfahren Sie, wie dies geht, und lernen noch andere wichtige Vorgänge zur Kompilierung für die Verteilung kennen.

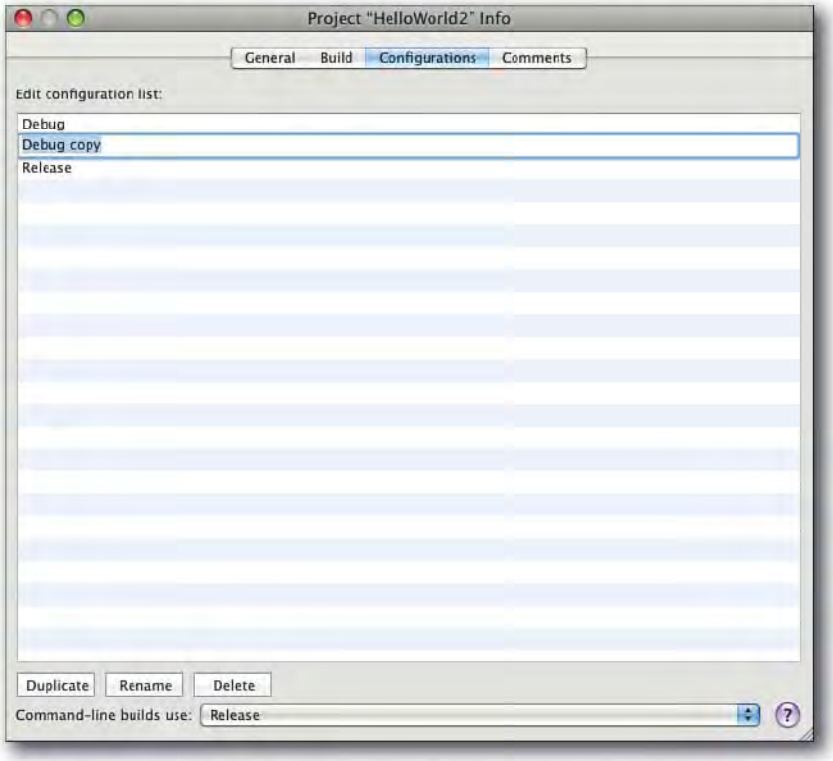
2.13.1 Konfigurationen erstellen und bearbeiten

In Xcode dienen Konfigurationen dazu, Build-Einstellungen festzuhalten. Sie dienen als schnelle Referenz dafür, wie Sie die verschiedenen Dinge einrichten möchten, sodass Sie die Kompilierung für ein Gerät oder für den App Store einfach dadurch vorbereiten können, dass Sie eine Konfiguration auswählen. Standardmäßig gibt es in Xcode-Projekten die Konfigurationen **Debug** und **Release**. Sie können auch eigene erstellen, z. B. für die reguläre und die Ad-hoc-Verteilung.

Wenn Sie den Beispielen in diesem Kapitel bis hierhin gefolgt sind, haben Sie bereits ein »Hello World«-Projekt eingerichtet und seine Build-Einstellungen für das Debugging bearbeitet. Zum Signieren der Anwendung wird ein Platzhalterprofil für die Verteilung verwendet. Anstatt jedes Mal die Build-Einstellungen zu bearbeiten, wenn Sie das Signierprofil ändern möchten, können Sie stattdessen auch eine neue Konfiguration anlegen.

Wählen Sie im Fenster **Project** ganz oben in der Spalte **Groups & Files** das Projekt **HelloWorld,** und klicken Sie auf die blaue **Info**-Schaltfläche, um das **Project Info**-Fenster zu öffnen. Dort finden Sie die vier Titel **General**, **Build**, **Configuratons** und **Comments**. Öffnen Sie den Titel **Configurations**.

Wählen Sie die Konfiguration **Debug** aus, die Sie bereits angepasst haben, und klicken Sie auf die Schaltfläche **Duplicate** unten links im Fenster. Xcode erstellt eine Kopie und öffnet ein Texteingabefeld für den Namen, wie Sie in *Abbildung 2.21* sehen. Ändern Sie den Namen von **Debug** in **Distribution**. In der Praxis werden Sie eher die Konfiguration **Release** bearbeiten oder kopieren statt **Debug**. In diesem Beispiel verwenden wir jedoch **Debug**, da diese Konfiguration bereits angepasst ist.



▸ *Abbildung 2.21: Unter dem Titel* **Configuration** *des Fensters* **Project Info** *können Sie neue Konfigurationen erstellen und damit Voreinstellungen für den Build wie Signieridentitäten festlegen.*

Klicken Sie als Nächstes auf den Titel **Build,** und wählen Sie die neue Option **Distribution** aus dem Einblendmenü **Configuration**. Dieser Schritt ist sehr wichtig, da Sie sonst die Konfiguration ändern, die zuletzt verwendet worden ist. Suchen Sie den Eintrag **Code Signing Identity,** und setzen Sie **Any iPhone OS Device** auf Ihr Platzhalterprofil für die Verteilung. Schließen Sie danach das Fenster **Project Info**.

Mit diesen Schritten haben Sie eine Konfiguration für die Verteilung zu Ihrem Projekt hinzugefügt, die Sie auswählen können, wenn es an die Kompilierung geht. Denken Sie daran, dass Sie für jedes Projekt eine eigene Konfiguration anlegen müssen. Konfigurationen werden nicht von einem Projekt zum nächsten übernommen, sondern im Rahmen der Projekteinstellungen gespeichert.

2.14 Saubere Builds

Bei einem sauberen Build wird jeder Teil des Projekts von Grund auf neu kompiliert. Ein solcher Aufräumvorgang sorgt auch dafür, dass der Projekt-Build die aktuellen Versionen der Projektbestandteile wie Bilder und Klänge enthält. Einen sauberen Build können Sie dadurch erzwingen, dass Sie den Build-Ordner innerhalb des Projektordners löschen. Sie können aber auch ein internes Hilfsprogramm von Xcode nutzen. Wählen Sie **Build ▸ Clean** ⌘ + ⇧ + K. Wie *Abbildung 2.22* zeigt, fragt Xcode Sie, ob Sie auch Abhängigkeiten und vorkompilierte Header aufräumen möchten. Im Allgemeinen schadet es nichts, dem zuzustimmen. Klicken Sie auf **Clean**, und warten Sie darauf, dass Xcode die Arbeit verrichtet.

▸ *Abbildung 2.22: Xcode kann Ihre Projekte gründlich von kompilierten Artefakten reinigen.*

Apple empfiehlt, vor der Kompilierung von Anwendungen für den App Store stets einen Aufräumvorgang durchzuführen, und das ist eine gute Vorgehensweise, die Sie sich angewöhnen sollten. Ich selbst kombiniere dabei die beiden Methoden, indem ich den Build-Ordner lösche und dann die Abhängigkeiten und vorkompilierten Header entfernen lasse. Dadurch erhalte ich ein eindeutiges Produkt, das sich leicht auffinden und nicht mit anderen Build-Versionen verwechseln lässt.

2.14.1 Kompilierung für den App Store

Um Anwendungen gemäß den Einreichungsrichtlinien für den App Store zu erstellen, müssen Sie sie mit einem gültigen Nutzungsprofil für die Verteilung und einer aktiven Entwickleridentität signieren. Wenn Sie eine korrekte Entwicklerkonfiguration eingerichtet haben, müssen Sie sich um den Großteil dieses Vorgangs nicht mehr kümmern. Übrig bleiben nur folgende Aufgaben:

- Wählen Sie **Device** als aktives SDK. Sie glauben ja gar nicht, wie viele Leute schon versucht haben, Simulator-Builds im App Store einzureichen, und sich dann stundenlang abgequält haben, bis sie bemerkt haben, wo der Fehler lag.

- Wählen Sie **Distribution** als aktive Konfiguration. Sie sollten auch das Fenster **Target Info** öffnen, um sicherzustellen, dass der Anwendungsbezeichner und die Identität für die Codesignierung korrekt festgelegt sind. Achten Sie darauf, dass **Configuration** ganz oben im Fenster auf **Active (Distribution)** oder **Distribution** gesetzt ist. Im Einblendmenü **Overview** des Projektfensters sollte **Device | Distribution** angezeigt werden.
- Kompilieren Sie die Anwendung mit **Build ▸ Compile** ⌘ + K. Die Anwendung sollte ohne Fehler kompiliert werden. Wenn nicht, sind Sie offensichtlich noch nicht so weit, dass Sie die Anwendung an den App Store übermitteln können.
- Suchen Sie das kompilierte Produkt. In der Spalte **Groups & Files** finden Sie die Gruppe **Products**. Öffnen Sie sie, und klicken Sie mit der rechten Maustaste bzw. bei gedrückter Ctrl-Taste auf die kompilierte Anwendung. Sie sollte schwarz angezeigt werden, nicht rot. Wählen Sie im Kontextmenü **Reveal in Finder**.
- Schauen Sie im Finder-Fenster nach, ob sich der Build tatsächlich in dem Ordner befindet, dessen Name auf `iphoneos` endet. (Sie können im App Store keine Simulator-Builds einreichen.)
- Klicken Sie mit der rechten Maustaste (bzw. bei gedrückter Ctrl-Taste) auf die Anwendung und komprimieren Sie sie. Anschließend übermitteln Sie die Zip-Datei über iTunes Connect an den App Store.

Wenn die Anwendung größer als 10 Mbyte ist, verwenden Sie das Mac OS X-Dienstprogramm *Application Loader*, um es an den App Store zu übermitteln. Dieses Dienstprogramm können Sie kostenlos auf der Seite **Managing Your Applications** von iTunes Connect herunterladen. Blättern Sie auf dieser Seite bis ganz nach unten, und klicken Sie auf **Get Applicaton Loader**.

2.14.2 Fehler beim Hochladen beseitigen

Manchmal kann es schwierig sein, eine Anwendung zum App Store hochzuladen. Sie melden sich bei iTunes Connect an, machen Angaben zur Anwendung und bereiten alles vor, um die Binärdatei hochzuladen, doch wenn Sie das dann tun, weist iTunes Connect die Datei ab. Die Website zeigt groß in Rosa die Meldung an, dass der Upload fehlgeschlagen ist. Liegt wirklich ein Signaturproblem vor? Sind die Zertifikate ungültig? Manchmal bestehen wirklich Schwierigkeiten aufgrund der Signatur, aber nicht immer. Mit den folgenden Schritten können Sie das Problem lösen. Manche davon kennen Sie bereits aus dem vorhergehenden Abschnitt, doch einige sind auch neu. Arbeiten Sie die ganze Liste ab, bis der Fehler beseitigt ist.

Als Erstes suchen Sie das Portal des Entwicklungsprogramms auf und vergewissern sich, dass Ihr Entwicklerzertifikat gültig ist. Es läuft nach einem bestimmten Zeitraum ab (gewöhnlich nach einem Jahr), und solange Sie es nicht erneuert haben, können Sie im App Store keine Software einreichen. Ist das Zertifikat abgelaufen, müssen Sie ein neues anfordern und neue Nutzungsprofile dafür erstellen. In den meisten Fällen, in denen Sie einem »rosa Upload der Verdammnis« begegnen, sind die Zertifikate jedoch gültig, und auch Xcode ist korrekt eingerichtet.

Kehren Sie zu Xcode zurück, und prüfen Sie, ob Sie als aktives SDK eine der Geräteoptionen eingestellt haben, z. B. **Device - 3.0**. Ein häufiger Grund für die rosafarbene Ablehnung besteht darin, dass jemand bei den Build-Einstellungen versehentlich die Simulatoreinstellung beibehalten hat. Stellen Sie als Nächstes sicher, dass Sie eine Build-Konfiguration ausgewählt haben, die Ihr Verteilungszertifikat nutzt (nicht das Entwicklerzertifikat). Dazu doppelklicken Sie auf das Ziel in der Spalte **Groups & Files** links im Projektfenster, woraufhin sich das Fenster **Target Info** öffnet. Klicken Sie auf den Titel **Build**, und prüfen Sie den Eintrag **Code Signing Identity**. Er sollte auf **iPhone Distribution:**, gefolgt von Ihrem Namen oder Firmennamen, eingestellt sein.

Auch oben links im Projektfenster können Sie die Einstellungen und Konfigurationen ablesen. Dort sollte **Device | Distribution** oder etwas Ähnliches stehen. Damit werden das aktive SDK und die aktive Konfiguration angezeigt. Wenn Sie den Upload immer noch nicht durchführen können, obwohl alle Einstellungen korrekt sind, verwenden Sie einen sauberen Build. Wählen Sie **Build ▸ Clean** ⌘ + ⇧ + K, und klicken Sie auf **Clean**. Alternativ können Sie auch manuell im Finder den Build-Ordner aus dem Projektordner entfernen. Nachdem Sie aufgeräumt haben, erstellen Sie einen neuen Build.

Verwenden Sie in den Namen der Zip-Archive, die Sie zu iTunes Connect hochladen, keine Leer- und Sonderzeichen. Die Anwendungsdatei können Sie nicht umbenennen, aber das Zip-Archiv. Namensprobleme können beim Hochladen von Anwendungen Schwierigkeiten verursachen. Die Daten innerhalb des Zip-Archivs müssen die richtige Anwendung umfassen, aber solange das der Fall ist, kann die Zip-Datei selbst einen beliebigen Namen tragen.

Wenn Sie damit immer noch nicht erreichen können, dass Ihre komprimierte Anwendung in iTunes Connect geladen wird, beenden Sie Xcode und starten es erneut. Mit diesem einfachen Trick können Sie mehr Signaturprobleme und Ablehnungen in Rosa lösen als mit den anderen bereits erwähnten Maßnahmen. Beenden Sie Xcode, starten Sie es neu, bereinigen Sie den Build, erstellen Sie einen neuen Build, komprimieren Sie die Datei, und übertragen Sie sie. Bei den meisten Entwicklern ist dieser letzte Schritt das einzige, was sie tun müssen, um über den Bildschirm mit der Ablehnung hinauszugelangen.

Wenn Sie immer noch vor einem Problem stehen, laden Sie sich das Mac OS X-Dienstprogramm *Application Loader* von der Seite **Manage Your Application** von iTunes Connect herunter. Anstatt die Anwendung direkt hochzuladen, klicken Sie auf das Markierungsfeld **Check Here to Upload Your Binary Later** und übermitteln das Archiv mithilfe von Application Loader.

Können Sie die Anwendung immer noch nicht an den App Store übermitteln, komprimieren Sie sie mit dem Programm eines Drittanbieters, oder kopieren Sie sie auf den Schreibtisch, bevor Sie sie komprimieren. Manchmal wird dadurch das Problem gelöst, dass sich eine völlig korrekt signierte Anwendung nicht erfolgreich einreichen lässt. Manche von der iTunes Connect-Website abgelehnten Anwendungen können fehlerfrei mithilfe von Application Loader hochgeladen werden.

Sie können auch versuchen, das Terminal zu starten und dort zu der kompilierten Anwendung zu wechseln. Führen Sie `codesign -vvv` *`Anwendungsname`*`.app` aus (wobei Sie natürlich den tatsächlichen Anwendungsnamen einfügen müssen), um zu prüfen, ob irgendwelche Fehlermeldungen über ungültige Signaturen auftreten.

Wenn Sie all diese Maßnahmen durchgeführt haben und die Anwendung immer noch nicht übertragen können, nehmen Sie Kontakt mit Apple auf. Senden Sie eine E-Mail an iTunes Connect (es gibt keine öffentliche Service-Telefonnummer), und erklären Sie die Situation. Geben Sie an, dass Sie die Zertifikate geprüft haben und dass diese gültig sind, und erwähnen Sie die bereits ausprobierten Maßnahmen. Wahrscheinlich kann man Ihnen bei Apple helfen, herauszufinden, warum Sie immer noch rosarot abgelehnt werden, wenn Sie versuchen, Ihre Anwendung einzureichen. In den meisten Fällen sollte aber die hier aufgezeigte Checkliste ausreichen, um die Übermittlungsprobleme zu lösen und die Anwendung zur Überprüfung einzureichen.

Hinweis

Beim Erneuern der Entwickler- und Verteilungszertifikate müssen Sie alle Nutzungsprofile neu ausgeben. Löschen Sie die alten, und erstellen Sie neue mit der neuen Entwickleridentität. Entfernen Sie veraltete Zertifikate aus dem Schlüsselbund, wenn Sie sie durch neue ersetzen.

2.15 Anwendungen für die Ad-hoc-Verteilung erstellen

Apple erlaubt Ihnen über die Ad-hoc-Verteilung, Ihre Anwendungen auch außerhalb des App Stores zu verbreiten. Dabei senden Sie Ihre Anwendungen an bis zu 100 registrierte Geräte und führen sie über eine besondere Art von Nutzungsprofil aus, durch die sie unter den FairPlay-Einschränkungen des iPhones laufen können. Die Ad-hoc-Verteilung eignet sich vor allem für Beta-Tests und für die Übermittlung von Rezensionsversionen an News-Websites und Magazine.

2.15.1 Geräte registrieren

Für die Ad-hoc-Verteilung müssen Sie als Erstes die Geräte registrieren. Fügen Sie im Portal des iPhone-Entwicklerprogramms Ihrem Account Gerätebezeichner (**Program Portal**, **Device**) und Namen hinzu. Die Bezeichner können Sie auf den iPhones selbst abrufen (mithilfe der in Kapitel 9, *Steuerelemente erstellen und verwenden*, vorgestellten `UIDevice`-Aufrufe), im Organizer von Xcode (kopieren Sie den Bezeichner, der unter dem Titel **Summary** erscheint), in iTunes (klicken Sie im Titel **Übersicht** des iPhones auf **Seriennummer**) im System-Profiler (wählen Sie **USB ▸ iPhone ▸ Seriennummer**) und mithilfe von *Ad Hoc Helper* in iTunes. Geben Sie den Bezeichner und einen eindeutigen Benutzernamen an.

2.15.2 Das Ad-hoc-Nutzungsprofil erstellen

Wenn Sie noch kein Ad-hoc-Nutzungsprofil erstellt haben, müssen Sie das jetzt tun. Wählen Sie dazu **Program Portal ▸ Provisioning ▸ Distribution**, klicken Sie auf **Add Profile**, wählen Sie **Ad Hoc**, geben Sie einen Profilnamen und Ihren üblichen Platzhalter-Anwendungsbezeichner ein (z. B. `com.ihrname.*`), und wählen Sie die Geräte aus, auf denen Sie die Anwendungen bereitstellen möchten. Vergessen Sie nicht, Ihre Identität zu prüfen, und klicken Sie dann auf **Submit**. Warten

Sie, bis Apple das neue Nutzungsprofil erstellt hat, laden Sie die Profildatei herunter, und ziehen Sie sie auf das Xcode-Anwendungssymbol. Sie brauchen diese Datei, um Ihre Anwendung zu erstellen. Nach dem Hinzufügen des Profils sollten Sie Xcode neu starten.

2.15.3 Eine Berechtigungsdatei zum Projekt hinzufügen

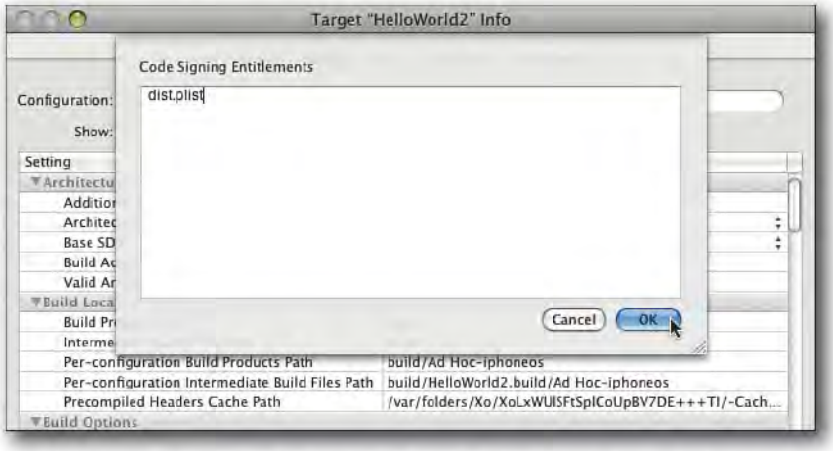
In Ad-hoc-Projekten müssen Sie eine besondere Berechtigungsdatei einfügen (siehe Apple Technical Note TN2242). Wählen Sie dazu in Xcode **File ▸ New File ▸ Code Signing ▸ Entitlements,** und klicken Sie auf **Next**. Erstellen Sie eine neue Berechtigung namens `dist.plist`, und klicken Sie auf **Finish**. Dadurch wird eine neue Datei mit willkürlichem Namen erstellt und Ihrem Projekt hinzufügt.

Suchen Sie die neue Berechtigungsdatei. Sie enthält eine einzige Eigenschaft, die Sie ändern müssen. Öffnen Sie die Datei mit einem Doppelklick, und deaktivieren Sie `get-task-allow`. (Das heißt: Setzen Sie die Eigenschaft auf den booleschen Wert `FALSE`.)

2.15.4 Die Berechtigung den Einstellungen hinzufügen

Nachdem Sie Ihre Berechtigung eingerichtet haben, müssen Sie sie zu Ihren Zieleinstellungen hinzufügen. Öffnen Sie das Fenster **Target Info**, während die Ad-hoc-Konfiguration ausgewählt ist. Achten Sie darauf, dass auch in diesem Fenster im Einblendmenü für die Konfiguration **Ad Hoc** angezeigt wird. Wenn nicht, wählen Sie diese Konfiguration aus.

Wählen Sie im Titel **Build** Ihre Ad-hoc-Konfiguration unter **Code Signing Identity** aus. Doppelklicken Sie dann auf **Code Signing Entitlements**, woraufhin sich ein interaktives Dialogfeld öffnet. Klicken Sie auf **+** und fügen Sie den Dateinamen `dist.plist` zu **Code Signing Entitlement** hinzu (siehe *Abbildung 2.23*). Klicken Sie dann auf **OK**. Alternativ können Sie die Berechtigungsdatei auch auf das Feld **Code Signing Entitlements** ziehen.



▸ *Abbildung 2.23: Fügen Sie* `dist.plist` *als neue Berechtigung für die Codesignatur von Builds für die Ad-hoc-Verteilung hinzu*

2.15.5 Die Ad-hoc-Anwendung erstellen

Jetzt sind Sie so weit, dass Sie die Anwendung erstellen können. Stellen Sie sicher, dass die Identität für die Codesignatur auf Ihr Ad-hoc-Nutzungsprofil gesetzt ist, und wählen sie **Build ▸ Build** ⌘ + B. Das neu kompilierte Produkt finden Sie in der Gruppe **Products** des Projektfensters. Klicken Sie mit der rechten Maustaste (oder bei gedrückter Ctrl-Taste) darauf und wählen Sie **Reveal in Finder**. Daraufhin wird ein Finder-Fenster geöffnet, in dem Sie das kompilierte Element sehen.

Verteilen Sie eine Kopie der Anwendung, die Sie mit dem Ad-hoc-Profil kompiliert haben, zusammen mit dem von Apple heruntergeladenen Profil selbst. Die Benutzer können dann das Profil und die Anwendung in iTunes ziehen, bevor sie die Anwendung auf ihr iPhone übertragen. Die Anwendung läuft nur auf den von Ihnen registrierten Geräten. Damit haben Sie eine sichere Möglichkeit, um sie direkt an die Benutzer zu verteilen.

2.15.6 Grafiken zu Ad-hoc-Verteilungen hinzufügen

Normal zeigt iTunes keine Grafiken für Ad-hoc-Programme an, sondern nur ein stilisiertes A. Zum Glück können Sie dieses Verhalten aber umgehen. Der iPhone-Entwickler Malcolm Hall hat mir gezeigt, wie Sie Ad-hoc-Anwendungen so einrichten, dass für sie das gewünschte Bild angezeigt wird.

Erstellen Sie im Finder einen Ordner mit zwei Elementen, nämlich einem JPEG-Bild namens `iTunesArtwork` mit den Abmessungen 512 × 512 und einen Ordner namens `Payload`. Fügen Sie das Anwendungsbundle (unkomprimiert) in den Unterordner `Payload` ein, komprimieren Sie den gesamten Ordner und benennen Sie die Zip-Datei in *Anwendungsname*`.ipa` um, wobei der Anwendungsname derjenige des Bundles in `Payload` sein muss.

Das IPA-Paket (iPhone Application) ahmt die Art und Weise *nach*, auf die Apple Anwendungen in iTunes bereitstellt. Wenn iTunes die Datei `iTunesArtwork` findet, wird diese für das Bild verwendet, dass Sie in der Anwendungsbibliothek sehen.

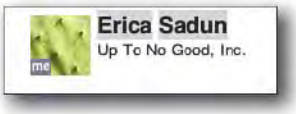
Die Datei `iTunesArtwork` darf nicht über eine Dateinamenerweiterung verfügen. Ist eine solche Erweiterung vorhanden, entfernen Sie sie, indem Sie die Datei an der Kommandozeile umbenennen. Die Datei muss zwar das JPEG-Format aufweisen, darf aber nicht die übliche Benennung mit der Endung `.jpeg` oder `.jpg` aufweisen.

Hinweis

Wenn Sie Ad-hoc-Builds an Windows Vista-Clients verteilen, müssen Sie die Benutzer anweisen, zuerst die IPA-Datei zu entkomprimieren und dann den unkomprimierten Ordner in iTunes hinzuzufügen. Vista dekomprimiert die Datei falsch, was zu Fehlern bei der Verifizierung der Anwendung führt.

2.16 Xcode-Identitäten anpassen

Xcode ist so eingerichtet, dass es den folgenden Header in Ihren Quellcode einbaut. Jedes der Elemente innerhalb der französischen Anführungszeichen (`&laquo;`, `&raquo;`) ist eine Variable, die beim Anlegen des Codes gesetzt wird. Benutzer- und Organisationsname werden aus dem Adressbuch abgerufen, wo sie als Ihre persönlichen Kontaktinformationen auftreten. Das Symbol für diesen Kontakt ist mit **Ich** (im Englischen mit **me**) gekennzeichnet, wie Sie in *Abbildung 2.24* sehen.



▸ *Abbildung 2.24: Im Adressbuch von Mac OS X ist der Kontakt, der für die Personalisierung von Xcode-Dateien verwendet wird, mit* **Ich** *bzw. im Englischen mit* **me** *in der Ecke des Benutzersymbols gekennzeichnet.*

```
/*
* main.m
* &laquo;PROJECTNAME&raquo;
*
* Created by &laquo;FULLUSERNAME&raquo; on &laquo;DATE&raquo;.
* Copyright (c) &laquo;YEAR&raquo; &laquo;ORGANIZATIONNAME&raquo;.
All rights reserved.
*
*/
```

Diese Einstellungen können Sie mit zwei Vorgaben ändern, die Sie über die Kommandozeile festlegen. Die folgenden `defaults`-Befehle legen Werte für den Organisations- und den Benutzernamen fest, die sich von denen im Adressbuch unterscheiden. Wenn Sie solche benutzerdefinierten Einstellungen verwenden, überschreiben sie diejenigen, die aus dem Adressbuch abgerufen werden.

```
defaults write com.apple.Xcode PBXCustomTemplateMacroDefinitions 'ORGA-
NIZATIONNAME= "Apple, Inc." ; FULLUSERNAME = "Jonathan I."}'
```

Sie können den Organisationsnamen auch projektweise in den Einstellungen unter **Projects Info** ▸ **General** ändern.

Ausgerechnet den String, den die meisten iPhone-Entwickler gern ändern möchten, können Sie leider nicht mit Vorgaben überschreiben. Der Bezeichner `com.yourcompany`, der in neuen Projekten erscheint, ist in den Xcode-Templates hartkodiert. Wenn Sie ihn ändern möchten, müssen Sie die mitgelieferten Templates von Apple bearbeiten oder, was noch besser ist, die Templates kopieren und in Ihrer eigenen Benutzerbibliothek ändern.

2.17 Eigene Xcode-Templates erstellen

Wenn Sie neue Projekte in Xcode erstellen, können Sie ein Template aus den Kategorien **iPhone** und **Mac OS X** auswählen, in denen Sie eine Reihe vorgefertigter Programmgerüste finden. Für das iPhone gibt es Templates für ansichtsbasierte Anwendungen und für Anwendungen, die mit OpenGL ES erstellt werden. Für den Mac können Sie u. a. dynamische Bibliotheken, Kommandozeilen-Dienstprogramme und Cocoa-Anwendungen erstellen.

Manchmal führen Sie jedoch immer wieder dieselben Schritte durch, um Ihre Projekte für ein bestimmtes firmeneigenes Design anzupassen, z. B. indem Sie den Firmenbezeichner ändern. Glücklicherweise können Sie in Xcode eigene Templates zu den vorgefertigten hinzufügen, sodass Sie bei neuen Projekten genau da anfangen können, wo Sie wirklich beginnen wollen, und nicht dort, wo Apple Sie stehen gelassen hat. Jay Abbot von *TinyPlay.com* hat mir als Erster gezeigt, wie Sie dazu vorgehen. Sie müssen eines der Templates von Apple kopieren, es in einen Ordner unter `Library/Application Support` ziehen und dort anpassen.

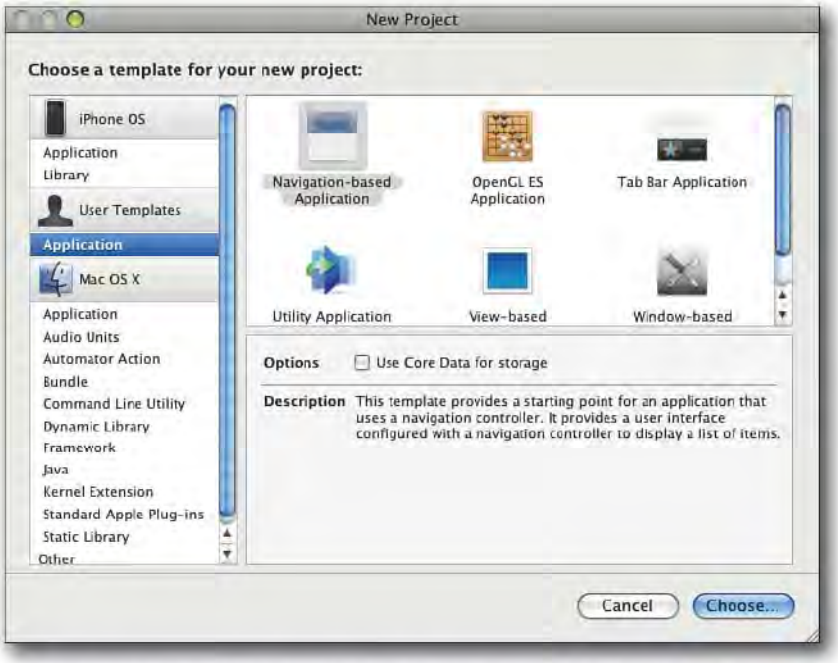
Die Projekt-Templates befinden sich im Verzeichnis `/Developer`, wobei die iPhone-Vorlagen in `/Developer/Platforms/iPhoneOS.platform/Developer/Library/Xcode/Project Templates/Application` liegen. Jeder Ordner in diesem Verzeichnis gehört zu genau einem Template.

2.17.1 Den Bezeichner »com.yourcompany« überschreiben

Eine der einfachsten Schönheitsreparaturen, die Sie durchführen können, besteht darin, `com.yourcompany` zu ersetzen. Kopieren Sie als Erstes den gesamten `Application`-Ordner aus dem Pfad für die Entwickler-Templates auf den Schreibtisch. Durchsuchen Sie alle Ordner nach Vorkommen von `com.yourcompany` innerhalb der `Info.plist`-Dateien, und ändern Sie alle in den tatsächlichen Bezeichner für Ihre Platzhalter-Nutzungsprofile. Suchen Sie auch in den Unterordnern der einzelnen Templates, um alle `Info.plist`-Dateien zu finden. Seien Sie bei der Bearbeitung vorsichtig, und ändern Sie keine der normalen Formatierungsinformationen.

Nachdem Sie die Korrekturen vorgenommen haben, wechseln Sie zum Ordner `~/Library/Application Support/Developer/Shared/Xcode` in Ihrem Benutzerverzeichnis. Erstellen Sie dort den Ordner `Project Templates`, und verschieben Sie den Ordner `Application` vom Schreibtisch dorthin. Wenn Sie Xcode das nächste Mal starten, erscheint darin der neue Abschnitt **User Templates**, in dem Ihre Versionen der Anwendungs-Templates erscheinen (siehe *Abbildung 2.25*).

Wenn Sie ein Template aus dem Abschnitt **User Templates** auswählen statt aus der Kategorie **iPhone OS**, wird Ihre angepasste Version mit der geänderten `Info.plist`-Datei geladen. Mit dieser Vorgehensweise zum Anlegen neuer Projekte sorgen Sie dafür, dass bereits ein Anwendungsbezeichner voreingestellt ist, der mit Ihren Nutzungsprofilen übereinstimmt.



▸ *Abbildung 2.25: In Xcode können Sie neue Projekte auch auf der Grundlage selbst erstellter Templates anlegen. Diese Templates befinden sich in einem besonderen Xcode-Verzeichnis Ihres Benutzerbibliotheksordners.*

2.17.2 Andere Templates erstellen

Mit eigenen Templates können Sie sehr viel mehr anfangen, als nur einen einzigen String zu ändern. Benutzer-Templates sind Ausgangspunkte für alle möglichen Arten der Projektentwicklung. Sie können eigene Bilder wie Ihr Firmenlogo sowie häufig benutzte Klassen hinzufügen. Alles, was Sie einem Template hinzufügen, steht Ihnen in Xcode zur Verwendung in neuen Projekten zur Verfügung. Wenn Sie feststellen, dass Sie in den Templates von Apple immer wieder dieselben Anpassungen vornehmen, können Sie eigene Templates verwenden, um die wiederholte Ausführung dieser Aufgaben zu vermeiden. Eigene Templates helfen Ihnen, viel Arbeit zu sparen. Wenn Sie den Vorgang der Projektinitialisierung einmal sorgfältig durchgehen, können Sie bei allen weiteren Projekten auf dieser gut vorbereiteten Grundlage aufbauen.

Untersuchen Sie die vorhandenen Templates, und kopieren Sie dasjenige, das Ihren Zielen am weitesten entgegenkommt, auf den Schreibtisch. Passen Sie den Ordner an, indem Sie die Dateien darin bearbeiten, einige Dateien entfernen oder neue hinzufügen. Um das Projekt in Xcode einzurichten, müssen Sie es aktualisieren. Sie können Verteilungs- und Ad-hoc-Konfigurationen einschließlich der Ad-hoc-Berechtigungsdatei hinzufügen. Vermeiden Sie es aber, Nutzungsprofile im Fenster **Target Info** einzurichten, denn wenn in einem Template eine Signieridentität fest verankert ist, wird es schwierig, auf andere Konfigurationen umzuschalten. Nehmen Sie so viele Bearbeitungen vor, wie Sie benötigen.

Stellen Sie sicher, dass das Template kompiliert werden kann und – bei iPhone-Quellcode – dass es problemlos im Simulator läuft. Speichern Sie die Vorlage, und löschen Sie den Build-Ordner sowie die benutzerspezifischen Dateien im Unterverzeichnis `xcodeproj` mit Ihrem Benutzernamen. (Wenn Sie das Projekt erneut bearbeiten, müssen Sie diese Dateien wiederum löschen.)

Geben Sie einen Gruppennamen für Ihre neuen Templates an, z. B. **My Custom Templates**. Dieser Name gilt für die Gruppe, die das Template besitzt, nicht aber für das Template selbst, und entspricht der Gruppe **Application** für die Templates von Apple. Ziehen Sie das bearbeitete Template in den neuen Gruppenordner, und benennen Sie den Template-Ordner sinnvoll um. Der Name des Ordners wird in Xcode als Name des Templates angezeigt.

Zum Abschluss bearbeiten Sie die Template-Beschreibung in der Datei `TemplateInfo.plist` des Ordners `xcodeproj` und optional die Bilder in der Datei `TemplateIcons.icns`. Xcode wird mit einem Symboleditor ausgeliefert, mit dem Sie nach Belieben Grafiken in ICNS-Dateien einfügen können. Anderenfalls werden die Standardsymbole des ursprünglichen, kopierten Templates verwendet.

Nachdem Sie alle diese Schritte durchgeführt haben, verfügen Sie über ein eigenes Template, das Sie in Xcode zum Erstellen neuer Projekte verwenden können. Dieses Template können Sie auch an andere weitergeben, indem Sie die Ordner komprimieren. Es ist am besten, mit dem Komprimieren auf der Ebene der Template-Gruppe anzufangen, und das Archiv dann in den Ordner `Project Templates` zu verschieben.

2.18 Ein letzter Punkt: Code nebeneinander anzeigen

Wenn Sie neue Klassen erstellen, ist es hilfreich, die Header- und die Methodendatei nebeneinander zu öffnen. Anstatt zwischen zwei verschiedenen Fenstern hin- und herzuschalten, bietet Xcode eine elegante Möglichkeit, um beide gleichzeitig zu bearbeiten. Als Erstes öffnen Sie dazu die `.m`-Datei in einem Standardeditor.

In der oberen rechten Ecke des Bearbeitungsbereichs, gleich unter den Schaltflächen **Ungrouped** und **Project** finden Sie mehrere Symbole. Genau in der Ecke ist ein Vorhängeschloss zu sehen und darunter ein geteiltes Viereck. Fahren Sie mit dem Mauszeiger über das Viereck. Der Tooltipp, der daraufhin angezeigt wird, sollte **Click to Split the Editor View** lauten.

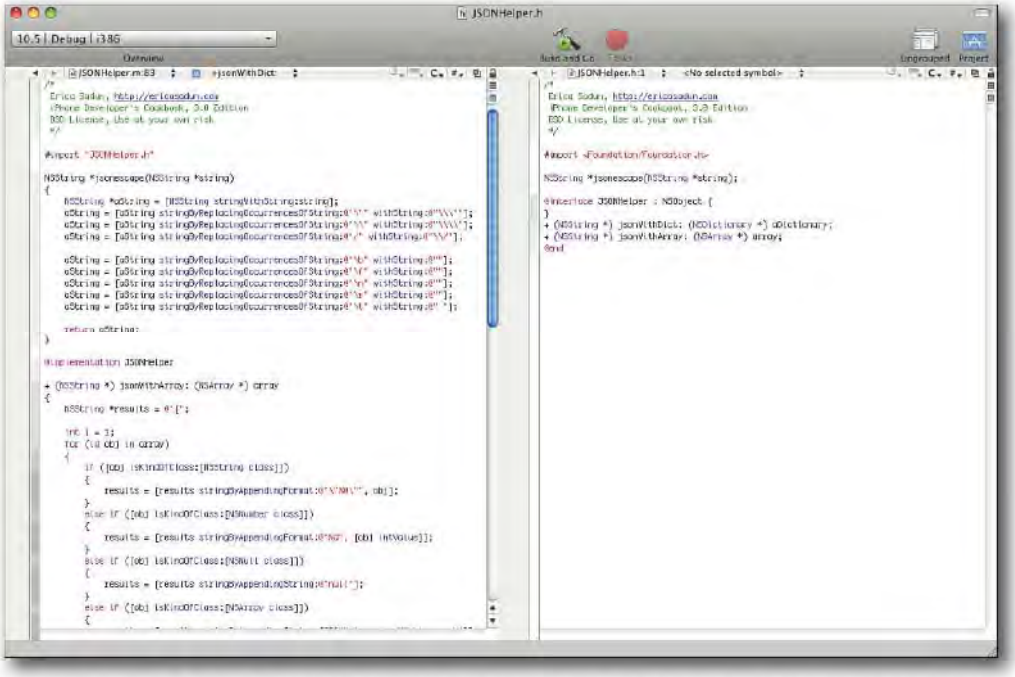
Klicken Sie bei gedrückter `Alt`-Taste auf das Viereck. Durch die Kombination mit `Alt` rufen Sie eine vertikale statt der horizontalen Aufteilung hervor, die ein Klick auf die Schaltfläche normalerweise auslöst. Unter der Schaltfläche zum Aufteilen erscheint nun eine weitere zum Zusammenfügen. Wenn Sie so weit sind, klicken Sie darauf, um das Fenster wieder auf die ungeteilte Anzeige zurückzusetzen. Belassen Sie die Anzeige jetzt jedoch im aufgeteilten Zustand.

Fahren Sie als Nächstes mit dem Mauszeiger nach oben und links von dem Vorhängeschloss in der Ecke. Der Tooltipp für diese Schaltfläche sollte **Go to Counterpart** lauten. Damit schalten Sie zwischen der `.h`- und der `.m`-Ansicht um. Klicken Sie darauf. (Alternativ können Sie auch `⌘` + `Alt` + `↑` drücken.) Anschließend wird der Bildschirm wie in *Abbildung 2.26* aktualisiert, um sowohl die `.m`- (links) als auch die `.h`-Version (rechts der Klassendefinition zusammen im Editor anzuzeigen. Dadurch haben Sie beide Ansichten in einem einzigen Fenster, was das Nachschlagen vereinfacht.

Wenn es nötig sein sollte, können Sie auch die Größe des Fensters und das Verhältnis der beiden Bereiche ändern. Die Leiste zur Größenänderung erscheint rechts neben dem Rollbalken der linken Ansicht. Sie lässt sich zunächst nur schwer erkennen, wenn Sie aber mit dem Mauszeiger über die richtige Stelle fahren, verwandelt sich der Cursor in einen Doppelpfeil zur Größenänderung. Klicken Sie, und ziehen Sie den Zeiger, um die Größe der Fenster zu ändern.

Hinweis

Wenn Sie bei gedrückter ⌘-Taste auf eine beliebige Klasse oder Methode doppelklicken, laden Sie automatisch die zugehörige Headerdatei.



▶ *Abbildung 2.26: In Xcode können Sie Header- und Quelldateien für Methoden und Klassen in einem einzigen Fenster nebeneinander bearbeiten.*

2.19 Zusammenfassung

In diesem Kapitel haben wir sehr viel besprochen. Sie haben gesehen, wie Sie Xcode-Projekte von Anfang bis Ende erstellen, kompilieren und debuggen. Außerdem haben Sie die meisten der wichtigen Xcode-Komponenten kennengelernt, die Sie täglich verwenden werden, und Sie haben etwas über die vielen verschiedenen Möglichkeiten gelernt, um iPhone-Projekte zu erstellen und auszuführen. Aus diesem Kapitel können Sie die folgenden Punkte mitnehmen:

- Xcode bietet zwar leicht zu verwendende Templates an, doch sollten Sie sie als Ausgangs- und nicht als Endpunkte ansehen. Sie können Projekte nach Ihren Vorstellungen anpassen und bearbeiten, und, wie Sie in diesem Kapitel erfahren haben, die bearbeiteten Projekte in neue Templates umwandeln.
- Mit Interface Builder können Sie Ansichten sehr leicht gestalten. Rein technisch gesehen, rufen Sie damit dieselben Methodenaufrufe und Eigenschaftenzuweisungen hervor wie beim manuellen Entwurf, doch können Sie diese Designaufgaben in der eleganten Benutzeroberfläche von Interface Builder grafisch erledigen, was viele Entwickler bevorzugen.
- Zu lernen, wie Sie sich in der integrierten Dokumentation von Xcode zurechtfinden, ist sehr wichtig, um ein iPhone-Entwickler zu werden. Niemand kann alle Informationen im Kopf bewahren. Je besser Sie mit der Oberfläche der Dokumentation vertraut werden, umso schneller finden Sie die Klassen, Methoden und Eigenschaften, die Ihnen weiterhelfen.
- Alles ist einem stetigen Wandel unterworfen. Abonnieren Sie die iPhone OS-Dokumentation in Xcode, um sicherzugehen, dass die Dokumentation stets so aktuell wie möglich ist.
- Der integrierte Debugger von Xcode sowie *Instruments* helfen Ihnen dabei, Bugs schneller zu beheben, als wenn Sie versuchen würden, sie selbst zu finden. Diese Werkzeuge mögen auf den ersten Blick kompliziert aussehen, doch es lohnt sich für die täglichen Entwicklungsarbeiten, sie gut zu beherrschen.
- Lernen Sie den Organizer kennen, und verwenden Sie ihn! Dort erhalten Sie wichtige Rückmeldungen darüber, welche Geräte angeschlossen sind und in welchem Zustand sie sich befinden. Die weiteren Werkzeuge, z. B. das Screenshot-Programm und die Konsole, erweitern das Leistungsvermögen des Organizers noch.
- Durch Konfigurationen können Sie sich wiederholende Aufgaben vermeiden. Sobald eine Konfiguration einmal festgelegt ist, können Sie ohne großen Aufwand auswählen, wie eine Anwendung kompiliert und signiert werden soll.

3 Objective-C-Trainingslager

Bei der iPhone-Entwicklung arbeiten Sie mit Objective-C, der Standardprogrammiersprache für sowohl das iPhone als auch Mac OS X. Diese Sprache ist sehr leistungsstark und erlaubt Ihnen, beim Schreiben von Anwendungen auf die Frameworks Cocoa und Cocoa Touch von Apple zurückzugreifen. In diesem Kapitel erhalten Sie grundlegende Kenntnisse in Objective-C, um mit der iPhone-Programmierung loslegen zu können. Sie erfahren etwas über Schnittstellen, Methoden, Eigenschaften, Speicherverwaltung usw. Zur Abrundung sehen wir uns neben Objective-C auch noch Cocoa an, um Ihnen die wichtigsten Klassen zu zeigen, die Sie bei Ihrer täglichen Programmierarbeit verwenden werden. Außerdem erhalten Sie konkrete Beispiele zum Einsatz dieser Klassen.

3.1 Die Sprache Objective-C

Objective-C ist eine echte Obermenge von ANSI C. Bei C handelt es sich um eine kompilierte, prozedurale Programmiersprache, die in den frühen 1970er-Jahren von AT&T entwickelt wurde. Objective-C, erdacht von Brad J. Cox, ergänzt C um objektorientierte Merkmale und Prinzipien von Smalltalk-80. Smalltalk ist eine der ältesten und bekanntesten objektorientierten Sprachen und wurde von Xerox PARC entwickelt. Cox hat das Objekt- und Nachrichtensystem auf das standardmäßige C aufgesetzt und so seine neue Sprache geformt. Dadurch können Programmierer weiterhin in der vertrauten Sprache C arbeiten, haben aber in dieser Sprache gleichzeitig Zugriff auf objektorientierte Merkmale. In den späten 1980er-Jahren wurde Objective-C als Hauptentwicklungssprache für das Betriebssystem NeXTStep der neuen Computerfirma NeXT von Steve Jobs übernommen. NeXTStep war die Inspirationsquelle und der Vorläufer von OS X. Die aktuelle Version 2.0 von Objective-C wurde zusammen mit OS X Leopard im Oktober 2007 veröffentlicht.

Bei der objektorientierten Programmierung werden Merkmale eingesetzt, die es im standardmäßigen C nicht gibt. Objekte sind Datenstrukturen mit einer festgelegten Gruppe von Funktionsaufrufen. Jedes Objekt in Objective-C verfügt über *Instanzvariablen*, bei denen es sich um die Felder der Datenstruktur handelt, und über *Methoden*, also die Funktionsaufrufe, die das Objekt ausführen kann. In objektorientiertem Code werden diese Objekte und Methoden verwendet, um in der Programmierung Abstraktionen zu ermöglichen, die die Lesbarkeit und Zuverlässigkeit des Codes verbessern.

Bei der objektorientierten Programmierung können Sie wiederverwendbare Codeeinheiten erstellen, die sich vom normalen Fluss der prozeduralen Entwicklung entkoppeln lassen. Objektorientierte Programme stützen sich nicht auf den Prozessablauf, sondern werden auf der Grundlage intelligenter Datenstrukturen aufgebaut, die aus den Objekten und deren Methoden hervorgehen. Cocoa Touch auf dem iPhone und Cocoa auf Mac OS X bieten eine umfassende Bibliothek solcher intelligenter Objekte. Mit Objective-C haben Sie Zugriff auf diese Bibliothek, sodass Sie den Werkzeugkasten von Apple nutzen können, um mit einem Minimum an Anstrengung und Code leistungsfähige Anwendungen zu schreiben.

Hinweis

Die Klassennamen von Cocoa Touch auf dem iPhone, die mit `NS` beginnen, z. B. `NSString` und `NSArray`, gehen auf NeXT zurück. *NS* steht für NeXTStep, das Betriebssystem der NeXT-Computer.

3.2 Klassen und Objekte

Objekte bilden das Kernstück der objektorientierten Programmierung. Um sie zu definieren, legen Sie Klassen an, die als Vorlagen zur Objekterstellung dienen. In Objective-C gibt eine Klassendefinition an, wie neue Objekte der betreffenden Klasse erstellt werden. Um also ein »Dingsbums«-Objekt anzulegen, definieren Sie die Klasse `Dingsbums` und erstellen damit bei Bedarf neue Objekte.

In jeder Klasse sind die Instanzvariablen und Methoden in einer öffentlichen Headerdatei mit der standardmäßigen C-Endung `.h` aufgeführt. Beispielsweise können Sie für ein Auto ein `Car`-Objekt wie das in *Listing 3.1* erstellen. Die hier gezeigte Headerdatei `Car.h` enthält die Schnittstelle, die deklariert, welche Struktur ein `Car`-Objekt aufweist. Die Namen aller Klassen in Objective-C sollten mit einem großen Anfangsbuchstaben beginnen.

```
#import <Foundation/Foundation.h>
@interface Car : NSObject
{
   int year;
   NSString *make;
   NSString *model;
}
```

```
- (void) setMake:(NSString *) aMake andModel:(NSString *) aModel
    andYear: (int) aYear;
- (void) printCarInfo;
- (int) year;
@end
```

▶ *Listing 3.1: Deklaration der Schnittstelle* `Car` *(*`Car.h`*)*

In Objective-C dient das Symbol @ zur Kennzeichnung bestimmter Schlüsselwörter. Die beiden entsprechenden Elemente in diesem Listing (`@interface` und `@end`) grenzen Start und Ende der Klassenschnittstellendefinition ab. Die Klassendefinition selbst beschreibt ein Objekt mit den drei Instanzvariablen `year`, `make` und `model`, die wiederum in den geschweiften Klammern zu Beginn der Schnittstelle deklariert werden.

Die Instanzvariable `year` wird als Integer deklariert (mit `int`), `make` und `model` dagegen sind Strings, genauer gesagt, Instanzen von `NSString`. In Objective-C wird eher diese objektorientierte Klasse verwendet als die mit `char *` definierten Bytestrings von C. Wie Sie in diesem Buch immer wieder sehen werden, bietet `NSString` weit mehr Möglichkeiten als C-Strings. Mit dieser Klasse können Sie die Länge eines Strings ermitteln, nach Teilstrings suchen und sie ersetzen, Strings umkehren, Dateierweiterungen herausziehen usw. All diese Möglichkeiten sind in die elementare Objektbibliothek von Cocoa Touch eingebaut.

Diese Klassendefinition deklariert auch drei öffentliche Methoden. Die erste heißt `setMake:andModel:andYear:`, wobei die gesamte dreiteilige Deklaration einschließlich der Doppelpunkte den Namen einer einzigen Methode bildet. In C würden Sie dazu eine Funktion wie `setProperties(char *c1, char *c2, int i)` verwenden. Der Ansatz von Objective-C wirkt zwar etwas sperriger als der von C, bietet aber mehr Klarheit und Selbstdokumentation. Sie müssen nicht raten, was es mit `c1`, `c2` und `i` auf sich hat, da die Verwendung direkt innerhalb des Namens deklariert ist:

```
[myCar setMake:c1 andModel:c2 andYear:i];
```

Die drei Methoden sind als `void`, `void` und `int` typisiert. Wie in C bezieht sich dies auf den Typ der Daten, die die Methode zurückgibt. Die ersten beiden Methoden geben keine Daten zurück, die dritte einen Integer. In C würden die Funktionsdeklarationen der zweiten und dritten Methode `void printCarInfo()` und `int year()` lauten.

Der Ansatz von Objective-C, die Argumente in die Methodennamen einzustreuen, mag für Einsteiger ungewohnt sein, entwickelt sich aber schnell zu einem geschätzten Merkmal. Es ist nicht mehr nötig, zu raten, welches Argument übergeben werden muss, da der Name einer Methode bereits anzeigt, welche Elemente sie entgegennimmt. In Objective-C werden Methodennamen auch als *Selektoren* bezeichnet, und zwar gerade bei der iPhone-Programmierung, vor allem, wenn Sie Aufrufe von `performSelector:` verwenden, um Objekten zur Laufzeit Nachrichten zu senden.

Beachten Sie, dass in der Headerdatei `#import` zum Laden von Headern verwendet wird und nicht `#include`. Beim Importieren von Headern werden in Objective-C automatisch die Dateien übersprungen, die bereits hinzugefügt sind. Dadurch können Sie `#import`-Direktiven in den verschiedenen Quelldateien mehrfach verwenden, ohne Leistungseinbußen zu riskieren.

Hinweis

Den Code für dieses Beispiel und alle anderen Beispiele in diesem Kapitel finden Sie im Beispielcode zu diesem Buch. Informationen darüber, wie Sie diesen Beispielcode aus dem Internet herunterladen können, erhalten Sie im Vorwort.

3.2.1 Objekte erstellen

Um ein Objekt zu erstellen, weisen Sie Objective-C an, Speicher dafür zuzuweisen und einen Zeiger auf das Objekt zurückzugeben. Da Objective-C eine objektorientierte Sprache ist, sieht ihre Syntax etwas anders aus als die des regulären C. Anstatt einfach Funktionen aufzurufen, weisen Sie ein Objekt an, etwas zu tun, und zwar in der Form zweier Elemente in eckigen Klammern. Das erste Element ist das Objekt, das die Nachricht empfängt, das andere die Nachricht selbst: `[objekt nachricht]`.

In unserem Beispiel sendet der Quellcode die Nachricht `alloc` an die Klasse `Car` und dann die Nachricht `init` an das neu zugewiesene `Car`-Objekt. Diese Verschachtelung ist typisch für Objective-C.

```
Car *myCar = [[Car alloc] init];
```

Das Muster »erst zuweisen, dann initiieren«, das Sie hier sehen, ist der übliche Weg zur Instanziierung eines neuen Objekts. Die Klasse `Car` führt die Methode `alloc` aus. Sie weist einen neuen Speicherblock zu, der groß genug ist, um darin alle in der Klassendefinition aufgeführten Instanzvariablen aufzunehmen, setzt die Instanzvariablen auf null und gibt einen Zeiger zurück. Der neu zugewiesene Block wird als *Instanz* bezeichnet und steht für ein einzelnes Objekt im Arbeitsspeicher.

Einige Klassen, z. B. Ansichten, weisen besondere Initialisierer wie `initWithFrame:` auf, Sie können aber auch selbst Initialisierer wie `initWithMake: andModel: andYear:` schreiben. Das Muster, dass zum Erstellen eines neuen Objekts erst eine Zuweisung und dann eine Initialisierung erfolgt, gilt jedoch immer. In jedem Fall erstellen Sie das Objekt im Speicher und setzen dann wichtige Instanzvariablen auf bestimmte Werte.

3.2.2 Speicherzuweisung

In diesem Beispiel wird Speicher von 16 Byte zugewiesen. Wie die Sternchen anzeigen, sind sowohl `make` als auch `model` Zeiger. In Objective-C zeigen Objektvariablen auf das Objekt selbst. Der Zeiger weist eine Größe von vier Byte auf, weshalb `sizeof(myCar)` den Wert 4 zurückgibt. Das Objekt besteht aus zwei Zeigern zu je vier Byte, einem Integer sowie einem zusätzlichen Feld, das nicht aus der Klasse `Car` stammt.

Dieses zusätzliche Feld entspringt der Klasse `NSObject`. In *Listing 3.1* sehen Sie rechts neben dem Doppelpunkt hinter dem Wort `Car` in der Klassendefinition die Angabe `NSObject`. Damit ist `NSObject` die Elternklasse von `Car`, von der `Car` alle Instanzvariablen und Methoden erbt. Das bedeutet, dass `Car` vom Typ `NSObject` ist und alle Speicherzuweisungen erbt, die `NSObject`-Instanzen benötigen. Dies ist der Ursprung der zusätzlichen vier Bytes.

Die Gesamtgröße des zugewiesenen Objekts beträgt 16 Byte. Darin sind die beiden `NSString`-Zeiger mit je vier Byte, ein `int` mit vier Byte und die von `NSObject` geerbte Zuweisung mit vier Byte enthalten. Mit der C-Funktion `sizeof` können Sie die Größe der Objekte auf einfache Weise ausgeben. Im folgenden Code werden mit der Anweisung `printf` des standardmäßigen C Textinformationen an die Konsole gesendet. `printf`-Befehle funktionieren in Objective-C ebenso wie in ANSI C.

```
NSObject *object = [[NSObject alloc] init];
Car *myCar = [[Car alloc] init];

// Gibt 4 zurück, die Größe eines Objektzeigers
printf("object pointer: %d\n", sizeof(object));

// Gibt 4 zurück, die Größe eines NSObject-Objekts
printf("object itself: %d\n", sizeof(*object));

// Gibt 4 zurück, wiederum die Größe eines Objektzeigers
printf("myCar pointer: %d\n", sizeof(myCar));

// Gibt 16 zurück, die Größe eines Car-Objekts
printf("myCar object: %d\n", sizeof(*myCar));
```

3.2.3 Speicher freigeben

In C weisen Sie Speicher mit `malloc()` oder einem ähnlichen Aufruf zu und geben ihn mit `free()` wieder frei; in Objective-C verwenden Sie dazu `alloc` bzw. `release`. (Es gibt in Objective-C auch einige andere Möglichkeiten, um Speicher zuzuweisen, z. B. das Kopieren anderer Objekte.)

```
[object release];
[myCar release];
```

Wie wir in Kapitel 2, *Ein erstes Projekt erstellen*, besprochen haben, ist die Freigabe von Speicher etwas komplizierter als in standardmäßigem C, da Objective-C ein Speichersystem mit Referenzzähler verwendet. Jedes Objekt im Speicher weist einen Beibehaltungszähler auf, den Sie einsehen können, indem Sie dem Objekt die Nachricht `retainCount` senden. Bei seiner Erstellung hat jedes Objekt den Wert 1 in seinem Beibehaltungszähler. Dieser Zähler wird jedes Mal um 1 verringert, wenn das Objekt die Nachricht `release` empfängt. Erreicht der Zähler für ein Objekt den Wert 0, wird es in den allgemeinen Speicherpool freigegeben.

```
// Der Beibehaltungszähler beträgt nach dem Erstellen 1
printf("The retain count is %d\n", [myCar retainCount]);

// Hierdurch wird der Beibehaltungszähler auf 0 herabgesetzt
[myCar release];
```

```
// Der folgende Befehl führt zu einem Fehler, da das Objekt bereits
// freigegeben ist
printf("Retain count is now %d\n", [myCar retainCount]);
```

Wenn Sie Nachrichten an freigegebene Objekte senden, bringen Sie Ihre Anwendung zum Absturz. Bei der Ausführung des zweiten `printf`-Befehls geht die `printf`-Nachricht an das bereits freigegebene `myCar`-Objekt. Dies führt zu einer Speicherzugriffsverletzung, weshalb das Programm abgebrochen wird.

```
The retain count is 1
objc[10754]: FREED(id): message retainCount sent to freed
object=0xd1e520
```

Auf dem iPhone gibt es keine Garbage Collection, weshalb Sie als Entwickler Ihre Objekte selbst verwalten müssen. Erhalten Sie sie so lange aufrecht, wie sie verwendet werden, und geben Sie den Speicher frei, wenn Sie sie nicht mehr benötigen. Weitere Informationen über grundlegende Verfahren zur Speicherverwaltung finden Sie weiter hinten in diesem Kapitel.

3.3 Methoden, Nachrichten und Selektoren

Zum Zuweisen und Initialisieren von Daten verwenden Sie in Standard-C zwei Funktionsaufrufe. Im Gegensatz zur Anweisung `[[Car alloc] init]` von Objective-C sieht das wie folgt aus:

```
Car *myCar = malloc(sizeof(Car));
init(myCar);
```

In Objective-C wird die Syntax *`funktionsname(argument)`* nicht verwendet. Stattdessen senden Sie in der Schreibweise mit eckigen Klammern Nachrichten an Objekte. Eine solche Nachricht weist das Objekt an, eine Methode auszuführen. Die Verantwortung dafür, die Methode zu implementieren und ein Ergebnis hervorzurufen, liegt beim Objekt. Das erste Element innerhalb der eckigen Klammern ist der Empfänger der Nachricht, das zweite ein Methodenname, unter Umständen mit Argumenten für diese Methode. Zusammengenommen bildet dies die Nachricht, die Sie senden möchten.

In C schreiben Sie beispielsweise Folgendes:

```
printCarInfo(myCar);
```

In Objective-C dagegen heißt das:

```
[myCar printCarInfo];
```

Trotz der Unterschiede in der Syntax sind Methoden im Grunde genommen Funktionen, die auf Objekten operieren. Sie weisen auch die gleichen Typen auf, die in Standard-C zur Verfügung stehen. Anders als bei Funktionsaufrufen gibt es in Objective-C jedoch Einschränkungen dafür, wer Methoden implementieren und aufrufen darf. Methoden gehören zu Klassen, und die Klassenschnittstelle definiert, welche dieser Methoden für die Außenwelt deklariert ist.

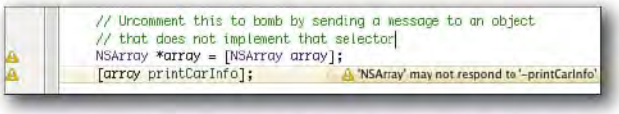
3.3.1 Dynamische Typisierung

Neben der statischen Typisierung wird in Objective-C auch eine dynamische verwendet. Bei der statischen Typisierung wird eine Variablendeklaration zur Kompilierungszeit auf eine bestimmte Klasse beschränkt. Dagegen ist bei der dynamischen Typisierung das Laufzeitsystem und nicht der Compiler dafür verantwortlich, die Objekte zu fragen, welche Methoden sie ausführen können und zu welcher Klasse sie gehören. Das bedeutet, dass während der Programmausführung ausgewählt werden kann, welche Nachrichten gesendet werden und an welche Objekte sie gehen. Dieses Merkmal bietet sehr viele Möglichkeiten und wird normalerweise mit Interpreter-Systemen wie Lisp verbunden. Sie können ein Objekt auswählen, im Programm eine Nachricht anlegen und diese Nachricht an das Objekt senden, ohne zu wissen, welches Objekt zur Kompilierungszeit ausgewählt wird und welche Nachricht ihm gesendet wird.

Mit diesen Möglichkeiten geht natürlich auch Verantwortung einher. Sie können Nachrichten nur an Objekte senden, die die im Selektor beschriebene Methode auch tatsächlich implementieren (es sei denn, die Klasse kann Nachrichten verarbeiten, die keine Implementierungen aufweisen – dazu müssen Sie die Aufrufweiterleitung von Objective-C verwenden, die weiter hinten in diesem Kapitel besprochen wird). Wenn Sie `printCarInfo` beispielsweise an ein Arrayobjekt senden, führt dies zu einem Laufzeitfehler und dem Absturz des Programms, denn in Arrays ist diese Methode nicht definiert. Nur Objekte, die die gegebene Methode implementieren, können korrekt auf die Nachricht reagieren und den angeforderten Code ausführen.

```
2009-05-08 09:04:31.978 HelloWorld[419:20b] *** -[NSCFArray printCarInfo]:
➥unrecognized selector sent to instance 0xd14e80
2009-05-08 09:04:31.980 HelloWorld[419:20b] *** Terminating app due to
➥uncaught exception 'NSInvalidArgumentException', reason: ,*** -[NSCFArray
➥printCarInfo]: unrecognized selector sent to instance 0xd14e80'
```

Bei der Kompilierung überprüft Objective-C anhand der statischen Typisierung der Objekte die an sie gesendeten Nachrichten. Die Arraydefinition in *Listing 3.1* ist statisch deklariert und teilt dem Compiler mit, dass das fragliche Objekt vom Typ `(NSArray *)` ist. Wenn der Compiler Objekte findet, die auf die angeforderten Methoden möglicherweise nicht reagieren können, gibt er eine Warnung aus.



▶ *Abbildung 3.1: Der Objective-C-Compiler von Xcode gibt eine Warnung aus, wenn er eine Methode findet, die der Empfänger anscheinend nicht implementiert.*

Diese Warnungen bedeuten nicht, dass die Kompilierung fehlschlägt, Es wäre auch durchaus möglich, dass `NSArray` die Methode `printCarInfo` tatsächlich implementiert, diese Implementierung aber lediglich nicht in der veröffentlichten Schnittstelle deklariert hätte, sodass der Code trotzdem problemlos läuft. Da `NSArray` die Methode jedoch wirklich nicht implementiert hat, führt die Ausführung dieses Codes zur Laufzeit zu dem bereits gezeigten Absturz.

Aufgrund der dynamischen Typisierung können Sie in Objective-C auf verschiedene Art und Weise auf ein und dieselbe Art von Objekt zeigen. Zwar wurde `array` als statisch typisiertes `(NSArray *)`-Objekt deklariert, doch nutzt es die gleichen internen Objektdatenstrukturen wie ein als `id` deklariertes Objekt. Der Typ `id` kann auf jedes Objekt unabhängig von seiner Klasse zeigen und ist mit `(NSObject *)` gleichwertig. Die folgende Zuweisung ist gültig und ruft keinerlei Warnungen zur Kompilierungszeit hervor:

```
NSArray *array = [NSArray array];
// Diese Zuweisung ist gültig
id untypedVariable = array;
```

Zur weiteren Veranschaulichung betrachten Sie ein veränderbares Array. `NSMutableArray` ist eine Unterklasse von `NSArray` und ermöglicht es, Arrays zu erstellen, die sich ändern und bearbeiten lassen. Wenn Sie ein veränderbares Array erstellen und typisieren, es aber zu einem Arrayzeiger zuweisen, wird dies fehlerfrei kompiliert. Im folgenden Beispiel ist `anotherArray` zwar statisch als `NSArray` typisiert, doch wenn Sie es wie folgt erstellen, wird dadurch zur Laufzeit ein Objekt erstellt, das alle Instanzvariablen und Verhalten der Klasse für veränderbare Arrays enthält.

```
NSArray *anotherArray = [NSMutableArray array];
// Dieser Aufruf einer Methode, die nur für veränderbare Arrays gilt, ist
// gültig, führt bei der Kompilierung aber zu einer Warnung
[anotherArray addObject:@"Hello World"];
```

Was hier zu einer Warnung führt, ist nicht die Erstellung und Zuweisung, sondern die Verwendung. Wenn wir `addObject:` an `anotherArray` senden, nutzen wir unser Wissen darüber aus, dass dieses Array in Wirklichkeit veränderbar ist, auch wenn es statisch als `(NSArray *)` typisiert wurde. Dies kann der Compiler jedoch nicht erkennen, weshalb die Verwendung zur Kompilierungszeit zu der Warnung »`'NSArray' may not respond to ,-addObject:'`« (»`'NSArray'` kann möglicherweise nicht auf `,-addObject:'` reagieren«) führt. Zur Laufzeit jedoch funktioniert der Code fehlerlos.

Das Objekt einer Kindklasse zu einem Zeiger der Elternklasse zuzuordnen, funktioniert zur Laufzeit zwar im Allgemeinen, doch den umgekehrten Weg zu gehen ist weit gefährlicher. Jedes veränderbare Array ist eine Art von Array und kann daher alle Nachrichten empfangen, die für Arrays gelten. Dagegen ist nicht jedes Array veränderbar. Wenn Sie die Nachricht `addObject:` an ein reguläres Array senden, ist das fatal. Zur Laufzeit ist die Katastrophe da, weil Arrays diese Methode nicht implementieren.

```
NSArray *standardArray = [NSArray array];
NSMutableArray *mutableArray;
// Diese Zeile führt zu einer Warnung
mutableArray = standardArray;
// Diese Zeile führt zur Laufzeit zu einem Desaster
[mutableArray addObject:@"Hello World"];
```

Der hier gezeigte Code ruft nur eine Warnung hervor, nämlich in der Zeile, in der das standardmäßige Arrayobjekt einem Zeiger für veränderbare Arrays zugeordnet wird. Die Warnung lautet: »assignment from distinct Objective-C type« (»Zuweisung von gesondertem Objective-C-Typ«). Zuweisungen von

Eltern- zu Kindobjekten führen nicht zu dieser Warnung, aber Zuweisungen von Kind- zu Elternobjekten. Nehmen Sie Zuweisungen also nur zwischen Klassen vor, die völlig unabhängig voneinander sind. Ignorieren Sie diese Warnungen nicht, sondern korrigieren Sie den Code, da Sie anderenfalls das Fundament für einen Absturz zur Laufzeit legen. Objective-C ist eine kompilierte Sprache mit dynamischer Typisierung und führt viele der Laufzeitüberprüfungen nicht durch, die interpretierte objektorientierte Sprachen vornehmen.

Hinweis

Den Compiler von Xcode können Sie so einrichten, dass er Warnungen als Fehler behandelt, indem Sie das Flag `GCC_TREAT_WARNINGS_AS_ERRORS` im Fenster **Project Info ▸ Build ▸ User-Defined** setzen. Da Objective-C so dynamisch ist, kann der Compiler im Gegensatz zu Compilern statischer Sprachen nicht jedes Problem erfassen, das zur Laufzeit zu einem Absturz führen kann. Achten Sie also auf die Warnungen, und versuchen Sie, deren Ursachen zu beseitigen.

3.3.2 Methoden erben

Objekte erben sowohl Methodenimplementierungen als auch Instanzvariablen. Ein `Car`-Objekt ist eine Art von `NSObject`-Objekt und kann daher auf die gleichen Nachrichten reagieren wie Letzteres. Aus diesem Grunde kann `myCar` mit `alloc` und `init` zugewiesen und initialisiert werden: Diese Methoden werden durch `NSObject` definiert und können daher von allen Instanzen von `Car` verwendet werden, da `Car` von der Klasse `NSObject` abgeleitet ist.

Ebenso sind alle `NSMutableArray`-Instanzen eine Art von `NSArray`-Objekt. Alle Methoden von Arrays können auch von veränderbaren Arrays und deren Kindklassen verwendet werden. Sie können die Elemente in einem Array zählen, ein Objekt anhand seiner Indexzahl abrufen usw.

Eine Kindklasse kann die Implementierung einer Methode ihrer Elternklasse überschreiben, aber das Vorhandensein der Methode nicht aufheben. Kindklassen erben stets sämtliche Verhalten und Eigenschaften ihrer Elternklassen.

3.3.3 Methoden deklarieren

Wie *Listing 3.1* zeigt, definiert eine Klassenschnittstelle die Instanzvariablen und Methoden, die eine neue Klasse neben denen ihrer Elternklasse hat. Die Schnittstelle befindet sich normalerweise in einer Headerdatei mit der Erweiterung `.h`. In *Listing 3.1* hat die Schnittstelle die drei folgenden Methoden deklariert:

```
- (void) setMake:(NSString *) aMake andModel:(NSString *) aModel
    andYear: (int) aYear;
- (void) printCarInfo;
- (int) year;
```

Die drei Methoden geben `void`, `void` und `int` zurück. Beachten Sie den Bindestrich zu Beginn der Methodendeklaration. Er zeigt an, dass die Methoden in den Objektinstanzen implementiert werden. So rufen Sie beispielsweise `[myCar year]` auf und nicht `[Car year]`, denn anderenfalls würde die Nachricht an die Klasse `Car` gesendet werden und nicht an ein `Car`-Objekt. Erläuterungen zu Klassenmethoden (die durch + statt durch - gekennzeichnet sind) folgen weiter hinten in diesem Abschnitt.

Wie ich bereits erwähnt habe, können Methodenaufrufe kompliziert werden. Der folgende Aufruf sendet eine Methodenanforderung mit drei Parametern, die in den Methodenaufruf eingeflochten sind. Der Name der Methode, also ihr Selektor, lautet `setMake: andModel: andYear:`, wobei die drei Doppelpunkte anzeigen, wo die Parameter eingefügt werden müssen. Die Typen der einzelnen Parameter sind in der Schnittstelle hinter den Doppelpunkten definiert, und zwar als `(NSString *)`, `(NSString *)` und `(int)`. Da diese Methode `void` zurückgibt, werden die Ergebnisse keiner Variable zugewiesen.

```
[myCar setMake:@"Ford" andModel:@"Prefect" andYear:1946];
```

3.3.4 Methoden implementieren

Das Paar aus Methoden- und Headerdatei zusammen enthält alle notwendigen Informationen, um eine Klasse zu implementieren und für den Rest der Anwendung bekannt zu machen. Der Implementierungsabschnitt einer Klassendefinition umfasst den Code zur Implementierung der Funktionalität. Dieser Quelltext befindet sich gewöhnlich in einer `.m`-Datei (für »Methode«).

Listing 3.2 zeigt die Implementierung für das Beispiel der Klasse `Car`. Es umfasst den Code für alle drei in der Headerdatei von *Listing 3.1* deklarierten Methoden und fügt eine vierte hinzu. Diese zusätzliche Methode definiert `init` neu. Die `Car`-Version von `init` setzt `make` und `model` auf `nil`, den NULL-Zeiger für Objective-C-Objekte. Außerdem initialisiert diese Methode das Baujahr auf 1901.

Die Sondervariable `self` verweist auf das Objekt, das die Methode implementiert, und wird durch das zugrunde liegende Laufzeitsystem von Objective-C bereitgestellt. In diesem Fall verweist `self` auf die aktuelle Instanz der Klasse `Car`. Der Aufruf `[self nachricht]` weist Objective-C an, eine Nachricht an das Objekt zu senden, das zurzeit die Methode ausführt.

Zu der hier gezeigten Methode `init` gibt es einiges zu bemerken. Erstens gibt sie einen Wert vom Typ `(id)` zurück. Wie ich in diesem Kapitel bereits erwähnt habe, ist der Typ `id` mehr oder weniger gleichwertig mit `(NSObject *)`, allerdings zumindest theoretisch etwas allgemeiner gefasst. Dieser Typ kann auf jegliches Objekt jeglicher Klasse zeigen (auch auf `Class`-Objekte selbst). Resultate geben Sie wie in C mit `return` zurück. Der Zweck von `init` besteht darin, eine korrekt initialisierte Version des Empfängers mithilfe von `return self` zurückzugeben.

Zweitens ruft die Methode `[super init]` auf. Dadurch wird Objective-C angewiesen, eine Nachricht an eine andere Implementierung zu senden, nämlich an die in der Oberklasse des Objekts. Die Oberklasse von `Car` ist `NSObject`, wie aus *Listing 3.1* hervorgeht. Dieser Aufruf besagt: »Bitte führe die Initialisierung durch, die meine Elternklasse normalerweise erledigt, und wende erst dann mein eigenes Verhalten an.«

Schließlich ist noch die Überprüfung `if (!self)` bemerkenswert. In seltenen Fällen können Speicherprobleme auftreten, sodass der Aufruf `[super init]` den Wert `nil` zurückgibt. Dann aber gibt die Methode `init` die Steuerung zurück, ohne irgendwelche Instanzvariablen zu setzen. Da ein `nil`-Objekt nicht auf zugewiesenen Speicher zeigt, ist es nicht möglich, über `nil` auf Instanzvariablen zuzugreifen.

Die anderen Methoden verwenden `year`, `make` und `model`, als ob dies lokal deklarierte Variablen wären. Als Instanzvariablen sind sie innerhalb des Kontextes des aktuellen Objekts definiert und können wie in diesem Beispiel gezeigt gesetzt und gelesen werden. Die Methode `UTF8String`, die an die Instanzvariablen `make` und `model` gesendet wird, konvertiert diese `NSString`-Objekte in C-Strings, die mit dem Formatspezifizierer `%s` ausgegeben werden können.

Hinweis

Sie können beliebige Nachrichten an `nil` senden, z. B. `[nil beliebigeMethode]`. Das Ergebnis ist wiederum `nil` (oder genauer gesagt, 0 umgewandelt in `nil`). Mit anderen Worten: Dieser Vorgang hat keinerlei Auswirkungen. Dank dieses Verhaltens können Sie Methodenaufrufen mit einer Absicherung für den Fall verschachteln, dass eine der einzelnen Methoden fehlschlägt und `nil` zurückgibt. Wenn während der Zuweisung mit `[[Car alloc] init]` der Speicher ausgeht, wird die `init`-Nachricht an `nil` gesendet, wodurch die gesamte `alloc`/`init`-Anforderung `nil` zurückgibt.

```
#import "Car.h"

@implementation Car
- (id) init
{
   self = [super init];
   if (!self) return nil;

   make = nil;
   model = nil;
   year = 1901;

   return self;
}

- (void) setMake:(NSString *) aMake andModel:(NSString *) aModel
    andYear: (int) aYear
{
   make = [NSString stringWithString:aMake];
   model = [NSString stringWithString:aModel];
   year = aYear;
}
```

```
- (void) printCarInfo
{
    if (!make) return;
    if (!model) return;

    printf("Car Info\n");
    printf("Make: %s\n", [make UTF8String]);
    printf("Model: %s\n", [model UTF8String]);
    printf("Year: %d\n", year);
}

- (int) year
{
    return year;
}
@end
```

▶ *Listing 3.2: Implementierung der Klasse* `Car` *(*`Car.m`*)*

3.3.5 Klassenmethoden

Klassenmethoden werden mit dem Pluszeichen (+) als Präfix definiert statt mit dem Bindestrich (-). Deklaration und Implementierung erfolgen aber wie bei Instanzmethoden. Beispielsweise können Sie einer Schnittstelle die folgende Methodendeklaration hinzufügen:

```
+ (NSString *) motto;
```

In der Implementierung kodieren Sie sie dann wie folgt:

```
+ (NSString *) motto
{
    return(@"Ford Prefects are Mostly Harmless");
}
```

Klassenmethoden unterscheiden sich von Instanzmethoden darin, dass sie generell keinen Status verwenden können. Das heißt, sie haben keinen Zugriff auf Instanzvariablen, da diese Elemente nur beim Zuweisen von Objekten im Speicher erstellt werden.

Warum also sollte man überhaupt Klassenmethoden verwenden? Dazu gibt es drei Gründe. Erstens liefern Klassenmethoden Ergebnisse, ohne dass ein Objekt instanziiert werden muss. Die Methode `motto` ruft ein hartkodiertes Ergebnis hervor, das nicht vom Zugriff auf Variablen abhängt. Hilfsmethoden wie diese sind in Klassen häufig besser aufgehoben als in der Form von Instanzmethoden.

Stellen Sie sich eine Klasse für geometrische Operationen vor. In dieser Klasse können Sie eine Umrechnung zwischen Bogenmaß und Grad implementieren, ohne dass dafür eine Instanz erforderlich ist, z. B. in der Form `[GeometryClas convertAngleToRadians:theta];`. C-Funktionen, die in der Headerdatei deklariert sind, eignen sich auch sehr gut für diese Aufgabe.

Der zweite Grund besteht darin, dass Klassenmethoden Singletons verbergen können. Dabei handelt es sich um statisch zugewiesene Instanzen, die im iPhone SDK massenhaft vorhanden sind. So gibt `[UIApplication sharedApplication]` z. B. einen Zeiger auf das Singleton-Objekt zurück, das Ihre Anwendung ist. `[UIDevice currentDevice]` ruft ein Objekt ab, das für die Hardwareplattform steht, auf der Sie arbeiten.

Durch die Kombination einer Klassenmethode mit einem Singleton können Sie von überall in Ihrer Anwendung auf diese statische Instanz zugreifen, ohne dazu einen Zeiger auf das Objekt oder eine Instanzvariable zu dessen Speicherung zu benötigen. Die Klassenmethode ruft die Referenz des Objekts ab und gibt sie auf Verlangen zurück.

Drittens fügen sich Klassenmethoden in Speicherverwaltungsverfahren ein. Nehmen wir an, Sie wollen ein neues `NSArray` zuweisen. Dies können Sie mit `[[NSArray alloc] init]` erledigen, aber auch mit `[NSArray array]`. Die Klassenmethode im zweiten Fall gibt ein Arrayobjekt zurück, das initialisiert und für die automatische Freigabe (*autorelease*) eingerichtet wurde. Wie Sie in diesem Kapitel noch lesen werden, hat Apple einen Standard für Klassenmethoden aufgestellt, die Objekte erstellen, sodass diese Methoden die Objekte stets mit automatischer Freigabe zurückgeben. Daher ist die Verwendung von Klassenmethoden ein elementarer Bestandteil des standardmäßigen Speicherverwaltungssystems auf dem iPhone.

3.3.6 Schnelle Auflistung

Die schnelle Auflistung wurde in Objective-C 2.0 eingeführt und bietet eine einfache und elegante Möglichkeit, um Sammlungen wie Arrays und Mengen aufzulisten. Hierbei wird zusätzlich eine `for`-Schleife verwendet, die die Sammlung mit einer knappen `for/in`-Syntax durchläuft. Die Auflistung ist sehr effizient und erfolgt rasch, außerdem ist sie sicher. Versuche, die Sammlung zu ändern, während sie aufgelistet wird, führen zu einer Laufzeitausnahme.

```
NSArray *colors = [NSArray arrayWithObjects:
   @"Black", @"Silver", @"Gray", nil];
for (NSString *color in colors)
   printf("Consider buying a %s car \r\n", [color UTF8String]);
```

Hinweis

Seien Sie vorsichtig, wenn Sie Methoden wie `arrayWithObjects:` oder `dictionaryWithKeysAndValues:` verwenden, da sie recht fehleranfällig sind. Häufig verwenden Entwickler diese Methoden für Instanzvariablen, ohne zuerst sicherstellen, dass die Werte nicht `nil` sind.

3.4 Klassenhierarchie

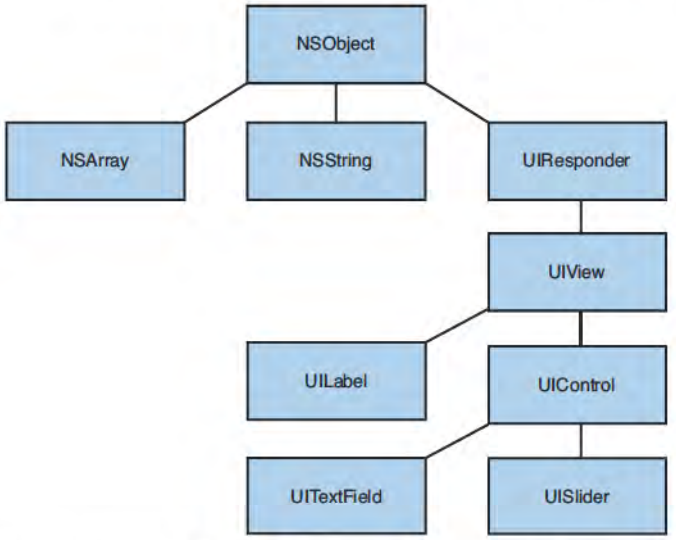
In Objective-C wird jede neue Klasse von einer bereits bestehenden abgeleitet. Die in den *Listings 3.1* und *3.2* beschriebene Klasse `Car` ist auf `NSObject` aufgebaut, der Wurzelklasse des Objective-C-Klassenbaums. Jede Unterklasse erbt Eigenschaften und Verhalten von ihrer Eltern- oder Oberklasse

und verändert sie bzw. fügt eigene Eigenschaften und Verhalten hinzu. Beispielsweise ergänzt die Klasse `Car` das geerbte gewöhnliche `NSObject` um mehrere Instanzvariablen und Methoden.

Abbildung 3.2 zeigt einige der Klassen auf dem iPhone sowie die Beziehungen zwischen ihnen in der Klassenhierarchie. Strings und Arrays stammen ebenso von `NSObject` ab wie die Klasse `UIResponder`, wobei `UIResponder` der Urahn aller iPhone-Bildschirmelemente ist. Ansichten, Beschriftungen, Textfelder und Schieberegler sind Kinder, Enkel und weitere Nachkommen von `UIResponder` und `NSObject`.

Jede Klasse außer `NSObject` stammt von einer anderen Klasse ab. `UITextField` ist eine Art von `UIControl`, das wiederum eine Art von `UIView` ist, und dies wiederum ist ein `UIResponder`, der selbst ein `NSObject` ist. In Objective-C dreht sich alles darum, neue Dinge innerhalb dieser Objekthierarchie einzufügen. Kindklassen können Folgendes tun:

- Sie können neue Instanzvariablen hinzufügen, die ihre Eltern- oder Oberklasse nicht zuweist. Die Klasse `Car` ergänzt drei davon, nämlich die Strings `make` und `model` sowie den Integer `year`.
- Sie können neue Methoden hinzufügen, die in der Elternklasse nicht definiert sind. In `Car` sind mehrere neue Methoden definiert, mit denen Sie die Werte der Instanzvariablen setzen und einen Bericht über das Auto ausgeben können.
- Sie können Methoden überschreiben, die von der Elternklasse definiert werden. Die Methode `init` der Klasse `Car` überschreibt die Version von `NSObject`. Wenn einem `Car`-Objekt eine `init`-Nachricht gesendet wird, führt es seine eigene Version der Methode aus, nicht die von `NSObject`. Gleichzeitig sorgt der Code für `init` dafür, dass die Methode `init` von `NSObject` via `[super init]` aufgerufen wird. Die Elternimplementierung zu erweitern und gleichzeitig auf sie zu verweisen ist eines der grundlegenden Entwurfsprinzipien von Objective-C.



▸ *Abbildung 3.2: Alle Cocoa Touch-Klassen stammen von* `NSObject` *ab, der Wurzel des Klassenhierarchiebaums.*

3.5 Informationen festhalten

Nachdem Sie jetzt die Grundlagen zu Klassen und Objekten kennen, müssen Sie wissen, wie Sie Informationen darüber festhalten. Neben `printf` bietet Objective-C die grundlegende Protokollierungsfunktion `NSLog`. Sie funktioniert ähnlich wie `printf`, die Ausgabe erfolgt aber in `stderr` statt in `stdout`. Außerdem verwendet `NSLog` einen `NSString`-Formatstring statt eines C-Strings.

`NSStrings` werden anders deklariert als C-Strings, nämlich mit einem voranstehenden @-Symbol. Ein typischer `NSString` sieht `@"wie folgt"` aus, der gleichwertige C-String dagegen `"wie folgt"`, also ohne das @. Während C-Strings auf einen Zeiger zu einem Bytestring verweisen, handelt es sich bei `NSStrings` um Objekte. Einen C-String können Sie bearbeiten, indem Sie die in den einzelnen Bytes gespeicherten Werte ändern, `NSStrings` dagegen sind nicht veränderbar. Sie haben keinen Zugriff auf die Bytes, um sie zu bearbeiten, und die eigentlichen Stringdaten sind nicht innerhalb des Objekts gespeichert.

```
// Dies sind 12 Bytes adressierbaren Speichers
printf("%d\n", sizeof("Hello World"));

// Dies ist ein 4-Byte-Objekt, das auf nicht adressierbaren Speicher zeigt
NSString *string = @"Hello World";
printf("%d\n", sizeof(*string));
```

`NSLog` verwendet nicht nur die standardmäßigen C-Formatspezifizierer, sondern führt auch den Objektspezifizierer `%@` ein, mit dem sich Objekte ausgeben lassen. Damit können Sie

```
printf("Make: %s\n", [make UTF8String]);
```

in folgende Form umschreiben:

```
NSLog(@"Make: %@", make);
```

Tabelle 3.1 zeigt einige der häufigsten Formatspezifizierer. Diese Liste ist jedoch alles andere als vollständig. Weitere Einzelheiten erfahren Sie im *String Programming Guide for Cocoa* von Apple (*http://developer.apple.com/mac/library/documentation/cocoa/conceptual/Strings*).

Spezifizierer	Bedeutung
`%@`	Objective-C-Objekt, das die Ergebnisse von `description` oder `descriptionWithLocale:` verwendet
`%%`	Das Literalzeichen „%“
`%d`	Integer mit Vorzeichen (32 Bit)
`%u`	Integer ohne Vorzeichen (32 Bit)
`%f`	Fließkommazahl (64 Bit)
`%e`	Fließkommazahl in wissenschaftlicher Notation mit Exponent (64 Bit)

%c	Zeichendaten ohne Vorzeichen (8 Bit)
%C	Unicode-Zeichendaten (16 Bit)
%s	Zeichendatenarray mit NULL-Terminierung (String, 8 Bit)
%S	Unicode-Zeichendatenarray mit NULL-Terminierung (16 Bit)
%p	Zeigeradresse mit Hexausgabe in Kleinbuchstaben und führendem 0x
%x	Hexwert mit Kleinbuchstaben ohne Vorzeichen (32 Bit)
%X	Hexwert mit Großbuchstaben ohne Vorzeichen (32 Bit)

► *Tabelle 3.1: Häufige Stringformatspezifizierer*

NSLog braucht kein hartkodiertes Wagenrücklaufzeichen, sondern hängt automatisch eine neue Zeile an. Darüber hinaus fügt es auch jedem Protokoll einen Zeitstempel hinzu, sodass das Ergebnis des zuvor gezeigten NSLog-Ausrufs wie folgt aussieht:

```
2009-05-07 14:19:08.792 HelloWorld[11197:20b] Make: Ford
```

Über die Nachricht description wird fast jedes Objekt in einen String verwandelt. NSLog verwendet description, um den Inhalt der mit %@ formatierten Objekte zu zeigen. Dabei wird ein NSString mit einer Beschreibung des Empfängerobjekts in Textform zurückgegeben. Sie können Objekte auch außerhalb von NSLog beschreiben, indem Sie ihnen die Nachricht description senden. Dies ist vor allem im Gegensatz zu den Befehlen printf und fprintf nützlich, die sonst keine Objekte ausgeben können.

```
fprintf(stderr, "%s\n", [[myCar description] UTF8String]);
```

Eine weitere hilfreiche Protokollierungsfunktion ist CFShow(). Sie nimmt ein Argument entgegen, nämlich ein Objekt, und gibt eine Beschreibung des Objekts als Momentaufnahme an stderr aus.

```
CFShow(make);
```

Wie NSLog sendet auch CFShow eine description-Nachricht an die Objekte, die es anzeigt, überflutet im Gegensatz zu NSLog aber nicht Ihre Debuggerkonsole mit Zeitstempeln. Damit eignet sich diese Funktion für diejenigen, die lieber auf diese zusätzlichen Informationen verzichten. Für CFShow sind keine Formatierungsstrings erforderlich, was es einfacher macht, diese Funktion zum Code hinzuzufügen. Sie kann aber nur für Objekte eingesetzt werden, nicht für Integer- oder Fließkommawerte.

3.6 Eigenschaften

Eigenschaften legen Variablen und Methoden über sogenannte *Zugriffsmethoden* nach außen offen, also über Methoden, die auf Informationen zugreifen. Die Verwendung von Eigenschaften scheint überflüssig zu sein, da in der Klassendefinition von *Listing 3.1* bereits öffentliche Methoden bekannt gemacht werden. Wozu braucht man also noch Eigenschaften? Die Verwendung von Eigenschaften hat jedoch einige Vorteile gegenüber dem Einsatz selbst gestrickter Methoden, vor allem die Punktschreibweise und die Speicherverwaltung.

3.6.1 Punktschreibweise

Dank der Punktschreibweise können Sie auf Objektinformationen zugreifen, ohne Klammern verwenden zu müssen. Anstatt `[myCar year]` aufzurufen, um die Instanzvariable `year` abzurufen, schreiben Sie `myCar.year`. Das mag zwar so aussehen, als würden Sie unmittelbar auf die Instanzvariable zugreifen, doch das ist nicht der Fall. Eigenschaften rufen stets Methoden auf, die wiederum auf die Daten eines Objekts zugreifen. Tatsächlich verletzen Sie damit die Kapselung von Objekten nicht, da die Eigenschaften auf Methoden zurückgreifen, um die Daten aus dem Objekt herauszuholen.

Da diese Methoden verborgen sind, wird das Erscheinungsbild Ihres Codes bei der Verwendung von Eigenschaften einfacher. Um z. B. den Text einer Tabellenzelle mithilfe von Eigenschaften zu setzen, gehen Sie wie folgt vor:

```
myTableViewCell.textLabel.text = @"Hello World";
```

Das ist weniger mühselig als die folgende Variante ohne Eigenschaften:

```
[[myTableViewCell textLabel] setText:@"Hello World"];
```

Die Codeversion mit Eigenschaften lässt sich leichter lesen und deutlich einfacher warten.

3.6.2 Eigenschaften und Speicherverwaltung

Eigenschaften vereinfachen die Speicherverwaltung. Sie können Eigenschaften erstellen, die Instanzvariablen während der Lebensdauer der Objekte automatisch beibehalten und sie freigeben, sobald Sie diese Variablen auf `nil` setzen. Wenn Sie eine beibehaltene Eigenschaft setzen, wird der Speicher erst dann freigegeben, wenn Sie dies anfordern.

Die Methode `arrayWithObjects:` gibt normalerweise ein Objekt mit automatischer Freigabe zurück, dessen Speicherzuweisung am Ende des Zyklus der Ereignisschleife aufgehoben wird. (Einzelheiten über Autorelease-Pools finden Sie in Kapitel 1, *Einführung in das iPhone SDK*. Eine ausführlichere Erläuterung der Speicherverwaltung folgt in diesem Kapitel.) Wenn Sie das Array zu einer beibehaltenen Eigenschaft zuweisen, bleibt es dadurch dauerhaft bestehen:

```
self.colors = [NSArray arrayWithObjects:
   @"Gray", @"Silver", @"Black"];
```

Wenn Sie das Array nicht mehr brauchen und seinen Speicher freigeben möchten, setzen Sie die Eigenschaft auf `nil`. Dies funktioniert, da Objective-C Zugriffsmethoden synthetisieren kann, um die Änderung der Werte von Instanzvariablen korrekt zu verwalten. In Wirklichkeit setzen Sie die Variable gar nicht auf `nil`, sondern weisen Objective-C an, eine Methode auszuführen, um ein zuvor gesetztes Objekts freizugeben und dann die Instanzvariable auf `nil` zu setzen. All dies geschieht aber hinter den Kulissen. Für Sie als Programmierer sieht es einfach so aus, als weisen Sie der Variable `nil` zu.

```
self.colors = nil;
```

Senden Sie keine `release`-Nachrichten an beibehaltene Eigenschaften, etwa in der Form `[self.colors release]`. Dies wirkt sich nämlich nicht auf die Zuweisung der Instanzvariable `colors` aus, sodass die Variable danach auf einen wahrscheinlich freigegebenen Speicherbereich zeigt. Wenn Sie der beibehaltenen Eigenschaft später ein Objekt zuweisen, empfängt der Speicher, auf den `self.colors` zeigt, eine weitere `release`-Nachricht, was wahrscheinlich zu einem Absturz aufgrund einer Ausnahme wegen doppelter Freigabe führt.

3.6.3 Eigenschaften erstellen

Es gibt zwei grundlegende Arten von Eigenschaften, nämlich schreibbare (`readwrite`) und schreibgeschützte (`readonly`). Bei den schreibbaren Eigenschaften, die den Standardtyp darstellen, können Sie die Werte ändern, auf die Sie zugreifen, was bei den schreibgeschützten Eigenschaften nicht möglich ist. Sie brauchen auch zwei Arten von Zugriffsmethoden, nämlich Set- und Get-Methoden.

Set-Methoden legen Informationen fest, Get-Methoden rufen sie ab. Bei der Definition dieser Methoden können Sie willkürliche Namen wählen, sich aber auch an die üblichen Vereinbarungen in Objective-C halten: Für die Methode zum Abrufen des Objekts verwenden Sie den Namen der Instanzvariable, zum Setzen fügen Sie das Präfix `set` an. Objective-C kann diese Methoden sogar für Sie synthetisieren. Nehmen wir z. B. an, dass Sie in der Klassenschnittstelle von `Car` eine Eigenschaft wie `year` auf folgende Weise deklarieren:

```
@property int year;
```

In der Klassenimplementierung lassen Sie diese Eigenschaft dann wie folgt synthetisieren:

```
@synthesize year;
```

Anschließend können Sie die Instanzvariable lesen und setzen, ohne dafür weiteren Code schreiben zu müssen. Objective-C erstellt zwei Methoden, um den aktuellen Wert abzurufen (`[myCar year]`) und ihn zu setzen (`[myCar setYear:1962]`), und macht damit zusätzlich die beiden folgenden Kürzel in Punktschreibweise möglich:

```
myCar.year = 1962;
NSLog(@"%d", myCar.year);
```

Um eine schreibgeschützte Eigenschaft zu erstellen, müssen Sie sie in der Schnittstelle mit dem Attribut `readonly` deklarieren. Für schreibgeschützte Eigenschaften gibt es Get-, aber keine Set-Methoden. Die folgende Methode gibt z. B. einen formatierten Textstring mit Angaben über ein Auto zurück:

```
@property (readonly) NSString *carInfo;
```

Objective-C kann schreibgeschützte Eigenschaften synthetisieren, Sie können die Get-Methoden aber auch selbst erstellen und der Klassenimplementierung hinzufügen. Die folgende Methode gibt eine Beschreibung des Autos mithilfe von `stringWithFormat:` zurück, wobei ein Formatstring wie `sprintf` verwendet wird, um einen neuen String zu erstellen.

```
- (NSString *) carInfo
{
   if (!self.make) return @"";
   if (!self.model) return @"";
   return [NSString stringWithFormat:
      @"Car Info\nMake: %@nYear: % %d",
      self.make, self.model, self.year];
}
```

Diese Methode steht zur Verwendung in der Punktschreibweise zur Verfügung, z. B. als `CFShow(myCar.carInfo);`.

Wenn Sie die Get-Methode für eine schreibgeschützte Eigenschaft synthetisieren lassen, sollten Sie Vorsicht walten lassen. Achten Sie darauf, dass Sie die Instanzvariable für die Eigenschaft innerhalb der Implementierungsdatei nicht in der Punktschreibweise zuweisen. Wenn Sie z. B. `model` als schreibgeschützte Eigenschaft deklariert haben, können Sie sie wie folgt zuweisen:

```
model = @"Prefect";
```

Die folgende Schreibweise dagegen ist nicht möglich:

```
self.model = @"Prefect";
```

Bei dieser zweiten Version würde `setModel:` aufgerufen, aber eine solche Methode ist für eine schreibgeschützte Eigenschaft nicht definiert.

3.6.4 Eigene Get- und Set-Methoden erstellen

Wenn Sie Eigenschaften mit `@synthesize` synthetisieren lassen, erstellt Objective-C automatisch die Methoden dafür, doch können Sie diesen Vorgang auch übergehen und die Methoden selbst anlegen. Diese Methoden können so einfach aussehen wie die folgenden. Beachten Sie, dass das zweite Wort im Namen der Set-Methode mit großem Anfangsbuchstaben geschrieben wird. Vereinbarungsgemäß erwartet Objective-C, dass die Namen von Set-Methoden der Form `setInstanz:` folgen, wobei der erste Buchstabe des Namens der Instanzvariable großgeschrieben wird.

```
-(int) year
{
   return year;
}

- (void) setYear: (int) aYear
{
   year = aYear;
}
```

Wenn Sie Ihre eigenen Set- und Get-Methoden schreiben, können Sie dabei auch für die Speicherverwaltung sorgen. Die folgenden Methoden behalten Elemente bei und geben frühere Werte frei:

```
- (NSString *) model
{
   return model;
}

- (void) setModel: (NSString *) newModel
{
   if (newModel != model) {
      [model release];
      model = [newModel retain];
   }
}
```

Hinweis

In dem seltenen Fall, dass `newModel` auf irgendeine Weise ein Kind von `model` ist, kann der Aufruf `[model release]` unter Umständen auch den Speicher des `newModel`-Objekts freigeben. Aus diesem Grund sollten Sie in einer umfassenderen Set-Methode dafür sorgen, dass `newModel` beibehalten wird, bevor Sie `[model release]` aufrufen.

Sie können sogar noch weiter gehen und kompliziertere Routinen erstellen, die neben Zuweisung und Abruf noch zusätzlichen Nutzen mit sich bringen. Beispielweise können Sie zählen, wie oft ein Wert abgerufen oder geändert wurde, oder programminterne Nachrichten an andere Objekte senden. Dem Objective-C-Compiler ist das gleich, sofern er nur zu jeder Eigenschaft eine Get-Methode (gewöhnlich mit dem Namen der Eigenschaft) und eine Set-Methode (gewöhnlich `set`*`Eigenschaftsname`*) findet. Sie können sogar die Namenskonvention von Objective-C umgehen, indem Sie die Namen der Set- und Get-Methoden in der Eigenschaftsdeklaration angeben. Mit der folgenden Deklaration erstellen Sie die neue boolesche Eigenschaft `forSale` mit einem selbst definierten Paar aus Get- und Set-Methode. Eigenschaftsdeklarationen werden immer zur Klassenschnittstelle hinzugefügt.

```
@property (getter=isForSale, setter=setSalable:) BOOL forSale;
```

Synthetisieren Sie diese Methoden dann wie üblich in der Klassenimplementierung. Die Implementierung wird gewöhnlich in der `.m`-Datei gespeichert, die die `.h`-Headerdatei begleitet.

```
@synthesize forSale;
```

Mit diesem Ansatz erstellen Sie sowohl die normalen Set- und Get-Methoden mit Punktschreibweise als auch die beiden eigenen Methoden `isForSale` und `setSalable:`. Sie können `forSale` mit der Punktschreibweise zuweisen und abrufen, aber merkwürdigerweise nicht die entsprechenden Methoden einsetzen und auch nicht die eigene Set-Methode in der Punktschreibweise verwenden, wie Sie im Folgenden sehen:

```
Car *myCar = [Car car];

// Sie können die synthetisierten Set- und Get-Methoden verwenden
[myCar setSalable:YES];
printf("The car %s for sale\n",
   myCar.isForSale ? "is" : "is not");

// Die normalen Get- und Set-Methoden funktionieren auch in
// Punktschreibweise
myCar.forSale = NO;
printf("The car %s for sale\n",
   myCar.forSale ? "is" : "is not");

// Bei den Methodenversionen geht das dagegen nicht.
// Hierbei ergeben sich Laufzeitfehler.
// [myCar setForSale:YES];
// printf("The car %s for sale\n",
//    [myCar forSale] ? "is" : "is not");
// Sie können die selbst geschriebene Set-Methode nicht in der
// Punktschreibweise verwenden.
// Dies führt zu einem Kompilierungsfehler.
// myCar.setSalable = YES;
```

3.6.5 Eigenschaftsattribute

Neben den Attributen `readwrite` bzw. `readonly` können Sie auch angeben, ob eine Eigenschaft beibehalten werden soll (`retain`) und ob sie elementar ist (`atomic`). Das Standardverhalten für Eigenschaften ist die Zuweisung (`assign`). Mit einer solchen Eigenschaft ist kein besonderes Beibehaltungs- oder Freigabeverhalten verbunden, aber dadurch, dass Sie eine Variable zu einer Eigenschaft machen, sorgen Sie dafür, dass sie außerhalb der Klasse über die Punktschreibweise zugänglich wird. Eine folgendermaßen deklarierte Eigenschaft weist das Zuweisungsverhalten auf:

```
@property NSString *make;
```

Wenn Sie für die Eigenschaft das Attribut `retain` festlegen, hat das zweierlei Auswirkungen. Erstens wird das bei der Zuweisung übergebene Objekt beibehalten, zweitens wird der vorherige Wert freigegeben, bevor die neue Zuweisung erfolgt. Mit dem Attribut `retain` nutzen Sie die Vorteile der Speicherverwaltung, die wir im vorhergehenden Abschnitt behandelt haben. Um beibehaltene Eigenschaften zu erstellen, fügen Sie in der Deklaration das betreffende Attribut in Klammern ein:

```
@property (retain) NSString *make;
```

Ein drittes Attribut namens `copy` sendet eine Kopiernachricht an das übergebene Objekt, behält es bei und gibt einen eventuellen vorherigen Wert frei:

```
@property (copy) NSString *make;
```

Sie können das Objekt auch bei der Zuweisung beibehalten:

```
myCar.make = @"Ford";
[myCar.make retain];
```

Bei der Entwicklung in einer Multithread-Umgebung ist es sinnvoll, elementare Methoden zu verwenden. Xcode synthetisiert elementare Methoden, um Objekte automatisch zu sperren, bevor sie abgerufen oder verändert werden, und um sie anschließend wieder zu entsperren. Dadurch wird sichergestellt, dass der Vorgang zum Festlegen oder Abrufen eines Objektwerts unabhängig von gleichzeitig ablaufenden Threads vollständig ausgeführt wird. Es gibt jedoch kein Schlüsselwort `atomic`, denn alle Methoden werden automatisch als elementar synthetisiert. Sie können jedoch das Gegenteil verlangen, sodass Objective-C nicht elementare Zugriffsmethoden erstellt:

```
@property (nonatomic, retain) NSString *make;
```

Bei nicht elementaren Methoden erfolgt der Zugriff schneller. Es kann jedoch Probleme geben, wenn zwei konkurrierende Threads versuchen, gleichzeitig dieselbe Eigenschaft zu verändern. Elementare Eigenschaften stellen durch ihr Sperrverhalten sicher, dass ein Objekt vollständig aktualisiert wird, bevor die Eigenschaft für einen anderen Lese- oder Änderungsvorgang freigegeben wird.

3.7 Einfache Speicherverwaltung

Im Grunde gibt es bei der Speicherverwaltung nur zwei einfache Regeln. Der Beibehaltungszähler eines Objekts hat bei dessen Erstellung den Wert 1 und bei der Freigabe den Wert 0. Ihnen als Entwickler obliegt es, sich um die Beibehaltung des Objekts während seiner Lebensdauer zu kümmern. Stellen Sie sicher, dass es von Anfang bis Ende vorhanden ist, ohne vorzeitig freigegeben zu werden, und sorgen Sie dafür, dass es schließlich freigegeben wird, wenn es an der Zeit ist, das zu tun. Was die Sache etwas komplizierter macht, ist der Autorelease-Pool von Objective-C. Wenn einige Objekte automatisch und andere manuell freigegeben werden müssen, wie handhaben Sie dann am besten die Objekte? Die folgende einfache Anleitung gibt Ihnen eine Grundvorstellung davon, wie Sie die Speicherverwaltung erledigen.

3.7.1 Objekte erstellen

Jedes Mal, wenn Sie nach dem `alloc/init`-Muster ein Objekt erstellen, setzen Sie seinen Beibehaltungszähler auf 1. Dabei spielt es keine Rolle, welche Klasse Sie verwenden oder was für ein Objekt Sie anlegen.

```
id myObject = [[SomeObject alloc] init];
```

Wenn Sie bei Variablen mit lokalem Gültigkeitsbereich das Objekt nicht vor dem Ende einer Methode freigeben, tritt ein Speicherleck auf. Die Referenz auf den Speicher wird entfernt, aber der Speicher selbst bleibt zugewiesen. Auch der Beibehaltungszähler behält den Wert +1 bei.

```
- (void) leakyMethod
{
    // Hier tritt ein Speicherleck auf
    NSArray *array = [[NSArray alloc] init];
}
```

Die richtige Art und Weise, das `alloc/init`-Muster zu verwenden, besteht darin, das Objekt zu erstellen, zu verwenden und freizugeben. Durch die Freigabe wird der Beibehaltungszähler auf o gesetzt. Am Ende der Methode wird die Zuweisung des Objekts aufgehoben.

```
- (void) properMethod
{
   NSArray *array = [[NSArray alloc] init];
   // Hier wird das Array verwendet
   [array release];
}
```

Für Autorelease-Objekte ist bei Variablen mit lokalem Gültigkeitsbereich eine explizite `release`-Anweisung nicht erforderlich (tatsächlich dürfen Sie eine solche Anwendung gar nicht verwenden, da ihr Programm sonst aufgrund einer doppelten Freigabe abstürzt). Wenn Sie einem Objekt die Nachricht `autorelease` senden, wird es für die automatische Freigabe gekennzeichnet. Wird der Autorelease-Pool am Ende einer Ereignisschleife geleert, so sendet er an alle Objekte in seinem Besitz eine `release`-Nachricht.

```
- (void) anotherProperMethod
{
   NSArray *array = [[[NSArray alloc] init] autorelease];
   // Im Gegensatz zur Verwendung von release stürzt das Programm
   // hierbei nicht ab
   printf("Retain count is %d\n", [array retainCount]);
   // Hier wird das Array verwendet
}
```

Vereinbarungsgemäß geben alle Methoden zum Erstellen von Klassenobjekten Objekte mit automatischer Freigabe zurück. Die Klassenmethode `array` von `NSArray` gibt ein neu initialisiertes Array zurück, das bereits für die automatische Freigabe vorgesehen ist. Das Objekt kann innerhalb der Methode verwendet werden und wird freigegeben, wenn der Autorelease-Pool geleert wird.

```
- (void) yetAnotherProperMethod
{
   NSArray *array = [NSArray array];
   // Hier wird das Array verwendet
}
```

Am Ende dieser Methode kehrt das Autorelease-Array in den allgemeinen Speicherpool zurück.

3.7.2 Autorelease-Objekte erstellen

Wenn Sie eine andere Methode anweisen, ein Objekt zu erstellen, zeugt es von gutem Programmierstil, wenn dieses Objekt mit automatischer Freigabe zurückgegeben wird. Wenn Sie dies konsequent tun, können Sie sich auf die folgende Regel verlassen: »Wenn ich es nicht selbst zugewiesen habe, wurde es für mich erstellt und als Autorelease-Objekt zurückgegeben.«

```
- (Car *) fetchACar
{
   Car *myCar = [[Car alloc] init];
   return [myCar autorelease];
}
```

Dies gilt vor allem für Klassenmethoden. Vereinbarungsgemäß erstellen alle Klassenmethoden neue Objekte als Autorelease-Objekte. Im Allgemeinen spricht man hier von Hilfsmethoden. Objekte, die Sie selbst zuweisen, werden dagegen nicht für die automatische Freigabe gekennzeichnet, es sei denn, dass Sie dies ausdrücklich selbst verlangen.

```
// Wird nicht automatisch freigegeben
Car *car1 = [[Car alloc] init];

// Wird automatisch freigegeben
Car *car2 = [[[Car alloc] init] autorelease];

// Dies *sollte* vereinbarungsgemäß ein automatisch freigegebenes
// Objekt sein
Car *car3 = [Car car];
```

Um eine Klassen-Hilfsmethode zu erstellen, müssen Sie sie mit dem Präfix + statt - definieren und das Objekt zurückgeben, nachdem Sie ihm eine `autorelease`-Nachricht gesendet haben.

```
+ (Car *) car
{
   return [[[Car alloc] init] autorelease];
}
```

Lebensdauer von Autorelease-Objekten

Wie lange können Sie ein Autorelease-Objekt verwenden? Welche Garantien haben Sie? Die einfache Regel lautet, dass das Objekt so lange Ihnen gehört, bis das nächste Element der Ereignisschleife verarbeitet wird. Die Ereignisschleife wiederum wird dadurch ausgelöst, dass ein Benutzer etwas antippt oder eine Taste drückt, dass ein Ereignis aufgrund eines abgelaufenen Zeitraums eintritt usw. Diese Zeiträume sind nach menschlichem Maßstab unglaublich kurz, im Bezugsrahmen des iPhone-Prozessors aber ziemlich lang. Allgemeiner gesagt, können Sie davon ausgehen, dass ein Autorelease-Objekt während der Dauer eines Methodenaufrufs bestehen bleibt.

Nachdem eine Methode die Steuerung zurückgegeben hat, können Sie sich jedoch auf nichts mehr verlassen. Wenn Sie ein Array auch außerhalb des Gültigkeitsbereichs einer einzigen Methode oder für einen längeren Zeitraum benötigen (z. B. können Sie eine benutzerdefinierte Ausführungsschleife in einer Methode beginnen und verlängern, solange die Methode besteht), gelten andere Regeln. Dann müssen Sie Autorelease-Objekte beibehalten, um ihren Zählerstand zu erhöhen und zu verhindern, dass ihre Zuweisung beim Leeren des Pools aufgehoben wird. Wenn der Autorelease-Pool die Freigabe ihres Speichers verlangt, beläuft sich ihr Beibehaltungszähler dann immer noch auf mindestens 1.

Hinweis

Weisen Sie Eigenschaften nicht sich selbst zu, z. B. `myCar.colors = myCar.colors`. Das Verhalten der Eigenschaften nach dem Motto »erst freigeben, dann beibehalten« kann dazu führen, dass die Zuweisung dieser Objekte aufgehoben wird, bevor sie erneut zugewiesen und beibehalten werden können.

3.7.3 Autorelease-Objekte beibehalten

Sie können `retain` genauso an Autorelease- wie an jegliche anderen Objekte senden. Wenn Objekte, die für die automatische Freigabe gekennzeichnet sind, beibehalten werden, können sie über die Dauer einer einzelnen Methode hinaus fortbestehen. Nachdem ein Autorelease-Objekt beibehalten wurde, ist es ebenso anfällig für Speicherlecks wie ein Objekt, das Sie mit `alloc/init` erstellen. Beispielsweise kann ein Speicherleck auftreten, wenn Sie wie folgt ein Objekt mit dem Gültigkeitsbereich einer lokalen Variable beibehalten:

```
- (void)anotherLeakyMethod
{
   // Nach der Rückgabe der Steuerung verlieren Sie die lokale
   // Referenz auf das Array und können es nicht mehr freigeben.
   NSArray *array = [NSArray array];
   [array retain];
}
```

Wenn `array` erstellt wird, hat es einen Beibehaltungszähler von +1. Sobald Sie diesem Objekt eine `retain`-Nachricht senden, wird der Zählerwert auf +2 erhöht. Wenn die Methode endet und der Autorelease-Pool geleert wird, empfängt das Objekt eine einzige `release`-Nachricht, sodass der Zählerwert wieder auf 1 sinkt. Jetzt steckt das Objekt fest. Bei einem Zählerstand von +1 kann seine Zuweisung nicht aufgehoben werden, aber da keine Referenz mehr übrig ist, die auf das Objekt zeigt, kann es auch keine endgültige `release`-Nachricht empfangen, um seinen Lebenszyklus zu beenden. Aus diesem Grund ist es von entscheidender Bedeutung, Referenzen auf beibehaltene Objekte zu erstellen.

Durch eine Referenz können Sie ein beibehaltenes Objekt während seiner Lebensdauer verwenden und es freigeben, sobald Sie es nicht mehr brauchen. Referenzen setzen Sie über eine Instanzvariable (die bevorzugte Vorgehensweise) oder über eine statische Variable, die in der Klassenimplementierung definiert ist. Um den Code einfach und zuverlässig zu gestalten, verwenden Sie beibehaltene Eigenschaften, die aus diesen Instanzvariablen erstellt wurden. Im nächsten Abschnitt lernen Sie, wie beibehaltene Eigenschaften funktionieren und warum sie die bevorzugte Lösung für Entwickler sind.

3.7.4 Beibehaltene Eigenschaften

Beibehaltene Eigenschaften halten die Daten fest, die Sie ihnen zuweisen, und verwerfen sie, wenn Sie einen neuen Wert setzen. Daher fügen sie sich problemlos in die elementare Speicherverwaltung ein. Im Folgenden erfahren Sie, wie Sie beibehaltene Eigenschaften in Ihren iPhone-Anwendungen erstellen und verwenden.

Als Erstes deklarieren Sie die beibehaltene Eigenschaft in der Klassenschnittstelle, indem Sie das Schlüsselwort `retain` in Klammern angeben:

```
@property (retain) NSArray *colors;
```

Anschließend synthetisieren Sie in der Implementierung die Eigenschaftsmethoden:

```
@synthesize colors;
```

Ist die Direktive `@synthesize` vorhanden, erstellt Objective-C automatisch Routinen zur Handhabung der beibehaltenen Eigenschaft. Diese Routinen behalten ein Objekt automatisch bei, wenn Sie es zu einer Eigenschaft zuweisen. Dieses Verhalten gilt unabhängig davon, ob das Objekt für die automatische Freigabe eingerichtet ist oder nicht. Wenn Sie der Eigenschaft ein anderes Objekt zuweisen, wird das vorherige automatisch freigegeben.

Werte zu beibehaltenen Eigenschaften zuweisen

Bei der Arbeit mit beibehaltenen Eigenschaften müssen Sie sich darüber im Klaren sein, dass es zwei Formen der Zuweisung gibt. Welchem Muster Sie folgen, hängt davon ob, ob Sie ein Autorelease-Objekt zuweisen oder ein normales. Bei Objekten mit automatischer Freigabe verwenden Sie eine einfache Zuweisung. Das folgende Beispiel zeigt eine Zuweisung, die die Eigenschaft `colors` auf das neue Array setzt und dieses beibehält:

```
myCar.colors = [NSArray arrayWithObjects:
   @"Black", @"Silver", @"Gray", nil];
```

Das Array wird erstellt und als Autorelease-Objekt mit dem Beibehaltungszähler +1 zurückgegeben. Durch die Zuweisung zur beibehaltenen Eigenschaft `colors` wird der Zählerwert auf +2 heraufgesetzt. Am Ende der aktuellen Ereignisschleife sendet der Autorelease-Pool eine `release`-Nachricht an das Array, sodass der Zähler auf +1 zurückfällt.

Normale Objekte (ohne automatische Freigabe) geben Sie nach der Zuweisung frei. Wird ein normal zugewiesenes Objekt erstellt, beträgt sein Beibehaltungszähler +1. Bei der Zuweisung zu einer beibehaltenen Eigenschaft steigt dieser Wert auf +2. Die Freigabe des Objekts setzt den Zähler wieder auf +1 zurück.

```
// Nicht automatisch freigegebenes Objekt. Der Beibehaltungszähler
// beläuft sich bei der Erstellung auf +1
NSArray *array = [[NSArray alloc]
   initWithObjects:@"Black", @"Silver", @"Gray", nil];

// Zählerstand steigt bei der Zuweisung zu einer beibehaltenen Eigenschaft
// auf +2
myCar.colors = array;

// Durch die Freigabe wird der Zähler wieder auf +1 gesenkt
[array release];
```

Dem Muster »Erstellen – Zuweisen – Freigeben« begegnen Sie bei der iPhone-Entwicklung häufiger. Sie können es verwenden, wenn Sie eine neu im Speicher zugewiesene Ansicht zu einem Ansichtscontrollerobjekt zuweisen, z. B. wie folgt:

```
UIView *mainView = [[UIView alloc] initWithFrame:aFrame];
self.view = mainView;
[mainView release];
```

In diesen drei Schritten wird der Beibehaltungszähler des Objekts von +1 auf +2 erhöht und dann wieder auf +1 gesenkt.

Ein Zählerendstand von +1 sorgt dafür, dass Sie das Objekt beliebig lange verwenden können. Gleichzeitig ist auch sichergestellt, dass das Objekt freigegeben wird, wenn die Eigenschaft auf einen neuen Wert gesetzt und für den früheren Wert `release` aufgerufen wird. Bei dieser Freigabe wird der Zähler von +1 auf 0 gesetzt, sodass die Zuweisung des Objekts automatisch aufgehoben wird.

Beibehaltene Eigenschaften erneut zuweisen

Wenn Sie eine beibehaltene Eigenschaft nicht mehr brauchen, setzen Sie sie (unabhängig davon, wie Sie sie erstellt haben) auf `nil` oder ein anderes Objekt. Dadurch wird dem zuvor zugewiesenen Objekt eine `release`-Nachricht gesendet.

```
myCar.colors=nil;
```

Haben Sie die Eigenschaft `colors` wie in unserem Beispiel auf ein Array gesetzt, erhält dieses Array automatisch eine `release`-Nachricht. Da bei jeder Zuweisung der Beibehaltungszähler des Objekts auf +1 steigt, setzt die Neuzuweisung ihn wieder von +1 auf 0 zurück. Damit endet die Lebensdauer des Objekts.

Fallgruben bei der Zuweisung vermeiden

Innerhalb der Klassenimplementierung erweist sich die Verwendung von Eigenschaften als praktisch, da Sie damit die Möglichkeiten der Speicherverwaltung nutzen können. Vermeiden Sie dabei die unmittelbare Verwendung von Instanzvariablen, da bei einer solchen direkten Zuweisung weder das Array beibehalten noch ein früherer Wert freigegeben wird. Dies ist eine Fallgrube, in die viele Anfänger stürzen. Denken Sie beim Zugriff auf Instanzvariablen auch an die Punktschreibweise.

```
colors = [NSArray arrayWithObjects:
   @"Black", @"Silver", @"Gray", nil];
```

Dieselben Vorsichtsmaßnahmen gelten auch für Eigenschaften, die als `assign` definiert sind. Sehen Sie sich die folgenden Verhaltsweisen genau an. Sowohl bei

```
@property NSArray *colors;
```

als auch bei

```
@property NSArray (assign) *colors;
```

können Sie die Punktschreibweise verwenden, doch bei der Zuweisung über diese Eigenschaften werden keinerlei Objekte beibehalten oder freigegeben. Mit `assign` definierte Eigenschaften machen die Instanzvariable `colors` zwar von außen zugänglich, bieten aber nicht die Vorteile der Speicherverwaltung, die sich bei `retain`-Eigenschaften ergeben.

HINWEIS

Als Faustregel rät Apple davon ab, Eigenschaften in `init`-Funktionen zu verwenden. Stattdessen sollten Sie dort direkt Instanzvariablen einsetzen.

3.7.5 Hohe Werte für den Beibehaltungszähler

Wenn der Wert für den Beibehaltungszähler über +1 hinausgeht und dort auch bleibt, heißt das nicht unbedingt, dass Sie etwas falsch gemacht haben. Betrachten Sie das folgende Codefragment, in dem eine Sicht erstellt und zu verschiedenen Arrays hinzugefügt wird. Der Beibehaltungszähler steigt dabei von +1 auf +4.

```
// Wenn die Ansicht erstellt wird, beträgt ihr Beibehaltungszähler +1
UIView *view = [[[UIView alloc] init] autorelease];
printf("Count: %d\n", [view retainCount]);

// Wenn Sie die Ansicht zu einem Array hinzufügen, erhöht sich der Wert
// auf +2
NSArray *array1 = [NSArray arrayWithObject:view];
printf("Count: %d\n", [view retainCount]);

// Bei einem weiteren Array steigt der Beibehaltungszähler auf +3
NSArray *array2 = [NSArray arrayWithObject:view];
printf("Count: %d\n", [view retainCount]);

// Noch ein Array führt zum Wert +4
NSArray *array3 = [NSArray arrayWithObject:view];
printf("Count: %d\n", [view retainCount]);
```

Jedes Array wurde mit einer Klassen-Hilfsmethode erstellt und gibt ein Autorelease-Objekt zurück. Auch die Ansicht ist für die automatische Freigabe gekennzeichnet. Manche Sammlungsklassen wie `NSArray` behalten Objekte automatisch bei, wenn sie sie zu einem Array hinzufügen, und geben sie frei, wenn das Objekt entfernt wird (dies gilt nur für veränderbare Objekte) oder wenn die Sammlung freigegeben wird. Dieser Code weist keine Speicherlecks auf, da jedes der vier Objekte so eingerichtet ist, dass es sich selbst und seine Kindobjekte korrekt freigibt, wenn der Autorelease-Pool geleert wird.

Wenn `release` zu den drei Arrays gesendet wird, gibt jedes von ihnen die Ansicht frei, was deren Beibehaltungswert von +4 auf +1 senkt. Die letzte `release`-Nachricht, die an das Objekt selbst geht, bringt den Zähler von +1 auf 0 herunter, sodass die Zuweisung der Ansicht am Ende der Methode aufgehoben werden kann. Kein Leck, keine weitere Beibehaltung, keine Probleme!

3.7.6 Andere Möglichkeiten zum Erstellen von Objekten

Sie haben gesehen, wie Sie mit `alloc` Speicher zuweisen. In Objective-C gibt es jedoch auch andere Möglichkeiten, um neue Objekte zu erstellen. Die Methoden dafür unterscheiden sich bei den einzelnen Klassen und Frameworks, weshalb Sie sich die Klassendokumentation ansehen müssen, um genauere Einzelheiten zu erfahren. Wenn Sie ein Objekt mit einer Methode erstellen, in deren Namen `alloc`, `new`, `create` oder `copy` vorkommt, haben Sie immer die Verantwortung dafür, das Objekt freizugeben. Im Gegensatz zu Klassen-Hilfsmethoden geben Methoden mit solchen Namen im Allgemeinen keine Autorelease-Objekte zurück.

Wenn Sie z. B. eine `copy`-Nachricht an ein Objekt senden, duplizieren Sie es. Dabei wird ein Objekt mit einem Beibehaltungswert von +1, aber ohne Zuweisung zum Autorelease-Pool zurückgegeben. Verwenden Sie `copy`, wenn Sie ein Objekt duplizieren und Änderungen an der Kopie vornehmen, das Original aber aufbewahren wollen. Beachten Sie, dass Objective-C von Sammlungen wie Arrays und Dictionarys meistens flache Kopien erstellt. Dabei werden zwar die Struktur der Sammlung und die Adressen der einzelnen Zeiger übernommen, die gespeicherten Inhalte aber nicht kopiert.

Objektzuweisung im C-Stil

Da Objective-C eine Obermenge von C ist, werden in Objective-C-Programmen für das iPhone häufig APIs verwendet, in denen Objekte wie in C erstellt und verwaltet werden. *Core Foundation* (CF) ist ein Cocoa Touch-Framework mit Funktionsaufrufen auf C-Basis. Wenn Sie in Objective-C mit CF-Objekten arbeiten, erstellen Sie sie mit `CFAllocators` und verwenden häufig die Funktion `CFRelease()`, um den Objektspeicher freizugeben.

Es gibt jedoch keine einfachen Regeln. Wie der folgende Code zeigt, ist es möglich, dass Sie `free()`, `CFRelease()` und eigene Methoden wie `CGContextRelease()` alle im selben Gültigkeitsbereich neben den standardmäßigen Klassen-Hilfsmethoden von Objective-C wie `imageWithCGImage:` verwenden. Um das Kontextobjekt zu erstellen, haben wir hier die Funktion `CGBitmaßContextCreate()` verwendet, die wie die meisten CF-Funktionsaufrufe kein Autorelease-Objekt zurückgibt. Das folgende Codefragment erstellt ein Objekt der iPhone-Klasse `UIImage`, das zum Speichern von Bilddaten dient:

```
UIImage *buildImage(int imgsize)
{
   // Erstellt Kontext mit zugewiesenen Bits
   CGContextRef context =
      MyCreateBitmapContext(imgsize, imgsize);
   CGImageRef myRef =
      CGBitmapContextCreateImage(context);
   free(CGBitmapContextGetData(context));
   CGContextRelease(context);
   UIImage *img = [UIImage imageWithCGImage:myRef];
   CFRelease(myRef);
   return img;
}
```

Carbon und Core Foundation

Da Sie noch oft genug mit Core Foundation arbeiten müssen, sollten Sie darauf vorbereitet sein, die darin enthaltenen Konstrukte zu verwenden, vor allem die Frameworks. Dabei handelt es sich um Bibliotheken mit Klassen, die Sie in Ihren Anwendungen einsetzen können.

Tabelle 3.2 erläutert die wichtigsten Begriffe. Zusammenfassend sei gesagt, dass in den ersten Versionen von Mac OS X das auf C aufgebaute Framework Core Foundation als Übergangssystem für die Entwicklung von Anwendungen verwendet wurde, die sowohl auf klassischen Mac-Systemen als auch auf Mac OS X laufen konnten. Core Foundation verwendet zwar objektorientierte Erweiterungen zu C, die Funktionen und Konstrukte basieren aber auf C, nicht auf Objective-C.

Begriff	Erklärung
Foundation	Die Kernklassen für die Objective-C-Entwicklung, die alle grundlegenden Datentypen und Dienste bereitstellen, die für Cocoa und Cocoa Touch erforderlich sind. In einem Abschnitt am Ende dieses Kapitels erhalten Sie eine Einführung in einige der wichtigsten Foundation-Klassen für Ihre Anwendungen.
Core Foundation	Eine Bibliothek mit C-Klassen, die zwar auf der Foundation-API beruhen, aber in C implementiert sind. Core Foundation verwendet objektorientierte Daten, ist aber nicht aus Objective-C-Klassen aufgebaut.
Carbon	Ein älterer Satz von Apple bereitgestellter Bibliotheken mit einer prozeduralen API. Carbon bot Unterstützung für Ereignisbehandlung, eine Grafikbibliothek und viele weitere Frameworks. Manche Carbon-APIs bestehen in Core Foundation fort. Carbon wurde für das klassische Mac OS eingeführt und erschien zum ersten Mal in Mac OS 8.1.
Cocoa	Die von Apple bereitgestellte Sammlung von Frameworks, APIs und Laufzeitumgebungen, die das moderne Laufzeitsystem von Mac OS X bilden. Die Frameworks sind größtenteils in Objective-C geschrieben, obwohl in einigen weiterhin C bzw. C++ verwendet wird.
Cocoa Touch	Das Äquivalent zu Cocoa für iPhone OS, bei dem die Frameworks für die berührungsempfindliche Benutzeroberfläche des iPhone angepasst sind. Einige iPhone-Frameworks wie Core Audio und Open GL ES werden nicht als Bestandteil von Cocoa Touch betrachtet.
Toll Free Bridging	Eine Methode zur Kombination von Cocoa und Carbon. Hierbei geht es um einen Satz von austauschbaren Datentypen. Beispielsweise können das Foundation-Objekt `(NSString *)` von Cocoa und das Core Foundation-Objekt `CFStringRef` von Carbon gleichwertig verwendet werden. Bei diesem »zollfreien Brückenschlag« wird die auf C aufgebaute Core Foundation mit der Welt von Objective-C-Foundation verbunden.

▸ *Tabelle 3.2: Schlüsselbegriffe für die OS X-Entwicklung*

Die Core Foundation-Technologie besteht in Cocoa fort. Wenn Sie iPhone-Anwendungen in Objective-C programmieren, wird Ihnen irgendwann die C-artige Core Foundation begegnen. Die Besonderheiten der Core Foundation-Programmierung sind jedoch nicht Thema dieses Kapitels. Am besten lernen Sie sie kennen, indem Sie sie für sich betrachten und nicht, während Sie sich mit den Grundlagen der Programmierung in Objective-C beschäftigen.

3.7.7 Zuweisung von Objekten aufheben

Auf dem iPhone wird Objective-C mit einer Speicherverwaltung durch Referenzzähler verwendet. Es gibt auf dem iPhone keine Garbage Collection, und die Wahrscheinlichkeit dafür, dass es sie jemals geben wird, ist sehr gering. Jedes Objekt räumt hinter sich selbst auf. Was heißt das aber für die Praxis? Die folgende kurze Zusammenfassung zeigt, wie Sie die Existenz eines Objekts beenden, wie Sie seine Instanzvariablen aufräumen und die Aufhebung der Zuweisung vorbereiten.

Instanzvariablen müssen beibehaltene Objekte freigeben, bevor die Zuweisung aufgehoben wird. Dabei müssen Sie als Entwickler dafür sorgen, dass der Beibehaltungswert dieser Objekte auf 0 sinkt, bevor das Elternobjekt freigegeben wird. Dazu implementieren Sie die Methode `dealloc`, die das Laufzeitsystem automatisch aufruft, wenn ein Objekt freigegeben werden soll. Wenn Sie eine Klasse verwenden, die Objektinstanzvariablen aufweist (also nicht nur Fließkommazahlen, Integer und boolesche Werte), müssen Sie wahrscheinlich eine Methode zum Aufheben der Zuweisung implementieren. Die Grundstruktur einer `dealloc`-Methode sieht wie folgt aus:

```
- (void) dealloc
{
    // Aufräumvorgang durch die Klasse
    Hier räume ich meine eigenen Instanzvariablen auf

    // Aufräumvorgang der Oberklasse
    [super dealloc]
}
```

Die Methode, die Sie schreiben, sollte zwei Phasen aufweisen. Als Erstes werden jegliche Instanzvariablen Ihrer Klasse aufgeräumt, anschließend weisen Sie die Oberklasse an, ihre eigene Aufräumroutine durchzuführen. Das Schlüsselwort `super` verweist auf die Oberklasse des Objekts, das die `dealloc`-Methode ausführt. Wie Sie den Aufräumvorgang gestalten, hängt davon ab, ob die Instanzvariablen automatisch beibehalten werden.

Sie haben bereits gelesen, wie Sie Objekte erstellen, wie Sie Referenzen auf Objekte anlegen und wie Sie sicherstellen, dass der Beibehaltungswert eines Objekts nach dessen Erstellung bei +1 bleibt. Jetzt sehen wir uns den letzten Schritt in der Existenz eines Objekts an, nämlich die Verringerung des Zählerwerts auf 0, damit die Zuweisung des Objekts aufgehoben werden kann.

Beibehaltene Eigenschaften

Beibehaltene Eigenschaften müssen Sie mit einer Zuweisung in Punktschreibweise auf `nil` setzen. Dadurch wird die von Objective-C synthetisierte benutzerdefinierte Set-Methode aufgerufen und das Objekt freigegeben, das der Eigenschaft zuvor zugewiesen war. Hatte dieses frühere Objekt einen Beibehaltungswert von +1, wird der Zähler durch diese Freigabe auf 0 gesetzt.

```
self.make = nil;
```

Variablen

Wenn Sie einfache Instanzvariablen (keine Eigenschaften) verwenden oder Formateigenschaften zuweisen, senden Sie die Nachricht `release`, wenn die Zuweisung aufgehoben werden soll. Nehmen wir an, Sie haben die Instanzvariable `salesman` definiert. Sie kann zu einem beliebigen Zeitpunkt während der Lebensdauer des Objekts gesetzt werden. Eine Zuweisung von `salesman` kann wie folgt aussehen:

```
// Freigabe des früheren Werts
[salesman release];

// Neue Zuweisung erfolgt. Beibehaltungswert beträgt +1
salesman = [[SomeClass alloc] init];
```

Bei einer Zuweisung dieser Art kann `salesman` zu jedem Zeitpunkt der Lebensdauer des Objekts auf ein Objekt mit einem Beibehaltungswert von +1 zeigen. In Ihrer `dealloc`-Methode müssen Sie daher das Objekt, das zurzeit zu `salesman` zugewiesen ist, freigeben, um den Zählerwert auf 0 zu verringern.

```
[salesman release];
```

Beispiel für eine dealloc-Methode

Betrachten wir als Beispiel unsere erweiterte Klasse `Car` mit beibehaltenen Eigenschaften für `make`, `model` und `colors` sowie der einfachen Instanzvariable `salesman`. Die endgültige `dealloc`-Methode sieht dann wie im Folgenden gezeigt aus. Die Instanzvariablen `year` und `forSale` sind Integer- bzw. boolesche Werte und keine Objekte, weshalb sie nicht auf diese Weise verwaltet werden müssen.

```
- (void) dealloc
{
    self.make = nil;
    self.model = nil;
    self.colors = nil;
    [salesman release];
    [super dealloc];
}
```

Eine Obergrenze für den Beibehaltungswert ist entscheidend dafür, dass die Speicherverwaltung von Objective-C funktioniert. Kein Objekt sollte einen Beibehaltungswert größer als +1 haben, nachdem es erstellt und zugewiesen wurde. Die Grenze von +1 stellt sicher, dass die endgültige Freigabe in `dealloc` den Wert auf 0 herabsetzt.

Zusätzliche Aufräumvorgänge

Die Methode `dealloc` ist der ideale Ort, um aufzuräumen. Beispielsweise kann es sein, dass Sie einen Audio Toolbox-Klang loswerden oder andere Wartungsaufgaben verrichten müssen, bevor eine Klasse freigegeben werden kann. Solche Aufgaben betreffen fast ausnahmslos ältere Frameworks wie Core Foundation, Core Graphics und Core Audio oder ähnliche Frameworks im C-Stil.

```
if (snd) AudioServicesDisposeSystemSoundID(snd);
```

Betrachten Sie `dealloc` als Ihre letzte Gelegenheit, um lose Enden abzuschneiden, bevor das Objekt für immer verschwindet. Ob Sie nun offene Sockets herunterfahren, Dateizeiger schließen oder Ressourcen freigeben müssen – nutzen Sie diese Methode, um sicherzustellen, dass der Code in einen so sauberen Zustand wie möglich zurückkehrt.

3.8 Singletons verwenden

Über die Klassen `UIApplication` und `UIDevice` haben Sie Zugriff auf die zurzeit laufende Anwendung und die Gerätehardware, auf der sie ausgeführt wird. Zu diesem Zweck bieten diese Klassen Singletons an, also eine einzige Instanz einer Klasse im aktuellen Prozess. Beispielsweise gibt `[UIApplication sharedApplication]` ein Singleton zurück, das Informationen darüber zurückliefert, welcher Delegate verwendet wird, ob die Anwendung die Bearbeitung durch Schütteln unterstützt, welche Fenster das Programm definiert usw.

Die meisten Singleton-Objekte sind Steuerungszentralen. Unter anderem koordinieren sie Dienste, stellen Schlüsselinformationen bereit und lenken den externen Zugriff. Wenn Sie eine zentrale Funktion benötigen, z. B. einen Manager, der auf einen Webdienst zugreift, können Sie durch die Verwendung eines Singletons sicherstellen, dass alle Teile der Anwendung mit demselben zentralen Manager koordiniert werden.

Zum Erstellen eines Singletons benötigen Sie nur sehr wenig Code. Sie definieren eine statische, gemeinsam genutzte Instanz innerhalb der Klassenimplementierung und führen eine Klassenmethode hinzu, die auf diese Instanz zeigt. In dem folgenden Fragment, das aus dem Tagging-Beispiel von Kapitel 6, *Ansichten und Animationen zusammenstellen*, stammt, wird die Instanz erstellt, sobald sie zum ersten Mal angefordert wird.

```
@implementation ViewIndexer
static ViewIndexer *sharedInstance = nil;

+(ViewIndexer *) sharedInstance {
   if(!sharedInstance)
      sharedInstance = [[self alloc] init];
   return sharedInstance;
}

// Hier ist das Verhalten der Klasse definiert

@end
```

Um dieses Singleton zu verwenden, rufen Sie `[ViewIndexer sharedInstance]` auf. Dadurch wird das gemeinsam genutzte Objekt zurückgegeben, sodass Sie Zugriff auf das vom Singleton gebotene Verhalten haben. Um andere Klassen daran zu hindern, eine zweite Instanz zu erstellen, überschreiben Sie `allocWithZone:`. (Für die meisten Verwendungszwecke ist das jedoch schon so übervorsichtig, dass es ans Paranoide grenzt.) Die hier verwendete Direktive `@synchronized()` verhindert, dass dieser Code durch mehr als einen Thread auf einmal ausgeführt wird.

```
+ (id)allocWithZone:(NSZone *)zone
{
    @synchronized(self) {
        if (sharedInstance == nil) {
            sharedInstance = [super allocWithZone:zone];
            return sharedInstance;
        }
    }
    return nil;
}
```

3.9 Kategorien (zur Erweiterung von Klassen)

Die Möglichkeit, bereits bestehende Klassen zu erweitern, ist eines der leistungsstärksten Merkmale von Objective-C. Die Erweiterung des Verhaltens wird als *Kategorie* bezeichnet. Kategorien erweitern den Funktionsumfang von Klassen, ohne dass Sie Unterklassen bilden müssen. Stattdessen wählen Sie einen beschreibenden Namen für die Erweiterung aus, erstellen einen Header und implementieren die Funktionalität in einer Methodendatei. Kategorien können Methoden auch zu bestehenden Klassen hinzufügen, die beim Erstellen der Kategorie noch nicht definiert waren und deren Quellcode nicht vorlag.

Um eine Kategorie zu erstellen, müssen Sie eine neue Schnittstelle deklarieren. Geben Sie den Namen der Kategorie (Sie können ihn beliebig wählen) in Klammern an, wie Sie im folgenden Beispiel sehen. Führen Sie dann alle neuen öffentlichen Methoden und Eigenschaften auf, und speichern Sie die Headerdatei. Die Kategorie `Orientation` im Beispiel erweitert die Klasse `UIDevice`, bei der es sich um die SDK-Klasse für die Meldung von Gerätemerkmalen wie Ausrichtung, Ladezustand der Batterie und Status des Annäherungssensors handelt. Diese Schnittstelle fügt `UIDevice` eine einzige Eigenschaft hinzu, die einen schreibgeschützten booleschen Wert zurückgibt. Die neue Eigenschaft `isLandscape` meldet, ob das Gerät zurzeit quer ausgerichtet ist.

```
@interface UIDevice (Orientation)
@property (nonatomic, readonly) BOOL isLandscape;
@end
```

Anders als beim Erstellen von Unterklassen können Sie in einer Kategorieschnittstelle keine neuen Instanzvariablen hinzufügen. Stattdessen erweitern Sie das Verhalten der Klasse, wie der Quellcode von *Listing 3.3* zeigt. Hier wird die Überprüfung der Ausrichtung durch einen Blick auf die Standardeigenschaft `orientation` von `UIDevice` implementiert.

Diese neue Eigenschaft können Sie wie folgt verwenden:

```
NSLog(@"The device orientation is%@landscape",
   [UIDevice currentDevice].isLandscape ? @" " : @" not ");
```

Hier ist die Prüfung der Querausrichtung nahtlos in die SDK-Klasse `UIDevice` eingebaut, und zwar über eine Eigenschaft, die es vor der Erweiterung der Klasse nicht gab. Nebenbei bemerkt, enthält `UIKit` Makros zum Thema Geräteausrichtung (`UIDeviceOrientationPortrait` und `UIDeviceOrientationLandscape`), denen Sie aber einen vom Gerät abgerufenen Ausrichtungswert übergeben müssen.

Hinweis

Mithilfe von Kategorien können Sie nicht nur neues Verhalten zu bestehenden Klassen hinzufügen, sondern auch verwandte Methoden für selbst erstellte Klassen in getrennten Dateien gruppieren. Bei umfangreichen, komplexen Klassen kann dies die Wartung und die Verwaltung der einzelnen Quelldateien erleichtern. Wenn Sie eine Kategoriemethode hinzufügen, die die gleiche Signatur hat wie eine bestehende Methode, verwendet das Laufzeitsystem von Objective-C Ihre Implementierung und überschreibt das Original.

```
@interface UIDevice (Orientation)
@property (nonatomic, readonly) BOOL isLandscape;
@end

@implementation UIDevice (Orientation)
- (BOOL) isLandscape
{
   return (self.orientation == UIDeviceOrientationLandscapeLeft) ||
      (self.orientation == UIDeviceOrientationLandscapeRight);
}
@end
```

▸ *Listing 3.3: Die Kategorie* `Orientation` *für die Klasse* `UIDevice` *erstellen*

3.10 Protokolle

In Kapitel 1 wurden Delegates eingeführt, mit deren Hilfe sich Einzelheiten implementieren lassen, die während der Definition der Klasse noch nicht bekannt sind. Beispielsweise ist einer Tabelle zwar klar, wie die Zellenreihen angezeigt werden sollen, sie weiß aber nicht, was zu geschehen hat, wenn der Benutzer auf eine Zelle tippt. Die Bedeutung einer solchen Berührung hängt von der Anwendung ab, die die Tabelle implementiert. Beim Antippen kann ein anderer Bildschirm geöffnet, eine Nachricht an einen Webserver gesendet oder irgendeine andere denkbare Aufgabe ausgeführt werden. Durch

Delegierung kann die Tabelle mit einem intelligenten Objekt kommunizieren, das für die Handhabung solcher Berührungen zuständig ist, dessen Verhalten aber nicht beim Erstellen der Tabellenklasse geschrieben wird, sondern zu einem davon völlig unabhängigen Zeitpunkt.

Die Delegierung stellt im Grunde eine Sprache für die Vermittlung zwischen einem Objekt und seinem Handler bereit. Eine Tabelle teilt ihrem Delegate mit, dass sie angetippt, dass in ihr geblättert oder dass ihr Status auf andere Weise verändert wurde. Der Delegate entscheidet dann, wie auf diese Nachrichten zu reagieren ist, und führt auf der Grundlage der Semantik seiner Anwendung die entsprechenden Aktualisierungen herbei.

Datenquellen funktionieren ähnlich, doch vermitteln Sie keine Reaktionen auf Vorgänge, sondern stellen Daten auf Anforderung bereit. Wenn eine Tabelle fragt, welche Informationen sie in Zelle 1 und 2 packen soll, antwortete die Datenquelle mit den angeforderten Informationen. Wie bei der Delegierung können Tabellen auch bei Datenquellen Anfragen an ein Objekt richten, das die jeweiligen Anforderungen kennt.

In Objective-C werden sowohl die Delegierung als auch die Verwendung von Datenquellen durch das System der sogenannten *Protokolle* ermöglicht. Protokolle definieren im Voraus, wie Klassen miteinander kommunizieren können. Sie enthalten eine Liste von Methoden, die nicht innerhalb von Klassen definiert sind. Einige dieser Methoden sind erforderlich, andere optional. Wenn eine Klasse die erforderlichen Methoden implementiert, so spricht man davon, dass sie dem Protokoll gehorcht oder es erfüllt.

3.10.1 Ein Protokoll definieren

Nehmen wir als Beispiel einen Springteufel – einen kleinen Kasten mit einer Kurbel. Wenn Sie die Kurbel drehen, spielt Musik, und manchmal springt eine Puppe aus dem Kasten. Stellen Sie sich jetzt vor, dass Sie dieses Spielzeug (besser gesagt, eine grobe Annäherung) in Objective-C implementieren. Es gibt eine mögliche Aktion, nämlich das Drehen der Kurbel, und zwei mögliche Ergebnisse: Musik und das Erscheinen des Teufelchens.

Betrachten wir nun einen Programmclient für dieses Spielzeug. Er könnte beispielsweise auf die Ergebnisse reagieren, indem er beim Abspielen der Musik nach und nach einen Langeweilezähler erhöht und beim Erscheinen des Teufelchens Überraschung zeigt. In Objective-C müssen Sie für diesen Client also zwei Reaktionen implementieren, einen für die Musik und einen für das Teufelchen (im Englischen »jack« genannt). Ein mögliches Clientprotokoll sieht wie folgt aus:

```
@protocol JackClient
- (void) musicDidPlay;
- (void) jackDidAppear;
@end
```

Dieses Protokoll legt fest, dass etwas auf das Abspielen der Musik und das Erscheinen des Springteufels reagieren muss, um ein Client des Spielzeugs zu sein. Die Auflistung dieser Methoden innerhalb eines `@protocol`-Containers definiert das Protokoll. Alle hier aufgeführten Methoden sind erforderlich, sofern Sie sie nicht ausdrücklich als `@optional` deklarieren. Mehr darüber erfahren Sie im nächsten Abschnitt.

3.10.2 Ein Protokoll eingliedern

Stellen Sie sich als Nächstes vor, dass Sie eine Klasse für das Spielzeug selbst entwerfen. Sie stellt eine mögliche Aktion bereit, nämlich das Drehen der Kurbel, und erfordert ein zweites Objekt, das das Protokoll implementiert und in diesem Fall als Client bezeichnet wird. Die Schnittstelle der Klasse legt fest, dass der Client eine Art von Objekt (`id`) sein muss, das dem Protokoll `JackClient` gehorcht (`id <JackClient>`). Darüber hinaus weiß die Klasse zu diesem Zeitpunkt nichts darüber, was für eine Art von Objekt dieser Dienst bereitstellt.

```
@interface JackInTheBox : NSObject
{
   id <JackClient> client;
}
- (void) turnTheCrank;
@property (retain) id <JackClient> client;
@end
```

3.10.3 Callbacks hinzufügen

Über Callbacks wird die Klasse für das Spielzeug mit ihrem Client verbunden. Da der Client das Protokoll `JackClient` erfüllen muss, können Sie ihm die Nachrichten `jackDidAppear` und `musicDidPlay` senden, ohne dass es bei der Kompilierung zu einem Fehler kommt. Das Protokoll sorgt dafür, dass der Client diese Methoden implementiert. Im folgenden Code wird die Callback-Methode zufällig ausgewählt. Bei ungefähr neun von zehn Aufrufen spielt die Musik, sodass `musicDidPlay` an den Client gesendet wird.

```
- (void) turnTheCrank
{
   // Sie brauchen einen Client, der auf das Drehen der Kurbel reagiert
   if (!self.client) return;

   // Zufällig ausgewählte Reaktion auf die Kurbeldrehung
   int action = random() % 10;
   if (action < 1)
      [self.client jackDidAppear];
   else
      [self.client musicDidPlay];
}
```

3.10.4 Optionale Callbacks deklarieren

In Protokollen treten zwei Arten von Callbacks auf, nämlich erforderliche und optionale, wobei erforderliche Callbacks den Standard bilden. Eine Klasse, die dem Protokoll gehorcht, muss diese Methode implementieren, da ansonsten eine Compiler-Warnung erfolgt. Mit den Schlüsselwörtern `@required` und `@optional` können Sie festlegen, zu welcher Art eine Methode gehört. Alle hinter

`@required` aufgeführten Methoden sind erforderlich, alle hinter `@optional` aufgeführten sind optional. Daraus können Sie Ihr Protokoll nach Bedarf zusammensetzen.

```
@protocol JackClient <NSObject>
- (void) musicDidPlay; // erforderlich
@required
- (void) jackDidAppear; // ebenfalls erforderlich
@optional
- (void) nothingDidHappen; // optional
@end
```

In der Praxis ist es unsinnig, wenn mehr Schlüsselwörter erscheinen als ein einziges Mal `@optional`. Das gleiche Protokoll können Sie auch einfacher deklarieren. Wenn Sie gar keine optionalen Elemente verwenden, lassen Sie das Schlüsselwort ganz weg. Beachten Sie die Deklaration `<NSObject>` in dem Beispiel. Sie ist erforderlich, damit optionale Protokolle implementiert werden können. Diese Deklaration besagt, dass ein `JackClient`-Objekt eine Art von `NSObject` ist.

```
@protocol JackClient <NSObject>
- (void) musicDidPlay;
- (void) jackDidAppear;
@optional
- (void) nothingDidHappen;
@end
```

3.10.5 Optionale Callbacks implementieren

Bei optionalen Protokollmethoden hat der Client die Wahl, ob er sie implementieren will. Dadurch verringern Sie die Implementierungspflichten für denjenigen, der den Client schreibt, laden der Klasse mit der Protokolldefinition aber zusätzliche Arbeit auf. Wenn Sie nicht sicher sind, ob eine Klasse eine bestimmte Methode implementiert oder nicht, müssen Sie dies erst überprüfen, bevor Sie eine Nachricht senden. Zum Glück geht das in Objective-C und der Klasse `NSObject` einfach:

```
// Optionale Clientmethode
if ([self.client respondsToSelector: @selector(nothingDidHappen)])
   [self.client nothingDidHappen];
```

`NSObject` stellt die Methode `respondsToSelector:` bereit, die den booleschen Wert `YES` zurückgibt, wenn ein Objekt die Methode implementiert, und anderenfalls `NO`. Wenn Sie den Client als `<NSObject>` deklarieren, teilen Sie dem Compiler mit, dass der Client diese Methode handhaben kann, sodass Sie den Client auf Kompatibilität prüfen können, bevor Sie ihm eine Methode senden.

3.10.6 Ein Protokoll erfüllen

Die Konformität mit einem Protokoll ist bei Klassen in ihrer Schnittstellendeklaration enthalten. Wenn ein Ansichtscontroller das Protokoll `JackClient` implementiert, so ist dies in spitzen Klammern angegeben. Eine Klasse kann auch mehreren Protokollen gehorchen. Dazu geben Sie alle Protokollnamen innerhalb des Klammernpaars an und trennen sie jeweils durch Kommata.

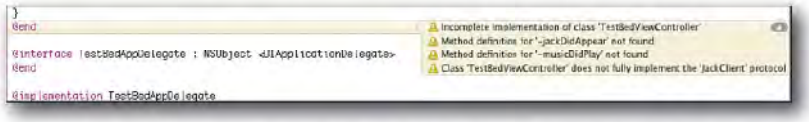
```
@interface TestBedViewController :
   UIViewController <JackClient>
{
   JackInTheBox *jack;
}
@property (retain) JackInTheBox *jack;
@end
```

Wenn Sie das Protokoll `JackClient` deklarieren, können Sie die Eigenschaft `client` des Hosts zuweisen. Der folgende Code wird fehlerlos kompiliert, da die Klasse für `self` im Einklang mit `JackClient` deklariert wurde.

```
self.jack = [JackInTheBox jack];
self.jack.client = self;
```

Hätten Sie die Protokolldeklaration in der Schnittstelle weggelassen, würde diese Zuweisung bei der Kompilierung zu einem Fehler führen.

Nachdem Sie das Protokoll in spitzen Klammern angegeben haben, *müssen* Sie dessen gesamten erforderlichen Methoden in Ihrer Klasse implementieren. Wenn Sie auch nur eine davon auslassen, führt das zu den Compiler-Warnungen, die Sie in *Abbildung 3.3* sehen. Der Compiler teilt Ihnen mit, welche Methoden fehlen und zu welchem Protokoll sie gehören.



► *Abbildung 3.3: Sie müssen alle erforderlichen Methoden implementieren, um ein Protokoll zu erfüllen. Objective-C gibt bei unvollständigen Implementierungen Warnungen aus.*

Die meisten Protokollmethoden im iPhone SDK sind optional. Sowohl die erforderlichen als auch die optionalen Methoden sind in der Entwicklerdokumentation ausführlich beschrieben. Beachten Sie, dass die Dokumentation der Protokolle von der Dokumentation der Klassen, die sie unterstützen, getrennt ist. Beispielsweise enthält die Xcode-Dokumentation drei verschiedene Referenzseiten für `UITableView`: eine für die Klasse `UITableView`, eine für das Protokoll `UITableViewDelegate` und eine für das Protokoll `UITableViewDataSource`.

3.11 Foundation-Klassen

Es gibt einige wenige Schlüsselklassen, mit denen Sie sich unbedingt vertraut machen müssen, bevor Sie sich auf das Programmieren stürzen, und das gilt vor allem, wenn Ihnen Objective-C noch neu ist. Dazu gehören Strings, Zahlen und Sammlungen, die entscheidende Bausteine für das Erstellen von Anwendungen bilden. Beispielsweise ist die Klasse `NSString` das Arbeitstier für fast alle Aufgaben der Textbearbeitung in Objective-C. Wie andere grundlegende Klassen ist sie jedoch nicht in Objective-C selbst definiert, sondern gehört zum Framework *Foundation*, in dem Sie nahezu alle wichtigen Klassen finden, die Sie für die tägliche Arbeit brauchen.

Foundation umfasst mehr als ein Dutzend Objektfamilien und Hunderte von Objektklassen – von Wertobjekten zum Speichern von Zahlen und Daten über Strings zum Festhalten von Zeichendaten und Sammlungen für andere Objekte bis zu Klassen, die auf das Dateisystem zugreifen und Daten von URLs abrufen. Foundation wird manchmal auch (nicht ganz korrekt) als Cocoa bezeichnet. (Cocoa und seine iPhone-Entsprechung Cocoa Touch enthalten tatsächlich alle Frameworks für die Mac OS X-Programmierung.) Wenn Sie Foundation beherrschen, beherrschen Sie auch die Programmierung in Objective-C, aber eine gründliche Abhandlung dieses Themas würde ein eigenes Buch ergeben.

Ich kann Ihnen in diesem Abschnitt keine umfassende Einführung in die Foundation-Klassen geben, sondern nur einen knappen Überblick als »Überlebenshilfe« für Programmierer. Im Folgenden finden Sie die Klassen, die Sie kennen müssen, und einige absolut notwendige Grundsätze, um mit ihnen zu arbeiten. Außerdem habe ich viele Codebeispiele beigefügt, die die Verwendung der einzelnen Klassen veranschaulichen, sodass Sie wenigstens einen Ausgangspunkt haben.

3.11.1 Strings

Wie ihre Vettern, die C-Strings vom Typ `(char *)`, dienen auch Cocoa-Strings zum Speichern von Zeichendaten. Allerdings handelt es sich bei ihnen um Objekte statt um Bytearrays. Anders als in C kann die Klasse `NSString` in Cocoa nicht verändert werden. Zwar können Sie Strings heranziehen, um daraus andere Strings zu erstellen, Sie können aber die Strings, die Sie bereits haben, nicht bearbeiten. Stringkonstanten sind durch Anführungszeichen und ein @-Symbol begrenzt. Im Folgenden sehen Sie eine typische Stringkonstante, die einer Stringvariable zugewiesen wird:

```
NSString *myString = @"A string constant";
```

Strings erstellen

Sie können Strings mit einer Formatierung erstellen, ganz ähnlich wie bei der Verwendung von `sprintf`. Wenn Sie bereits damit vertraut sind, `printf`-Anweisungen zu schreiben, können Sie Ihre Kenntnisse unmittelbar auf die Stringformatierung übertragen. Um Objekte in Strings aufzunehmen, verwenden Sie den Formatspezifizierer `%@`. Die verschiedenen Stringformatspezifizierer sind ausführlich im *Cocoa String Programming Guide* dokumentiert, den Sie über das Dokumentationsfenster von Xcode einsehen können ⌘ + Alt + ?. Die am häufigsten verwendeten Formate sind in *Tabelle 3.1* aufgeführt.

```
NSString *myString = [NSString stringWithFormat:
   @"The number is %d", 5];
```

Sie können Strings aneinanderhängen, um neue Strings zu bilden. Der folgende Aufruf gibt »The number is 522« aus. Dabei wird eine neue Instanz aus anderen Strings aufgebaut.

```
NSLog(@"%@", [myString stringByAppendingString:@"22"]);
```

Durch das Anhängen von Formaten gewinnen Sie noch weitere Möglichkeiten. Sie können den Formatstring und die Bestandteile angeben, aus denen das Ergebnis aufgebaut wird:

```
NSLog(@"%@", [myString stringByAppendingString:@"%d", 22]);
```

Längen und Zeichen mit Indexwert

Jeder String kann seine Länge angeben (über `length`) und auf Aufforderung ein Zeichen mit einem bestimmten Indexwert ausgeben. Die beiden folgenden Aufrufe führen zu der Ausgabe 15 bzw. e für den zuvor gezeigten String `@"The number is 5"`. Cocoa-Zeichen weisen den Typ `unichar` auf, in dem Unicode-Zeichen abgelegt werden.

```
NSLog(@"%d", myString.length);
printf("%c", [myString characterAtIndex:2]);
```

Konvertierung aus und in C-Strings

Auch wenn Sie in Objective-C arbeiten, machen sich häufig die Umstände der Programmierung in C bemerkbar, weshalb es sehr wichtig ist, zwischen C- und Cocoa-Strings wechseln zu können. Um einen `NSString` in einen C-String zu konvertieren, senden Sie ihm entweder `UTF8String` oder `cStringUsingEncoding:`. Beide Verfahren sind gleichwertig und ergeben dieselben C-Bytes:

```
printf("%s\n", [myString UTF8String]);
printf("%s\n", [myString cStringUsingEncoding: NSUTF8StringEncoding]);
```

Sie können auch umgekehrt vorgehen und einen C-String in einen `NSString` umwandeln, indem Sie `stringWithCString: encoding:` verwenden. Die hier gezeigten Beispiele weisen eine UTF-8-Kodierung auf, aber Objective-C unterstützt eine breite Palette von Möglichkeiten, z. B. ASCII, Japanisch, Lateinisch, Windows-CP1251 usw.

```
NSLog(@"%@", [NSString stringWithCString:"Hello World" encoding:
   NSUTF8StringEncoding]);
```

Strings in Dateien schreiben und aus Dateien lesen

Eine praktische Möglichkeit zum Speichern und Abrufen von Daten besteht darin, Strings in das lokale Dateisystem zu schreiben und von dort zu lesen. Das folgende Fragment zeigt, wie Sie einen String in eine Datei schreiben:

```
NSString *myString = @"Hello World";
NSError *error;
NSString *path = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/file.txt"];
if (![myString writeToFile:path atomically:YES
encoding:NSUTF8StringEncoding
error:&error])
{
   NSLog(@"Error writing to file: %@", [error localizedDescription]);
   return;
}
NSLog(@"File successfully written to file");
```

Der Pfad für die Datei ist `NSHomeDirectory()`, eine Funktion, die einen String mit einem Pfad zurückgibt, der auf die Anwendungs-Sandbox zeigt. Beachten Sie die besondere Methode, mit der der Teilpfad `Documents/file.txt` angehängt wird.

In Cocoa gibt es bei den meisten Routinen für den Dateizugriff eine Option, um den Vorgang als Einheit durchzuführen. Wenn Sie den Parameter `atomically` auf `YES` setzen, schreibt das iPhone die Datei in eine temporäre Hilfsdatei und benennt diese dann um. Durch einen solchen als Einheit durchgeführten Schreibvorgang vermeiden Sie eine Beschädigung der Datei.

Die hier gezeigte Anforderung gibt einen booleschen Wert zurück, nämlich `YES`, wenn der String geschrieben wurde, und `NO`, wenn dies nicht der Fall ist. Sollte die Schreibanforderung fehlschlagen, protokolliert dieser Code den Fehler mit einer übersetzten Beschreibung. Die Fehlerinformation wird in einer Instanz der Klasse `NSError` gespeichert, und der Selektor `localizedDescription` wird gesendet, um die Informationen in eine für den Menschen lesbare Form zu übersetzen. Wenn eine iPhone-Methode Fehler zurückgibt, können Sie mit diesem Ansatz bestimmen, welcher Fehler ausgelöst wurde.

Das Lesen eines Strings aus einer Datei folgt dem gleichen Muster, gibt aber kein boolesches Ergebnis zurück. Stattdessen müssen Sie überprüfen, ob der zurückgegebene String `nil` ist, und den zurückgegebenen Fehler anzeigen, wenn dies der Fall ist.

```
NSString *inString = [NSString stringWithContentsOfFile:path
encoding:NSUTF8StringEncoding error:&error];
if (!inString)
{
NSLog(@"Error reading from file % %@", [path lastPathComponent],
[error localizedDescription]);
return;
}
NSLog(@"File successfully read from file");
NSLog(@"%@", inString);
```

Zugriff auf Teilstrings

In Cocoa gibt es eine Reihe von Möglichkeiten, um Teilstrings aus Strings zu extrahieren. Die folgende kurze Übersicht zeigt einige der typischen Ansätze. Wie Sie wahrscheinlich erwarten, nimmt die Bearbeitung von Strings einen großen Teil jeder flexiblen API ein. Cocoa bietet sehr viel mehr Routinen und Klassen, um Strings zu analysieren und zu interpretieren, als die wenigen, die hier aufgeführt sind. In dieser groben Übersicht über `NSString` werden z. B. `NSScanner`, `NSXMLParser` usw. nicht erwähnt.

Strings in Arrays konvertieren

Sie können einen String in ein Array umwandeln, indem Sie seine Bestandteile an dem dazwischen auftretenden Begrenzungszeichen abtrennen. Im folgenden Beispiel wird der String an den Leerzeichen aufgetrennt und in einzelne Wörter zerlegt. Die Leerzeichen werden verworfen, sodass nur ein Array mit den einzelnen Zahlwörtern übrig bleibt.

```
NSString *myString = @"One Two Three Four Five Six Seven";
NSArray *wordArray = [myString componentsSeparatedByString: @" "];
NSLog(@"%@", wordArray);
```

Teilstrings nach Index abrufen

Sie können Teilstrings vom Anfang eines Strings bis zu einem bestimmten Index oder von einem Index bis zum Ende anfordern. Bei den beiden folgenden Beispielen mit den `to`- und `from`-Versionen der Teilstringindexabfrage wird `@"One Two"` bzw. `@"Two Three Four Five Six Seven"` zurückgegeben. Wie in Standard-C beginnen die Indizes von Arrays und Strings bei 0.

```
NSString *sub1 = [myString substringToIndex:7];
NSLog(@"%@", sub1);
NSString *sub2 = [myString substringFromIndex:4];
NSLog(@"%@", sub2);
```

Teilstrings aus Bereichen gewinnen

Mithilfe von Bereichen können Sie genau angeben, wo ein Teilstring anfangen und aufhören soll. Der folgende Code gibt `@"Tw"` zurück, was beim Zeichen mit dem Index 4 beginnt und zwei Zeichen lang ist. In `NSRange` können Sie einen Abschnitt innerhalb einer Zeichenfolge definieren. Solche Bereiche setzen Sie bei indizierenden Elementen wie Strings und Arrays ein.

```
NSRange r;
r.location = 4;
r.length = 2;
NSString *sub3 = [myString substringWithRange:r];
NSLog(@"%@", sub3);
```

Suchen und ersetzen in Strings

In Cocoa können Sie einen String auf einfache Weise nach einem Teilstring durchsuchen. Die Suche gibt einen Bereich zurück, der durch eine Anfangsposition und eine Länge gekennzeichnet ist. Überprüfen Sie stets die Anfangsposition! Die Position `NSNotFound` bedeutet, dass die Suche fehlgeschlagen ist. Der folgende Code gibt eine Position von 18 und eine Länge von 4 zurück:

```
NSRange searchRange = [myString rangeOfString:@"Five"];
if (searchRange.location != NSNotFound)
   NSLog(@"Range location: %d, length: %d", searchRange.location,
      searchRange.length);
```

Nachdem Sie einen Bereich gefunden haben, können Sie ihn durch einen neuen String ersetzen. Dabei muss der neue String nicht genauso lang sein wie der ursprüngliche, sodass der resultierende String länger oder kürzer sein kann als derjenige, von dem Sie ausgegangen sind.

```
NSLog(@"%@", [myString stringByReplacingCharactersInRange:
   searchRange withString: @"New String"]);
```

Mit einem allgemeineren Ansatz können Sie alle Vorkommen eines bestimmten Strings ersetzen. Das folgende Codefragment tauscht die einzelnen Leerzeichen durch ein Muster mit einem Sternchen aus, sodass sich `@"One * Two * Three * Four * Five * Six * Seven"` ergibt:

```
NSString *replaced = [myString stringByReplacingOccurrencesOfString:
@" " withString: @" * "];
NSLog(@"%@", replaced);
```

Groß- und Kleinschreibung ändern

Cocoa bietet drei einfache Methoden, um die Groß- und Kleinschreibung eines Strings zu ändern. Die drei folgenden Beispiele geben den String einmal vollständig in Großbuchstaben, einmal vollständig in Kleinbuchstaben sowie in der Form englischer Überschriften zurück, bei denen jedes Wort mit einem Großbuchstaben beginnt (»Hello World. How Do You Do?«). Da Cocoa Vergleiche ohne Berücksichtigung der Groß- und Kleinschreibung ermöglicht, werden Sie selten zwischen Groß- und Kleinschreibung umwandeln müssen, wenn Sie Strings miteinander vergleichen.

```
NSString *myString = @"Hello world. How do you do?";
NSLog(@"%@", [myString uppercaseString]);
NSLog(@"%@", [myString lowercaseString]);
NSLog(@"%@", [myString capitalizedString]);
```

Strings prüfen

Auf dem iPhone gibt es verschiedene Möglichkeiten, um Strings zu vergleichen und zu prüfen. Die drei einfachsten sind die Überprüfung auf Gleichheit zweier Strings, auf Übereinstimmung mit einem Stringpräfix (den Zeichen, mit denen der String beginnt) bzw. dem Suffix (den Zeichen, mit denen der String endet). Bei komplizierteren Vergleichen werden `NSComparisonResults`-Konstanten verwendet, um anzuzeigen, wie die einzelnen Elemente im Vergleich zueinander angeordnet sind.

```
NSString *s1 = @"Hello World";
NSString *s2 = @"Hello Mom";
NSLog(@"%@ %@ %@", s1, [s1 isEqualToString:s2] ?
   @"equals" : @"differs from", s2);
NSLog(@"%@ %@ %@", s1, [s1 hasPrefix:@"Hello"] ?
   @"starts with" : @"does not start with", @"Hello");
NSLog(@"%@ %@ %@", s1, [s1 hasSuffix:@"Hello"] ?
   @"ends with" : @"does not end with", @"Hello");
```

Zahlen aus Strings gewinnen

Sie können Strings mithilfe einer Wertmethode in Zahlen umwandeln. Die folgenden Beispiele geben 3, 1, 3,141592 und 3,141592 zurück:

```
NSString *s1 = @"3.141592";
NSLog(@"%d", [s1 intValue]):
NSLog(@"%d", [s1 boolValue]);
NSLog(@"%f", [s1 floatValue]);
NSLog(@"%f", [s1 doubleValue]);
```

Veränderbare Strings

`NSMutableString` ist eine Unterklasse von `NSString` und bietet Ihnen die Möglichkeit, mit Strings zu arbeiten, deren Inhalte geändert werden können. Nachdem ein String instanziiert ist, können Sie neue Inhalte an ihn anhängen, um so ein Ergebnis zusammenzubauen, bevor es von einer Methode zurückgegeben wird. Das folgende Beispiel zeigt »Hello World. The results are in now.« an.

```
NSMutableString *myString = [NSMutableString stringWithString:
   @"Hello World. "];
[myString appendFormat:@"The results are %@ now.", @"in"];
NSLog(@"%@", myString);
```

3.11.2 Zahlen und Datumsangaben

Foundation umfasst auch eine große Familie von Wertklassen, darunter auch solche für Zahlen und Datumsangaben. Anders als die Fließkommazahlen, Integer usw. von Standard-C sind all dies Objekte. Sie können zugewiesen und freigegeben und in Sammlungen wie Arrays, Dictionarys und Mengen verwendet werden. Die folgenden Beispiele zeigen Zahlen und Datumsangaben in Aktion und geben einen grundlegenden Überblick über diese Klassen.

Mit Zahlen arbeiten

Mit der Klasse `NSNumber` können Sie Zahlen als Objekte behandeln. Neue Instanzen von `NSNumber` erstellen Sie mit einer Reihe von Hilfsmethoden, z. B. mit `numberWithInt:`, `numberWithFloat:`, `numberWithBool:` usw. Wenn die Werte einmal gesetzt sind, können Sie sie mit `intValue`, `floatValue`, `boolValue` usw. abrufen und mit normalen mathematischen Operationen von C Berechnungen damit anstellen.

Sie sind nicht darauf beschränkt, ein Objekt mit demselben Datentyp abzurufen, mit dem es gesetzt wurde, sondern können z. B. einen Fließkommawert festlegen und daraus einen Integer abrufen. Zahlen lassen sich auch in Strings konvertieren.

```
NSNumber *number = [NSNumber numberWithFloat:3.141592];
NSLog(@"%d", [number intValue]);
NSLog(@"%@", [number stringValue]);
```

Einer der wichtigsten Gründe für die Verwendung von `NSNumber`-Objekten anstelle von `int`- und `float`-Werten usw. besteht darin, dass Sie sie in Cocoa-Routinen und -Klassen verwenden können. Beispielweise können Sie keinen Benutzerstandardwert (also eine Voreinstellung) auf den Integerwert 23 setzen, etwa für die Meldung: »Sie haben dieses Programm bereits 23-mal verwendet.« Es ist jedoch möglich, das Objekt `[NSNumber numberWithInt:23]` zu speichern und später den Integerwert aus diesem Objekt abzurufen, um daraus eine Meldung für den Benutzer zu machen.

> **Hinweis**
>
> Die Klasse `NSDecimalNumber` stellt einen praktischen objektorientierten Wrapper für die Arithmetik von Dezimalzahlen dar.

Arbeiten mit Datumsangaben

Wie im standardmäßigen C und bei `time()` wird auch in `NSDate`-Objekten die Anzahl der Sekunden seit einer Epoche, also einer genormten globalen Zeitreferenz, zur Darstellung des aktuellen Datums verwendet. Die iPhone-Epoche begann um Mitternacht am 1. Januar 2001, die Unix-Standardepoche begann um Mitternacht am 1. Januar 1970.

Jedes `NSTimeInterval` stellt eine Zeitspanne in Sekunden dar, die mit einer Fließkommagenauigkeit im Bereich von Sekundenbruchteilen gespeichert werden. Der folgende Code zeigt, wie Sie ein neues Datumsobjekt mit der aktuellen Zeit erstellen und wie Sie mithilfe eines Intervalls auf einen Zeitpunkt in der Zukunft (oder Vergangenheit) verweisen:

```
// Aktueller Zeitpunkt
NSDate *date = [NSDate date];

// Zeitpunkt zehn Sekunden in der Zukunft
date = [NSDate dateWithTimeIntervalSinceNow:10.0f];
```

Datumsangaben vergleichen Sie, indem Sie das Zeitintervall dazwischen festlegen bzw. überprüfen. Das folgende Codefragment zwingt die Anwendung dazu, fünf Sekunden lang zu ruhen, und vergleicht dann das aktuelle Datum mit dem in `date` gespeicherten:

```
// Ruht fünf Sekunden lang und prüft dann das Zeitintervall
[NSThread sleepUntilDate:[NSDate dateWithTimeIntervalSinceNow:5.0f]];
NSLog(@"Slept %f seconds", [[NSDate date] timeIntervalSinceDate:date]);
```

Die standardmäßige Beschreibungsmethode für Datumsangaben gibt einen für Menschen lesbaren String zurück, der das aktuelle Datum und die Uhrzeit zeigt:

```
// Zeigt das Datum
NSLog(@"%@", [date description]);
```

Um Datumsangaben in rundum formatierte Strings zu verwandeln, anstatt einfach die Standardbeschreibung zu verwenden, setzen Sie eine Instanz von `NSDateFormatter` ein. Sie geben das Format (z. B. `YY` für die Jahresangabe mit zwei Stellen und `YYYY` für die mit vier Stellen) über die Eigenschaft `dateFormat` des Objekts an. Eine vollständige Liste der Formatspezifizierer finden Sie in der mitgelieferten Xcode-Dokumentation. Mit dieser Klasse können Sie nicht nur für eine formatierte Ausgabe sorgen, sondern auch vorformatierte Datumsausgaben aus Strings lesen. Dies überlassen wir Ihnen als Übungsaufgabe.

```
// Gibt einen formatierten String aus, der für das aktuelle Datum steht
NSDateFormatter *formatter = [[[NSDateFormatter alloc] init]
   autorelease];
formatter.dateFormat = @"MM/dd/YY HH:mm:ss";
NSString *timestamp = [formatter stringFromDate:[NSDate date]];
NSLog(@"%@", timestamp);
```

Timer

Manchmal ist es notwendig, dafür zu sorgen, dass eine Aktion zu einem bestimmten Zeitpunkt in der Zukunft ausgeführt wird. Cocoa bietet einen einfach zu verwendenden Timer, der nach dem von Ihnen angegebenen Zeitintervall ausgelöst wird, nämlich die Klasse `NSTimer`. Der im Folgenden gezeigte Timer wird nach einer Sekunde wiederholt ausgelöst, bis er deaktiviert wird:

```
[NSTimer scheduledTimerWithTimeInterval: 1.0f target: self selector:
@selector(handleTimer:) userInfo: nil repeats: YES];
```

Jedes Mal, wenn der Timer ausgelöst wird, ruft er sein Ziel auf und sendet ihm die Selektornachricht, mit der er initialisiert wurde. Die Callback-Methode nimmt ein Argument entgegen (beachten Sie den einzelnen Doppelpunkt), nämlich den Timer selbst. Um einen Timer zu deaktivieren, senden Sie ihm die Nachricht `invalidate`. Dadurch wird das Timerobjekt freigegeben und aus der aktuellen Ausführungsschleife entfernt.

```
- (void) handleTimer: (NSTimer *) timer
{
   printf("Timer count: %d\n", count++);
   if (count > 3)
   {
      [timer invalidate];
      printf("Timer disabled\n");
   }
}
```

Informationen über Indexpfade abrufen

Die Klasse `NSIndexPath` wird bei iPhone-Tabellen verwendet. In ihr sind die Abschnitts- und die Zeilennummer einer Benutzerauswahl gespeichert, also die Stelle, auf die ein Benutzer auf der Tabelle tippt. Wenn diese Zahlen mit Indexpfaden versehen sind, können Sie sie mit den Eigenschaften `myIndexPath.row` und `myIndexPath.section` abrufen. Weitere Informationen über diese Klasse und ihre Verwendung erhalten Sie in Kapitel 11, *Tabellenansichten erstellen und verwalten*.

3.11.3 Sammlungen

Auf dem iPhone werden hauptsächlich drei Arten von Sammlungen verwendet: Arrays, Dictionarys und Mengen (Sets). Arrays verhalten sich wie C-Arrays. Sie bestehen aus einer indizierten Liste von Objekten, die Sie abrufen können, indem Sie angeben, bei welchem Index nachgeschlagen werden soll. In Dictionarys sind Werte dagegen so gespeichert, dass Sie sie anhand von Schlüsseln nachschlagen können. Beispielsweise können Sie das Alter von Personen in einem Dictionary ablegen, wobei das Alter des Vaters das `NSNumber`-Objekt 57 und das Alter des Kindes das Objekt 15 ist. Mengen sind ungeordnete Gruppen von Objekten und werden auf dem iPhone vor allem in Verbindung mit dem Lesen von Benutzerberührungen vom Bildschirm eingesetzt. Von jeder dieser Klassen gibt es wie bei `NSString` eine normale und eine veränderbare Version.

Arrays erstellen und darauf zugreifen

Arrays erstellen Sie mit der Hilfsmethode `arrayWithObjects:`, die ein zur automatischen Freigabe gekennzeichnetes Array zurückgibt. Wenn Sie diese Methode aufrufen, führen Sie alle Objekte auf, die Sie zu dem Array hinzufügen möchten, und schließen die Liste mit `nil` ab. (Wenn Sie auf das `nil` verzichten, stürzt die Anwendung zur Laufzeit ab.) Einem Array können Sie jede beliebige Art von Objekt hinzufügen, auch andere Arrays und Dictionarys. Das folgende Beispiel zeigt, wie Sie ein Array mit drei Elementen erstellen:

```
NSArray *array = [NSArray arrayWithObjects:@"One", @"Two", @"Three", nil];
```

Die Eigenschaft `count` gibt die Anzahl der Objekte im Array an. Arrays werden beginnend bei 0 indiziert, sodass der höchste Indexwert um 1 kleiner ist als der Wert von `count`. Wenn Sie versuchen, auf `[array objectAtIndex:array.count]` zuzugreifen, führt dies zu einer Ausnahme aufgrund einer Indexüberschreitung und zum Absturz. Seien Sie beim Abrufen von Objekten also stets vorsichtig, und achten Sie darauf, weder die obere noch die untere Grenze für das Array zu überschreiten.

```
NSLog(@"%d", array.count);
NSLog(@"%@", [array objectAtIndex:0]);
```

Die veränderbare Variante von `NSArray` heißt `NSMutableArray`. Veränderbare Arrays lassen sich bearbeiten, sodass Sie nach Belieben Objekte hinzufügen oder daraus entfernen können. Der folgende Code kopiert das vorstehende Array in ein veränderbares und bearbeitet Letzteres, indem er ein Objekt hinzufügt und ein anderes entfernt. Dadurch ergibt sich das Array `[@"One", @"Two", @"Four"]`:

```
NSMutableArray *marray = [NSMutableArray arrayWithArray:array];
[marray addObject:@"Four"];
[marray removeObjectAtIndex:2];
NSLog(@"%@", marray);
```

Sowohl veränderbare als auch nicht veränderbare Arrays können Sie stets zu einem neuen kombinieren, das alle Elemente der einzelnen Arrays enthält. Hierbei wird nicht geprüft, ob dabei Duplikate auftreten. Der folgende Code erstellt ein Array aus sechs Elementen, das die Zahlwörter *one*, *two* und *three* des ursprünglichen Arrays und *one*, *two* und *four* des veränderbaren Arrays enthält:

```
NSLog(@"%@", [array arrayByAddingObjectsFromArray:marray]);
```

Arrays prüfen

Sie können prüfen, ob ein Array ein bestimmtes Objekt enthält, und den Index eines gegebenen Objekts abrufen. Der folgende Code sucht nach dem ersten Vorkommen von »Four« und gibt den Index des betreffenden Objekts zurück. Der Test in der `if`-Anweisung stellt sicher, dass es mindestens eine solche Fundstelle gibt.

```
if ([marray containsObject:@"Four"])
   NSLog(@"The index is %d",
      [marray indexOfObject:@"Four"]);
```

Arrays in Strings umwandeln

Wie zu anderen Objekten können Sie auch zu einem Array die Nachricht `description` senden, woraufhin ein `NSString` mit einer Beschreibung des Arrays zurückgegeben wird. Außerdem können Sie ein `NSArray` mit `componentsJoinedByString` in einen String umwandeln. Der folgende Code gibt `@"One Two Three"` zurück.

```
NSArray *array = [NSArray arrayWithObjects:@"One", @"Two", @"Three", nil];
NSLog(@"%@", [array componentsJoinedByString:@" "]);
```

Dictionarys erstellen und darauf zugreifen

In `NSDictionary`-Objekten werden Schlüssel und Werte gespeichert, sodass Sie die Objekte mithilfe von Strings nachschlagen können. Die veränderbare Version von Dictionarys, `NSMutableDictionary`, erlaubt es Ihnen, diese Dictionarys zu bearbeiten, indem Sie bei Bedarf Elemente hinzufügen und entfernen. Bei der iPhone-Programmierung verwenden Sie die veränderbare Klasse häufiger als die statische, weshalb in diesen Beispielen die veränderbaren Versionen vorgeführt werden.

Dictionarys erstellen

Mit der Hilfsmethode `dictionary` erstellen Sie ein neues veränderbares Dictionary, wie Sie im Folgenden sehen. Dadurch wird ein neu initialisiertes Dictionary zurückgegeben, das Sie bearbeiten können. Sie füllen es mit `setObject: forKey:`.

```
NSMutableDictionary *dict = [NSMutableDictionary dictionary];
[dict setObject:@"1" forKey:@"A"];
[dict setObject:@"2" forKey:@"B"];
[dict setObject:@"3" forKey:@"C"];
NSLog(@"%@", [dict description]);
```

Dictionarys durchsuchen

Ein Dictionary zu durchsuchen bedeutet, es nach dem Schlüsselnamen abzufragen. Sie verwenden `objectForKey:`, um das Objekt zu finden, das zu dem gegebenen Schlüssel gehört. Wird der Schlüssel nicht gefunden, gibt das Dictionary `nil` zurück.

```
NSLog(@"%@", [dict objectForKey:@"A"]);
NSLog(@"%@", [dict objectForKey:@"F"]);
```

Objekte ersetzen

Wenn Sie für einen bereits belegten Schlüssel ein neues Objekt festlegen, ersetzt Cocoa das ursprüngliche Objekt im Dictionary. Der folgende Code ersetzt den Wert `3` für den Schlüssel `C` durch `foo`:

```
[dict setObject:@"foo" forKey:@"C"];
NSLog(@"%@", [dict objectForKey:@"C"]);
```

Objekte entfernen

Sie können Objekte auch aus Dictionarys entfernen. Mit dem folgenden Code löschen Sie das Objekt, das zum Schlüssel `B` gehört. Anschließend sind sowohl der Schlüssel als auch das Objekt nicht mehr im Dictionary vorhanden.

```
[dict removeObjectForKey:@"B"];
```

Schlüssel auflisten

Dictionarys können die Anzahl der in ihnen gespeicherten Einträge melden und ein Array aller zurzeit verwendeten Schlüssel ausgeben. Anhand dieser Schlüsselliste können Sie ablesen, welche Schlüssel schon in Gebrauch sind. Bevor Sie ein Element zum Dictionary hinzufügen, können Sie einen Vergleich mit dieser Liste durchführen, damit Sie kein vorhandenes Schlüssel/Wert-Paar überschreiben.

```
NSLog(@"The dictionary has %d objects", [dict count]);
NSLog(@"%@", [dict allKeys]);
```

Zugriff auf Mengenobjekte

In Mengen (Sets) sind ungeordnete Objektsammlungen gespeichert. Mengen begegnen Ihnen fast ausschließlich bei der Arbeit mit dem Multitouch-Bildschirm des iPhones. Die Klasse `UIView` empfängt Aktualisierungen von Fingerbewegungen, die Berührungen als `NSSet` ausgeben. Um solche Berührungen zu handhaben, müssen Sie fast immer `allObjects` verwenden und mit dem daraufhin zurückgegebenen Array arbeiten. Nach dieser Konvertierung können Sie normale Arrayaufrufe verwenden, um die Berührungen aufzulisten, abzufragen und zu durchlaufen.

Speicherverwaltung bei Sammlungen

Arrays, Mengen und Dictionarys behalten Objekte, die zu ihnen hinzugefügt werden, automatisch bei, und geben solche frei, die aus ihnen entfernt werden. Freigabenachrichten werden auch gesendet, wenn die Zuweisung der Sammlung aufgehoben wird. Sammlungen kopieren keine Objekte, sondern verlassen sich auf die Beibehaltungszähler, um die Objekte festzuhalten und nach Bedarf zu verwenden.

Sammlungen in eine Datei schreiben

Sowohl Arrays als auch Dictionarys können mit der Methode `writeToFile: atomically:` in Dateien gespeichert werden, sofern die Typen innerhalb der Sammlung `NSData`, `NSDate`, `NSNumber`, `NSString`, `NSArray` oder `NSDictionary` sind. Als erstes Argument übergeben Sie den Pfad, als zweites einen booleschen Wert. Wie beim Speichern von Strings bestimmt das zweite Argument, ob die Datei zuerst als temporäre Hilfsdatei gespeichert und dann umbenannt werden soll. Die Methode gibt den booleschen Wert `YES` zurück, wenn die Datei gespeichert wurde, und anderenfalls `NO`. Zum Speichern von Arrays und Dictionarys werden standardmäßige Eigenschaftslistendateien erstellt.

```
NSString *path = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/ArraySample.txt"];
if ([array writeToFile:path atomically:YES])
   NSLog(@"File was written successfully");
```

Um ein Array oder ein Dictionary aus einer Datei abzurufen, verwenden Sie die Hilfsmethode `arrayWithContentsOfFile:` bzw. `dictionaryWithContentsOfFile:`. Gibt die Methode `nil` zurück, konnte die Datei nicht gelesen werden.

```
NSArray *newArray = [NSArray arrayWithContentsOfFile:path];
NSLog(@"%@", newArray);
```

URLs konstruieren

`NSURL`-Objekte zeigen auf Ressourcen, bei denen es sich um lokale Dateien, aber auch um URLs im Web handeln kann. Um URL-Objekte zu erstellen, übergeben Sie einen String an eine Klassen-Hilfsfunktion, wobei es für die einzelnen Arten von URLs unterschiedliche Funktionen gibt. Ist ein `NSURL`-Objekt aber einmal erstellt, kann es nicht mehr verändert werden. Cocoa kümmert sich nicht darum, ob die Ressource lokal ist oder auf ein Objekt zeigt, das nur im Internet existiert. Der folgende Code zeigt, wie Sie URLs für die beiden Typen – lokaler Pfad und Web – erstellen:

```
NSString *path = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/foo.txt"];
NSURL *url1 = [NSURL fileURLWithPath:path];
NSLog(@"%@", url1);

NSString *urlpath = @"http://ericasadun.com";
NSURL *url2 = [NSURL URLWithString:urlpath];
NSLog(@"%d characters read",
   [[NSString stringWithContentsOfURL:url2] length]);
```

Mit NSData arbeiten

Wenn `NSString`-Objekte die Entsprechung der mit o abgeschlossenen C-Strings sind, dann sind `NSData` mit Puffern zu vergleichen. `NSData` bietet Datenobjekte zum Speichern und Handhaben von Bytes. Häufig müssen Sie `NSData` mit dem Inhalt einer Datei eines URLs füllen. Die zurückgegebenen Daten können die Länge angeben, damit Sie wissen, wie viele Bytes abgerufen wurden. Das folgende Codefragment ruft den Inhalt eines URLs ab und gibt die Anzahl der gelesenen Bytes aus:

```
NSData *data = [NSData dataWithContentsOfURL:url2];
NSLog(@"%d", [data length]);
```

Um auf den grundlegenden Bytepuffer eines `NSData`-Objekts zuzugreifen, verwenden Sie `bytes`. Dadurch wird ein `(const void *)`-Zeiger auf die eigentlichen Daten zurückgegeben.

Wie bei anderen Cocoa-Objekten können Sie einerseits die standardmäßige Version der Klasse `NSData` verwenden, andererseits aber auch die veränderbare Kindklasse `NSMutableData`. Die meisten Cocoa-Programme, die auf das Web zugreifen, vor allem diejenigen, die asynchrone Downloads durchführen, »ziehen« immer nur wenige Daten auf einmal. Für diese Fälle sind `NSMutableData`-Objekte sinnvoll. Sie können veränderbare Daten langsam anwachsen lassen, indem Sie `appendData:` verwenden, um neue Informationen anzuhängen, sobald sie empfangen worden sind.

Dateiverwaltung

Der Dateimanager für das iPhone ist ein Singleton, das von der Klasse `NSFileManager` bereitgestellt wird. Es kann den Inhalt von Ordnern auflisten, um zu bestimmen, welche Dateien vorhanden sind, und grundlegende Aufgaben des Dateisystems durchführen. Der folgende Code ruft eine Dateiliste für zwei Ordner ab. Zuerst schaut er in den `Documents`-Ordner der Sandbox und dann in das Anwendungsbundle selbst.

```
NSFileManager *fm = [NSFileManager defaultManager];

// Führt die Dateien im Documents-Ordner der Sandbox auf
NSString *path = [NSHomeDirectory() stringByAppendingPathComponent:@"
   Documents"];
NSLog(@"%@",[fm directoryContentsAtPath:path]);


// Führt die Dateien im Anwendungsbundle auf
path = [[NSBundle mainBundle] bundlePath];
NSLog(@"%@",[fm directoryContentsAtPath:path]);
```

Beachten Sie hier die Verwendung von `NSBundle`. Dadurch können Sie das Anwendungsbundle finden und dessen Pfad dem Dateimanager übergeben. Außerdem können Sie `NSBundle` verwenden, um den Pfad für ein Element abzurufen, das in dem Bundle enthalten ist. (Sie können jedoch niemals in das Anwendungsbundle schreiben.) Der folgende Code gibt den Pfad zum Bild `Default.png` der Anwendung zurück. Beachten Sie, dass der Dateiname und die Erweiterung jeweils für sich stehen und dass bei beiden auf die Groß- und Kleinschreibung geachtet wird.

```
NSBundle *mb = [NSBundle mainBundle];
NSLog(@"%@", [mb pathForResource:@"Default" ofType:@"png"]);
```

Der Dateimanager bietet die ganze Palette dateispezifischer Aufgaben. Er kann Dateien verschieben, kopieren und entfernen sowie das System nach Merkmalen und dem Besitzer von Dateien abfragen. Die folgenden Beispiele zeigen einige der einfacheren Routinen, die Sie in Ihren Anwendungen einsetzen können:

```
// Eine Datei erstellen
NSString *docspath = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents"];
NSString *filepath = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/testfile"];
```

```
NSArray *array = [@"One Two Three" componentsSeparatedByString:@" "];
[array writeToFile:filepath atomically:YES];
NSLog(@"%@", [fm directoryContentsAtPath:docspath]);


// Eine Datei kopieren
NSString *copypath = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/copied"];
if (![fm copyItemAtPath:filepath toPath:copypath error:&error])
{
   NSLog(@"Copy Error: %@", [error localizedDescription]);
   return;
}
NSLog(@"%@", [fm directoryContentsAtPath:docspath]);


// Eine Datei verschieben
NSString *newpath = [NSHomeDirectory()
stringByAppendingPathComponent:@"Documents/renamed"];
if (![fm moveItemAtPath:filepath toPath:newpath error:&error])
{
   NSLog(@"Move Error: %@", [error localizedDescription]);
   return;
}
NSLog(@"%@", [fm directoryContentsAtPath:docspath]);


// Eine Datei entfernen
if (![fm removeItemAtPath:copypath error:&error])
{
   NSLog(@"Remove Error: %@", [error localizedDescription]);
   return;
}
NSLog(@"%@", [fm directoryContentsAtPath:docspath]);
```

Hinweis

Ein weiterer bequemer Kniff für den Umgang mit Dateien besteht darin, in Pfadnamen eine Tilde zu verwenden, z. B. `"~/Library/Preferences/foo.plist"`, und die `NSString`-Methode `stringByExpandingTildeInPath` anzuwenden.

3.12 Zum guten Schluss: Nachrichtenweiterleitung

Objective-C bietet zwar keine echte mehrfache Vererbung, aber eine Notlösung, damit Objekte auf Nachrichten reagieren können, die in anderen Klassen implementiert sind. Wenn ein Objekt auf die Nachrichten einer anderen Klasse reagieren können soll, können Sie in Ihrer Anwendung die Nachrichtenweiterleitung verwenden, um Zugriff auf die Methoden des Objekts zu erhalten.

Normalerweise führt es zu einem Laufzeitfehler, wenn Sie eine nicht erkannte Nachricht senden, sodass die Anwendung abstürzt. Doch vor dem Absturz gibt das Laufzeitsystem des iPhones jedem Objekt eine zweite Chance, um die Nachricht zu handhaben. Wenn Sie die Nachricht abfangen, können Sie sie zu dem Objekt umleiten, das sie versteht und darauf reagieren kann.

Kehren wir zu der Beispielklasse `Car` zurück, die wir in diesem Kapitel behandelt haben. Die auf halbem Wege eingeführte Eigenschaft `carInfo` gibt einen String zurück, der den Hersteller, das Modell und das Baujahr des Autos beschreibt. Nehmen wir nun an, dass eine Instanz von `Car` auf `NSString`-Nachrichten antworten kann, indem sie sie an diese Eigenschaft weiterleitet. Wenn Sie dann `length` an ein `Car`-Objekt senden, stürzt die Anwendung nicht ab – stattdessen gibt das Objekt die Länge des `carInfo`-Strings zurück. Senden Sie `stringByAppendingString:`, so hängt das Objekt den String an den Eigenschaftsstring an. Dies wäre so, als hätte die Klasse `Car` das gesamte Stringverhalten geerbt (oder zumindest geliehen).

Objective-C ermöglicht ein solches Verhalten durch die sogenannte *Nachrichtenweiterleitung*. Wenn Sie eine Nachricht an ein Objekt senden, das mit dem Selektor nicht umgehen kann, wird dieser Selektor an die Methode `forwardInvocation:` weitergeleitet. Das mit dieser Nachricht gesendete Objekt, eine `NSInvocation`-Instanz, speichert den ursprünglichen Selektor und die anforderten Argumente. Sie können `forwardInvocation:` überschreiben und die Nachricht an ein anderes Objekt senden.

3.12.1 Die Nachrichtenweiterleitung umsetzen

Um die Nachrichtenweiterleitung in einem eigenen Programm umzusetzen, müssen Sie zwei Methoden überschreiben, nämlich `methodSignatureForSelector:` und `forwardInvocation:`. Die erste dieser beiden Methoden erstellt eine gültige Methodensignatur für Nachrichten, die von einer anderen Klasse implementiert werden, die zweite leitet den Selektor an ein Objekt weiter, das die Nachricht tatsächlich implementiert.

Eine Methodensignatur erstellen

Die erste Methode gibt eine Methodensignatur für den angeforderten Selektor zurück. In unserem Beispiel kann eine Instanz von `Car` keine korrekte Signatur für einen Selektor erstellen, der von einer anderen Klasse implementiert wird, in diesem Fall von `NSString`. Wenn Sie eine Überprüfung auf eine nicht wohlgeformte Signatur hinzufügen (also auf die Rückgabe von `nil`), geben Sie der

Methode die Gelegenheit, die einzelnen Möglichkeiten der Pseudovererbung durchzugehen und zu versuchen, ein gültiges Ergebnis hervorzurufen. Im folgenden Beispiel werden Methoden nur einer einzigen anderen Klasse über `self.carInfo` herangezogen:

```
- (NSMethodSignature*) methodSignatureForSelector:(SEL)selector
{
   // Prüft nach, ob Car die Nachricht handhaben kann
   NSMethodSignature* signature = [super
      methodSignatureForSelector:selector];
   // Wenn nicht, wird geprüft, ob der String carInfo etwas mit der
   // Nachricht anfangen kann
   if (!signature)
      signature = [self.carInfo methodSignatureForSelector:selector];

   return signature;
}
```

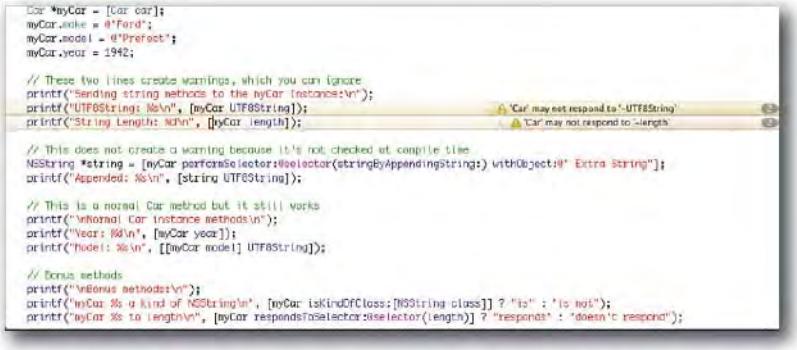
Weiterleiten

Die zweite Methode, die Sie überschreiben müssen, ist `forwardInvocation`. Sie wird nur dann aufgerufen, wenn ein Objekt nicht in der Lage war, mit einer Nachricht umzugehen. Diese Methode gibt dem Objekt eine zweite Chance, indem sie ihm erlaubt, die Nachricht weiterzuleiten. Sie prüft, ob der String `self.carInfo` auf den Selektor reagiert. Wenn ja, weist sie das aufrufende Objekt an, sich selbst mit dem Stringobjekt als Empfänger aufzurufen.

```
- (void)forwardInvocation:(NSInvocation *)invocation
{
   SEL selector = [invocation selector];

   if ([self.carInfo respondsToSelector:selector])
   {
      printf("[forwarding from %s to %s] ", [[[self class] description]
         UTF8String], [[NSString description] UTF8String]);
      [invocation invokeWithTarget:self.carInfo];
   }
}
```

Weitergeleitete Nachrichten verwenden

Der Aufruf von Nicht-Klassenmethoden wie `UTF8String` oder `length` führt zu Compiler-Warnungen, die Sie aber ignorieren können. Der in *Abbildung 3.4* gezeigte Code ruft zwei solche Warnungen hervor, wird aber kompiliert und (was schließlich wichtiger ist) fehlerfrei ausgeführt. Wie Sie in der Abbildung sehen, können Sie einer `Car`-Instanz sowohl Methoden senden, die von der Klasse selbst definiert sind, als auch solche, die `NSString` implementiert.



► *Abbildung 3.4: Der Compiler gibt Warnungen über weitergeleitete Methoden aus, der Code läuft aber fehlerfrei.*

Aufräumen

Die Aufrufweiterleitung ahmt zwar die mehrfache Vererbung nach, doch `NSObject` setzt diese beiden Verfahren nicht gleich. Methoden wie `respondsToSelector:` und `isKindOfClass:` überprüfen nur die Vererbungshierarchie und kümmern sich nicht um die Weiterleitung.

Es gibt einige optionale Methoden, mit denen eine Klasse ihre Nachrichtenkonformität mit einer anderen Klasse besser ausdrücken kann. Wenn Sie `respondsToSelector:` und `isKindOfClass:` neu implementieren, können andere Klassen Ihre Klasse abfragen. Im Gegenzug gibt die Klasse bekannt, dass sie nicht nur auf ihre eigenen Methoden, sondern auch auf alle Stringmethoden reagiert und »eine Art von« String (»kind of«) ist. Dadurch wird der Ansatz der Pseudovererbung noch erweitert.

```
// Erweitert Selektorkonformität
- (BOOL)respondsToSelector:(SEL)aSelector
{
    // Die Klasse Car kann die Nachricht handhaben
    if ( [super respondsToSelector:aSelector] )
        return YES;

    // Der String carInfo kann die Nachricht handhaben
    if ([self.carInfo respondsToSelector:aSelector])
        return YES;

    // Anderenfalls...
    return NO;
}
```

```
// Erlaubt das Auftreten als Klasse
- (BOOL)isKindOfClass:(Class)aClass
{
   // Überprüft Car
   if (aClass == [Car class]) return YES;
   if ([super isKindOfClass:aClass]) return YES;

   // Überprüft NSString
   if ([self.carInfo isKindOfClass:aClass]) return YES;

   return NO;
}
```

Weiterleitung kinderleicht

Das Methodenpaar aus `methodSignatureForSelector:` und `forwardInvocation:` bildet eine bewährte Möglichkeit, um die Weiterleitung für Ihre Klassen einzusetzen. Auf dem iPhone ist auch ein einfacherer, aber weniger gut dokumentierter Ansatz möglich, den Sie auf eigene Gefahr verwenden können. Ersetzen Sie dazu die beiden Methoden durch eine einzige, die die gesamte Arbeit mit weniger Programmieraufwand verrichtet. Sie lässt sich bequem einsetzen, doch wenn Sie sich die Xcode-Dokumentation ansehen, werden Sie feststellen, dass sie auf dem iPhone offiziell nicht unterstützt wird.

```
- (id)forwardingTargetForSelector:(SEL)sel
{
   if ([self.carInfo respondsToSelector:sel]) return self.carInfo;
   return nil;
}
```

3.13 Zusammenfassung

In diesem Kapitel haben Sie eine abgekürzte, aber sehr informative Einführung in Objective-C und Foundation erhalten. Sie haben gelesen, wie Objective-C die Sprache C erweitert und Unterstützung für die objektorientierte Programmierung bietet. Sie haben Eigenschaften und die Speicherverwaltung kennengelernt und eine Übersicht über die wichtigsten Foundation-Klassen erhalten. Was können Sie nun aus diesem Kapitel mitnehmen? Merken Sie sich die folgenden abschließenden Gedanken:

- Der Beispielcode zu diesem Kapitel enthält alle Beispiele, die in dieser Einführung gegeben wurden. Versuchen Sie, diesen Code direkt in Xcode auszuprobieren. Spielen Sie damit herum, ergänzen Sie Ihre eigenen Codebeispiele, oder erweitern Sie die vorgestellten. Praktische Übungen sind die beste Möglichkeit, um wichtige Fertigkeiten für die iPhone-Entwicklung zu erlangen.

- Um Objective-C und Cocoa zu lernen, bedarf es mehr als nur der Lektüre eines Kapitels. Wenn Sie die Programmierung für das iPhone ernsthaft lernen wollen und Ihnen die hier vorgestellten Prinzipien neu sind, sollten Sie zu einem Buch greifen, das sich auf die Einführung dieser Technologien für Entwickler konzentriert, für die diese Plattform neu ist. Empfehlenswert sind beispielsweise *Cocoa: Programmierung für Mac OS X* von Aaron Hillegass, *Objective-C 2.0: Anwendungen entwickeln für Mac und iPhone* von Stephen Kochan und *Xcode 3* von Fritz Anderson.
- In diesem Kapitel wurden die Frameworks *Core Foundation* und *Carbon* erwähnt. Diese Technologien wurden aber nicht tiefschürfend behandelt. Allerdings werden Sie auf dem iPhone irgendwann APIs auf der Grundlage von C begegnen, vor allem dann, wenn Sie mit dem Adressbuch, mit Quartz-2D-Grafiken, mit Core Audio und anderen Frameworks arbeiten. Alle diese Bereiche sind auf der Entwicklerwebsite von Apple ausführlich und mit Codebeispielen dokumentiert. Eine sicherere Grundlage in der Programmierung mit C (und vielleicht auch C++) hilft Ihnen, mit den Einzelheiten der Implementierung fertigzuwerden.

4 Benutzeroberflächen entwerfen

Das iPhone SDK hilft Ihnen bei der Gestaltung von Benutzeroberflächen. In diesem Kapitel werden die Klassen für das optische Erscheinungsbild eingeführt und deren Rolle für den Entwurf der Oberfläche erläutert. Sie erfahren hier etwas über die Controller, die mit diesen Klassen umgehen, und erfahren, wie Sie Aufgaben wie die Reaktion auf eine Drehung des Geräts erledigen. Anschließend geht es um Lösungen für das Layout und die Anpassung von Oberflächen. Sie lernen kombinierte Lösungen kennen, bei denen die Oberfläche teilweise mit Interface Builder und teilweise mit Objective-C-Code erstellt wird. Beim Durcharbeiten dieses Kapitels lernen Sie viele Möglichkeiten kennen, die Sie beim Entwurf eigener Anwendungen einsetzen können.

4.1 UIView und UIWindow

Fast alles, was auf dem Bildschirm des iPhones erscheint, ist ein Kindelement der Klasse `UIView`. Ansichten sind so etwas wie eine Leinwand, auf der Sie mit Farben malen, Bilder und Schaltflächen platzieren. Diese Elemente können Sie auf dem Bildschirm verschieben, ihre Größe ändern und sie übereinanderstapeln. Ansichten sind die grundlegenden Bestandteile von Benutzeroberflächen.

Die Regel für das iPhone lautet: ein Fenster, viele Ansichten. Wenn Sie das im Hinterkopf behalten, wird das Entwerfen von Benutzerschnittstellen einfacher. Bildlich gesprochen, ist `UIWindow` das Fernsehgerät, während die `UIViews` die Schauspieler Ihrer Lieblingsserie sind. Sie können sich auf dem Bildschirm bewegen, erscheinen, verschwinden sowie ihr Aussehen und ihr Verhalten mit der Zeit ändern.

Im Gegensatz dazu steht das Fernsehgerät normalerweise still. Die Größe des Bildschirms ändert sich nicht, auch wenn die virtuelle Welt, die Sie durch ihn sehen, praktisch unbegrenzt ist. Sie können im selben Haushalt sogar mehrere Fernsehgeräte besitzen (so wie Sie in derselben Anwendung mehrere Instanzen von `UIWindow` erstellen können), doch immer nur eins gleichzeitig anschauen.

`UIViews` sind die Bausteine für die grafische Benutzeroberfläche. Sie stellen die grafischen Elemente bereit, die auf dem Bildschirm angezeigt werden und die Benutzerinteraktion ermöglichen. Jede Benutzerschnittstelle des iPhones besteht aus `UIViews`, die in einem `UIWindow` angezeigt werden, das selbst eine besondere Art von `UIView` ist. Das Fenster fungiert als Container und ist die Wurzel der Anzeigehierarchie. Es enthält alle sichtbaren Anwendungskomponenten.

Neben `UIView` und `UIWindow` gibt es noch eine Vielzahl an spezialisierten Ansichten, z. B. `UIImageView` und `UITextView`, mit denen Sie Schnittstellen aus vorgefertigten Teilen aufbauen können. In diesem Abschnitt erhalten Sie einen Überblick über diese Ansichten. Sie finden sie in der Bibliothek des Interface Builder und können sie in Ihre Anwendungsschnittstellen aufnehmen, um eine eigene grafische Benutzeroberfläche zusammenzustellen.

Hinweis

Das Kürzel `UI`, mit dem die Namen mancher Klassen beginnen (z. B. `UIView`) steht für *User Interface*, also *Benutzerschnittstelle*.

4.1.1 Ansichten zur Anzeige von Daten

Eine der wichtigsten Aufgaben einer Ansicht ist die optische Darstellung von Daten. In Cocoa Touch dienen die folgenden Klassen zur Anzeige von Informationen auf dem Bildschirm:

- Die Klasse `UITextView` zeigt dem Benutzer Textpassagen an und ermöglicht ihm, eigene Texte über die Tastatur einzugeben. Dabei müssen Sie festlegen, ob der angezeigte Text bearbeitet werden kann. In einer Textansicht wird durchgängig eine Schriftart mit einer einzigen Textgröße verwendet.
- Instanzen von `UILabel` zeigen kurze, schreibgeschützte Textansichten an. Wie der Name (*label*, also *Beschriftung*) angibt, wird diese Klasse verwendet, um Elemente auf dem Bildschirm statisch zu beschriften. Farbe, Schriftart und Schriftschnitt der Beschriftung legen Sie mithilfe der Eigenschaften der Ansicht fest. Die Wörter »Fahrenheit« und »Celsius« in *Abbildung 4.8* werden über `UILabel` angezeigt.
- `UIImageViews` zeigen Bilder an. In diesen Ansichtstyp laden Sie `UIImage`-Objekte, die die Instanzen einer abstrakten Klasse zum Speichern von Bildern darstellen. Nachdem die Ansicht geladen ist, legen Sie ihre Position und Größe fest, wobei `UIImageView` automatisch ihre Inhalte an diese Bedingungen anpasst. Ein besonderes Merkmal dieser Klasse besteht darin, dass Sie nicht nur ein einzelnes Bild laden können, sondern auch eine Serie von Bildern, um eine Animation zu gestalten.

- Wenn Sie HTML-, PDF- oder anderen Webinhalt anzeigen möchten, bietet die Klasse `UIWebView` alles, was Sie dazu benötigen. Instanzen von `UIWebView` bieten eine reichhaltige Quelle von Anzeigemöglichkeiten, mit denen Sie nahezu alle vom eingebauten Safari-Browser unterstützten Datentypen anzeigen können. Diese Ansichten ermöglichen eine einfache Suche im Web mit integriertem Verlauf, im Grunde also ein vorgefertigtes Safari-ähnliches Objekt, das Sie in Ihre Programme einfügen können. Manchmal verwenden Entwickler `UIWebView`-Instanzen auch zur Anzeige von formatierten Textblöcken. Als Dreingabe sind bei diesen Instanzen auch das Zoomen und Blättern ohne zusätzlichen Programmieraufwand möglich.
- `MKMapViews` (`MK` steht für *Map Kit*) betten Landkarten in Ihre Anwendungen ein. Benutzer können Karteninformationen einsehen und auf den Inhalt der Karte zugreifen wie in der Anwendung *Karten*. Diese Klasse, die in der SKD-Version 3.0 eingeführt wurde, erlaubt es Ihnen, eine Karte mit eigenen Anmerkungen zu versehen. Dazu werden die Klassen `MKAnnotationView` und `MKPinAnnotationView` herangezogen.
- Mit Instanzen von `UIScrollView` können Sie Inhalte anzeigen, die umfangreicher sind als der Anwendungsbildschirm. Die Benutzer blättern mithilfe der horizontalen und vertikalen Rollbalken durch den Inhalt, um alles sehen zu können. In solchen Ansichten ist auch das Zoomen möglich, sodass Sie die iPhone-Standardgesten des Zusammen- und Auseinanderziehens verwenden können, um die Größe des Inhalts zu ändern.

4.1.2 Auswahlansichten

Auf dem iPhone gibt es zwei Hauptklassen, um dem Benutzer Auswahlmöglichkeiten zu bieten. Die Klasse `UIAlertView` ruft die blauen Popup-Fenster hervor, die Sie schon in vielen Anwendungen gesehen haben. Dabei legen Sie den Meldungstext fest und passen die Schaltflächen an, um dem Benutzer Fragen zu stellen, auf die er antworten kann. Beispielsweise können Sie einen Benutzer bitten, eine Aktion im Programm zu bestätigen oder abzubrechen. Sie können aber nicht nur Entscheidungen verlangen, sondern auch Informationen ausgeben. Warnansichten, die nur eine Schaltfläche (gewöhnlich **OK**) anzeigen, bilden eine einfache Möglichkeit, um dem Benutzer Text anzuzeigen.

Die zweite Klasse für Entscheidungen ist `UIActionSheet` mit Menüs, die sich vom unteren Rand des Bildschirms aus entrollen. Solche »Aktionsseiten« zeigen eine Meldung sowie Schaltflächen an, aus denen der Benutzer auswählen kann. Diese Seiten sehen zwar anders aus als Warnansichten, funktionieren im Grunde aber auf ähnliche Weise. Im Regelfall verwenden Sie Aktionsseiten, wenn es mehrere Auswahlmöglichkeiten gibt, und Warnansichten, wenn Sie höchstens zwei oder drei Optionen anbieten.

Beide Arten der Darstellung sind modal und erfordern, dass der Benutzer eine Auswahl trifft, bevor es weitergeht. Aus diesem Grunde ist es ein Gebot der Höflichkeit, neben den anderen Auswahlmöglichkeiten auch eine zum Abbrechen des Vorgangs anzubieten.

4.1.3 Steuerelemente

Steuerelemente sind Objekte auf dem Bildschirm, die Berührungen des Benutzers an Callback-Methoden weiterleiten. Sie können auch numerische oder Textwerte bereitstellen, die die Anwendung verarbeitet. Zu den Steuerelementen zählen u. a. Schaltflächen, Umschalter und Schieberegler, die ziemlich genau den Steuerelementen in der Programmierung für Desktop-Computer entsprechen. Die folgende Übersicht zeigt die wichtigsten Klassen in Cocoa Touch und die Steuerungsmöglichkeiten, die sie jeweils bieten:

- Instanzen von `UIButton` zeigen Schaltflächen auf dem Bildschirm an. Benutzer können darauf tippen, um über die Target-Action-Logik einen Callback auszulösen. Dabei geben Sie an, wie die Schaltfläche aussehen soll, welcher Text auf ihr angezeigt wird und wie die Schaltfläche ausgelöst wird. Die häufigste Art eines Auslösers ist das Aufheben der Berührung im Inneren (*touch up inside*), bei der die Berührung durch den Benutzer innerhalb der Grenzen der Schaltfläche endet. Wenn es Ihnen sonderbar erscheint, eine Schaltfläche durch das Loslassen und nicht das Senken des Fingers auszulösen, denken Sie daran, dass es auf dem iPhone den De-facto-Standard gibt, die Betätigung einer Schaltfläche dadurch zu widerrufen, dass der Benutzer seinen Finger seitwärts von der Schaltfläche zieht, bevor er ihn hebt.

  Im Interface Builder werden Schaltflächen als *Round Rect Buttons* (abgerundete Rechtecke) bezeichnet. Dort finden Sie auch Schaltflächen, die wie Ansichten aussehen und sich auch so verhalten, in Wirklichkeit aber gar keine Ansichten sind. Leistenschaltflächenelemente (`UIBarButtonItem`) speichern die Eigenschaften der Schaltflächen auf der Symbol- und Navigationsleiste, sind aber selbst keine Schaltflächen. Diese Leisten verwenden Beschreibungen, um sich selbst aufzubauen, doch die eigentlichen Schaltflächenansichten sind für Sie als Entwickler generell nicht zugänglich.

> **Hinweis**
>
> Im Interface Builder können Sie die Ansichtsbibliothek sowohl nach den Klassennamen durchsuchen (z. B. `UIButton`) als auch nach der Benennung im Interface Builder selbst (z. B. `round` oder `button`).

- `UISegmentedControl` zeigt eine Zeile von Schaltflächen gleicher Größe an, die sich wie die Tasten an einem guten alten Radio verhalten, d. h., dass jeweils nur eine Schaltfläche auf einmal aktiviert sein kann. Sie können diese Schaltflächen als Bilder oder als Text darstellen. Eine besondere Option (`momentary`) erlaubt es Ihnen, das normale Verhalten zu ersetzen und zu verhindern, dass die Schaltflächen anzeigen, welche von ihnen zuletzt ausgewählt wurde.
- In Cocoa Touch dient die Klasse `UISwitch` als einfacher Umschalter. Diese Klasse ermöglicht die Auswahl zwischen Ein und Aus und sieht wie ein normaler Lichtschalter an der Wand aus.
- Bei der Klasse `UISlider` kann der Benutzer einen Wert aus einem gegebenen Bereich auswählen, indem er einen Indikator (»Griff«) an einer horizontalen Linie verschiebt. Die Position des Indikators steht für die aktuelle Einstellung des Steuerelements, wobei der Wert

durch die relative Position des Griffs bestimmt wird. Ein typisches Beispiel für einen solchen Schieberegler auf dem iPhone ist der Lautstärkeregler in den Anwendungen *iPod* und *Musik*.

- Mit Seitensteuerelementen kann sich der Benutzer zwischen Seiten bewegen, gewöhnlich im Rahmen einer Implementierung von `UIScrollView`. Die Klasse `UIPageControl` gibt eine Reihe von kleinen Punkten aus (wie die auf dem Startbildschirm des iPhones), die die aktuelle Seite anzeigen und den Benutzer zur nächsten oder vorhergehenden Seite wechseln lassen.
- `UITextFields` sind eine Form von Steuerelement, in das Sie Text eingeben können. Diese Felder weisen nur eine einzige Zeile für die Eingabe auf und dienen zur Abfrage kurzer Textelemente (z. B. Benutzername und Kennwort) von den Benutzern. In *Abbildung 4.8* sehen Sie zwei Textfelder.

4.1.4 Tabellen und Picker-Ansichten

Tabellen zeigen eine Liste von Auswahlmöglichkeiten an, durch die der Benutzer blättern kann. Die am häufigsten verwendete Art von Tabelle bietet die Klasse `UITableView`, die Sie z. B. in den Anwendungen *Kontakte*, *YouTube* und *iPod/Musik* sehen. Tabellen bestehen aus Zeilen mit Informationen (bereitgestellt durch die Klasse `UITableViewCell`), durch die die Benutzer blättern und dann eine Auswahl treffen können.

Die Klasse `UIPickerView` zeigt eine Tabelle an, bei der die Benutzer ihre Auswahl treffen, indem sie verschiedene Rädchen drehen. Eine besondere Version dieser Klasse ist `UIDatePicker` mit einem vorgefertigten Verhalten für die Datums- und Uhrzeitauswahl. Diese Klasse wird in den Anwendungen *Kalender* und *Uhr* ausgiebig genutzt.

4.1.5 Leisten

Auf dem iPhone gibt es vier Arten von Leistenansichten. Leisten sind kompakte Ansichten (gewöhnlich mit einer kleineren Höhe als 50 Pixel), die sich quer über den Bildschirm spannen. Am häufigsten wird `UINavigationBar` verwendet (siehe *Abbildung 4.2*), die auf vielen Schnittstellen oben angezeigt wird, um die Navigation zu ermöglichen. Als Entwickler arbeiten Sie fast nie direkt mit den Instanzen dieser Klasse. Stattdessen wird die Sicht durch Instanzen von `UINavigationController` generiert und verwaltet, über die Sie in den folgenden Abschnitten mehr erfahren.

Tab Bars bieten die Art von Auswahlmöglichkeiten, die Sie am unteren Rand der Anwendungen *YouTube* und *iPod/Musik* sehen, z. B. **Highlights**, **Topvideos**, **Alben** und **Podcasts**. In *Abbildung 4.3* (oben) sehen Sie oben eine typische Instanz von `UITabBar`. Suchleisten (`UISearchBar`) fügen eine Textansicht hinzu, die bei der Navigationsleiste von Tabellen angezeigt wird, z. B. in der Anwendung *Kontakte*. Wie bei Navigationsleisten arbeiten Sie gewöhnlich mit Instanzen von `UITabBarController` bzw. `UISearchDisplayController`, anstatt die Sichten direkt zu erstellen und zu verwalten.

Unter allen iPhone-Leisten ist nur die Klasse `UIToolBar` für die direkte Verwendung gedacht. Sie zeigt ähnlich wie `UISegmentedControl` eine Reihe von Schaltflächen an, aber mit einem anderen Erscheinungsbild (siehe *Abbildung 4.3*, unten). Die Formatierung von Symbolleisten beschränkt sich darauf, die aktuelle Auswahl hervorzuheben. Der Zweck von Symbolleisten besteht darin, die mög-

lichen Aktionen bereitzustellen, die in der derzeitigen Ansicht vorgenommen werden können. Beispielsweise können Sie mit der Symbolleiste von *Mail* Nachrichten löschen oder darauf antworten. Auf den einzelnen Schaltflächen von Symbolleisten werden einfarbige Bilder angezeigt.

Wenn Sie in Ihrem Entwurf Tab Bars und Symbolleisten verwenden möchten, sollten Sie sich die Zeit nehmen, die *Human Interface Guidelines* von Apple zu lesen, die im Rahmen der iPhone-Standarddokumentation erhältlich sind. Apple lehnt Anwendungen ab, in denen Leisten auf eine Weise eingesetzt werden, die nicht diesen Richtlinien entspricht.

Hinweis

Wie die Schaltflächen von Symbolleisten erscheinen auch die Navigationselemente im Interface Builder, sodass Sie sie wie Ansichten in Ihren Projekten platzieren können. Wie ihre Vettern sind auch die Navigationselemente selbst keine Ansichten, sondern speichern Informationen darüber, was in der Navigationsleiste vor sich geht, und sie dienen dazu, die Leiste zu konstruieren, die schließlich angezeigt wird.

4.1.6 Fortschritt und Aktivität

Cocoa Touch enthält zwei Klassen, die dazu dienen, den Benutzer über fortdauernde Tätigkeiten zu informieren. Die Klasse `UIActivityIndicatorView` zeigt ein rotierendes Rad an, während eine Aufgabe abläuft. Daran kann der Benutzer erkennen, dass die Aufgabe irgendwann beendet sein wird, aber nicht, wann. Wenn Sie den Fortschritt angeben möchten, verwenden Sie die Klasse `UIProgressView`. Deren Instanzen zeigen eine Leiste an, die von links nach rechts gefüllt wird und damit angibt, wie weit die Aufgabe gediehen ist.

4.2 UIViewController

Auf dem iPhone zentralisieren Controller einige Formen der Ansichtsverwaltung. Sie sind praktische Hilfen, da sie die Ansichten mit der physischen Realität Ihres Geräts verbinden. Ansichtscontroller verarbeiten Neuausrichtungsereignisse wie das Drehen des iPhones ins Querformat oder werden bei Navigationsvorgängen aktiv, bei denen der Benutzer von Ansicht zu Ansicht wechselt.

Ansichtscontroller sind keine Ansichten, sondern abstrakte Klassen ohne grafische Darstellung. Nur Sichten bieten eine grafische »Leinwand«. Stattdessen helfen Controller dabei, Ansichten in der Entwurfsumgebung umfangreicherer Anwendungen zu verwenden. Einen Rahmen legen Sie hier nicht so fest wie bei einer normalen `UIView`. In `UIView` wird `initWithFrame:` verwendet, in `UIViewController` dagegen `init`.

Das iPhone SDK umfasst viele Klassen für Ansichtscontroller, von allgemeinen bis zu speziellen. Spezialisierte Controller sind sowohl ein Segen als auch ein Fluch. Positiv ist, dass sie einen enormen Funktionsumfang ohne zusätzlichen Programmieraufwand bieten. Der Nachteil besteht darin, dass sie oft so spezialisiert sind, dass sie Kernmerkmale verbergen, mit denen manche Entwickler lieber arbeiten würden.

Beispielsweise gibt es keine einfache Klasse für den Zugriff auf die Kamera. Sie müssen sich durch die Klasse `UIImagePickerController` arbeiten, um Fotos zu schießen. Diese Klasse mit ihrer vorgefertigten grafischen Benutzeroberfläche ist elegant und gut entworfen, verweigert Entwicklern aber den direkten Zugriff auf die Kamera und damit zu eigenen Benutzerschnittstellen, die sie vielleicht lieber erstellen möchten. Sie können keine aktuellen Daten von der Kamera abrufen und in einer Datenbank speichern. Stattdessen muss der Benutzer das Bild aufnehmen, bestätigen, dass es seinen Wünschen entspricht, und dann die Steuerung zurück an die Anwendung übergeben.

Die folgende Übersicht stellt einige der Ansichtscontroller vor, denen Sie beim Erstellen von Anwendungsschnittstellen für das iPhone begegnen werden.

4.2.1 UIViewController

`UIViewController` ist die Elternklasse für Ansichtscontroller, mit der Sie Ihre Hauptansichten verwalten. Hierbei handelt es sich um den Lastesel der Ansichtscontroller. Einen Großteil Ihrer Zeit werden Sie damit zubringen, diese eine Klasse anzupassen. Die grundlegende Klasse `UIViewController` verwaltet alle Hauptansichten vom Anfang bis zum Ende von deren Lebenszyklus und kümmert sich um die Änderungen, auf die die Ansicht dabei reagieren muss.

Beispielsweise erledigen `UIViewController` die Aufgabe der Neuausrichtung, sodass Sie sowohl für das Quer- als auch für das Hochformat programmieren können. `UIViewController` entscheiden, ob sie ihre Ausrichtung ändern müssen, wenn der Benutzer das iPhone kippt, und sie geben an, wie diese Neuausrichtung geschehen soll. Dazu verwenden sie Instanzmethoden wie `shouldAutorotateToInterfaceOrientation:`. Ohne Ansichtscontroller könnte die Schnittstelle nicht automatisch ihre Ausrichtung korrigieren. Viele Entwickler haben festgestellt, dass es schwierig ist, `UIViews` direkt und ohne Hilfe durch eine Ansichtscontrollerklasse zu drehen.

Die Instanzen von `UIViewController` sind dafür verantwortlich, wie eine Ansicht aussieht und welche Unteransichten angezeigt werden. Häufig laden sie dazu die betreffenden Informationen aus `.xib`-Dateien. Mit Instanzmethoden wie `loadView` und `viewDidLoad` können Sie ein Verhalten für die Zeit während oder nach der Einrichtung der Ansicht hinzufügen.

Darauf zu reagieren, dass Ansichten angezeigt oder entfernt werden, ist eine weitere Aufgabe für Ansichtscontroller. Das sind die Realitäten, die umfangreiche Anwendungen mit sich bringen. Dank Methoden wie `viewDidAppear:` und `viewWillDisappear:` können Sie die Einrichtungs- und Bereinigungsaufgaben der Ansichtsverwaltung erledigen. Beispielsweise können Sie zu erwartende Daten vorab laden oder Speicher aufräumen, der nicht verwendet wird, während die Ansicht nicht auf dem Bildschirm zu sehen ist.

Jede der hier genannten Aufgaben legt fest, wie sich eine Ansicht in eine Anwendung einfügt und auf einem bestimmten Gerät funktioniert. Der `UIViewController` vermittelt zwischen Ansichten und externen Erfordernissen, sodass sich die Ansicht an diese Anforderungen anpassen kann.

4.2.2 UINavigationController

Wie der Name schon andeutet, ermöglicht der Navigationscontroller es Ihnen, sich auf und ab durch die Baumstruktur von Ansichtshierarchien zu bewegen. Sie erstellen eine einfarbige Navigationsleiste, wie sie am oberen Rand vieler iPhone-Standardanwendungen erscheint. Diese Navigationscontroller erleben Sie im Einsatz, wenn Sie sich durch irgendeine Art von Hierarchie bewegen, z. B. in der Anwendung *Kontakte* oder im App Store auf dem iPhone. Beide Programme wurden mit Navigationscontrollern erstellt.

Durch Navigationscontroller können Sie neue Ansichten in die richtige Position hieven und automatisch **Zurück**-Schaltflächen erstellen, die den Titel des aufrufenden Ansichtscontrollers zeigen. Alle Navigationscontroller nutzen einen Ansichtscontroller als »Wurzel«, um die Spitze des Navigationsbaums festzulegen, sodass die **Zurück**-Schaltflächen Sie letzten Endes zu einer Hauptansicht zurückführen. Navigationscontroller und ihre Baumstrukturen werden weiter hinten in diesem Kapitel noch ausführlicher besprochen.

Wenn Sie Verantwortung an den Navigationscontroller abgeben, können Sie sich bei Ihrer Entwurfstätigkeit besser darauf konzentrieren, einzelne Ansichtscontrollerbildschirme zu erstellen. Sie müssen sich nicht um die Einzelheiten der Navigation kümmern, sondern dem Navigationscontroller lediglich mitteilen, zu welcher Ansicht er als Nächstes wechseln soll. Die Verantwortung für den Verlaufsstapel und die Navigationsschaltflächen wird Ihnen abgenommen. In Kapitel 5, *Mit Ansichtscontrollern arbeiten*, erfahren Sie mehr über Ansichtscontroller und lernen Rezepte für deren Verwendung kennen.

4.2.3 UITabBarController

Parallele Ansichten sind wie Sender im Radio. Eine Tab Bar ermöglicht Benutzern die Auswahl, welcher `UIViewController` »angehört« wird, ohne dass dafür eine eigene Navigationshierarchie vorhanden sein muss. Am besten ist dies in Programmen wie YouTube und iPod zu erkennen, wo Benutzer sich entweder die Top-25-Liste oder Alben- bzw. Titellisten ansehen können. Jede dieser parallelen Welten funktioniert unabhängig voneinander, und alle können ihre eigene Navigationshierarchie haben. Sie erstellen die Ansichts- oder Navigationscontroller für die einzelnen Symbole, und Cocoa Touch kümmert sich um die Einzelheiten der Mehrfachansicht.

Wenn `TabBar`-Instanzen z. B. mehr als fünf Ansichtscontroller gleichzeitig anbieten, können Benutzer sie über **Mehr ▸ Bearbeiten** anpassen. Dort können sie die favorisierten Controller in die Schaltflächenleiste am unteren Rand des Bildschirms ziehen. Dazu ist keine zusätzliche Programmierung erforderlich. Bearbeitbare Symbole erhalten Sie ohne Zusatzaufwand, Sie müssen sie lediglich über die Eigenschaft `customizableViewControllers` anfordern. In Kapitel 5 lesen Sie mehr darüber, wie Sie Anwendungen mit Tab Bars schreiben und die Bilder festlegen, die die einzelnen Schaltflächen schmücken.

4.2.4 Tabellencontroller

Tabellenansichtscontroller machen die Verwendung von Tabellen in iPhone-Projekten einfacher. Die Klasse `UITableViewController` bietet eine standardmäßige, bereits verbundene Instanz von `UITableView` und richtet die Delegation und die Datenquellen automatisch auf sich selbst aus. Sie müssen dann nur noch die Delegate- und Datenquellenmethoden bereitstellen, um die Tabelle mit Daten zu füllen und auf Benutzerberührungen zu reagieren. `UITableViewController` wird ausführlicher in Kapitel 11, *Tabellenansichten erstellen und verwalten*, erläutert.

Der Suchanzeigecontroller ist eine Art von Tabellenansicht, enthält aber über `UISearchBar` eine integrierte Suchleiste. Damit können die Benutzer nach Daten suchen, die von einem anderen Ansichtscontroller bereitgestellt werden, dem *Inhaltscontroller*. Während die Benutzer die Suchinformationen aktualisieren, passt der Inhaltscontroller seine Datenquelle an, um ihm die Elemente hinzuzufügen, die mit der Suchabfrage übereinstimmen.

Es mag merkwürdig erscheinen, einen anderen Controller dazu zu zwingen, diese Aufgabe zu erledigen, aber in der Praxis funktioniert das sehr gut. Der Inhaltscontroller ist fast immer ein Tabellenansichtscontroller, der den Suchcontroller auf Anforderung anzeigt. Die Suche durchläuft dann die Daten der ursprünglichen Tabelle und zeigt eine Teilmenge von deren Informationen an, bis die Suche wieder ausgeschaltet wird.

Auch `NSFetchedResultsController` ist eine Art von Tabellencontroller. Streng genommen handelte es sich hierbei zwar nicht um einen Ansichtscontroller, doch hilft diese Klasse dabei, eine `UITableView` mit Objekten zu füllen, die aus einem Core Data-Speicher abgerufen worden sind. In Kapitel 19, *Core Data*, finden Sie ein Beispiel, das diese Klasse in Aktion zeigt.

4.2.5 Adressbuchcontroller

Das Framework für die Adressbuch-Benutzerschnittstelle (`AddressBookUI.framework`) umfasst verschiedene Ansichtscontroller, mit denen Sie eine Person aus einem Adressbuch auswählen, deren Kontaktinformationen anzeigen, eine neue Person hinzufügen oder einen bereits bestehenden Eintrag bearbeiten können. Diese Ansichtscontroller gliedern sich auch in das C-Framework `ABAddressBook` ein, das Funktionen zur Abfrage und Aktualisierung des integrierten Adressbuchs auf dem iPhone bietet. In Kapitel 18, *Verbindung mit dem Adressbuch*, werden das Adressbuch und seine UI-Controller ausführlicher besprochen.

4.2.6 UIImagePickerControl

Mit dem Hilfscontroller `UIImagePickerControl` können Benutzer Bilder aus Alben auf dem Gerät auswählen sowie Fotos mit der iPhone-Kamera schießen. Damit erhalten Sie Vollzugriff auf die meisten der Ordnungsmöglichkeiten, die Benutzer in den Anwendungen *Kamera* und *Fotos* haben. In Wirklichkeit handelt es sich dabei nicht um zwei verschiedene Programme, sondern um ein einziges, das sich nur als zwei unterschiedliche Anwendungen ausgibt. Es gibt nur einen Controller, der Zugriff auf die Kamera und auf Auswahlmöglichkeiten für Fotos bietet.

Für die Auswahl von Fotos hat Apple eine fortschrittliche Schnittstelle bereitgestellt. Die Benutzer können sich in der Hierarchie der Fotoalben nach oben und unten bewegen, bis sie das gewünschte Bild finden. Der Picker kümmert sich automatisch um den Zugriff auf das integrierte Fotoalbum, sodass Ihnen kaum mehr zu tun übrig bleibt, als auszuwählen, wie das ausgewählte Bild verwendet werden soll.

Die Kameraschnittstelle ist ebenso eindrucksvoll. Der Controller erlaubt den Benutzern sogar, ein Bild auszurichten und zu vergrößern und somit benutzerdefinierte Bearbeitungen an dem aufgenommenen Foto vorzunehmen. Eine vollständige Erörterung dieser Klasse mit Anleitungen sowohl für die Kamera- als auch die Bildauswahlversion finden Sie in Kapitel 7, *Mit Bildern arbeiten*.

4.2.7 Mailerstellung

Mit `MFMailComposeViewController` können Sie E-Mail-Nachrichten erstellen, die Benutzer direkt in Ihrem Programm anpassen können. Das iPhone unterstützt zwar schon lange `mailto:`-ULRs, an die E-Mail-Nachrichten gesendet werden können, doch diese in der Version 3.0 des SDK eingeführte neue Klasse gibt Ihnen weit mehr Kontrolle über Inhalt und Anhänge von E-Mails. Vor allem aber können die Benutzer weiterhin in Ihrem Programm arbeiten, ohne es zwangsweise verlassen und auf die Anwendung *Mail* zugreifen zu müssen.

Der Mailerstellungscontroller ist einfach zu verwenden und wird in Kapitel 7 benutzt, um Fotos per E-Mail zu verschicken. Er gehört zum Framework MessageUI, und das Präfix `MF` steht für *Message Framework*.

4.2.8 GKPeerPickerController

Der *GameKit Peer Picker* bietet eine grafische Standardoberfläche, um andere iPhones aufzuspüren und sich mit Ihnen in Verbindung zu setzen. Er zeigt eine elegante Schnittstelle an, auf der andere verfügbare iPhones aufgelistet werden, zu denen eine Verbindung möglich ist. Dieser Controller ist zwar Teil von GameKit, seine Technologie lässt sich aber auch für Anwendungen anpassen, die nichts mit Spielen zu tun haben, z. B. Dateiübertragung, Messaging usw.

Sie können den Picker auf Bluetooth- oder Internetverbindungen einrichten. Dem Benutzer gegenüber zeigt er dann nur die Verbindungen des jeweiligen Typs an, wobei die Benutzer die Art der Auswahl in der Schnittstelle nicht selbst festlegen können.

Mehr über den Peer-Picker-Controller erfahren Sie in Kapitel 12, *Verbindungen mit GameKit und Bonjour*.

4.2.9 Media Player-Controller

Das Framework *Media Player* enthält mehrere Controller, mit denen Sie Musik und Filme auswählen und abspielen können. `MPMediaPickerController` zeigt eine grafische Benutzeroberfläche zur Medienauswahl, auf der Benutzer Musik, Podcasts und Hörbücher auswählen können. Dabei legen Sie fest, welche Medien angezeigt werden. Eine Wiedergabe der Medien ist über eine Instanz von `MPMusicPlayerController` möglich.

Wenn Ihre Benutzer einen Film ansehen oder eine Audioaufnahme anhören müssen, ist dies mit einer Instanz von `MPMoviePlayerController` möglich. Dazu geben Sie lediglich einen Pfad zur Medienressource an und bringen den Controller in die Ansicht. Der Controller zeigt eine **Fertig**-Schaltfläche für den Benutzer an oder gibt am Ende der Wiedergabe einen Delegate-Aufruf zurück.

Wenn Sie mehr über die Auswahl und Wiedergabe von Medien wissen möchten, lesen Sie Kapitel 15, *Audio, Video und Media Kit*.

4.3 Geometrie von Ansichten

Die iPhone-Hardware ist in der Theorie nicht auf eine Anzeige von 320 × 480 Pixel beschränkt. Entwerfen Sie Ihre Anwendungen so unabhängig von der Auflösung wie möglich. Trotzdem spielen aber gewisse Umstände der Geometrie eine Rolle für den Entwurf von Anwendungen für die derzeitige Generation von iPhones. Dies gilt vor allem, wenn Sie an einen Grafikdesigner Muster für die Bearbeitung in Photoshop weitergeben.

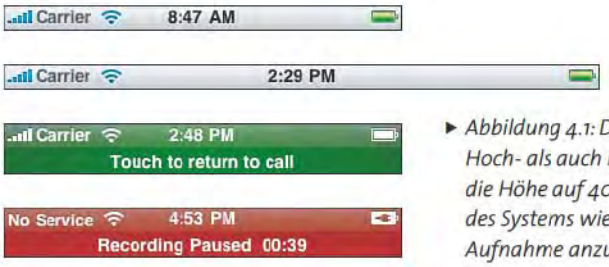
Der folgende Überblick zeigt die Bildschirmelemente, bei denen Sie für die Gestaltung von Schnittstellen meistens darauf vertrauen können, dass die Geometrie unverändert bleibt. Verlassen Sie sich nach Möglichkeit nicht auf die Größen, sondern behalten Sie beim Entwerfen ihre Proportionen und Größenverhältnisse im Hinterkopf.

Künftige iPhone-Modelle oder verwandte iPhone OS-Geräte weisen möglicherweise nicht dieselbe Bildschirmgröße oder -form auf. So hat das neue iPad beispielsweise eine Auflösung von 1024 × 768 Pixel. Alle in diesem Abschnitt genannten Maße gelten nur für die ersten fünf Mitglieder der iPhone OS-Familie, die einen Bildschirm mit den Abmessungen 320 × 480 haben: die erste Generation des iPhones, das iPhone 3G bzw. 3G S sowie die erste und zweite Generation des iPod touch.

4.3.1 Statusleiste

Die Statusleiste ganz oben auf dem iPhone-Bildschirm zeigt die Uhrzeit, den Verbindungsstatus, den Ladezustand der Batterie sowie den Netzbetreiber (iPhone) bzw. das Modell des Geräts (iPod) an. Diese Leiste ist im Normalzustand 20 Pixel hoch, wird bei Telefongesprächen oder bei der Anzeige von Meldungen jedoch auf 40 Pixel vergrößert (allerdings nur im Hochformat). Leider bietet das SDK keine öffentlichen Zugriffsmöglichkeiten für das Nachrichtenanzeigesystem, sodass Sie hier keine eigenen Meldungen einblenden können. Die 40 Pixel hohe, farbige Statusanzeige können Sie sehen, wenn Sie eine Voice Memo-Aufzeichnung anhalten, Nike+ verwenden oder – beim Modell 3G oder jünger – das iPhone an den Computer anschließen.

Abbildung 4.1 zeigt die Statusleiste im Hoch- und Querformat sowie im 40-Pixel-Modus. Sie können die Statusleiste auch ausblenden, aber dadurch verhindern Sie, dass die Benutzer die Uhrzeit und den Ladezustand der Batterie sehen, sofern Sie diese Angaben nicht irgendwo anders in der Schnittstelle Ihrer Anwendung machen. Es ist auch möglich, die Statusleiste in Grau, Schwarz oder transparentem Schwarz anzuzeigen. Letzteres ermöglicht es, die dahinter liegende Ansicht durchscheinen zu lassen, um die Farben in der Anwendung besser abstimmen zu können.



▶ *Abbildung 4.1: Die Statusleiste ist normalerweise sowohl im Hoch- als auch im Querformat 20 Pixel hoch. Manchmal wird die Höhe auf 40 Pixel vergrößert, um fortdauernde Tätigkeiten des Systems wie einen Telefonanruf oder eine angehaltene Aufnahme anzuzeigen.*

Wenn Sie diese 20 Pixel Bildschirmfläche lieber für einen anderen Verwendungszweck nutzen möchten, können Sie die Statuszeile vollständig verbergen. Verwenden Sie dazu den `UIApplication`-Aufruf `[[UIApplication sharedApplication] setStatusBarHidden:YES animated:NO]`. Alternativ können Sie den Schlüssel `UIStatusBarHidden` in der Datei `Info.plist` auf `<true/>` setzen.

Wenn die Statusleiste angezeigt wird, steht für Ihre Anwendung auf dem Standard-iPhone ein Platz von 320 × 460 Pixel im Hoch- und 480 × 300 Pixel im Querformat zur Verfügung. Diese Zahlen können sich je nachdem ändern, welche anderen Elemente Sie der Schnittstelle hinzufügen, z. B. Navigationsleisten, Tab Bars usw. Darüber hinaus können sich die Pixelabmessungen für das Standard-iPhone im Laufe der Zeit ändern, wenn Apple neue Modelle und neue verwandte Produkte mit iPhone OS auf den Markt bringt.

Die Statusleiste wird sowohl im Hoch- als auch im Querformat angezeigt und passt sich jeweils an die Ausrichtung an. Um Ihre Anwendung ausschließlich im Querformat zu betreiben, setzen Sie die Ausrichtung der Statusleiste auf Querformat (`[[UIApplication sharedApplication] setStatusBarOrientation: UIInterfaceOrientationLandscapeRight]`), und zwar auch dann, wenn Sie die Leiste ausblenden wollen. Alternativ setzen Sie `UIInterfaceOrientation` in der Datei `Info.plist` auf `UIInterfaceOrientationLandscapeLeft` oder `UIInterfaceOrientationLandscapeRight`. Dadurch wird die Anzeige von Fenstern nebeneinander erzwungen und für eine Tastatur im Querformat gesorgt.

HINWEIS

Mit **HARDWARE ▸ TOGGLE IN CALL STATUS BAR** testen Sie die 40-Pixel-Statusleiste Ihrer Anwendung im Simulator.

4.3.2 Navigationsleisten, Symbolleisten und Tab Bars

Standardmäßig sind `UINavigationBar`-Objekte (siehe *Abbildung 4.2*) im Hochformat 44 Pixel hoch und im Querformat 32 Pixel. Sie reichen von einer Seite des Bildschirms zur gegenüberliegenden, sodass sie eine Fläche von 320 × 44 bzw. 480 × 32 Pixel einnehmen.

▸ *Abbildung 4.2: Navigationsleisten erstrecken sich von einer Seite des Bildschirms zur anderen. Die Höhe ist auf dem iPhone im Hochformat auf 44 Pixel und im Querformat auf 32 Pixel festgelegt. Bei Verwendung der selten genutzten Eingabeaufforderung, die Sie in den beiden unteren Bildern sehen, steigt die Höhe auf 74 Pixel.*

Für Navigationsleisten gibt es auch einen selten genutzten Modus mit Meldung, bei dem die Höhe um 30 Pixel heraufgesetzt wird. Im Hochformat nimmt die Leiste dann 320 × 74 Pixel ein, im Querformat 480 × 74, wobei im letzteren Fall statt der üblichen 32-Pixel-Version auch eine Navigationsleiste von 44 Pixel angezeigt wird.

Hinweis

Um eine Meldung zu einer Navigationsleiste hinzuzufügen, bearbeiten Sie das Navigationselement des Ansichtscontrollers: `self.navigationItem.prompt = @"Please click a button now";`.

Tab Bars sind in beiden Ausrichtungen 48 Pixel hoch, nehmen also 320 × 48 bzw. 480 × 48 Pixel ein. Um ausreichenden Platz für das Antippen zu gewähren, sollten die einzelnen Elemente auf einer Tab Bar laut Apple einen Antippbereich von mindestens 44 × 44 Pixel aufweisen. Das entspricht den üblichen Grafiken von 30 × 30 Pixel.

Abbildung 4.3 zeigt eine typische Tab Bar und eine nahe verwandte Klasse, die Symbolleiste. Symbolleisten weisen dieselbe Elementgröße von 44 Pixel auf, werden aber am unteren Rand des Bildschirms angezeigt.

▸ *Abbildung 4.3: Auf iPhones mit einer Bildschirmgröße von 320 × 480 Pixel haben Tab Bars eine Höhe von 48 Pixel (oben). Symbolleisten weisen ebenso wie Navigationsleisten eine Höhe von 44 Pixel auf.*

Diese beiden Elemente der grafischen Benutzerschnittstelle sind nicht für die Verwendung im Querformat geeignet. Dies können Sie bei den Anwendungen *iPod* und *YouTube* sehen. Die Ansicht im Hochformat beruht auf Tab Bars, beim Wechsel ins Querformat zeigt sich aber eine völlig andere Darstellung, nämlich Coverflow in *iPod* und File in *YouTube*.

Bei der Nutzung von Status-, Navigations- und Symbolleisten sowie Tab Bars müssen Sie etwas grundlegende Mathematik anwenden, um zu berechnen, wie viel Platz noch für den Entwurf des Hintergrunds zur Verfügung steht. Eine typische Anwendung mit einer Navigations- und einer Statusleiste lässt in der Mitte einen Bereich von 320 × 416 Pixel im Hoch- und 480 × 268 Pixel im Querformat frei. Wenn Sie Tab Bars oder Symbolleisten verwenden, verringern Sie die verfügbare Höhe um weitere 48 bzw. 44 Pixel.

4.3.3 Tastaturen und Picker-Ansichten

Die iPhone-Standardtastatur nimmt 320 × 216 Pixel im Quer- und 480 × 162 Pixel im Hochformat ein. In *Abbildung 4.4* sehen Sie die Tastatur in der Standardkonfiguration für beide Ausrichtungen. Wenn in einer Anwendung ein Textelement aktiv wird, so erscheint die Tastatur am unteren Rand des Bildschirms, wobei sie jegliche dort vorhandenen Elemente verdeckt und oben nur einen verkleinerten Raum zur Interaktion freilässt. Ausgefeiltere Tastaturlayouts können sogar noch mehr Platz auf dem Bildschirm einnehmen.

▸ *Abbildung 4.4: Die Tastatur nimmt sowohl im Hoch- als auch im Querformat einen großen Teil des iPhone-Bildschirms ein. Achten Sie bei der Gestaltung Ihrer Anwendung darauf.*

Sie sollten die Hauptansicht Ihrer Anwendung verkleinern, wenn die Tastatur angezeigt wird. Gibt es auf dem Bildschirm mehrere Elemente, die der Benutzer bearbeiten kann, dann fahren Sie am besten mit einer verkürzten Ansicht mit Rollbalken. Dadurch kann der Benutzer zu allen möglichen Bereichen blättern, ohne dass Textfelder oder Textansichten von der Tastatur zugedeckt werden. Verringern Sie den Frame der Hintergrundansicht je nach Ausrichtung um 216 bzw. 162 Pixel.

Geben Sie dem Benutzer eine Möglichkeit, die Tastatur wieder auszublenden, z. B. durch Antippen der Tastaturschaltfläche **Return** oder über eine **Fertig**-Schaltfläche, sodass die normale Ansicht wieder erscheint. Lassen Sie die Benutzer nicht mit eingeblendeter Tastatur zurück. In Kapitel 9, *Steuerelemente erstellen und verwenden*, erfahren Sie genauere Einzelheiten über das Ausblenden der Tastatur.

Hinweis

Sowohl `UIPickerView` als auch `UIDatePicker` verwenden dieselbe Geometrie wie die Standardtastatur. Ein `UISwitch` nimmt standardmäßig eine Fläche von 94 × 28 Pixel ein, während ein `UISegmentedControl` in der normalen textgestützten Form gewöhnlich 44 Pixel hoch ist.

4.3.4 Textfelder

Bei Instanzen von `UiTextField` müssen Sie mindestens 30 Pixel an Höhe vorsehen, damit die Benutzer genügend Platz haben, um Text mit der Standardschriftgröße einzugeben, ohne dass er abgeschnitten wird.

4.3.5 Die Klasse UIScreen

Das Objekt `UIScreen` steht für den physischen Bildschirm des iPhones, auf den Sie über `[UIScreen mainScreen]` zugreifen können. Dieses Objekt bildet die Grenzen des Standardfensterlayouts auf den Pixelraum ab, wobei jegliche verwendeten Symbol-, Status- und Navigationsleisten beachtet werden.

Um die Größe des gesamten Bildschirms abzurufen, verwenden Sie `[[UIScreen mainScreen] bounds]`. Dadurch wird ein Rechteck zurückgegeben, das die Gesamtgröße des iPhone-Bildschirms in Pixeln definiert. Wie ich bereits erwähnt habe, muss der iPhone-Bildschirm nicht zwangsläufig 320 × 480 Pixel groß sein, wenn Apple neue Geräte einführt.

Mit dem Methodenaufruf `[[UIScreen mainScreen] applicationFrame]` können Sie die zentrale Anwendungsfläche abrufen. Auf einem Gerät der ersten oder zweiten Generation wird für eine Anwendung mit einer Statusleiste von 20 Pixel und einer Navigationsleiste von 44 Pixel eine Größe von 320 × 416 Pixel zurückgegeben.

Mit diesen Zahlen können Sie den verfügbaren Platz auf dem iPhone-Bildschirm berechnen und das Layout der Ansichten Ihrer Anwendung bestimmen, sofern Sie nicht den Interface Builder verwenden.

4.4 Schnittstellen anlegen

Es gibt mehr als eine Möglichkeit, eine Schnittstelle zu konstruieren. Mit dem iPhone SDK können Sie eine grafische Benutzerschnittstelle manuell in Objective-C schreiben, Sie können sie aber auch grafisch im Interface Builder gestalten. Im Code geben Sie durch Programmbefehle an, wo die einzelnen Elemente auf dem Bildschirm erscheinen und wie sie sich verhalten. Im Interface Builder ordnen Sie die Elemente in einem grafischen Editor an. Beide Verfahrensweisen bieten Vorteile. Wägen Sie die Vorteile selbst ab, und kommen Sie zu einer eigenen Entscheidung.

Schließlich führen beide Vorgehensweisen zum selben Ziel. Der Objective-C-Code entspricht genau dem Layout im Interface Builder, und das im Interface Builder eingerichtete Callback-Verhalten führt zu denselben Ergebnissen wie das, das Sie in Objective-C schreiben.

Die Einzelheiten der Implementierung mögen abweichen. Bei einer manuellen Version verwenden Sie `loadView`, um die Hauptansicht zu erstellen und ihre Schnittstellenelemente hinzuzufügen. Dagegen schließt ein `.xib`-gestützter Ansichtscontroller seine eigene Einrichtung mit `viewDidLoad` ab, nachdem er die vorgefertigte Schnittstelle aus der `.xib`-Datei geladen hat. In Cocoa Touch können Sie beide Ansätze sowie eine Kombination daraus verwenden, bei der Sie `.xib`-Dateien direkt über Objective-C-Befehle laden.

In den nächsten Abschnitten lernen Sie diese Verfahrensweisen kennen, indem Sie den Vorgang einmal im Interface Builder und einmal in Xcode durchgehen. Im Anschluss daran sind zwei kombinierte Lösungen angegeben. Jede dieser vier Anleitungen führt zu identischen Endergebnissen mit identischem Funktionsumfang.

4.5 Schritt für Schritt: Ein Programm zur Temperaturumrechnung im Interface Builder erstellen

Der Interface Builder (IB) mit seinen GUI-Layoutwerkzeugen hilft Ihnen beim Gestalten von grafischen Inhalten. Er ermöglicht es Ihnen, interaktive Steuerelemente hinzuzufügen und auf dem Bildschirm zu verschieben, um eine eigene Schnittstelle zu gestalten. Im ersten Beispiel erstellen wir ein klassisches Umrechnungsprogramm für Fahrenheit und Celsius, für das wir ausschließlich standardmäßige Xcode/IB-Entwurfsvorlagen verwenden. Die Schnittstelle wird vollständig im Interface Builder gestaltet, wobei wir nur wenig Code in Xcode schreiben.

Hinweis

Sie sollten die Hello-World-Beispiele in Kapitel 2, *Ein erstes Projekt erstellen*, durchgearbeitet und damit ein Grundwissen über Xcode und den Interface Builder erworben haben. Die Beispiele in diesem Kapitel gehen tiefer, setzen aber voraus, dass Sie Grundkenntnisse in der Verwendung dieser Werkzeuge haben.

4.5.1 Ein neues Projekt erstellen

Starten Sie Xcode, und erstellen Sie ein neues Projekt. Wählen Sie dazu **File ▸ New Project ▸ iPhone OS ▸ Application ▸ Navigation-based Application**, und klicken Sie auf **Choose**. Nennen Sie das Projekt »Converter«, und speichern Sie es auf Ihrem Schreibtisch. Nachdem Sie das Projekt erstellt haben, öffnet sich in Xcode ein neues Projektfenster mit den beiden `.xib`-Dateien `MainWindow.xib` und `RootViewController.xib` sowie Klassen für den Anwendungs-Delegate und den Wurzelansichtscontroller.

Bei einem Projekt mit Navigation müssen Sie stets einen Wurzelansichtscontroller zuweisen. Dies ist der Ansichtscontroller an der Spitze des Navigationsbaums, von dem alle anderen Ansichtscontroller abzweigen. Der Name der `.xib`-Datei und der Klasse spiegeln diese Notwendigkeit wider.

4.5.2 Medien hinzufügen

Bevor Sie weitermachen können, müssen Sie dem Projekt zunächst einige grundlegende Medien hinzufügen. Kopieren Sie die Grafikdateien `icon.png` und `Default.png` aus dem Beispielcode-Ordner in das Projekt, indem Sie sie über der Gruppe **Resources** der Spalte **Groups & Files** loslassen. Aktivieren Sie **Copy Items into Destination Group's Folder (If Needed)**, bevor Sie auf **Add** klicken.

Hinweis

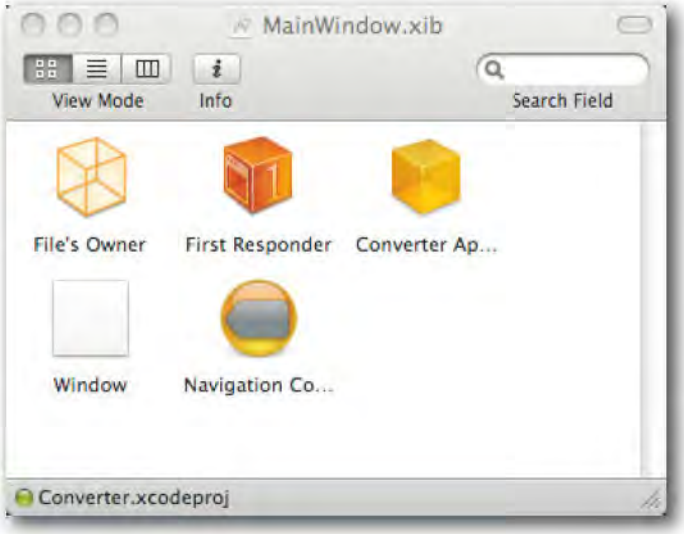
Wenn Sie ein Element in mehreren Projekten verwenden, können Sie die betreffende Datei hinzufügen, ohne sie kopieren zu müssen. Dadurch haben Sie eine einzige Quellversion, sodass sich alle Änderungen, die Sie daran vornehmen, auf sämtliche Projekte auswirken, die diese Datei verwenden. Aktivieren Sie dazu **Copy Items into Destination Group's Folder (If Needed)** einfach nicht. Der Nachteil ist, dass Sie versehentlich das Original löschen können, wenn Sie die Datei aus einem der Projekte entfernen, wovon mehrere Projekte betroffen wären.

Bei diesen beiden Elementen handelt es sich um die Grafiken für das Anwendungssymbol auf dem SpringBoard-Bildschirm des iPhones (`icon.png`) und für das Bild, das beim Start der Anwendung gezeigt wird (`Default.png`). Jedes Programm, das Sie schreiben, muss Grafiken für diese beiden Zwecke enthalten. Die Rolle dieser Elemente wird ausführlicher in Kapitel 1, *Einführung in das iPhone SDK*, beschrieben.

Fügen Sie als Nächstes `cover320x416.png` auf dieselbe Weise zur Gruppe **Resources** hinzu. Diese Datei enthält das Hintergrundbild für dieses Projekt. Die Grafik ist für eine Schnittstelle im Hochformat mit einer Status- und Navigationsleiste dimensioniert.

4.5.3 Interface Builder

Doppelklicken Sie auf die Datei `MainWindow.xib`, um sie im Interface Builder zu öffnen, und bringen Sie das Fenster mit dieser Datei in den Vordergrund. Falls Sie Schwierigkeiten haben, das Fenster zu finden, greifen Sie auf die Liste im Menü **Windows** zurück. In dem Fenster, das Sie in *Abbildung 4.5* sehen, finden Sie fünf Elemente. Die ersten beiden, `File's Owner` und `First Responder`, haben Sie bereits in Kapitel 2 kennengelernt. Die anderen sind neu: ein Converter-Anwendungs-Delegate (bezeichnet mit `Converter Ap...`), das Fenster (`Window`) und der Navigationscontroller (`Navigation Co...`).



▸ *Abbildung 4.5: Die Standardkomponenten von `MainWindow.xib`, die von der Xcode-Vorlage für Navigationsanwendungen erstellt werden*

Im Informationsfenster für Identitäten ⌘ + 4 können Sie sich die Klassen der einzelnen Objekte ansehen. Klicken Sie bei geöffnetem Informationsfenster auf die verschiedenen Objekte. Der Dateibesitzer (`Files's Owner`) ist eine Instanz von `UIApplication`, und sein Delegate ist der Hello-World-Anwendungs-Delegate. Dies entspricht dem in Kapitel 1 besprochenen Entwurfsmuster.

Das Fenster (`Windows`) ist eine Instanz von `UIWindow`. Es stellt eine bildschirmfüllende Ansicht bereit, die alle Anwendungsansichten besitzt, sobald diese hinzugefügt werden. Mit dieser Instanz arbeiten Sie nicht direkt, da sie bereits so eingerichtet ist, dass sie die vom Navigationscontroller definierten Ansichtsinhalte zeigt.

Die Rolle des Navigationscontrollers ist nicht so deutlich, da er eine Navigationsleiste mit einem optionalen Titel und vielleicht einige Schaltflächen bereitstellt, während eine andere Klasse für die Schnittstellenelemente unterhalb der Leiste zuständig ist. Jeder `UINavigationController` muss mit genau einem Wurzelansichtscontroller initialisiert werden. Dieser Ansichtscontroller legt die Ansicht fest, die den Rest des Bildschirms ausfüllt. Navigationscontroller werden weiter hinten in diesem Kapitel ausführlicher behandelt.

Wenn Sie auf ein Navigationscontrollerobjekt doppelklicken, öffnet sich ein Editorfenster, wie Sie in *Abbildung 4.6* sehen. An dem Screenshot können Sie ablesen, dass `MainWindow.xib` die Wurzelansicht nicht direkt definiert, sondern von `RootViewController.xib` lädt, der zweiten `.xib`-Datei, die durch die Auswahl der Vorlage für Navigationsanwendungen erstellt wurde.

▸ *Abbildung 4.6: Der Navigationscontroller liest seinen Wurzelansichtscontroller aus der zweiten* `.xib`*-Datei.*

Dadurch, dass Sie den Ansichtscontroller aus einer zweiten Datei laden, können Sie theoretisch die einzelnen Komponenten unabhängig voneinander gestalten, also die Ansicht getrennt vom Fenster und der Navigationsleiste entwerfen. (In der Praxis findet dieses Merkmal des Interface Builder allerdings nicht die ungeteilte Zustimmung.) Wenn Sie im Editorfenster die Wurzelansicht anwäh-

len und das Informationsfenster für Attribute öffnen ⌘ + 1, sehen Sie ein Popup-Fenster, in dem Sie eine `.xib`-Datei für den Wurzelansichtscontroller auswählen können. Ändern Sie diese Auswahl nicht, da Sie keine anderen `.xib`-Elemente für `UIViewController` haben.

4.5.4 Die Navigationsleiste bearbeiten

Zurück im Editor aus *Abbildung 4.6*, nehmen Sie die folgenden Änderungen vor: Als Erstes klicken Sie in die Mitte der blauen Leiste und geben das Wort »Converter« ein. Dadurch fügen Sie der Navigationsleiste einen Titel hinzu. Danach ziehen Sie ein Leistenschaltflächenelement (*Bar Buttom Item*) aus der Bibliothek (**Tools ▸ Library**, ⌘ + ⇧ + L auf die rechte Seite der Leiste. Doppelklicken Sie darauf, und ändern Sie das Wort »Item« in »Convert« (»Umrechnen«). *Abbildung 4.7* zeigt die Leiste, wie sie danach aussieht.

▸ *Abbildung 4.7: Sie können die Leiste des Navigationscontrollers direkt bearbeiten und ihr Schaltflächen hinzufügen.*

4.5.5 Die Hauptansicht ersetzen

In den Standardvorlagen stellt Ihnen Apple manchmal nicht genau das zur Verfügung, was Sie wirklich brauchen. Dies können Sie erkennen, wenn Sie `RootViewController.xib` im Interface Builder öffnen. In dieser Datei finden Sie einen Dateibesitzer, einen First Responder und eine Tabellenansicht (`Table View`). Für ein `Navigation Application`-Projekt in Xcode wird vorausgesetzt, dass Sie einen Tabellenansichtscontroller verwenden, aber in dieser Anleitung brauchen Sie einen `UIViewController`. Schließen Sie jetzt den Interface Builder, um den Tabellen- durch einen Ansichtscontroller zu ersetzen.

Markieren Sie in Xcode die Dateien `RootViewController.xib`, `RootViewController.h` und `RootViewController.m`, und löschen Sie sie, indem Sie die Löschtaste drücken. Wählen Sie **Also Move to Trash**. Dadurch entfernen Sie die tabellengestützten Vorgaben, mit denen Sie angefangen haben.

Wählen Sie **File ▸ New** ⌘ + N **▸ iPhone OS ▸ Cocoa Touch Class ▸ UIViewController subclass**. Aktivieren Sie **With XIB for User Interface**, und klicken Sie auf **Next**. Nennen Sie die Datei `RootViewController.m`, aktivieren Sie **also Create Root View Controller.h**, legen Sie als Speicherort den Hauptprojektordner fest, und klicken Sie auf **Finish**. Dadurch wird eine neue ansichtsgestützte Version der drei `RootViewController`-Dateien erstellt, die Sie für das Projekt benötigen – also eine `.xib`-Datei, eine `.h`-Headerdatei und eine `.m`-Implementierungsdatei. Anschließend ziehen Sie die neuen Klassendateien in die Gruppe **Classes** und die neue `.xib`-Datei in die Gruppe **Resources**.

Hinweis

Ich verwende Projekte mit Navigationscontrollern so häufig, dass ich meine eigene Vorlage erstellt habe, anstatt jedes Mal die Version mit der Tabellenansicht zu korrigieren. Anleitungen dafür, wie Sie in Xcode eigene Benutzervorlagen erstellen, finden Sie in Kapitel 2.

4.5.6 Simulierte Elemente hinzufügen

Doppelklicken Sie auf die neue `RootViewController.xib`-Datei, um sie im Interface Builder zu öffnen, und doppelklicken Sie dort auf **View**. Dadurch wird der Ansichtseditor geöffnet, der zu Anfang nur eine leere Ansicht zeigt, allenfalls mit einer Statusleiste. Bevor Sie fortfahren können, müssen Sie zunächst ein simuliertes Element hinzufügen, um sicherzustellen, dass der Entwurfsraum nicht mit den auf dem Bildschirm dargestellten Elementen in Konflikt gerät.

Öffnen Sie das Informationsfenster für Attribute ⌘ + 1. Die Statusleiste sollte bereits ausgewählt sein und als Attribut `Gray` anzeigen, anderenfalls müssen Sie selbst dafür sorgen. Wählen Sie dann **Top Bar ▸ Navigation Bar**, wodurch der Ansicht ein Platzhalter für eine Navigationsleiste hinzugefügt wird. Wählen Sie keine untere Leiste aus. Diese simulierten Elemente blockieren Teile des Bildschirms und beschränken Ihren Entwurfsraum auf die verbleibenden Bereiche.

4.5.7 Ein Hintergrundbild hinzufügen

Wenn Sie ein Bild in den Editor ziehen, wird es automatisch vergrößert, sodass es den gesamten verfügbaren Platz der Ansicht ausfüllt. Das Bild wird automatisch positioniert und bedeckt den Bereich unterhalb der Navigationsleiste.

Bevor Sie ein Bild in den Editor ziehen können, müssen Sie der zu bearbeitenden Ansicht das *Image View*-Element aus der Bibliothek hinzufügen. Dieses dient als Container für Bilder. Im Attribute-Informationsfenster wählen Sie dann in der Dropdown-Liste **Image** das Bild `cover320x416.png` aus. In dieser Liste sind alle in dem Xcode-Projekt verfügbaren Grafiken aufgeführt. (Um weitere Bilder hinzuzufügen, ziehen Sie sie in Xcode in das Projekt.) Nachdem Sie das `.png`-Bild ausgewählt haben, wird die Ansicht des Editors aktualisiert und zeigt die Grafik an.

Wählen Sie im Attribute-Informationsfenster anschließend **Interaction ▸ User Interaction Enabled**. Dies ist ein sehr wichtiger Schritt, der die Interaktion der Benutzer mit den Unteransichten ermöglicht. Wenn Sie ein Hintergrundbild verwenden, müssen Sie die Interaktion immer aktivieren. Dieser kleine »Haken« hat schon viele Entwickler geärgert, die diesen Schritt vergessen haben.

4.5.8 Beschriftungen und Ansichten hinzufügen

Ziehen Sie aus der Bibliothek zwei Textfelder und zwei Beschriftungen (*Labels*) in die Ansicht, und platzieren Sie sie ungefähr so wie in *Abbildung 4.8*. Doppelklicken Sie anschließend auf die Beschriftungen, und ändern Sie den Text oben in Fahrenheit und unten in Celsius.

Es ist wichtig, dass Sie genau angeben, wie die einzelnen Textfelder jeweils mit den Benutzern interagieren sollen. Neben anderen Merkmalen können Sie auswählen, welche Tastatur angezeigt wird, ob in dem Textfeld eine Eingabeaufforderung erscheint, ob die Wörter automatisch korrigiert oder automatisch in Großbuchstaben umgewandelt werden sollen usw.

▸ *Abbildung 4.8: Textfelder und Beschriftungen anordnen*

Markieren Sie das obere Textfeld, und wählen Sie im Attribut-Informationsfenster **Text Input Traits ▸ Keyboard ▸ Numbers & Punctuation**. Dadurch wird eine numerische Tastatur eingeblendet, wenn der Benutzer auf das obere Feld tippt.

Markieren Sie das untere Feld, und deaktivieren Sie **Control ▸ Content ▸ Enabled**. Das untere Feld soll die Ergebnisse anzeigen und darf nicht von den Benutzern bearbeitet werden können.

Hinweis

Je mehr Elemente Sie im Interface Builder hinzufügen, umso schwieriger wird es, das gewünschte durch einen Klick auszuwählen. Ein praktischer Trick besteht darin, in einem Bearbeitungsfenster des Interface Builder bei gedrückter Ctrl- und ⇧-Taste auf eine beliebige Ansicht zu klicken, woraufhin eine Liste aller Ansichten gezeigt wird, die sich zurzeit im Stapel befinden. Anschließend können Sie aus dieser Liste das gewünschte Element auswählen.

4.5.9 Die Schnittstelle testen

Speichern Sie die Änderungen, und kehren Sie zu Xcode zurück. Wählen Sie **Project ▸ Set Active SDK ▸ iPhone Simulator** und dann **Run ▸ Run**, oder klicken Sie auf **Build and Go**, um das Projekt im jetzigen Zustand zu kompilieren und im Simulator auszuführen. Stellen Sie dort sicher, dass sich beim Antippen des oberen Feldes eine numerische Tastatur öffnet, während das untere Feld nicht bearbeitet werden kann. Sie können auch auf die Schaltfläche **Convert** klicken, aber zurzeit geschieht dabei noch nichts. Sofern das Projekt in einem solchen Zustand ist, dass es sich kompilieren lässt, können Sie die jeweils aktuelle Form im Simulator oder auf einem Gerät testen.

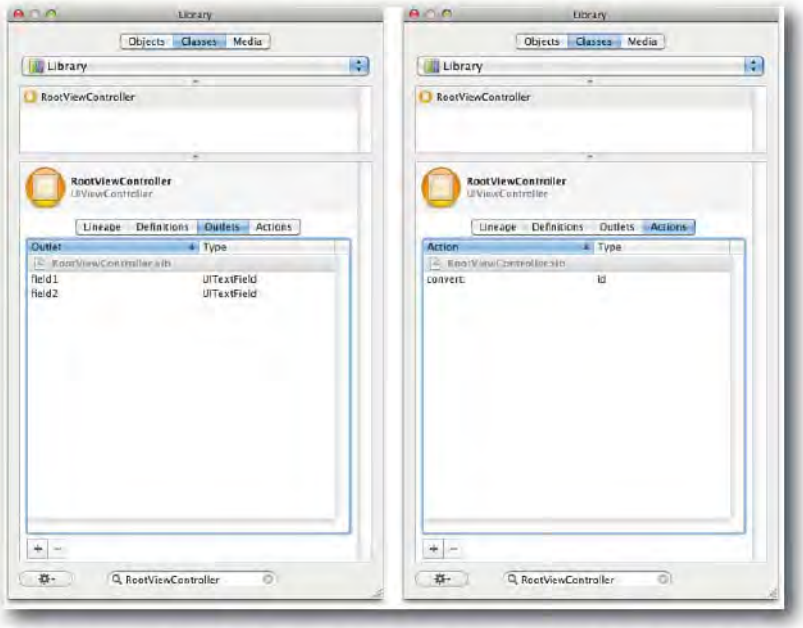
Es ist zwar auch möglich, eine Schnittstelle direkt im Interface Builder zu testen (mit **File ▸ Simulate Interface** oder ⌘ + R), doch ist diese Überprüfung weniger zuverlässig als ein Test in Xcode.

4.5.10 Outlets und Aktionen hinzufügen

Eine wichtige Rolle bei der Gestaltung im Interface Builder spielen Outlets und Aktionen. *Outlets* verbinden Schnittstellen mit Objekten und dienen im Grunde genommen als Platzhalter für Instanzvariablen. *Aktionen* sind Methoden, die eine im Interface Builder erstellte Schnittstelle aufrufen kann. Sie koppeln Ziele mit Vorgängen und senden Callbacks von Steuerelementansichten an Objekte. In diesem Projekt erstellen wir zwei Outlets und eine Aktion.

Bei der Verwendung von Xcode 3.1 wählen Sie im Interface Builder `File's Owner` und öffnen das Identitäts-Informationsfenster ⌘ + 4. Fügen Sie zwei Outlets hinzu, indem Sie im Bereich *Class Outlets* auf die Schaltfläche **+** klicken, und nennen Sie sie `field1` und `field2`. Xcode 3.2-Nutzer können wie in *Abbildung 4.9* gezeigt in der Bibliothek den Titel *Classes* auswählen und in der darunter erscheinenden Klassenliste *RootViewController* auswählen, um dann im Bereich *Outlets* ebenfalls über die **+**-Schaltfläche die neuen *Outlets* hinzuzufügen. Standardmäßig weisen Outlets den Typ `id` auf. Ändern Sie den Typ in `UITextField`. Der Typ legt fest, mit was für Ansichtsobjekten ein Outlet verbunden werden kann. Wenn Sie `UITextField` wählen, ist eine Verbindung mit den beiden Textfeldern in der Ansicht möglich, aber beispielsweise nicht mit Beschriftungen.

Als Nächstes fügen Sie eine Aktion hinzu. Xcode3.1-Nutzer klicken dazu im Bereich *Class Actions* wiederum auf **+**, während Xcode 3.2-Benutzer im Bereich *Actions* die Aktion hinzufügen. Ändern Sie den vorgegebenen Aktionsnamen in `convert:` (und vergessen Sie dabei nicht den Doppelpunkt!).



▸ *Abbildung 4.9: Xcode 3.2-Nutzer fügen ihren Klassen im Bibliotheksfenster Outlets und Actions hinzu.*

Der Interface Builder kann aus den von Ihnen hinzugefügten Aktionen und Outlets Klassendateien erstellen. Speichern Sie das Projekt, wählen Sie `File's Owner` und dann **File ▸ Write Class Files**. Behalten Sie den Klassennamen `RootViewController` (ohne Dateierweiterung) bei, und spei-

chern Sie die Datei im Hauptordner, nicht im Unterordner `Classes`. Als Sie `RootViewController` neu erstellt haben, wurden sowohl die `.xib`-Datei als auch die Klassendateien zum Hauptprojektordner hinzugefügt. Klicken Sie auf **Replace**, um diese Dateien zu ersetzen. Wenn Sie die Klasse `RootViewController` bereits angepasst haben und sie danach vom Interfacer Builder überschreiben lassen, können Daten verloren gehen.

Hinweis

Der Interface Builder kann Klassenheaderdateien aus Xcode lesen (**File ▸ Read Class Files**), Sie können aber auch Headerdateien in das Interface Builder-Dokument aufnehmen. Dadurch können Sie Instanzen eigener Klassen hinzufügen und Objekte zu ihnen zuweisen, ohne das Identitäts-Informationsfenster zu bemühen. Wenn Ihre `.xib`-Datei »vergisst«, zu welcher Klasse der Dateibesitzer gehört (was sich gewöhnlich in der Warnung zeigt, dass das Ansichts-Outlet verbunden wird, aber nicht mehr definiert ist), importieren Sie einfach Ihren Header der Ansichtscontrollerklasse erneut.

4.5.11 Den neuen Klassenheader untersuchen

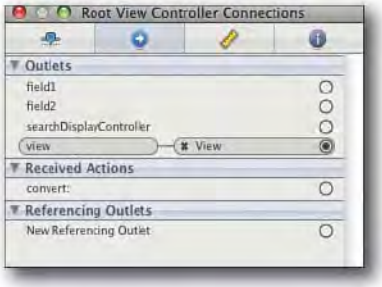
Öffnen Sie `RootViewController.h` in Xcode. Die frisch generierte Klassenschnittstelle enthält die Outlets und Aktionen, die Sie im Interface Builder erstellt haben. Sowohl `field1` als auch `field2` sind wie von Ihnen verlangt als `UITextField`-Instanzen typisiert und mit dem Schlüsselwort `IBOutlet` deklariert. Dieses Schlüsselwort gibt an, dass die Instanzvariable so eingerichtet wird, dass sie beim Laden des Ansichtscontrollers mit einem IB-Element übereinstimmt. Die Aktion `convert:` hat den Typ `IBAction`, was im Grunde genommen `(void)` entspricht.

```
@interface RootViewController : UIViewController {
    IBOutlet UITextField *field1;
    IBOutlet UITextField *field2;
}
- (IBAction)convert:(id)sender;
@end
```

Sie haben jetzt zwar diese beiden Outlets und die Aktion in der Klasse `RootViewController` definiert, aber noch keine Zuordnungen getroffen, die sie mit Elementen des Ansichtsobjekts verbinden. Es ist nun an der Zeit, dies nachzuholen.

4.5.12 Verbindungen hinzufügen

Wählen Sie im Interface Builder `File's Owner`, und öffnen Sie das Verbindungs-Informationsfenster ⌘ + 2. Dieses Dialogfeld, das Sie in *Abbildung 4.10* sehen, führt alle verfügbaren Outlets und Aktionen auf. Die leeren Kreise auf der rechten Seite zeigen an, dass die drei von Ihnen hinzugefügten Elemente noch nicht zugeordnet sind.



▶ *Abbildung 4.10: Leere Kreise kennzeichnen Outlets und Aktionen, die noch nicht mit Objekten verbunden sind.*

Ziehen Sie eine Verbindung vom Kreis von `field1` zum oberen Textfeld und vom Kreis von `field2` zum unteren. Mit diesen Verbindungen legen Sie fest, auf welche Objekte die einzelnen IB-Outlets verweisen. Speichern Sie die Änderungen.

Öffnen Sie `MainWindow.xib`, und doppelklicken Sie auf `Navigation Controller`, um das Editorfenster aus *Abbildung 4.6* zu öffnen. Ziehen Sie bei gedrückter Ctrl-Taste eine Verbindung von der Schaltfläche **Convert** zur Ansicht in der Mitte. Durch das Ziehen bei gedrückter Ctrl-Taste erstellen Sie ebenso Verbindungen wie durch das Ziehen der in *Abbildung 4.10* gezeigten Kreise. Lassen Sie die Maustaste los, wenn die Mittelansicht dunkler wird. Wie in *Abbildung 4.11* gezeigt, erscheint das Popup-Menü **Sent Actions**. Wählen Sie `convert:` aus, die einzige verfügbare Aktion, die zurzeit in `RootViewController` definiert ist. Speichern Sie.

▶ *Abbildung 4.11: Ziehen Sie eine Verbindung von dem Schaltflächenelement in der Leiste zur Mittelansicht, um die Schaltflächenaktion mit einer Methode zu verbinden, die vom Controller der Ansicht definiert wird.*

Hinweis

Wenn Sie bei gedrückter Ctrl-Taste (oder mit der rechten Maustaste) auf ein Objekt klicken, öffnet sich ein Popup-Menü mit vielen der Optionen, die im Verbindungs-Informationsfenster angezeigt werden.

Es widerspricht der Intuition, die Aktion einer Schaltfläche mit einer Ansicht zu verbinden, da die aufgerufene Methode nicht durch die Ansicht, sondern durch deren Controller definiert wird. Außerdem wirkt es merkwürdig, eine Schaltfläche, die in einer `.xib`-Datei definiert ist, mit einem Objekt aus einer anderen solchen Datei zu verbinden. Dies sind jedoch leider Eigenheiten des Interface Builder, mit denen Sie leben müssen.

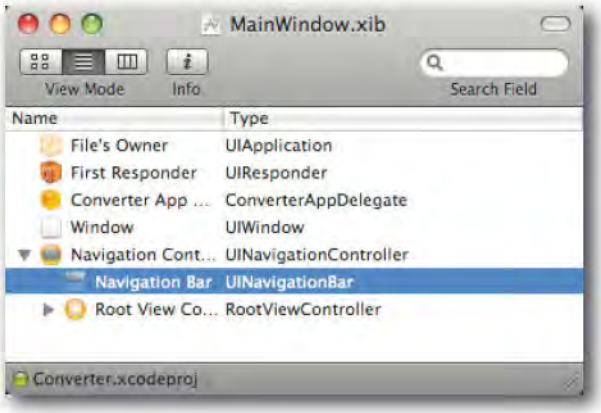
4.5.13 Die richtige Farbe aufnehmen

Der Entwurf für dieses Projekt sieht eine Navigationsleiste vor, deren Färbung zu der Hintergrundgrafik passt. Dazu müssen Sie die richtige Purpurnuance finden. Wechseln Sie wieder zu `RootViewController.xib`, und öffnen Sie das Editorfenster für `View`, in dem die Grafik `cover320x416.png` angezeigt wird.

Öffnen Sie das Farb-Informationsfenster (**Font ▸ Show Colors**), klicken Sie das Vergrößerungsglas an, bewegen Sie dieses über die purpurne Leiste in der Ansichtsgrafik, und klicken Sie. Dadurch wird der Purpurton dieser Ansicht aufgenommen und als aktuelle Farbe im Farb-Informationsfenster festgelegt.

Schließen Sie `RootViewController.xib`, und wechseln Sie wieder zu `MainWindow.xib`. Oben links in diesem Fenster, oberhalb des Symbols für `File's Owner`, finden Sie die Optionen für den Ansichtsmodus (**View Mode**). Klicken Sie auf die mittlere dieser drei Schaltflächen. Statt der Symbolansicht wird jetzt eine Liste angezeigt.

Öffnen Sie das Einblenddreieck neben `Navigation Controller` und wählen Sie `Navigation Bar`, wie Sie in *Abbildung 4.12* sehen können.



▸ *Abbildung 4.12: Um die Farbe der Navigationsleiste zu ändern, müssen Sie Elemente öffnen, die im Editorfenster nicht unmittelbar zugänglich sind.*

Öffnen Sie, während das Element `Navigation Bar` markiert ist, das Attribut-Informationsfenster ⌘ + 1. Stellen Sie sicher, dass die Ansicht (`View`) aus `RootViewController.xib` auf dem Bildschirm zu sehen ist. Ziehen Sie den übernommenen Purpurton aus der oberen Leiste der Palette **Colors** auf das Farbfeld (*Tint*) des Attribut-Informationsfensters. Speichern Sie die Datei, und schließen Sie `MainWindow.xib`.

4.5.14 Die Umrechnungsmethode definieren

Das Projekt ist jetzt komplett gestaltet und »verdrahtet«. Die Outlets sind mit dem Textfeld verbunden, die Schaltfläche mit der Aktion `convert:`. Allerdings tut diese Aktion noch nichts. Wechseln Sie wieder zu Xcode, und öffnen Sie `RootViewController.m`. Die Methode ist bis jetzt nur ein Gerüst, das keinerlei Handlung ausführt.

```
@implementation RootViewController
- (IBAction) convert: (id) sender {
}
@end
```

Die Methode in diesem Projekt soll Text, der in das obere Feld eingegeben wird, abrufen, in einen Fließkommawert umwandeln, dann von Fahrenheit in Celsius umrechnen und das Ergebnis im zweiten Textfeld platzieren. Die folgende Methode erledigt genau dies und blendet darüber hinaus nach der Umrechnung die Tastatur aus, indem sie `resignFirstResponder` aufruft. Fügen Sie diese Methode zu Ihrem Code hinzu, und speichern Sie.

```
- (IBAction) convert: (id) sender
{
    float invalue = [[field1 text] floatValue];
    float outvalue = (invalue - 32.0f) * 5.0f / 9.0f;
    [field2 setText:[NSString stringWithFormat:@"%3.2f", outvalue]];
    [field1 resignFirstResponder];
}
```

4.5.15 Die Anwendung ausführen

Nachdem Sie das Projekt jetzt vollständig bearbeitet haben, führen Sie das Programm erneut mit **Run ▸ Run** aus. Die Anwendung rechnet jetzt Fahrenheit-Werte in Celsius um. Dies können Sie beispielsweise mit den Werten 32 (0 °C), 98,6 (37 °C) und 212 (100 °C) testen. Bei der Eingabe der Fahrenheit-Werte verwenden Sie bitte einen Punkt anstelle eines Kommas als Dezimaltrennzeichen.

4.6 Schritt für Schritt: Die Schnittstelle für das Umrechnungsprogramm manuell erstellen

Alles, was Sie im Interface Builder entwerfen, können Sie auch direkt mit Objective-C und Cocoa Touch implementieren. Der Code in *Listing 4.1* ergibt das gleiche Beispielprojekt wie das, das Sie gerade erstellt haben. Anstatt die Schnittstelle aus einer `.xib`-Datei zu laden, gestalten Sie die Elemente manuell in der Methode `loadView`.

Der Code folgt dem bekannten Ansatz. Nach dem Erstellen einer Hintergrundansicht (entspricht `View` im Interface Builder-Projekt) fügt er darüber eine Bildansicht hinzu. Hierfür wird dieselbe Grafikdatei verwendet (`cover320x416.png`), und ebenfalls wie in Interface Builder ist das Flag `userInteractionEnabled` auf `YES` gesetzt.

Als Nächstes werden zwei Beschriftungen und zwei Textfelder hinzugefügt. Der Code setzt die Beschriftungen auf **Fahrenheit** und **Celsius**, ordnet dem ersten Feld die Verwendung der Tastatur für Zahlen und Satzzeichen zu und deaktiviert das zweite. Für die Position und Größe dieser Elemente werden die Ansichtsabmessungen aus der vorherigen Anleitung verwendet.

Um das Layout abzuschließen, färbt der Code die Navigationsleiste purpurn und fügt die Schaltfläche **Convert** hinzu. Diese Schaltfläche verwendet ebenso den Callback `convert:` wie im Interface Builder-Projekt und ruft denselben Code auf.

```
#import <UIKit/UIKit.h>
#define COOKBOOK_PURPLE_COLOR [UIColor colorWithRed:0.20392f
green:0.19607f \
   blue:0.61176f alpha:1.0f]
#define BARBUTTON(TITLE, SELECTOR) [[[UIBarButtonItem alloc] \
   initWithTitle:TITLE style:UIBarButtonItemStylePlain target:self \
   action:SELECTOR] autorelease]

@interface HelloWorldController : UIViewController {
   UITextField *field1;
   UITextField *field2;
}
-(void) convert: (id)sender;
@end

@implementation HelloWorldController
- (void) convert: (id) sender
{
   float invalue = [[field1 text] floatValue];
   float outvalue = (invalue - 32.0f) * 5.0f / 9.0f;
   [field2 setText:[NSString stringWithFormat:@"%3.2f", outvalue]];
   [field1 resignFirstResponder];
}
- (void)loadView
{
   UIView *contentView = [[UIView alloc] initWithFrame:
      [[UIScreen mainScreen] applicationFrame]];
   self.view = contentView;
   contentView.backgroundColor = [UIColor whiteColor];
   [contentView release];

   UIImageView *iv = [[UIImageView alloc] initWithImage:
```

```
    [UIImage imageNamed:@"cover320x416.png"]];
[self.view addSubview:iv];
iv.userInteractionEnabled = YES;

field1 = [[UITextField alloc] initWithFrame:
    CGRectMake(185.0, 31.0, 97.0, 31.0)];
field1.borderStyle = UITextBorderStyleRoundedRect;
field1.keyboardType = UIKeyboardTypeNumbersAndPunctuation;
field1.contentVerticalAlignment =
    UIControlContentVerticalAlignmentCenter;

field2 = [[UITextField alloc] initWithFrame:
    CGRectMake(185.0, 97.0, 97.0, 31.0)];
field2.borderStyle = UITextBorderStyleRoundedRect;
field2.enabled = NO;
field2.contentVerticalAlignment =
    UIControlContentVerticalAlignmentCenter;

UILabel *label1 = [[UILabel alloc] initWithFrame:
    CGRectMake(95.0, 34.0, 82.0, 21.0)];
label1.text = @"Fahrenheit";
label1.textAlignment = UITextAlignmentLeft;
label1.textColor = [UIColor colorWithRed:0.000 green:0.000
    blue:0.000 alpha:1.000];
label1.backgroundColor = [UIColor clearColor];

UILabel *label2 = [[UILabel alloc] initWithFrame:CGRectMake(121.0,
    102.0, 56.0, 21.0)];
label2.text = @"Celsius";
label2.textAlignment = UITextAlignmentLeft;
label2.textColor = [UIColor colorWithRed:0.000 green:0.000
    blue:0.000 alpha:1.000];
label2.backgroundColor = [UIColor clearColor];

[iv addSubview:field1];
[iv addSubview:field2];
[iv addSubview:label1];
[iv addSubview:label2];

[field1 release];
[field2 release];
[label1 release];
[label2 release];

[iv release];
```

```
    self.title = @"Converter";
    self.navigationItem.rightBarButtonItem = BARBUTTON(@"Convert",
        @selector(convert:));
    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
}
@end

@interface TestBedAppDelegate : NSObject <UIApplicationDelegate>
@end

@implementation TestBedAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    UIWindow *window = [[UIWindow alloc] initWithFrame:
        [[UIScreen mainScreen] bounds]];
    UINavigationController *nav = [[UINavigationController alloc]
        initWithRootViewController:[[HelloWorldController alloc] init]];
    [window addSubview:nav.view];
    [window makeKeyAndVisible];
}
@end

int main(int argc, char *argv[])
{
    NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
    int retVal = UIApplicationMain(argc, argv, nil,
        @"TestBedAppDelegate");
    [pool release];
    return retVal;
}
```

► *Listing 4.1: Programm zur Temperaturumrechnung mit manuell erstelltem Code*

4.6.1 Das Projekt zusammenstellen

Um dieses Projekt zu erstellen, müssen Sie eine der mitgelieferten Vorlagen von Xcode anpassen. Wählen Sie **File ▸ New Project** ⌘ + ⇧ + N **▸ iPhone OS ▸ Application ▸ Window-based Application**, klicken Sie auf **Choose**, nennen Sie das Projekt `HelloWorld2`, und sichern Sie es auf dem Schreibtisch.

Dieses vorlagengestützte Projekt erfordert einige Arbeiten an den Dateien. Löschen Sie `MainWindow.xib`, und wählen Sie dabei **Also Move to Trash**. Öffnen Sie als Nächstes `HelloWorld2-Info.plist`, löschen Sie dort die Zeile `Main nib file base name`, und speichern Sie die Änderungen. Dadurch machen Sie das Projekt unabhängig von einer Schnittstelle aus einer `.xib`-Datei.

Löschen Sie auch die Gruppe **Classes** mit den beiden darin enthaltenen Quelldateien, und wählen Sie **Also Move to Trash**. Dadurch entfernen Sie den Code aus der ursprünglichen `.xib`-Datei, sodass Sie Ihren eigenen Code einfügen können.

Kopieren Sie wie in der vorherigen Schritt-für-Schritt-Anleitung die drei Bilddateien `icon.png`, `Default.png` und `cover320x416.png` aus dem Beispielcode. Aktivieren Sie **Copy Items into Destination Group's Folder (If Needed)**, bevor Sie auf **Add** klicken. Verschieben Sie diese Dateien in die Gruppe **Resources** des Projekts.

Schließlich öffnen Sie die Datei `main.m`, fügen den Code aus *Listing 4.1* ein (Sie finden ihn im Ordner mit dem Beispielcode), kompilieren das Projekt und führen es im Simulator aus. Die Anwendung sieht genauso aus wie die Interface Builder-Version und verhält sich auch genauso. Anstatt die Schnittstelle aus einer `.xib`-Datei zu laden, haben Sie sie in dieser Version in der Implementierung der Ansichtscontrollerklasse durch Programmbefehle erstellt.

4.7 Schritt für Schritt: Ein kombinierter Ansatz zum Erstellen des Umrechnungsprogramms

Eine der schönen Eigenschaften von Cocoa Touch ist, dass Sie ein Programm nicht vollständig manuell schreiben oder vollständig im Interface Builder gestalten müssen. Sie können die grafischen Layoutmöglichkeiten des Interface Builder nutzen und mit der Programmierung in Xcode kombinieren, um eine bessere Lösung zu erstellen. Dabei verquicken Sie das statische Laden von `.xib`-Dateien aus dem Interface Builder mit dem besser wiederverwendbaren, codegesteuerten Ansatz zum dynamischen Laden. Sie können hierbei auf zwei verschiedene Weisen vorgehen: Entweder erstellen Sie im Interface Builder eine komplette, von `UIViewController` abgeleitete Klasse, oder Sie schreiben Ihre eigenen Ansichtscontrollerklassen und laden eine im Interface Builder entworfene `UIView`. In dieser Anleitung lernen Sie das erste Verfahren kennen, in der nächsten das zweite.

Wenn Xcode eine `.xib`-Datei findet, deren Name mit dem einer von `UIViewController` abgeleiteten Klasse übereinstimmt, lädt es beim Initialisieren einer Instanz automatisch diese Datei. Nehmen wir z. B. an, dass Sie einen neuen Navigationscontroller erstellen, und initialisieren wir dessen Wurzelansichtscontroller wie folgt:

```
UINavigationController *nav = [[UINavigationController alloc]
initWithRootViewController:[[RootViewController alloc] init]];
```

Wenn in dem Projekt eine Datei namens `RootViewController.xib` enthalten ist, verwendet Xcode sie, um die Ansicht des Ansichtscontrollers einzurichten. Wie der Name der Ansichtscontrollerklasse lautet, spielt dabei keine Rolle, sondern nur, dass es eine entsprechende `.xib`-Datei gibt. In dieser Anleitung nutzen wir dieses Verhalten, um eine Schnittstelle zu initialisieren.

4.7.1 Eine Standardvorlage anpassen

Wie beim vorherigen Projekt müssen Sie zu Anfang eine der mitgelieferten Vorlagen anpassen. Wählen Sie **File ▸ New Project** ⌘ + ⇧ + N **▸ iPhone OS ▸ Application ▸ Window-based Application**, klicken Sie auf **Choose**, nennen Sie das Projekt `HelloWorld3`, und sichern Sie es auf dem Schreibtisch.

Löschen Sie `MainWindow.xib`, und wählen Sie dabei **Also Move to Trash**. Öffnen Sie als Nächstes `HelloWorld3-Info.plist`, löschen Sie dort die Zeile `Main nib file base name`, und speichern Sie die Änderungen. Löschen Sie im Projektfenster die Gruppe **Classes** mit den beiden darin enthaltenen Quelldateien, und wählen Sie **Also Move to Trash**.

Kopieren Sie schließlich die drei Bilddateien `icon.png`, `Default.png` und `cover320x416.png` aus dem Beispielcode. Aktivieren Sie **Copy Items into Destination Group's Folder (If Needed)**, bevor Sie auf **Add** klicken. Verschieben Sie diese Dateien in die Gruppe **Resources** des Projekts.

4.7.2 Eine neue Ansichtscontrollerklasse mit .xib-Datei erstellen

Wählen Sie in Xcode **File ▸ New** ⌘ + N **▸ iPhone OS ▸ Cocoa Touch Class ▸ UIViewController subclass**, aktivieren Sie **With XIB for User Interface**, und klicken Sie auf **Next**. Nennen Sie die Datei `RootViewController.m`, aktivieren Sie **Also Create RootViewController.h**, legen Sie als Speicherort den Hauptprojektordner fest, und klicken Sie auf **Finish**.

Den Namen der Klasse können Sie beliebig wählen. Ich habe hier `RootViewController` gewählt, weil das die Rolle des Ansichtscontrollers beschreibt und das der Name ist, den ich bereits in den vorherigen Anleitungen verwendet habe.

4.7.3 Die Schnittstelle entwerfen

Wie in der ersten Anleitung doppelklicken Sie auf die neue Datei `RootViewController.xib`, um sie im Interface Builder zu öffnen. Dort doppelklicken Sie auf `View`, um den Ansichtseditor zu öffnen, und führen die folgenden Schritte aus:

1. Während `View` markiert ist, öffnen Sie das Attribut-Informationsfenster ⌘ + 1 und wählen **Top Bar ▸ Navigation Bar**.
2. Ziehen Sie eine Bildansicht in den Editor, und lassen Sie sie einrasten, sodass sie die Fläche unterhalb der Navigationsleiste ausfüllt.
3. Wählen Sie im Attribut-Informationsfenster in der Dropdown-Liste **Image** das Bild `cover320x416.png` aus, und aktivieren Sie **Interaction ▸ User Interaction Enabled**.
4. Ziehen Sie aus der Bibliothek zwei Textfelder und zwei Beschriftungen, und richten Sie sie so aus, dass das Layout ungefähr dem in *Abbildung 4.8* entspricht. Ändern Sie den Text der ersten Beschriftung in »Fahrenheit« und der zweiten in »Celsius«.
5. Markieren Sie das obere Textfeld, und wählen Sie im Attribut-Informationsfenster **Text Input Traits ▸ Keyboard ▸ Numbers & Punctuation**.
6. Markieren Sie das untere Feld, und deaktivieren Sie **Control ▸ Content ▸ Enabled**.
7. Wählen Sie `File's Owner`, und öffnen Sie das Identitäts-Informationsfenster ⌘ + 4. Fügen Sie zwei Outlets hinzu, indem Sie im Bereich *Outlets* auf + klicken, und nennen Sie sie `field1` und `field2`. Ändern Sie bei beiden Outlets den Typ von `id` in `UITextField`.
8. Fügen Sie im Bereich *Actions* ebenfalls mithilfe von + eine Aktion hinzu, und benennen Sie sie in `convert:` um. Vergessen Sie nicht den Doppelpunkt!

9. Öffnen Sie das Verbindungs-Informationsfeld ⌘ + 2. Ziehen Sie eine Verbindung vom Kreis von `field1` zum oberen Textfeld und vom Kreis von `field2` zum unteren.
10. Speichern Sie das geänderte Projekt.
11. Wählen Sie `File's Owner` und dann **File ▸ Write Class Files**. Behalten Sie den Dateinamen `RootViewController` (ohne Erweiterung) bei, und speichern Sie ihn im Hauptordner des Xcode-Projekts. Bestätigen Sie, dass bestehende Dateien ersetzt werden sollen.
12. Schließen Sie die Datei `RootViewController.xib`, und wechseln Sie wieder zu Xcode.

4.7.4 Die Implementierung des Ansichtscontrollers bearbeiten

Öffnen Sie in Xcode die Datei `RootViewController.m` und ersetzen Sie den Inhalt dieser Datei durch den folgenden Code. Dieser Code fügt die Methode `convert:` hinzu, die wir bereits in den beiden vorherigen Anleitungen verwendet haben, aber auch eine neue Methode namens `viewDidLoad`. Sie wird nach dem Laden der `.xib`-Datei aufgerufen und gibt dem Controller die Gelegenheit, die Initialisierung abzuschließen. In diesem Beispiel gibt sie den Titel an (»Converter«), fügt ein Schaltflächenelement zur Navigationsleiste hinzu (»convert«), legt den Callback fest (`convert:`) und färbt die Leiste purpurn ein.

```
#import "RootViewController.h"
#define COOKBOOK_PURPLE_COLOR [UIColor colorWithRed:0.20392f
green:0.19607f \
   blue:0.61176f alpha:1.0f]
#define BARBUTTON(TITLE, SELECTOR) [[[UIBarButtonItem alloc] \
   initWithTitle:TITLE style:UIBarButtonItemStylePlain target:self \
   action:SELECTOR] autorelease]

@implementation RootViewController
- (IBAction) convert: (id) sender
{
   float invalue = [[field1 text] floatValue];
   float outvalue = (invalue - 32.0f) * 5.0f / 9.0f;
   [field2 setText:[NSString stringWithFormat:@"%3.2f", outvalue]];
   [field1 resignFirstResponder];
}

- (void) viewDidLoad
{
   self.title = @"Converter";
   self.navigationItem.rightBarButtonItem = BARBUTTON(@"Convert",
      @selector(convert:));
   self.navigationController.navigationBar.tintColor =
      COOKBOOK_PURPLE_COLOR;
}
@end
```

4.7.5 »main.m« bearbeiten

Im letzten Schritt bearbeiten Sie `main.m` und ersetzen den Inhalt dieser Datei durch den folgenden Code, der das Hauptfenster und den Navigationscontroller einrichtet sowie eine neue Instanz von `RootViewController` als Wurzelansichtscontroller des Navigationscontrollers zuweist.

```
#import <UIKit/UIKit.h>
#import "RootViewController.h"

@interface HelloWorldAppDelegate : NSObject <UIApplicationDelegate>
@end

@implementation HelloWorldAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application {
   UIWindow *window = [[UIWindow alloc] initWithFrame:
      [[UIScreen mainScreen] bounds]];
   UINavigationController *nav = [[UINavigationController alloc]
      initWithRootViewController:[[RootViewController alloc] init]];
   [window addSubview:nav.view];
   [window makeKeyAndVisible];
}
@end

int main(int argc, char *argv[])
{
   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
   int retVal = UIApplicationMain(argc, argv, nil,
      @"HelloWorldAppDelegate");
   [pool release];
   return retVal;
}
```

4.7.6 Die Anwendung ausführen

Nachdem Sie das Programm nun vollständig bearbeitet haben, führen Sie es mit **Run ▸ Run** bzw. **Build an Go** aus. Die kompilierte Anwendung lädt ihre Schnittstelle aus der `.xib`-Datei. Anschließend werden die Einzelheiten wie die Navigationsleiste, deren Schaltfläche, deren Titel und Farbe finalisiert. Die Anwendung sieht genauso aus wie die in den beiden vorherigen Anleitungen und verhält sich auch genauso.

4.8 Schritt für Schritt: .xib-Daten direkt im Code laden

In Cocoa Touch können Sie Objekte aus einer beliebigen `.xib`-Datei abrufen, indem Sie `loadNibNamed: owner: options:` aufrufen. Dadurch wird ein Array aus Objekten zurückgegeben, das im `.xib`-Bundle initialisiert wird. Sie können es dann nehmen und in Ihrem Programm verwenden. In dieser Anleitung laden Sie auf diese Weise eine im Interface Builder entworfene Schnittstelle in einer ansonsten handkodierten Anwendung. Als Erstes kopieren Sie das Projekt aus der zweiten Anleitung, also derjenigen, in der Sie vollständig den Code erstellt haben. Diesen Code passen Sie anschließend so an, dass er eine aus einer `.xib`-Datei geladene Ansicht lädt.

Die Ansicht, die Sie hierbei verwenden, ist diejenige aus dem ersten Projekt. Kopieren Sie die Datei `RootViewController.xib` aus diesem Projekt, und fügen Sie sie zum neuen Projektordner hinzu. Diese `.xib`-Datei enthält das ursprüngliche Layout mit dem Hintergrundbild, den beiden Textfeldern usw. Benennen Sie die Datei in `mainview.xib` um. Dies ist wichtig, da Sie eine `.xib`-Datei verwenden müssen, deren Name nicht mit dem einer der Hauptansichtscontrollerklassen übereinstimmt. Wenn Sie dies vergessen, versucht die Anwendung, diese Sicht zu laden, was zu allerlei Laufzeitproblemen führt.

Öffnen Sie das Projekt in Xcode, und ziehen Sie die Datei `mainview.xib` aus dem Ordner in das Projekt. Sie können das Markierungsfeld **Copy** aktiviert lassen, auch wenn das nicht notwendig ist, denn die Datei befindet sich bereits in dem Ordner. Klicken Sie auf **Add**.

4.8.1 Die .xib-Datei aufräumen

Öffnen Sie `mainview.xib` im Interface Builder. Hier müssen Sie einige Änderungen vornehmen, damit die `.xib`-Datei korrekt von `loadView` geladen werden kann und Sie Zugriff auf die Unteransichten haben, die normalerweise zu `IBOutlet`-Instanzvariablen zugeordnet wären.

Ansichten mit Tags zu versehen, bedeutet, ihnen Nummern zuzuweisen. Alle Ansichtsklassen verfügen über ein Tag-Feld. Tags sind Integerzahlen, die Sie zur Bezeichnung von Ansichtsinstanzen verwenden können. Dabei wählen Sie aus, welche Nummer verwendet wird. Markieren Sie das obere Textfeld, öffnen Sie das Attribut-Informationsfeld ⌘ + 1, und ändern Sie das Feld **View ▸ Tag** auf `101`. (Möglicherweise müssen Sie nach unten blättern, um dieses Feld zu finden.) Markieren Sie das untere Feld, und ändern Sie dessen Tag-Feld auf `102`. Sobald Ansichten mit Tags versehen sind, können Sie sie von einer Elternansicht abrufen, indem Sie `viewWithTag:` aufrufen.

Entfernen Sie alle Verbindungen, die Sie zuvor im Interface Builder erstellt haben. Wählen Sie `File's Owner`, und öffnen Sie das Verbindungs-Informationsfenster ⌘ + 2. Löschen Sie alle Verbindungen (es sind drei), indem Sie bei allen auf das kleine **X** klicken. Speichern und schließen Sie die `.xib`-Datei. Dadurch stellen Sie sicher, dass die Anwendung nicht versucht, während der Kompilierung irgendwelche Outlet- oder Aktionsverbindungen herzustellen.

4.8.2 »loadView« aktualisieren

Öffnen Sie `main.m`, und ersetzen Sie die Methode `loadView` durch den folgenden Code, der eine Ansicht aus einer `.xib`-Datei lädt und als Hauptansicht für den Ansichtscontroller zuweist. Bei diesem Ansatz nutzen Sie die Tatsache, dass es in der `.xib`-Datei nur ein einziges Ansichtsobjekt gibt. In dieser Datei ist das die Haupt-`UIView`, die im Interface Builder `View` heißt. Weder `File's Owner` noch `First Responder` ist eine Ansicht.

```
- (void)loadView
{
   self.view = [[[NSBundle mainBundle] loadNibNamed:@"mainview"
      owner:self options:NULL] lastObject];
   field1 = (UITextField *)[self.view viewWithTag:101];
   field2 = (UITextField *)[self.view viewWithTag:102];
   self.title = @"Converter";
   self.navigationItem.rightBarButtonItem =
      BARBUTTON(@"Convert", @selector(convert:));
   self.navigationController.navigationBar.tintColor =
      COOKBOOK_PURPLE_COLOR;
}
```

Komplizierte `.xib`-Dateien können mehrere Ansichtsobjekte enthalten. Wenn Sie Ansichten aus `.xib`-Dateien laden, sollten Sie Tags verwenden, um zu markieren, welches Objekt welches ist, sodass Sie das richtige Objekt aus dem zurückgegebenen Array laden können.

Hinweis

Die Reihenfolge der Elemente im `.xib`-Dateiarray ist identisch mit der Reihenfolge im Projektfenster des Interface Builder. Da diese `.xib`-Datei genau ein Element der obersten Ebene enthält, könnten Sie im Code genauso einfach auch `objectAtIndex:0` als `lastObject` verwenden.

Objekte aus einer `.xib`-Datei werden mit einem Beibehaltungswert von 1 erstellt und automatisch freigegeben. Wenn Sie Elemente direkt aus einer `.xib`-Datei laden, müssen Sie Objekte aus dem zurückgegebenen Array, die Sie noch länger brauchen, selbst beibehalten. Durch die Verwendung der Set-Methode `self.view` wird eine Ansicht automatisch beibehalten. Die normale Speicherwarnungslogik für Ansichtscontroller stützt sich darauf, dass alle Ansichten, die zurzeit nicht eingeblendet sind, freigegeben und auf `nil` gesetzt werden. Dadurch werden letztlich alle `.xib`-Dateien aus dem Speicher entfernt.

4.9 Gestaltung für verschiedene Ausrichtungen

Auf dem iPhone müssen Sie mit einer Änderung der Geräteausrichtung rechnen. Eine übliche Frage beim Design lautet, wie die Anwendung auf solche Änderungen reagieren soll. Soll sich die Größe der Elemente auf dem Bildschirm ändern wie in Safari? Werden die Elemente an andere Plätze verschoben, um den veränderten Proportionen Rechnung zu tragen? Möchten Sie eine ganz andere Ansicht darstellen, wie es in den Anwendungen *YouTube* und *iPod/Musik* der Fall ist? Alle diese Varianten sind möglich. Welche Sie auswählen, hängt davon ab, was das Beste für Ihre Anwendung ist und welche grafischen Elemente im Spiel sind.

In den folgenden Abschnitten sehen wir uns diese Designmöglichkeiten an. Sie lernen die automatische Größenanpassung, die manuelle Platzierung der Ansicht sowie den Austausch von Ansichten kennen. Apple hat verlauten lassen, dass es irgendwann eine Unterstützung für getrennte Hoch- und Querformatansichten im SDK geben soll. Zurzeit ist dies aber noch nicht der Fall.

4.10 Die Neuausrichtung ermöglichen

Instanzen von `UIViewController` entscheiden, ob sie auf die Ausrichtung des iPhones reagieren, indem sie die optionale Methode `shouldAutorotateToInterfaceOrientation:` implementieren. Diese Methode gibt entweder `YES` oder `NO` zurück, je nachdem, ob Sie wollen, dass die Ansicht automatisch an die jeweilige Ausrichtung des Geräts angepasst wird. Um die automatische Drehung in alle möglichen Ausrichtungen zu erlauben, geben Sie einfach `YES` zurück.

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation
{
    return YES;
}
```

In dieser Methode können Sie die folgenden möglichen iPhone-Ausrichtungen auswerten:

- `UIDeviceOrientationUnknown`
- `UIDeviceOrientationPortrait`
- `UIDeviceOrientationPortraitUpsideDown`
- `UIDeviceOrientationLandscapeLeft`
- `UIDeviceOrientationLandscapeRight`
- `UIDeviceOrientationFaceUp`
- `UIDeviceOrientationFaceDown`

Von diesen Ausrichtungen wirken sich nur die Hoch- und Querformatvarianten (`Portrait` und `Landscape`) auf die automatische Drehung einer Ansicht aus. Wenn es Ihre Anwendung nur im Hoch- oder Querformat gibt, können Sie sie zwischen den beiden möglichen Ausrichtungen innerhalb des betreffenden Formats umschalten lassen. Der folgende Code nutzt das Symbol `||` für das logische `OR`, um zwei Tests zu einem einzigen Rückgabewert zu kombinieren:

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation
{
   return ((interfaceOrientation == UIDeviceOrientationPortrait) ||
      (interfaceOrientation == UIDeviceOrientationPortraitUpsideDown))
}
```

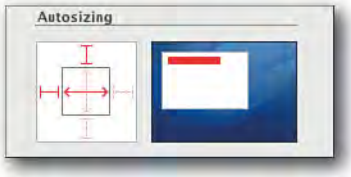
Beim Rückgabewert `YES` verwendet der Ansichtscontroller verschiedene Flags, um festzulegen, wie die automatische Drehung stattfinden soll. Beispielsweise können Sie Unteransichten sowohl horizontal als auch vertikal dehnen.

```
contentView.autoresizesSubviews = YES;
contentView.autoresizingMask = (UIViewAutoresizingFlexibleWidth |
   UIViewAutoresizingFlexibleHeight);
```

Diese Flags entsprechen den Einstellungen, die Sie im Größen-Informationsfenster des Interface Builder ⌘ + 3 festlegen. Mehr darüber erfahren Sie im nächsten Abschnitt.

4.11 Automatische Grössenanpassung

Wenn Sie das iPhone kippen, während Safari läuft, werden die Proportionen der Browseransicht der neuen Ausrichtung entsprechend angepasst. Dies geschieht durch die automatische Größenanpassung (*Autosizing*). Dabei werden einer Ansicht Regeln hinzugefügt, die sie anweisen, wie sie ihre Form zu ändern hat. Sie kann sich dehnen, ihre Größe beibehalten und in einem bestimmten Abstand vom Rand ihres Elternelements fixiert werden. Diese Eigenschaften können Sie manuell im Code festlegen, aber auch im Größen-Informationsfeld des Interface Builder ⌘ + 3, das Sie in *Abbildung 4.13* sehen.



▶ *Abbildung 4.13: Im Bereich* **Autosizing** *des Interface Builder legen Sie eine Maske für die automatische Größenänderung (*`autoresizingMask`*) einer Ansicht fest.*

In diesem Fenster können Sie die Regeln für die automatische Größenanpassung einer Ansicht festlegen. Das Steuerelement besteht aus einem inneren Rechteck mit zwei Doppelpfeilen sowie einem äußeren Rechteck mit vier Linien, die stumpfe Enden aufweisen. Jedes dieser Elemente können Sie mit einem Klick setzen oder aufheben. In aktiviertem Zustand erscheinen diese Elemente in hellem Rot, deaktiviert werden sie in einem dunkleren Rot angezeigt.

Die vier äußeren Linien werden als *Streben* bezeichnet (*struts*). Sie legen einen festen Abstand zwischen einer Ansicht und dem Rand von deren Elternelement fest. Nehmen wir an, dass Sie eine Ansicht mit einem oberen und einem linken Abstand von je 40 Pixel vom Rand der übergeordneten Sicht platzieren. Wenn Sie die obere und die linke Strebe aktivieren (wie in *Abbildung 4.13*), fixieren

Sie die Sicht damit in dieser Position. Um die Abstände unten und links beizubehalten, verwenden Sie die entsprechenden Streben. Damit die Ansicht beim Drehen die festen Seitenabstände einhalten kann, muss sie entweder verschoben werden oder ihre Größe ändern.

Die beiden inneren Linien sind die sogenannten *Federn* (*springs*) und steuern die Größenänderung einer Ansicht. Bei dem Beispiel in *Abbildung 4.13* ist die horizontale Feder aktiviert, sodass die Breite der Ansicht proportional zur Breite der Elternansicht horizontal geändert wird.

Für eine schwimmende Sicht, die weder fixiert ist noch automatisch ihre Größe ändert, deaktivieren Sie alle sechs Streben und Federn. Diese Möglichkeit besteht aber nur bei Unteransichten. Bei der im Interface Builder definierten Hauptansicht müssen beide Federn aktiviert sein.

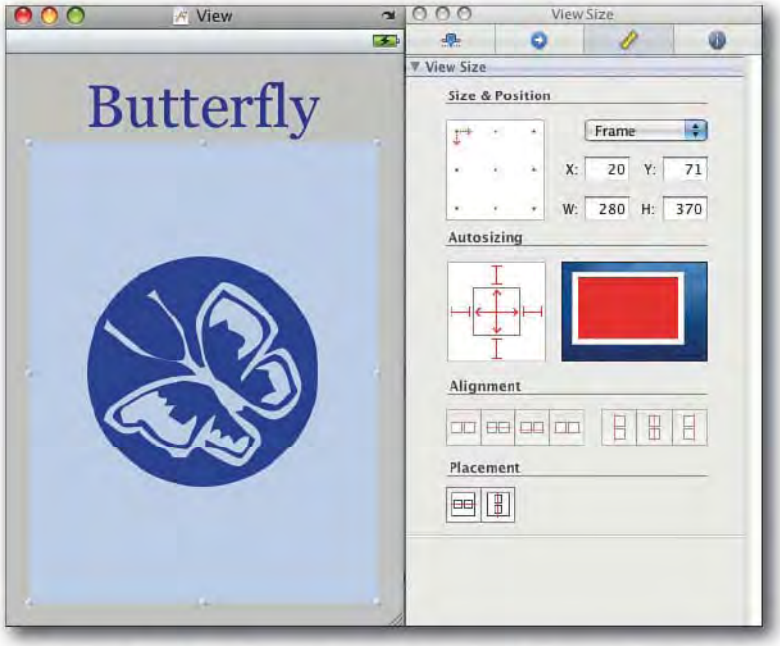
Wenn Sie diese Ansichtsaspekte lieber manuell setzen, verwenden Sie die beiden folgenden Eigenschaften. `autoresizesSubviews` ist ein boolescher Wert, der bestimmt, ob die Ansicht die Größenänderung ihrer Unteransichten erlaubt. Der Integer `autoresizingMask` wird mit dem bitweisen OR-Operator | aus den folgenden Flags kombiniert, sodass sich ein Eigenschaftswert ergibt:

- `UIViewAutoresizingNone` bedeutet, dass sich die Größe der Ansicht nicht ändert.
- Bei `UIViewAutoresizingFlexibleLeftMargin`, `UIViewAutoresizingFlexibleRightMargin`, `UIViewAutoresizingFlexibleTopMargin` und `UIViewAutoresizingFlexibleBottomMargin` wird die Größe einer Ansicht verändert, indem sie entlang des genannten Randes gedehnt oder zusammengezogen wird, ohne die Größe der enthaltenen Elemente zu beeinträchtigen. Diese Werte entsprechen den vier Streben im Bereich AUTOSIZING des Interface Builder (siehe *Abbildung 4.13*), wirken sich aber genau umgekehrt aus. Im Interface Builder dienen die Struts dazu, die Ränder zu fixieren, während diese Flags eine flexible Größenänderung entlang dieser Ränder erlauben.
- `UIViewAutoresizingFlexibleWidth` und `UIViewAutoresizingFlexibleHeight` regeln, ob sich die Größe einer Ansicht zusammen mit einer anderen ändert. Diese Werte entsprechen den Federn im Interface Builder. Sowohl die Federn als auch diese Flags ermöglichen eine flexible Größenänderung.

4.11.1 Beispiel für die automatische Größenanpassung

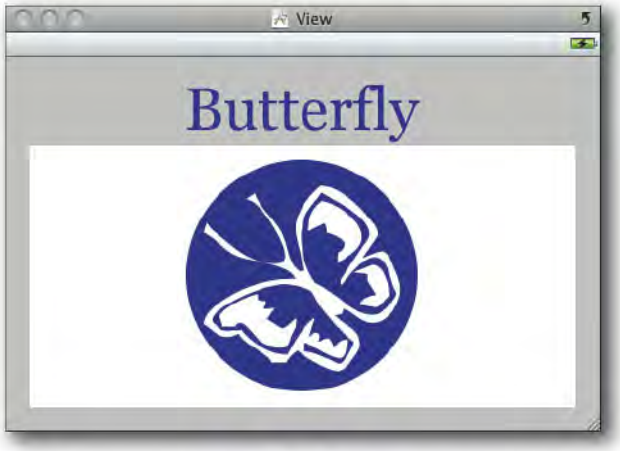
Betrachten Sie die Ansicht in *Abbildung 4.14*. Sie besteht aus einer Haupt- und drei Unteransichten, nämlich dem Titel, einem weißen Hintergrund und einer kleinen Grafik. Diese Unteransichten sind Beispiele für drei typische Situationen, denen Sie beim Entwurf von Anwendungen begegnen werden. Der Titel soll unabhängig von der Ausrichtung am Platz bleiben und seine Größe bewahren, der weiße Hintergrund muss sich dehnen oder zusammenziehen, um der Geometrie des Elternelements zu entsprechen, und die Schmetterlingsgrafik soll in ihrer Elternansicht schwimmen.

Das Verhalten der einzelnen Unteransichten in Bezug auf die automatische Größenanpassung legen Sie im Größen-Informationsfenster fest ⌘ + 3. Für den Titel ist nur eine einzige Strebe oben erforderlich, der Hintergrund muss seine Größe ändern können, während seine Abstände an allen Rändern erhalten bleiben. Dies erreichen Sie dadurch, dass Sie alle sechs Streben und Federn aktivieren (siehe *Abbildung 4.14*). Für die Unteransicht mit der Grafik ist die entgegengesetzte Einstellung nötig, bei der keine der Streben und Federn aktiviert sind.



► *Abbildung 4.14: Die automatische Größenanpassung einer Ansicht im Interface Builder festlegen*

Um die Ansicht in der anderen Ausrichtung zu testen, klicken Sie auf den kleinen, gekrümmten Pfeil oben rechts im Editorfenster der Ansicht. In *Abbildung 4.14* sehen Sie diesen Pfeil genau über der Akkuanzeige in der simulierten Statusleiste. *Abbildung 4.15* zeigt die Querformatversion der Ansicht mit diesen Einstellungen. Wenn Sie zwischen der Darstellung im Hoch- und Querformat umschalten, können Sie erkennen, ob Ihre Einstellungen für die automatische Größenanpassungen korrekt sind.



► *Abbildung 4.15: Dies ist die Querformatversion der Ansicht aus Abbildung 4.14 mit den genannten Einstellungen für die automatische Größenanpassung. Klicken Sie auf den Pfeil am rechten Ende der Titelleiste, um die Ansicht im Interface Builder zu drehen.*

Hinweis

Beim Start lädt das iPhone die letzte gespeicherte Ausrichtung. Wechseln Sie daher wieder zum Hochformat, bevor Sie die `.xib`-Datei sichern.

4.11.2 Eignet sich die automatische Größenanpassung für Ihre Anwendung?

Für manche iPhone-Klassen funktioniert die automatische Größenanpassung sehr gut. Umfangreiche, präsentationsgestützte Ansichten zeigen die besten Ergebnisse. Auch Web- und Textansichten lassen sich gut automatisch in der Größe ändern. Ihr Inhalt passt sich leicht an eine andere Gestalt der Anzeige an.

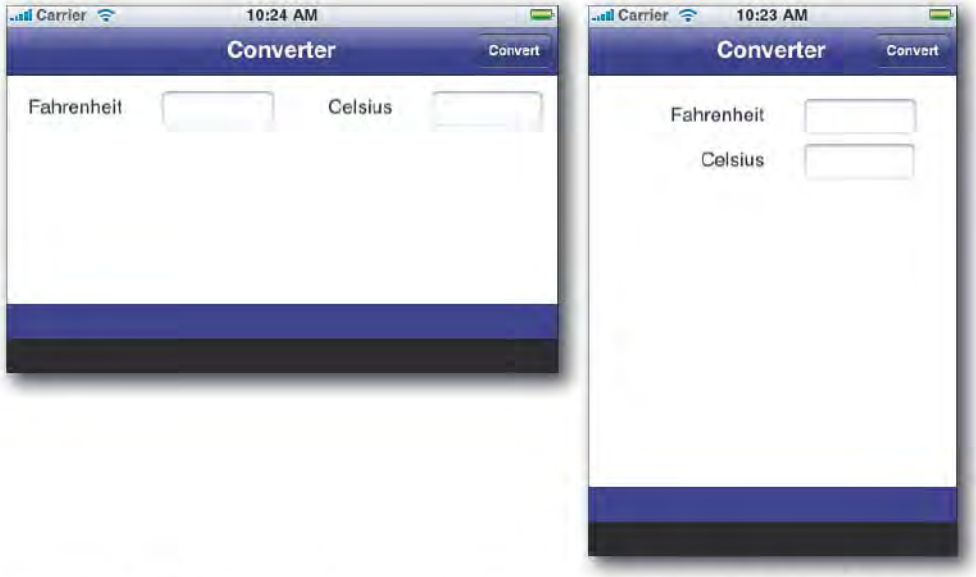
Kleine Steuerelemente, vor allem Textfelder, verhalten sich dabei jedoch weniger gut. Solche Ansichten sind nicht von Natur aus elastisch. Bei einem Wechsel vom Quer- ins Hochformat oder umgekehrt bleibt oft zu viel oder zu wenig Platz, um das vorhergehende Layout anzupassen. Bei solchen Ansichten sollten Sie lieber jedes Element an eine eigene Position stellen, anstatt sich auf die automatische Größenanpassung zu verlassen. Das heißt jedoch nicht, dass sich die automatische Größenanpassung überhaupt nicht für einfache Layouts eignet – genauso wenig, wie die automatische Größenanpassung grundsätzlich etwas für komplexe Ansichten mit vielen Unteransichten ist.

Auch für Bildansichten ist die automatische Größenanpassung nicht geeignet. Die meisten Bilder müssen ihre ursprünglichen Größenverhältnisse beibehalten. Ein Bild von 320 × 480 Pixel, das normalerweise im Hochformat angezeigt wird, müsste für das Querformat auf 213 × 320 Pixel verkleinert werden, also auf lediglich 45 % der Größe im Hochformat. Tauschen Sie solche Grafiken lieber gegen eine für das Querformat geeignete Version aus, anstatt zu versuchen, die hochformatigen Originale zu dehnen oder in der Größe zu ändern.

Denken Sie immer an die Tastatur, wenn Sie die automatische Größenanpassung einsetzen. Wenn Ihre Hauptansicht weder Rollbalken noch irgendwelche Möglichkeiten dafür aufweist, ihre Ansichten an zugängliche Positionen zu verschieben, kann die Tastatur möglicherweise Teile der Ansicht zudecken, für die sie da ist. Testen Sie Ihre Schnittstelle schon beim Entwurf, und zwar sowohl mit dem Umschalter im Interface Builder als auch im Simulator, um sicherzustellen, dass alle Elemente gut angeordnet und stets zugänglich sind.

4.12 Ansichten verschieben

Wenn eine automatische Größenanpassung als Reaktion auf eine Änderung der Ausrichtung impraktikabel ist, kann das Verschieben von Ansichten eine Lösung darstellen. Es ist allerdings mit mehr Aufwand verbunden. Prinzipiell funktioniert dieser Ansatz wie folgt: Nachdem der Ansichtscontroller die Ausrichtung abgeschlossen hat, ruft er die Delegierungsmethode `didRotateFromInterfaceOrientation:` auf. Hier können Sie eine Methode implementieren, die die einzelnen Ansichten positioniert, um Ergebnisse wie das in *Abbildung 4.16* zu erzielen. Wie Sie sehen, kann diese Vorgehensweise schnell mühselig werden, vor allem, wenn Sie es mit mehr als vier Unteransichten zu tun haben.



▶ *Abbildung 4.16: Durch das Verschieben von Ansichten können Sie das Layout nach einer Änderung der Ausrichtung anpassen.*

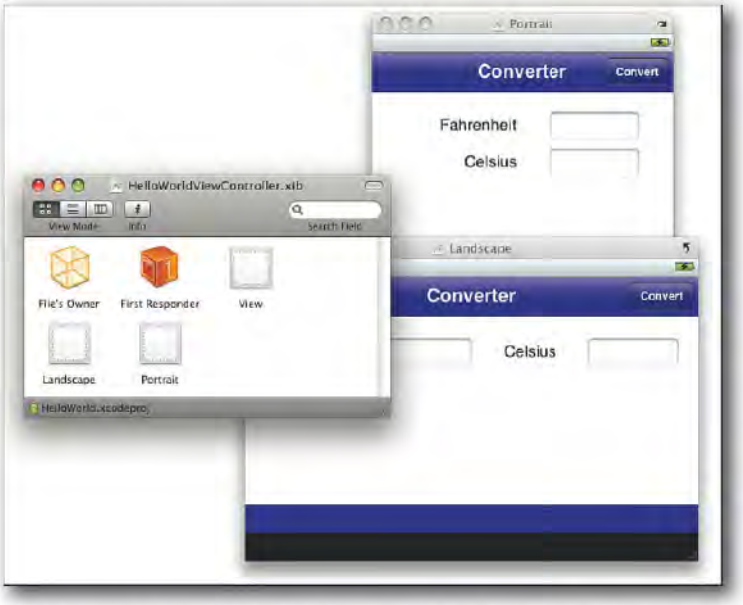
```
- (void)didRotateFromInterfaceOrientation:
(UIInterfaceOrientation)fromInterfaceOrientation
{
   UIInterfaceOrientation orientation = [[UIDevice currentDevice]
      orientation];
   UILabel *flabel = (UILabel *) [self.view viewWithTag:11];
   UILabel *clabel = (UILabel *) [self.view viewWithTag:12];
   UITextField *ffield = (UITextField *) [self.view viewWithTag:101];
   UITextField *cfield = (UITextField *) [self.view viewWithTag:102];

   switch (orientation)
   {
      case UIInterfaceOrientationLandscapeLeft:
      case UIInterfaceOrientationLandscapeRight:
      {
         flabel.center = CGPointMake(61,114);
         clabel.center = CGPointMake(321, 114);
         ffield.center = CGPointMake(184, 116);
         cfield.center = CGPointMake(418, 116);
         break;
      }
      case UIInterfaceOrientationPortrait:
      case UIInterfaceOrientationPortraitUpsideDown:
      {
```

```
            flabel.center = CGPointMake(113, 121);
            clabel.center = CGPointMake(139, 160);
            ffield.center = CGPointMake(236, 123);
            cfield.center = CGPointMake(236, 162);
            break;
        }
        default:
            break;
    }
}
```

Der große Vorteil der Verschiebung von Unteransichten gegenüber der Anzeige zweier völlig verschiedener Ansichten besteht darin, dass der Zugriff auf die ursprünglichen Unteransichten erhalten bleibt. Alle Instanzvariablen in Ihrem Code, die z. B. auf ein Textfeld zeigen, tun dies auch weiterhin, und zwar unabhängig davon, wo auf dem Bildschirm das Feld platziert ist. Die Datenstruktur des Ansichtscontrollers bleibt unverändert und unabhängig von der Position, was in hohem Maße dem MVC-Modell (Model-View-Controller) entspricht.

4.12.1 Ansichten durch die Duplizierung der Vorlage verschieben

Es gibt eine einfache Möglichkeit, um die Verschiebung mit weniger Arbeit zu erreichen. Duplizieren Sie dazu im Interface Builder zweimal die Hauptansicht, und bearbeiten Sie die beiden Kopien, um eine Hoch- und eine Querversion der Ansichtselemente zu erstellen. Da die Ansichten Kopien des Originals sind, behalten alle Unteransichten ihre Tags. *Abbildung 4.17* zeigt, wie solche Ansichten aussehen können.



▶ *Abbildung 4.17: Wenn Sie Ansichten im Interface Builder duplizieren, können Sie Vorlagen zum Verschieben von Unteransichten erstellen.*

4.12 Ansichten verschieben

In diesem Beispiel werden die ursprünglichen Ansichten und Unteransichten nach einem Wechsel der Ausrichtung weiterhin verwendet. (`allSubviews()` ist in *Rezept 6.2, Hilfsfunktionen für Unteransichten*, definiert.) Die beiden zusätzlichen Ansichten verwenden Sie als Vorlagen, um zu bestimmen, wo die einzelnen Unteransichten platziert werden sollen. Die Objekte schieben Sie in Übereinstimmung mit diesen Vorlagen auf ihre Position. Dieser Ansatz bietet zwei große Vorteile. Erstens müssen Sie die Positionen nicht hartkodieren, zweitens können Sie das Layout bei Bedarf im Interface Builder anpassen.

```
- (void)didRotateFromInterfaceOrientation:
(UIInterfaceOrientation)fromInterfaceOrientation
{
   UIView *template = nil;

   switch ([[UIDevice currentDevice] orientation])
   {
      case UIInterfaceOrientationLandscapeLeft:
      case UIInterfaceOrientationLandscapeRight:
      {
         template = landscapeTemplate;
         break;
      }
      case UIInterfaceOrientationPortrait:
      case UIInterfaceOrientationPortraitUpsideDown:
      {
         template = portraitTemplate;
         break;
      }
      default:
         break;
   }

   if (!template) return;

   for (UIView *eachView in allSubviews(template))
   {
      int tag = eachView.tag;
      if (tag < 10) continue;
      printf("About to move view %d\n", tag);
      [self.view viewWithTag:tag].frame = eachView.frame;
   }
}
```

Beachten Sie noch die folgenden Hinweise zu diesem Ansatz:

- Dieser Code ignoriert Ansichten ohne Tags und Tags mit einem Wert unter 10. Apple versieht Ansichten selten mit Tags, und wenn, dann haben diese Tags kleine Nummern wie 1, 2 oder 3. Sorgen Sie dafür, dass die Tags Ihrer Ansichten Nummern ab 10 aufweisen.
- In diesem Beispiel werden zwei Instanzvariablen (`portraitTemplate` und `landscapeTemplate`) verwendet, um unmittelbaren Zugriff auf die Vorlagen zu gewähren. Diese Variablen sind in der Headerdatei des Ansichtscontrollers als IB-Outlets definiert und über den Interface Builder verbunden. Beim Laden der `.xib`-Datei werden diese beiden Outlets automatisch festgelegt.
- Wenn Sie das Hochformatlayout bearbeiten wollen, so müssen Sie das im Interface Builder in der ursprünglichen Ansicht tun. Verwerfen Sie die frühere Hochformatansicht, und ersetzen Sie sie durch eine Kopie der überarbeiteten Ansicht. Verbinden Sie die Outlets von `File's Owner` mit der neuen Hochformatansicht. Damit ist sichergestellt, dass die Hochformat- und die Hauptansicht in der `.xib`-Datei identisch sind. Leider können Sie nicht einfach die Hauptansicht als Vorlage für das Hochformat verwenden. Nachdem sie einmal gedreht ist, sind die richtigen Positionen für die Hochformatansicht verloren.

4.13 Ansichten austauschen

In der Anwendung *iPod/Musik* wird nicht versucht, die Tabelle beim Drehen des iPhones ins Querformat umzustrukturieren. Stattdessen wird eine ganz andere Ansicht angezeigt, nämlich eine Coverflow-Darstellung der Alben. Um selbst einen solchen Austausch zu verwenden, bauen Sie zwei Ansichten in die `.xib`-Datei des Controllers ein: eine hoch- und eine querformatige. Ordnen Sie zu beiden Ansichten IB-Outlets zu, und setzen Sie die Eigenschaft `view` des Ansichtscontrollers für den Start auf die Hochformatversion.

Setzen Sie in der Implementierung des Ansichtscontrollers das Flag `autoresizesSubviews` für alle Hauptansichten auf `NO`, damit die Ansicht genau so erscheint, wie Sie sie im Interface Builder gestaltet haben. (Sie können sich auch den Spaß gönnen, die betreffenden Zeilen vorläufig auszukommentieren und im Simulator einige Drehungen durchzuführen. Das Ergebnis ist oft verblüffend.)

```
@implementation HelloWorldViewController
- (void)viewDidLoad {
    self.view.frame = [[UIScreen mainScreen] applicationFrame];
    landscapeView.autoresizesSubviews = NO;
    portraitView.autoresizesSubviews = NO;
}
```

Beim Drehen der Ansicht ins Quer- oder Hochformat wird `self.view` umgeschaltet, um auf die jeweils richtige Ansicht zu verweisen. Der folgende Code prüft zuerst, ob eine Querausrichtung vorliegt, und verwendet dann `else if` für die Ausrichtungen im Hochformat. Dies ist eine Absicherung gegen unbekannte Ausrichtungen und ein auf dem Kopf stehendes Hochformat.

```
- (BOOL)shouldAutorotateToInterfaceOrientation:
(UIInterfaceOrientation)interfaceOrientation {
    if ((interfaceOrientation == UIInterfaceOrientationLandscapeLeft)
    || (interfaceOrientation == UIInterfaceOrientationLandscapeRight))
        self.view = landscapeView;
    else if ((interfaceOrientation == UIInterfaceOrientationPortrait)
        || (interfaceOrientation ==
            UIInterfaceOrientationPortraitUpsideDown))
        self.view = portraitView;
    return YES;
}
@end
```

Wenn dieser Code ausgeführt wird, reagiert er auf Änderungen der Ausrichtung, indem er der Eigenschaft `view` des Ansichtscontrollers die Ansicht mit dem jeweils passenden Format zuweist.

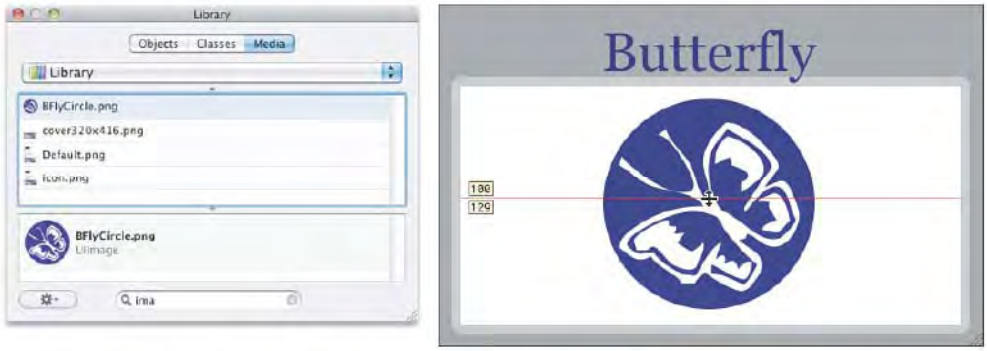
4.14 Zum guten Schluss: Ein halbes Dutzend grossartige Tipps zum Interface Builder

Es ist immer gut, bei der Entwicklung mit Interface Builder und Xcode einige zusätzliche Trümpfe im Ärmel zu haben. Im Folgenden zeige ich Ihnen meine sechs Lieblingstricks für die Arbeit im Interface Builder, die ich selbst regelmäßig verwende.

- Aus gestapelten Ansichten auswählen – In *Abbildung 4.12* haben Sie gesehen, wie Sie sich durch die Objekthierarchie des Interface Builder bewegen, um auf Unteransichten zuzugreifen. Eine andere Möglichkeit, um Unteransichten zu finden und auf sie zuzugreifen, besteht darin, bei gedrückter ⇧- und Ctrl-Taste auf die Ansicht zu klicken. Dadurch werden alle Ansichten angezeigt, die an dieser Position übereinander geschichtet sind (siehe *Abbildung 4.18*). Aus diesen Elementen können Sie jedes gewünschte Element unabhängig davon auswählen, ob es sich um die oberste Ansicht handelt oder nicht.

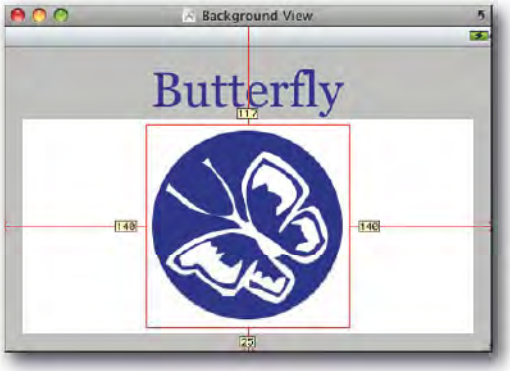
Abbildung 4.18: Wenn Sie bei gedrückter ⇧- und Ctrl-Taste klicken, wird ein Dialogfeld zur Ansichtsauswahl eingeblendet.

- Ansichten benennen – Geben Sie Ihren Ansichten im Identitäts-Informationsfenster ⌘ + 4 Namen wie diejenigen, die Sie in *Abbildung 4.18* sehen. Bearbeiten Sie dazu das Feld **Identity ▸ Name** im Interface Builder.
- Medien einfügen – In der Bibliothek des Interface Builder finden Sie den Titel **Media**, auf dem die zurzeit im Xcode-Projekt verfügbaren Medien aufgeführt sind (siehe *Abbildung 4.19*, links). Sie können Grafiken von dort in eine Ansicht ziehen. Der Interface Builder erstellt automatisch eine neue Instanz von `UIImageView` und fügt ihr die Grafik (als `UIImage`-Objekt) hinzu.



▸ *Abbildung 4.19: Sie können Medien aus der Bibliothek in Ihr Interface Builder-Projekt ziehen (links). An Hilfslinien können Sie Ihre Ansichten ausrichten (rechts).*

- Hilfslinien einblenden – Im Interface Builder können Sie ebensolche Layout-Hilfslinien verwenden wie z. B. in Photoshop. Im Menü **Layout** können Sie horizontale und vertikale Hilfslinien einblenden (**Add Horizontal Guide** bzw. **Add Vertical Guide**). Wie Sie in *Abbildung 4.19* (rechts) sehen, zeigt der Interface Builder immer pixelgenau die Position einer Hilfslinie an, während Sie sie verschieben.
- Objekte verschieben – Wenn Sie Unteransichten verschieben, können Sie sie mit den Pfeiltasten jeweils um ein Pixel in der betreffenden Richtung bewegen. Halten Sie die ⇧-Taste gedrückt, um eine Verschiebung um jeweils fünf Pixel zu erreichen.
- Objektlayout anzeigen – Wenn Sie die Alt-Taste gedrückt halten und mit der Maus über ein Objekt fahren, werden pixelgenaue Layoutinformationen darüber angezeigt, wie Sie in *Abbildung 4.20* sehen.



▸ *Abbildung 4.20: Wenn Sie bei gedrückter Alt-Taste mit der Maus über ein Objekt fahren, wird angezeigt, wie die Ansicht in ihrem Container angeordnet ist.*

4.15 Zusammenfassung

In diesem Kapitel wurden Sie in die Grundlagen des Schnittstellendesigns für das iPhone eingeführt. Sie haben gelernt, dass es nicht nur eine, sondern vier Vorgehensweisen gibt, um Schnittstellen zu konstruieren: Interface Builder, Xcode und zwei Mischungen dieser beiden Ansätze. Außerdem haben Sie die Neuausrichtung in Aktion erlebt und erfahren, wie Sie dafür sorgen, dass Ihre Anwendungen sowohl im Hoch- als auch im Querformat optimal angezeigt werden.

Bevor Sie zum nächsten Kapitel weiterblättern, sollten Sie sich die folgenden Punkte über das Layout von Schnittstellen einprägen:

- Der Interface Builder ist ein hervorragendes Werkzeug, um den Inhalt von `UIView`-Instanzen anzuordnen. Verwenden Sie es, um diese Ansichten mit den Ansichtscontrollern Ihres Programms zu verwenden und Oberflächen wie für das Beispielprogramm zur Temperaturumrechnung in diesem Kapitel auf grafischem Wege zu optimieren.
- Sie sollten erkennen können, wann der Interface Builder für eine Aufgabe nicht geeignet ist. Um Tab Bars und Navigationscontroller mit nur geringem Fensterdesign zu erstellen (z. B. für tabellen- oder textgestützte Anwendungen), brauchen Sie die Interface Builder-Werkzeuge für das Ansichtslayout nicht unbedingt. Wenn Sie auf den Interface Builder verzichten, müssen Sie die `.xib`-Datei aus dem Projekt löschen und den Schlüssel `Main NIB Window` aus `Info.plist` entfernen. Bearbeiten Sie auch die Datei `main.m`, um als viertes Argument von `UIApplicationMain()` den Klassennamen des Anwendungs-Delegates einzusetzen. Wenn Sie dies nicht tun, erhalten Sie eine Anwendung, die nur einen schwarzen Bildschirm zeigt und keinerlei Interaktion zulässt.
- Einige Ansichten eignen sich hervorragend für verschiedene Ausrichtungen, andere nicht. Zwingen Sie sich nicht dazu, eine Querformatversion Ihrer Anwendung anzubieten, deren Erscheinungsbild und Funktionsweise genau der Hochformatversion entsprechen.
- Speichern Sie immer, immer wieder Ihre Arbeit im Interface Builder! Vorher wird das Projekt auch nicht mit der jüngsten Version der `.xib`-Datei aktualisiert.
- Es gibt keine Universallösung für den Entwurf und die Implementierung von hoch- und querformatigen Layouts. Wählen Sie jeweils den Ansatz aus, der am besten Ihren Bedürfnissen entspricht und für die Benutzer am sinnvollsten ist.

5 Mit Ansichtscontrollern arbeiten

In vielen iPhone-Anwendungen kümmern sich `UIViewController` um die Handhabung der Ansicht. Im vorherigen Kapitel haben Sie gelernt, wie Sie mit Xcode und dem Interface Builder Anwendungen auf der Grundlage von Ansichtscontrollern erstellen. Jetzt geht es um ausgefeiltere Ansichtscontrollerklassen und ihre Verwendung in der Praxis. In diesem Kapitel erfahren Sie, wie Sie einfache Menüs aufbauen, Ansichtsnavigationsbäume erstellen und Anwendungen mit Tab Bars gestalten. Dabei erhalten Sie praktische Rezepte für den Umgang mit einer Vielzahl von Controllerklassen.

5.1 Entwicklung mit Navigationscontrollern

Die Klasse `UINavigationController` bietet die hochwertigen Eigenschaften einer auf `UINavigationBar` beruhenden Oberfläche bei lediglich geringem navigationsspezifischen Programmieraufwand. Navigationscontroller ermöglichen Benutzern den flüssigen Wechsel zwischen Ansichten (oder genauer gesagt, Ansichtscontrollern) mithilfe integrierter Animationen. Außerdem bieten sie gratis dazu eine Verlaufssteuerung ohne weiteren Programmieraufwand. Der Controller kümmert sich auch um die **Zurück**-Schaltfläche. Das bedeutet, dass der Titel des jeweiligen Eltern-Ansichtscontrollers automatisch als **Zurück**-Schaltfläche erscheint und Sie es den Benutzern ohne irgendeine Programmierung ermöglichen, das oberste Element vom Stack zu entfernen.

Als ob dies nicht schon genug wäre, bietet der Navigationscontroller außerdem eine einfache Menüleiste. Sie können Schaltflächen und sogar umfangreichere Steuerelemente zur Leiste hinzufügen, um Aktionen in Ihrer Anwendung bereitzustellen. Neben diesen drei Funktionen (Navigation, Verlauf und Menüs) fügen Navigationscontroller beeindruckende Effekte zu einfachen Paketen hinzu.

Die folgenden Rezepte stellen diese Kernfunktionen des Navigationscontrollers von der Erstellung von Menüs bis zum Aufbau eines Verlaufsstacks vor. In den Beispielen sehen Sie, wie Sie die Klasse `UINavigationBar` zum Erstellen von verschiedenen neuen und nützlichen Oberflächen nutzen können.

5.1.1 Einen Navigationscontroller erstellen

Unabhängig davon, ob Sie einen Navigationscontroller zur Vereinfachung der Bewegung zwischen Ansichten verwenden wollen (seine zugedachte Verwendung) oder als bequemen Halter für eine Menüschaltfläche, sollten Sie verstehen, wie er funktioniert. Auf der einfachsten Ebene verwalten Navigationscontroller den Stack von Ansichtscontrollern.

Jeder Navigationscontroller besitzt einen Wurzelansichtscontroller, der das Fundament des Stacks darstellt. Sie können durch Programmbefehle andere Controller auf den Stack verschieben, wodurch sich die Breadcrumb-Navigationspur erweitert und automatisch eine **Zurück**-Schaltfläche erstellt wird.

Immer wenn Sie auf eine dieser **Zurück**-Schaltflächen tippen, nehmen Sie einen Controller vom Stack. Die Benutzer können so lange zurückgehen, bis sie die Wurzel erreicht haben, aber dann geht es nicht mehr weiter. Die Wurzel ist die Wurzel, und Sie können nicht über sie hinaus blättern.

Dieses stackbasierte Design bleibt sogar dann bestehen, wenn Sie nur einen einzigen Ansichtscontroller verwenden wollen. Wenn Sie z. B. die integrierte Navigationsleiste von `UINavigationController` nutzen, um ein Menü mit zwei Schaltflächen zu erstellen, gehen Ihnen alle Navigationsvorteile des Stacks verloren. Sie müssen diesen einen Controller immer noch über `initWithRootViewController:` als Wurzel festlegen.

Um Schnittstellen mit Navigationsmöglichkeiten anzulegen, können Sie den Interface Builder und Xcode verwenden, wie Sie in Kapitel 4, *Benutzeroberflächen entwerfen*, gesehen haben. Sie können die gleichen Schnittstellen aber auch manuell entwerfen. Am einfachsten geht dies, wenn Sie den Navigationscontroller in der Methode `applicationDidFinishLaunching:` erstellen, die beim Start der Anwendung aufgerufen wird. Hier richten Sie die Fenster ein, legen den Navigationscontroller an und weisen ihm seine Wurzel zu.

```
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    UIWindow *window = [[UIWindow alloc] initWithFrame:
        [[UIScreen mainScreen] bounds]];
    UINavigationController *nav = [[UINavigationController alloc]
        initWithRootViewController:[[HelloWorldController alloc] init]];
    [window addSubview:nav.view];
    [window makeKeyAndVisible];
}
```

Dies ist eine der wenigen Stellen, an denen Sie sich wirklich keine Gedanken über Speicherverwaltung und Lecks machen müssen. Die Methode `dealloc` eines Anwendungs-Delegates wird niemals beim Beenden einer Anwendung aufgerufen. Zwar können Sie das Fenster und den Navigationscontroller zu Instanzvariablen zuweisen und diese Variablen in einer Methode zum Aufheben der Zuweisung verwenden, doch macht es überhaupt nichts aus, wenn Sie das nicht tun.

5.1.2 Ansichtscontroller hinzufügen und entfernen

Neue Objekte können hinzugefügt werden, indem Sie einen neuen Controller mit `pushViewController: animated:` auf den Navigationsstack schieben. Senden Sie diesen Aufruf zu dem Navigationscontroller, der einen `UIViewController` besitzt. Gewöhnlich erfolgt dieser Aufruf für `self.navigationController`. Beim Hinzufügen gleitet der neue Controller von rechts auf den Bildschirm (vorausgesetzt, Sie haben `animated` auf `YES` eingestellt). Gleichzeitig erscheint eine nach links weisende **Zurück**-Schaltfläche, mit der Sie im Stack einen Schritt zurückgehen können. Sie zeigt den Titel des vorausgegangenen Ansichtscontrollers an.

Es gibt viele Gründe, um eine neuen Ansicht einzufügen, z. B. den Wechsel zu einer Unteransicht wie einer Detailansicht oder das Durchlaufen einer Dateistruktur. Sie können Controller auf den Stack schieben lassen, nachdem der Benutzer auf eine Schaltfläche, ein Tabellenobjekt oder ein Erweiterungssymbol getippt hat.

Führen Sie die Anforderung eines neuen Controllers und andere Anpassungen der Navigationsleiste (z. B. die Einrichtung einer Schaltfläche auf der rechten Seite) innerhalb Ihrer `UIViewController`-Unterklassen durch. Es besteht kein Grund und keine Notwendigkeit, `UINavigationController` mit Unterklassen zu versehen. Meistens müssen Sie nicht direkt auf den Navigationscontroller zugreifen. Die beiden Ausnahmen von dieser Regel sind die Verwaltung der Schaltflächen in der Navigationsleiste und die Änderung des Erscheinungsbilds der Leiste.

Sie können den Stil und die Farbe der Leiste unmittelbar über die Eigenschaft `navigationBar` ändern:

```
self.navigationController.navigationBar.barStyle =
   UIBarStyleBlackTranslucent;
```

Um eine neue Schaltfläche hinzuzufügen, ändern Sie Ihr Navigationselement (`navigationItem`), das eine abstrakte Klasse zur Beschreibung des in der Navigationsleiste angezeigten Inhalts bietet. Wollen Sie eine Schaltfläche entfernen, weisen Sie ihr `nil` zu.

```
self.navigationItem.rightBarButtonItem = [[[UIBarButtonItem alloc]
   initWithTitle:@"Action" style:UIBarButtonItemStylePlain target:self
   action:@selector(performAction:)] autorelease];
```

5.1.3 Die Klasse »UINavigationItem«

Die Objekte der Navigationsleiste werden mit der Klasse `UINavigationItem` platziert, einer abstrakten Klasse, die Informationen über diese Objekte speichert. Zu den Eigenschaften von Navigationselementen gehören die linke und die rechte Schaltfläche in der Leiste, der in der Leiste angezeigte Titel, die Ansicht, die zur Anzeige des Titels dient, und eine eventuell vorhandene **Zurück**-Schaltfläche, um von der aktuellen Ansicht zur vorherigen zu wechseln.

Die Klasse `UINavigationItem` ermöglicht das Hinzufügen von Schaltflächen, Text und anderen Objekten der Oberfläche an drei Schlüsselpositionen: links, rechts und in der Mitte der Navigationsleiste. Normalerweise stellt sich dies als reguläre Schaltfläche auf der rechten Seite, Text in der Mitte (gewöhnlich der Titel des `UIViewControllers`) und als eine Art **Zurück**-Schaltfläche auf

der linken Seite dar. Sie sind aber nicht auf dieses Layout beschränkt, sondern können angepasste Steuerelemente zu allen drei Positionen hinzufügen. Es sind also auch Navigationsleisten mit Suchfeldern, unterteilten Steuerelementen, Symbolleisten, Bildern und mehr möglich.

Sie haben bereits gesehen, wie Sie eigene Leistenschaltflächen rechts und links von einem Navigationselement hinzufügen. Dem Titel eine eigene Ansicht hinzuzufügen, ist genauso einfach – statt eines Steuerelements ordnen Sie nur eine Ansicht zu. Der folgende Code fügt ein benutzerdefiniertes `UILabel` hinzu, aber es könnte sich hierbei auch um eine `UIImageView`, einen `UISwitch` oder irgendetwas anderes handeln:

```
self.navigationItem.titleView = [[[UILabel alloc]
    initWithFrame:CGRectMake(0.0f,0.0f, 120.0f, 36.0f)] autorelease];
```

Die einfachste Möglichkeit, um den Titel anzupassen, besteht darin, die Eigenschaft `title` des Kindansichtscontrollers zu verwenden statt des Navigationselements:

```
self.title = @"Hello";
```

Soll der Titel automatisch den Namen der laufenden Anwendung anzeigen, können Sie den folgenden Trick verwenden. Dadurch wird automatisch der kurze Anzeigename zurückgegeben, der in der Datei `Info.plist` des Bundles definiert ist.

```
self.title = [[[NSBundle mainBundle] infoDictionary]
    objectForKey:@"CFBundleName"];
```

Modale Darstellung

Bei normalen Navigationscontrollern bewegen Sie sich Ansicht für Ansicht weiter, wobei Sie auch gelegentlich anhalten und zur vorherigen Ansicht zurückkehren können. Eine Voraussetzung für diesen Ansatz ist, dass Sie sich auf und ab durch eine Datenmenge bewegen, die mit der von Ihnen verwendeten baumartigen Ansichtsstruktur übereinstimmt. Die modale Darstellung bietet eine andere Möglichkeit, um einen Ansichtscontroller anzuzeigen. Nachdem die Nachricht `presentModalViewController: animated:` gesendet wurde, gleitet ein neuer Controller auf den Bildschirm und übernimmt die Steuerung so lange, bis er mit `dismissModalViewControllerAnimated:` wieder entfernt wird. Dadurch können Sie Dialogfelder für besondere Zwecke in Ihre Anwendungen einbringen.

Normalerweise dienen modale Controller dazu, Daten wie Kontakte aus Adressbüchern oder Fotos aus der Bibliothek herauszusuchen, aber Sie können sie auch in jeder Situation verwenden, in der es sinnvoll ist, eine Aufgabe durchzuführen, die die Grenzen des aktiven Ansichtscontrollers überschreitet.

Ein modales Dialogfeld können Sie in drei verschiedenen Formen anzeigen, was Sie in der Eigenschaft `modalTransitionStyle` des Ansichtscontrollers festlegen. Bei der Standardeinstellung `UIModalTransitionStyleCoverVertical` wird die modale Ansicht von unten über den aktuellen Ansichtscontroller geschoben. Wird sie wieder entfernt, so gleitet sie nach unten weg. Bei `UIModalTransitionStyleFlipHorizontal` dreht sich die Ansicht scheinbar von rechts nach links, was so aussieht, als würde die Rückseite der aktuellen Ansicht freigelegt. Beim Entfernen dreht sich die Ansicht wieder von links nach rechts zurück. Der dritte mögliche Stil ist `UIModalTransitionCrossDissolve`. Hierbei wird die neue Ansicht über der vorherigen langsam eingeblendet und beim Entfernen wieder ausgeblendet.

5.2 Eine Hilfsfunktion

In einigen Rezepten dieses Buches verwende ich die Makro-Funktion `showAlert()`, die als grafische Version von `NSLog()` dient und eine Nachricht und Informationen über den Ursprung des Aufrufs anzeigt. Diese Funktion kann mit denselben Parametern aufgerufen werden wie `NSLog`, einschließlich Formatierungsstring und Argumenten. Aus Platzgründen habe ich den Warncode in den einzelnen Rezepten nicht angegeben. Der Aufruf wird in *Abbildung 5.1* gezeigt.

```
#define showAlert(format, ...) myShowAlert(__LINE__, (char *)__FUNC-
TION__, format, ##__VA_ARGS__)
// Einfache Warnfunktion
void myShowAlert(int line, char *functname, id formatstring,...)
{
   va_list arglist;
   if (!formatstring) return;
   va_start(arglist, formatstring);
   id outstring = [[[NSString alloc] initWithFormat:formatstring
      arguments:arglist] autorelease];
   va_end(arglist);

   NSString *filename = [[NSString stringWithCString:__FILE__]
      lastPathComponent];
   NSString *debugInfo = [NSString stringWithFormat:@"%@:%d\n%s",
      filename, line, functname];

   UIAlertView *av = [[[UIAlertView alloc] initWithTitle:outstring
      message:debugInfo delegate:nil
      cancelButtonTitle:@"OK"otherButtonTitles:nil] autorelease];
   [av show];
}
```

▸ *Abbildung 5.1: Ein einfaches Menü mit zwei Schaltflächen können Sie erstellen, indem Sie eigene Schaltflächen zu einer* `UINavigationController`*-Schnittstelle hinzufügen.*

5.3 Rezept: Ein einfaches Menü mit zwei Elementen erstellen

Obwohl viele Anwendungen nach ernst zu nehmenden Benutzerschnittstellen verlangen, ist oftmals kein komplexer Aufbau notwendig. Ein einfaches Menü mit einer oder zwei Schaltflächen kann vielem gerecht werden. Führen Sie die folgenden Schritte aus, um eine eigene Oberfläche für einfache Hilfsprogramme zu erstellen:

1. Erstellen Sie eine Unterklasse von `UIViewController`, die Sie als primären Interaktionsraum verwenden.
2. Stellen Sie einen Navigationscontroller bereit, und weisen Sie seiner Wurzelansicht eine Instanz Ihres angepassten Ansichtscontrollers zu.
3. Erstellen Sie im angepassten Ansichtscontroller eine oder mehrere Schaltflächen, und fügen Sie diese zu den Navigationselementen der Ansicht hinzu.
4. Erstellen Sie die Callback-Routinen, die ausgelöst werden, wenn ein Benutzer auf eine Schaltfläche tippt.

Rezept 5.1 veranschaulicht diese Schritte. Es erstellt einen einfachen Ansichtscontroller mit dem Namen `HelloController` und weist ihm einem `UINavigationController` als Wurzelansicht zu. In der Methode `loadView` befindet sich jeweils eine Schaltfläche auf der rechten und der linken Seite der benutzerdefinierten Slots für die Navigationsobjekte der Ansicht. Werden diese Schaltflächen angetippt, zeigen sie eine Meldung an, die angibt, welche Schaltfläche verwendet wurde. Dieses Rezept ist nicht sehr reich an Funktionen, doch es zeigt ein einfach zu erstellendes Menü mit zwei Objekten. In *Abbildung 5.1* sehen Sie die Oberfläche in Aktion.

In diesem Code wird ein praktisches Makro zum Erstellen von Leistenschaltflächen verwendet. Wenn Sie ihm einen Titel und einen Selektor übergeben, gibt das Makro ein korrekt initialisiertes und zur automatischen Freigabe gekennzeichnetes Schaltflächenelement zurück, das Sie einem Navigationselement zuweisen können.

```
#define BARBUTTON(TITLE, SELECTOR) [[[UIBarButtonItem alloc] \
    initWithTitle:TITLE style:UIBarButtonItemStylePlain target:self \
    action:SELECTOR] autorelease]
```

Sollten Sie mehr Komplexität brauchen, als zwei Objekte bieten können, lassen Sie die Schaltflächen `UIActionSheet`-Menüs auslösen. Action-Sheets werden in Kapitel 10, *Benutzer benachrichtigen*, beschrieben und ermöglichen die Auswahl von Aktionen aus einer kurzen Liste (mit normalerweise zwei bis fünf Optionen, wobei es jedoch auch möglich ist, längere Listen mit Rollbalken zu verwenden). Beispiele dafür finden Sie in den Programmen *Photo* und *Mail* für den Datenaustausch oder die Ablage.

Hinweis

Sie können zusätzlich zum Text oder an seiner Stelle auch Bilder zu den `UIBarButtonItem`-Instanzen in Ihrer Navigationsleiste hinzufügen. Verwenden Sie dazu `initWithImage: style: target: action:` anstelle des textbasierten Initialisierers.

```
@implementation TestBedViewController
- (void) rightAction: (id) sender
{
   showAlert(@"You pressed the right button");
}

- (void) leftAction: (id) sender
{
   showAlert(@"You pressed the left button");
}

- (void) loadView
{
   self.view = [[[NSBundle mainBundle] loadNibNamed:@"mainview"
      owner:self options:nil] lastObject];

   self.navigationController.navigationBar.tintColor =

      COOKBOOK_PURPLE_COLOR;
   self.navigationItem.rightBarButtonItem = BARBUTTON(@"Right",
      @selector(rightAction:));
   self.navigationItem.leftBarButtonItem = BARBUTTON(@"Left",
      @selector(leftAction:));
}
@end

@implementation TestBedAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application
{
   UIWindow *window = [[UIWindow alloc]
      initWithFrame:[[UIScreen mainScreen] bounds]];
   UINavigationController *nav = [[UINavigationController alloc]
      initWithRootViewController:[[TestBedViewController alloc] init]];
   [window addSubview:nav.view];
   [window makeKeyAndVisible];
}
@end
```

▸ *Rezept 5.1: Ein Menü mit zwei Schaltflächen unter Verwendung eines Navigationscontrollers erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 5 und öffnen das Projekt zu diesem Rezept.

5.4 Rezept: Ein Steuerelement mit einer Unterteilung hinzufügen

Das vorstehende Rezept hat gezeigt, wie Sie die zwei verfügbaren Schaltflächenslots in Ihrer Navigationsleiste zum Erstellen von Mini-Menüs verwenden. *Rezept 5.2* erweitert diese Idee, indem es ein sechsteiliges `UISegmentedControl` vorstellt und wie in *Abbildung 5.2* zur Navigationsleiste der Titelansicht hinzufügt. Jedes dieser Elemente aktualisiert die Hauptansicht mit seiner Nummer, wenn es angetippt wird.

Bei diesem Rezept müssen Sie besonders auf das Attribut `momentary` achten, das dem segmentierten Steuerelement zugewiesen ist. Es wandelt die Oberfläche von Optionsschaltern zu einem richtigen Menü um, in dem Objekte unabhängig voneinander und mehrmals ausgewählt werden können. Nachdem Sie also das dritte Objekt angetippt haben, können Sie erneut darauf tippen. Dies ist ein wichtiges Verhalten für die Menüinteraktion.

▶ *Abbildung 5.2: Wenn Sie ein segmentiertes Steuerelement zur Titelansicht hinzufügen, können Sie daraus ein Menü mit mehreren Objekten erstellen. Beachten Sie, dass kein Objekt nach einer Aktion hervorgehoben bleibt (in diesem Fall wurde die Schaltfläche* **One** *angetippt).*

Anders als bei *Rezept 5.1* lösen alle Objekte des segmentierten Steuerelements die gleiche Aktion aus (in diesem Fall `segmentAction:`). Bestimmen Sie, welche Aktion durchgeführt werden soll, indem Sie das Steuerelement nach seinem `selectedSegmentIndex` abfragen und den Wert mit dem gewünschten Verhalten abgleichen. In diesem Rezept wird die zentrale Textbeschriftung aktualisiert. Sie können aber je nach angetipptem Segment unterschiedliche Optionen wählen.

Hinweis

Wenn Sie diesen Code mit deaktivierter Eigenschaft `momentary` testen möchten, stellen Sie die Eigenschaft `selectedSegmentIndex` so ein, dass sie dem Wert entspricht, der ursprünglich angezeigt wird. Dabei entspricht Segment 0 der angezeigten Ziffer 1.

5.4 Rezept: Ein Steuerelement mit einer Unterteilung hinzufügen

Segmentierte Steuerelemente haben eine Einstellung für den Stil, die angibt, wie sie angezeigt werden sollen. In dem Beispiel aus *Abbildung 5.2* wird ein Leistenstil verwendet, der zur Verwendung in Leisten gedacht ist. Die beiden anderen möglichen Stile (`UISegmentedControlStyleBordered` und `UISegmentedControlStylePlain`) haben ein größeres, metallischeres Erscheinungsbild. Von diesen drei Stilen kann nur `UISegmentedControlStyleBar` auf die Änderungen an `tintcolor` reagieren, die in diesem Rezept eingesetzt werden.

```
-(void) segmentAction: (UISegmentedControl *) sender
{
   // Beschriftung mit der Segmentnummer aktualisieren
   UILabel *label = (UILabel *)[self.view viewWithTag:101];
   [label setText:[NSString stringWithFormat:
   @"%Od", sender.selectedSegmentIndex + 1]];
}

- (void) loadView
{
   self.view = [[[NSBundle mainBundle] loadNibNamed:@"mainview"
      owner:self options:nil] lastObject];

   self.navigationController.navigationBar.tintColor = COOKBOOK_
    PURPLE_COLOR;
   // Segmentiertes Steuerelement erstellen
   NSArray *buttonNames = [NSArray arrayWithObjects:@"One", @"Two",
      @"Three", @"Four", @"Five", @"Six", nil];
   UISegmentedControl* segmentedControl = [[UISegmentedControl alloc]
      initWithItems:buttonNames];
   segmentedControl.segmentedControlStyle =
      UISegmentedControlStyleBar;
   segmentedControl.momentary = YES;
   [segmentedControl addTarget:self action:@selector(segmentAction:)

      forControlEvents:UIControlEventValueChanged];

   // Zur Navigationsleiste hinzufügen
   self.navigationItem.titleView = segmentedControl;
   [segmentedControl release];
}
```

▸ *Rezept 5.2: Ein segmentiertes Steuerelement zu einer Navigationsleiste hinzufügen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 5 und öffnen das Projekt zu diesem Rezept.

5.5 Rezept: Navigation zwischen den Ansichtscontrollern

Navigationscontroller stellen nicht nur Menüs bereit, sondern führen auch die Aufgaben durch, für die sie eigentlich entwickelt wurden: die Verwaltung der Hierarchie beim Wechsel zwischen Ansichten. *Rezept 5.3* zeigt, wie Sie Navigationscontroller wirklich als Navigationscontroller einsetzen, die Ansichten im Stack verschieben.

Bei diesen Ansichten handelt es sich um dieselben Platzhalter zur Ziffernanzeige, die Sie bereits in den früheren Rezepten gesehen haben. Eine Instanzvariable speichert den aktuellen Tiefenwert, der verwendet wird, um die Nummer der derzeitigen Ebene anzuzeigen und zu entscheiden, ob eine weitere Verschiebeoption (**Push**) dargestellt werden soll. Die Maximaltiefe beträgt hier 6. In der Praxis verwenden Sie natürlich sinnvollere Ansichtscontroller und Inhalte, doch dieses Beispiel soll die Zusammenhänge auf einfachste Weise zeigen.

Wenn Sie einen neuen Controller für Ebene 3 auf den Stack schieben, legt der Navigationscontroller automatisch eine **Zurück**-Schaltfläche zu Ebene 2 (**Level 2**) an, wie Sie in *Abbildung 5.3* (links) sehen. Die Schaltfläche ganz rechts (**Push 4**) löst den Wechsel zum nächsten Controller aus, indem sie `pushViewController: animated:` aufruft. Wenn diese Schaltfläche angetippt wird, steht auf der nächsten **Zurück**-Schaltfläche **Level 3**, wie Sie in *Abbildung 5.3* (rechts) sehen.

▸ *Abbildung 5.3: Der Navigationscontroller erstellt automatisch korrekt beschriftete* **Zurück***-Schaltflächen. Nachdem Sie in der links dargestellten Schnittstelle auf* **Push 4** *getippt haben, schiebt der Navigationscontroller den Ansichtscontroller für Ebene 4 auf den Stack und erstellt die* **Zurück***-Schaltfläche* **Level 3***, die Sie in der Schnittstelle auf der rechten Seite sehen.*

Eine **Zurück**-Schaltfläche sorgt selbst dafür, dass der aktuelle Controller vom Stack entfernt wird. Dieses Verhalten müssen Sie nicht eigens programmieren. Beachten Sie, dass **Zurück**-Schaltflächen automatisch für Ansichtscontroller erstellt werden, die auf den Stack geschoben wurden, aber nicht für den Wurzelcontroller selbst, da sie dort nicht sinnvoll sind.

```
@interface TestBedViewController : UIViewController
{
   int depth;
}
@end

@implementation TestBedViewController
- (id) initWithDepth: (int) theDepth
{
   self = [super init];
   if (self) depth = theDepth;
   return self;
}

- (void) push
{
   TestBedViewController *tbvc = [[[TestBedViewController alloc]
      initWithDepth:(depth + 1)] autorelease];
   [self.navigationController pushViewController:tbvc animated:YES];
}

- (void) loadView
{
   self.view = [[[NSBundle mainBundle] loadNibNamed:@"mainview"
      owner:self options:nil] lastObject];

   self.navigationController.navigationBar.tintColor = COOKBOOK_
    PURPLE_COLOR;
   NSString *valueString = [NSString stringWithFormat:@"%d", depth];
   NSString *nextString = [NSString stringWithFormat:@"Push %d",
      depth + 1];

   // Titel festlegen
   self.title = [@"Level " stringByAppendingString:valueString];

   // Hauptebene festlegen
   ((UILabel *)[self.view viewWithTag:101]).text = valueString;

   // Schaltflächenelement zum Weiterschalten hinzufügen
   // Die maximale Tiefe beträgt 6
   if (depth < 6) self.navigationItem.rightBarButtonItem =
   BARBUTTON(nextString, @selector(push));
}
@end
```

▸ *Rezept 5.3: Ansichten mit* `UINavigationController` *durchlaufen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 5 und öffnen das Projekt zu diesem Rezept.

5.6 Rezept: Controller beliebig vom Stack entfernen

Normalerweise möchten Sie beim Antippen der **Zurück**-Schaltfläche zum vorherigen Ansichtscontroller zurückkehren. Es gibt jedoch auch Situationen, in denen Sie stattdessen den gesamten Stack leeren wollen. Stellen Sie sich z. B. vor, dass ein Benutzer einen interaktiven Test durchgespielt oder ein Museumsbesucher seine Führung beendet hat. In diesen Fällen ist es nicht sinnvoll, sich Bildschirm für Bildschirm durch eine lange Navigationsstruktur zurückzubewegen. Hier verwenden Sie stattdessen `popToRootViewControllerAnimated:`, um den Stack zu leeren, alle Ansichtscontroller außer der Wurzel zu entfernen und die Anzeige entsprechend zu aktualisieren.

Wenn Sie nicht zur Wurzel, sondern zu einem bestimmten anderen Controller zurückkehren möchten, verwenden Sie `popToViewController: animated:`. Dadurch wird der Stack so weit geleert, bis als oberste Ansicht der angegebene Ansichtscontroller übrig bleibt. Um einfach ein Element zurückzugehen, als hätte der Benutzer auf die **Zurück**-Schaltfläche getippt, verwenden Sie `popViewControllerAnimated:`.

5.6.1 Ein Ansichtscontrollerarray laden

Sie können ein Array aus `UIViewController`-Objekten erstellen und der Eigenschaft `viewControllers` eines `UINavigationController` zuweisen, wobei dieses Array für den aktuellen Controllerstack steht. Der oberste Ansichtscontroller (also der aktive) nimmt die letzte Position im Array ein (n - 1), während sich das Wurzelobjekt beim Index 0 befindet.

Es gibt verschiedene Gründe dafür, diese Arrayeigenschaft zu verwenden. Mit Controllerarrays können Sie frühere Zustände wiederherstellen, nachdem Sie die Anwendung verlassen und wieder aufgerufen haben. So können Sie etwa eine Statusliste in den Benutzervoreinstellungen speichern und diese beim Start wieder laden, sodass der Benutzer wieder zu der Position in der Controllerhierarchie zurückkehrt, an der er das Programm verlassen hat.

Arrays sind beim Hin- und Herspringen in einer konzeptionellen Baumstruktur sehr hilfreich. Wenn Sie sich beispielsweise durch Verzeichnisse bewegen, kann es sein, dass Sie über eine symbolische Verknüpfung irgendwo anders hinspringen müssen. Durch Einrichtung des gesamten Arrays vermeiden Sie die aufwendige Arbeit, Controller auf den Stack zu verschieben und anschließend wieder zu entfernen.

5.6.2 Temporäre Ansichten auf den Stack legen

Immer wieder begegne ich Entwicklern, die gern `UIViewController` anzeigen lassen möchten, die nicht im Stack verbleiben sollen. Beispielsweise möchten Sie mit Ansicht 1 beginnen, zu Ansicht 2 übergehen und dann zu Ansicht 3, wobei die **Zurück**-Schaltfläche von Ansicht 3 zur ersten Ansicht führt.

Eine solche Situation tritt häufiger auf, als Sie vielleicht denken mögen. Meistens kommt dies vor, wenn Sie in der zweiten Ansicht eine Einführung in die Aktion geben, die in der dritten Ansicht stattfindet. Gewöhnlich enthält der zweite Bildschirm dann Anweisungen, allgemeine Informationen oder eine Art Startbildschirm. Diese Inhalte sollen einmal angezeigt werden, dann aber aus der Benutzeroberfläche verschwinden, wobei sich jedoch der Navigationscontroller so normgerecht wie möglich verhalten soll. Dazu muss die **Zurück**-Schaltfläche die zweite, temporäre Ansicht ignorieren.

In *Rezept 5.4* sehen Sie, wie das geht. Wenn die zweite Ansicht zum Übergang zur dritten bereit ist, schiebt der Navigationscontroller die dritte auf den Stack. Dadurch entsteht für den Betrachter die korrekte Animation, nämlich der Übergang von Ansicht 2 zu Ansicht 3. Dann entfernt der Code ohne Animation die letzten beiden Ansichten, sodass im Stack Ansicht 1 obenauf liegt. Zum Abschluss legt der Code mit verzögerter Animation Ansicht 3 über Ansicht 1 auf den Stack, um die korrekte **Zurück**-Schaltfläche zu erhalten.

Die Hauptanimation zeigt zwar korrekt den Übergang von Ansicht 2 zu Ansicht 3, die Animation der Navigationsleiste vollführt aber einen Wechsel von der Wurzel zu Ansicht 3. Dies sollte zwar noch nicht dazu führen, dass Ihre Anwendung wegen Verletzung der Interaktionsrichtlinien im App Store abgelehnt wird, aber Sie sollten die Schnittstelle so gestalten, dass es möglichst nicht zu Unterbrechungen in der Anzeige kommt.

```
- (void) doPush: (id) nc
{
   // Wechselt zur Ansicht depth+1, wenn der Stack bei Ansicht 1 ist
   [nc pushViewController:[[TestBedViewController alloc]
      initWithDepth:depth+1] animated:YES];
}

- (void) push
{
   if (depth < 2)
   {
      [self.navigationController
         pushViewController:[[TestBedViewController alloc]
         initWithDepth:depth+1] animated:YES];
      return;
   }
```

```
    // Wechselt von der aktuellen Ansicht zur Ansicht depth+1 und zeigt eine
    // Animation
    [self.navigationController
       pushViewController:[[TestBedViewController alloc]
       initWithDepth:depth+1] animated:YES];

    // Vorbereitung auf den Wechsel von Ansicht 1 zu Ansicht depth+1
    [self performSelector:@selector(doPush:)

       withObject:self.navigationController afterDelay:0.05f];

    // Entfernt Ansicht depth+1 und dann Ansicht depth vom Stack
    [[self.navigationController topViewController] autorelease];
    [self.navigationController popViewControllerAnimated:NO];
    [[self.navigationController topViewController] autorelease];
    [self.navigationController popViewControllerAnimated:NO];
}
```

▸ *Rezept 5.4: Temporäre Ansichten auf den Stack legen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 5 und öffnen das Projekt zu diesem Rezept.

5.7 Rezept: Eine benutzerdefinierte modale Informationsansicht anzeigen

Modale Ansichtscontroller werden auf dem Bildschirm angezeigt, gehören aber nicht zum Ansichtscontroller-Standardstack. Sie dienen dazu, Daten auszuwählen oder Informationen darzustellen, erledigen also Aufgaben, die sich nicht gut in die normale Hierarchie einfügen. Alle Ansichts- und Navigationscontroller können einen modalen Controller anzeigen:

```
[self presentModalViewController:[[[InfoViewController alloc] init]
   autorelease] animated:YES];
```

Bei dem dargestellten Controller kann es sich sowohl um einen Ansichts- als auch um einen Navigationscontroller handeln. In jedem Fall wird eine **Fertig**-Schaltfläche angezeigt, damit der Benutzer den Controller wieder ausblenden kann. In *Abbildung 5.4* sehen Sie eine modale Darstellung, die auf einer Instanz von `UIViewController` beruht. Die Navigationsleiste oben in der Ansicht wurde mithilfe einer `UINavigationBar`-Instanz hinzugefügt, sodass sich diese Ansicht sehr einfach im Interface Builder erstellen lässt.

▸ *Abbildung 5.4: Diese modale Ansicht wurde auf der Grundlage eines* `UIViewControllers` *mithilfe einer* `UINavigationBar` *erstellt.*

Normalerweise sind für Ansichten auf der Grundlage von Navigationscontrollern zwei `.xib`-Dateien und zusätzliche Arbeit erforderlich, wie die Anleitungen in Kapitel 4 zum Erstellen einer Schnittstelle mit Navigationsvorrichtungen gezeigt haben. Durch die direkte Verwendung der Leiste können Sie diesen Aufwand vermeiden und eine elegante Lösung erzielen, die das normale Erscheinungsbild einer `UINavigationController`-Darstellung nachahmt.

Rezept 5.5 zeigt die beiden Hauptbestandteile dieser Darstellung. Die Anzeige erfolgt im Hauptansichtscontroller, wobei der Stil durch ein segmentiertes Steuerelement festgelegt wird. Das Ausblenden wird durch `InfoViewController` erledigt, also die Klasse, die dargestellt wird. Ihre **Fertig**-Schaltfläche (**Done**) wird im Interface Builder mit der Methode `doneReading` verknüpft. Diese Methode weist das Elternelement des Ansichtscontrollers an, den modal angezeigten Ansichtscontroller zu entfernen.

```
// Den Controller anzeigen
- (void) info
{
   int segment = [(UISegmentedControl *)self.navigationItem.titleView
      selectedSegmentIndex];
   int styles[3] = {UIModalTransitionStyleCoverVertical,
      UIModalTransitionStyleCrossDissolve,
      UIModalTransitionStyleFlipHorizontal};
   InfoViewController *ivc = [[[InfoViewController alloc] init]
      autorelease];
   ivc.modalTransitionStyle = styles[segment];
   [self presentModalViewController:ivc animated:YES];
}
```

Und:

```
// Den Controller entfernen
- (IBAction) doneReading
{
   [[self parentViewController]
      dismissModalViewControllerAnimated:YES];
}
```

► *Rezept 5.5:Einen modalen Controller anzeigen und entfernen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 5 und öffnen das Projekt zu diesem Rezept.

5.8 Rezept: Tab Bars

Die Klasse `UITabBarController` ermöglicht es Benutzern, sich zwischen mehreren Ansichtscontrollern zu bewegen und die Leiste am unteren Rand des Bildschirms anzupassen. Dies können Sie am besten in den Programmen *YouTube* und *iPod* erkennen. Beide bieten mit einem Tipp Zugriff auf verschiedene Ansichten und verfügen zusätzlich über eine **Mehr**-Schaltfläche, die dem Benutzer verschiedene Auswahl- und Bearbeitungsmöglichkeiten eröffnet.

► *Abbildung 5.5: Die Tab-Bar-Controller ermöglichen Benutzern die Auswahl von Ansichtscontrollern aus einer Leiste am unteren Rand des Bildschirms (links) und die Anpassung der Tab Bar über eine Liste verfügbarer Ansichtscontroller (rechts).*

Auf Tab Bars verschieben Sie Ansichten auf andere Weise als in Navigationsleisten. Sie erstellen eher eine Sammlung von Controllern (dies können `UIViewController`, `UINavigationController` oder jede andere Art von Controllern sein) und fügen sie in eine Tab Bar ein, indem Sie die Eigenschaften des `viewControllers` der Tab Bar einrichten. Mehr ist wirklich nicht zu tun. Cocoa Touch übernimmt die restliche Arbeit für Sie. Stellen Sie `allowsCustomizing` auf `YES`, um Benutzern die Neuanordnung der Tab Bar zu ermöglichen.

Rezept 5.6 erstellt elf einfache Ansichtscontroller der Klasse `BrightnessController`. Diese Klasse verwendet eine in `mainview.xib` eingebettete `UIView` und stellt den Hintergrund auf einen festgelegten Grauwert ein, in diesem Fall von 0 % bis 100 % in Zehn-Prozent-Schritten. *Abbildung 5.5* (links) zeigt die Schnittstelle in diesem Standardmodus, in dem die ersten vier Elemente und eine **Mehr**-Schaltfläche (**More**) angezeigt werden.

Beachten Sie, dass dieses Rezept die elf Controller zweimal hinzufügt. Das erste Mal werden sie auf die Liste der Ansichtscontroller gesetzt, die für den Benutzer zur Verfügung stehen:

```
tbarController.viewControllers = controllers;
```

Das zweite Mal wird festgelegt, dass der Benutzer beim interaktiven Verändern der Tab Bar aus der gesamten Liste auswählen kann:

```
tbarController.customizableViewControllers = controllers;
```

Die zweite Zeile ist optional, die erste zwingend erforderlich. Nachdem Sie die Ansichtscontroller eingerichtet haben, können Sie Teile oder alles davon zu der anpassbaren Liste hinzufügen. Sollten Sie dies nicht tun, sind die zusätzlichen Ansichtscontroller über die **Mehr**-Schaltfläche zwar noch sichtbar, doch können Benutzer sie nicht bei Bedarf in die Hauptleiste einschließen.

Grafiken werden auf dem **Mehr**-Bildschirm invertiert dargestellt. Apple gibt an, dass dies das erwartete und richtige Verhalten ist und dass auch keine Pläne bestehen, dies zu ändern. Dies bietet einen interessanten Ansichtskontrast, wenn beispielsweise Ihr 100 % weißes Farbfeld auf diesem Bildschirm in reinem Schwarz dargestellt wird.

```
@implementation BrightnessController
- (UIImage*) buildSwatch: (float) tint
{
   CGContextRef context = [GraphicsUtilities
      newBitmapContextWithWidth:30 andHeight:30];
   [GraphicsUtilities addRoundedRect:
      CGRectMake(0.0f, 0.0f, 30.0f, 30.0f) toContext:context
      withWidth:4.0f andHeight:4.0f];
   CGFloat gray[4] = {tint, tint, tint, 1.0f};
   CGContextSetFillColor(context, gray);
   CGContextFillPath(context);

   CGImageRef myRef = CGBitmapContextCreateImage (context);
   free(CGBitmapContextGetData(context));
   CGContextRelease(context);
```

```
    UIImage *img = [UIImage imageWithCGImage:myRef];
    CFRelease(myRef);
    return img;
}

-(BrightnessController *) initWithBrightness: (int) aBrightness
{
    self = [super init];
    brightness = aBrightness;
    self.title = [NSString stringWithFormat:@"%d%%", brightness * 10];
    [self.tabBarItem initWithTitle:self.title image:[self
        buildSwatch:(((float)brightness) / 10.0f)] tag:0];
    return self;
}

- (void) loadView
{
    self.view = [[[NSBundle mainBundle] loadNibNamed:@"mainview"
        owner:self options:nil] lastObject];
    UIView *bigSwatch = [self.view viewWithTag:101];
    bigSwatch.backgroundColor = [UIColor colorWithWhite:
        (brightness / 10.0f) alpha:1.0f];
}
@end

@interface TestBedAppDelegate : NSObject <UIApplicationDelegate,
    UITabBarControllerDelegate>
@end

@implementation TestBedAppDelegate
- (void)applicationDidFinishLaunching:(UIApplication *)application {
    NSMutableArray *controllers = [NSMutableArray array];

    for (int i = 0; i <= 10; i++)
    {
        BrightnessController *bControl = [[BrightnessController alloc]
            initWithBrightness:i];
        UINavigationController *nav = [[UINavigationController alloc]
            initWithRootViewController:bControl];
        nav.navigationBar.barStyle = UIBarStyleBlackTranslucent;
        [bControl release];
```

```
        [controllers addObject:nav];
        [nav release];
    }

    // Erstellt die Symbolleiste und fügt die Ansichtscontroller hinzu
    UITabBarController *tbarController = [[UITabBarController alloc]
        init];
    tbarController.viewControllers = controllers;
    tbarController.customizableViewControllers = controllers;
    tbarController.delegate = self;

    // Richtet das Fenster ein
    UIWindow *window = [[UIWindow alloc] initWithFrame:[[UIScreen
        mainScreen] bounds]];
    [window addSubview:tbarController.view];
    [window makeKeyAndVisible];
}
@end
```

► *Rezept 5.6: Einen Tab-Bar-Controller erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 5 und öffnen das Projekt zu diesem Rezept.

5.9 Rezept: Tab-Bar-Status beibehalten

Dauerhaftigkeit ist auf dem iPhone Gold. Wenn Sie eine Anwendung starten oder nach einer Unterbrechung wieder aufnehmen, sollten Sie die Benutzer stets zu einem Anwendungsstatus zurückführen, der möglichst genau dem entspricht, bei dem sie das Programm verlassen haben. Dadurch können die Benutzer Aufgaben wieder aufnehmen und erhalten eine Schnittstelle, die mit derjenigen der letzten Sitzung übereinstimmt. *Rezept 5.7* zeigt ein Beispiel dafür.

In diesem Rezept wird sowohl die aktuelle Reihenfolge der Tabs als auch der zurzeit ausgewählte Tab gespeichert. Dies geschieht bei jeder Aktualisierung dieser Elemente. Wenn ein Benutzer die Anwendung startet, sucht der Code nach den vorherigen Einstellungen und wendet sie an.

Der hier gezeigte Ansatz stützt sich auf zwei Delegate-Methoden. Die erste, `tabBarController:didEndCustomizingViewControllers:`, gibt das aktuelle Array der Ansichtscontroller an, nachdem der Benutzer die Ansicht auf dem Bearbeitungbildschirm angepasst hat. Der Code verwendet die Informationen in den Titeln (**10 %, 20 %** usw.), um mit den einzelnen Ansichtscontrollern Namen zu verbinden.

Die zweite Delegate-Methode ist `tabBarController: didSelectViewController:`. Jedes Mal, wenn ein Benutzer ein neues Symbol in der Tab Bar auswählt, sendet der Tab-Bar-Controller diese Methode. Der Code liest `selectedIndex` und speichert damit die Nummer des Controllers im aktuellen Array.

Zum Festlegen dieser Werte wird das integrierte System für Benutzervorgaben auf dem iPhone herangezogen, `NSUserDefaults`, das sich wie ein großes, veränderbares Dictionary verhält. Mit `setObject: forKey:` können Sie Werte für gegebene Schlüssel festlegen:

```
[[NSUserDefaults standardUserDefaults] setObject:titles
   forKey:@"tabOrder"];
```

Der Abruf erfolgt mit `objectForKey:`:

```
NSArray *titles = [[NSUserDefaults standardUserDefaults]
   objectForKey:@"tabOrder"];
```

Sie müssen die Einstellungen wie im Code gezeigt stets synchronisieren, damit das Dictionary für die Vorgabewerte die Änderungen widerspiegelt. Wenn Sie dies versäumen, werden die Vorgabewerte erst beim Beenden des Programms gesetzt. Beim Synchronisieren werden die Voreinstellungen bei Änderungen jedoch unmittelbar aktualisiert. Sämtliche Teile Ihrer Anwendung, die diese Einstellungen lesen müssen, haben daher immer Zugriff auf die neuesten Werte.

Beim Start sucht die Anwendung nach den Einstellungen für die letzte Symbolreihenfolge und das letzte aktivierte Symbol. Wenn sie solche Einstellungen findet, richtet sie damit die Tab Bar ein und macht den richtigen Tab aktiv. Da der gespeicherte Titel Informationen über den anzuzeigenden Helligkeitswert enthält, konvertiert der Code ihn aus dem Text- in ein Zahlenformat, dividiert die gefundene Zahl durch zehn und sendet das Ergebnis an die Initialisierungsfunktion.

Die meisten Anwendungen sind nicht auf so einem einfachen Zahlensystem aufgebaut. Wenn Sie in den Titeln die Reihenfolge in der Tab Bar speichern, müssen Sie die Ansichtscontroller sinnvoll benennen, sodass sie die Anordnung der Symbole widerspiegeln.

Hinweis

Sie können auch ein Array der Ansichts-Tags als `NSNumbers` speichern oder, was noch besser ist, die Klasse `NSKeyedArchiver` verwenden, die in Kapitel 8, *Gesten und Berührungen*, vorgestellt wird. Bei der schlüsselgestützten Archivierung können Sie Ansichten mithilfe von Statusinformationen wiederherstellen, die beim Beenden des Programms gespeichert werden.

```
@implementation TestBedAppDelegate
- (void)tabBarController:(UITabBarController *)tabBarController
didEndCustomizingViewControllers:(NSArray *)viewControllers
changed:(BOOL)changed
{
   // Speichert die Titel der Symbole in der aktuellen Reihenfolge
```

```
    NSMutableArray *titles = [NSMutableArray array];
    for (UIViewController *vc in viewControllers) [titles
        addObject:vc.title];
    [[NSUserDefaults standardUserDefaults] setObject:titles
        forKey:@"tabOrder"];
    [[NSUserDefaults standardUserDefaults] synchronize];
}

- (void)tabBarController:(UITabBarController *)tabBarController
        didSelectViewController:(UIViewController *)viewController
{
    // Aktualisiert die Angabe des zurzeit ausgewählten Symbols
    NSNumber *tabNumber = [NSNumber numberWithInt:[tabBarController
        selectedIndex]];
    [[NSUserDefaults standardUserDefaults] setObject:tabNumber
        forKey:@"selectedTab"];
    [[NSUserDefaults standardUserDefaults] synchronize];
}

- (void)applicationDidFinishLaunching:(UIApplication *)application {
    NSMutableArray *controllers = [NSMutableArray array];
    NSArray *titles = [[NSUserDefaults standardUserDefaults]
        objectForKey:@"tabOrder"];

    if (titles)
    {
        // Aus den Benutzervoreinstellungen abgerufene Titel
        for (NSString *theTitle in titles)
        {
            BrightnessController *bControl = [[BrightnessController
                alloc] initWithBrightness:([theTitle intValue] / 10)];
            UINavigationController *nav = [[UINavigationController
                alloc] initWithRootViewController:bControl];
            nav.navigationBar.barStyle = UIBarStyleBlackTranslucent;
            [bControl release];

            [controllers addObject:nav];
            [nav release];
        }
    } else {
        // Generiert alle neuen Controller
        for (int i = 0; i <= 10; i++)
        {
            BrightnessController *bControl = [[BrightnessController
                alloc] initWithBrightness:i];
```

```
            UINavigationController *nav = [[UINavigationController
               alloc] initWithRootViewController:bControl];
            nav.navigationBar.barStyle = UIBarStyleBlackTranslucent;
            [bControl release];

            [controllers addObject:nav];
            [nav release];
        }
    }

    // Erstellt die Symbolleiste und fügt die Ansichtscontroller hinzu
    UITabBarController *tbarController = [[UITabBarController alloc]
       init];
    tbarController.viewControllers = controllers;
    tbarController.customizableViewControllers = controllers;
    tbarController.delegate = self;

    NSNumber *tabNumber = [[NSUserDefaults standardUserDefaults]
       objectForKey:@"selectedTab"];
    if (tabNumber)
       tbarController.selectedIndex = [tabNumber intValue];

    // Richtet das Fenster ein
    UIWindow *window = [[UIWindow alloc] initWithFrame:[[UIScreen
       mainScreen] bounds]];
    [window addSubview:tbarController.view];
    [window makeKeyAndVisible];
}
@end
```

► *Rezept 5.7: Den Status der Tab Bar in den Benutzervoreinstellungen speichern*

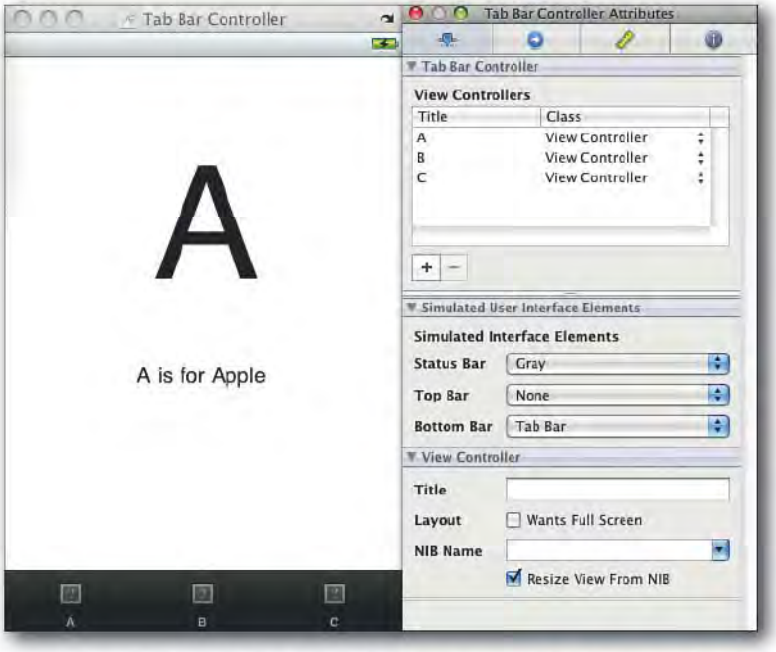
Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 5 und öffnen das Projekt zu diesem Rezept.

5.10 Zum guten Schluss: Tab-Bar-Controller im Interface Builder

In Xcode gibt es die leicht anzupassende Vorlage **Tab Bar Application** für den Aufbau von grafischen Benutzerschnittstellen mit Tab Bars im Interface Builder. Im Attribut-Informationsfenster des Tab-Bar-Controllers können Sie neue Tabs hinzufügen, indem Sie auf **+** klicken, wie *Abbildung 5.6* zeigt. In der Klassenspalte können Sie die Art des Ansichtscontrollers auswählen. Möglich sind hier **View Controller**, **Table View Controller**, **Navigation Controller** und **Image Picker Controller**.

Gewöhnlich erstellen Sie für jeden Tab der Tab Bar einen neuen Ansichtscontroller. Dazu erstellen Sie die Klasse (und die zugehörige `.xib`-Datei) in Xcode und wählen die `.xib`-Datei dann im Interface Builder aus. Klicken Sie auf die schwarzen Tabs in `MainWindow.xib` und dann auf die graue Darstellung von `View`. Wählen Sie im Attribut-Informationsfeld ⌘ + 1 die `.xib`-Datei aus. Der erste Tab erscheint vorausgefüllt mit einer Ansicht (siehe *Abbildung 5.6*), aber allen anderen muss noch eine `.xib`-Datei zugeordnet werden.



▸ *Abbildung 5.6: Der Interface Builder hat Werkzeuge zur Gestaltung von Tab-Bar-Controllern, bietet zum Erstellen einer eher logischen als grafischen Klasse aber kaum Vorteile.*

Um im Interface Builder Grafiken zu den Tabs hinzuzufügen, ziehen Sie `png`-Images von 20 × 20 Pixel aus dem Bereich **Library ▸ Media** auf die einzelnen Tab-Schaltflächen. Im **Media**-Bereich werden die Bilder aufgelistet, die Sie dem Xcode-Projekt hinzugefügt haben. Gestalten Sie die Bilder mit einem transparenten Hintergrund und einem weißen Vordergrund.

Der Interface Builder bietet zwar angenehme Möglichkeiten zur Gestaltung von einzelnen Ansichten, doch für Tab Bars und Navigationsleisten können Sie darauf verzichten, denn diese Klassen sind eher ein logisches Konstrukt als eine grafische Darstellung. Schließlich können Sie eine Tab Bar oder eine Navigationsleiste nicht auf dem Bildschirm verschieben. Alle Anpassungen lassen sich auch einfach im Code erledigen.

Sobald Sie anfangen, Delegate-Callbacks zu nutzen, die direkt im Code ansetzen, lohnt sich der Zusatzaufwand im Interface Builder ohnehin nicht mehr. Der Interface Builder eignet sich am besten für die Gestaltung von Ansichten, für Tab-Bar- und Navigationscontroller dagegen bietet er nur wenig Nutzen.

5.11 Zusammenfassung

In diesem Kapitel haben Sie die Klassen `UIViewController`, `UINavigationController` und `UITabBarController` in Aktion erlebt. Sie haben gelernt, wie Sie sie für die Darstellung von Ansichten und die Benutzernavigation einsetzen, wie Sie damit den virtuellen Interaktionsraum erweitern und mehrseitige Schnittstellen für Anwendungen erstellen. Bevor Sie zum nächsten Kapitel weiterblättern, sollten Sie sich die folgenden Punkte zum Thema Ansichtscontroller einprägen:

- Verwenden Sie Navigationsbäume, um hierarchische Schnittstellen zu erstellen. Sie eignen sich hervorragend für den Zugriff auf Dateistrukturen oder den Aufbau eines Einstellungsbaums. Wenn Sie mit »Einblendansichten« oder »Voreinstellungen« liebäugeln, sollten Sie dazu einen neuen Controller auf den Navigationsstack stellen.
- Scheuen Sie sich nicht, normale Elemente der grafischen Benutzerschnittstelle auf ungewöhnliche Art und Weise zu verwenden, sofern Sie damit im Rahmen der Apple-Richtlinien für Benutzeroberflächen bleiben. In diesem Kapitel haben Sie innovative Anwendungen von `UINavigationController` kennengelernt, bei denen es gar nicht um Navigation geht. Die Werkzeuge sind dazu da, benutzt zu werden.
- Sorgen Sie für Dauerhaftigkeit. Lassen Sie die Benutzer zu dem Status der grafischen Benutzerschnittstelle zurückkehren, in dem sie sie beim letzten Mal verlassen haben. `NSUserDefaults` ist ein integriertes System zum Speichern von Informationen über das Beenden einer Anwendung hinaus. Verwenden Sie diese Vorgabewerte, um den früheren Zustand der Oberfläche wiederherzustellen.
- Der Interface Builder eignet sich am besten für das grafische Layout. Viele Entwickler setzen ihn zur Gestaltung von Ansichten ein, meiden ihn aber beim Aufbau von Navigations- und Tab-Bar-Controllern.

6 Ansichten und Animationen zusammenstellen

Die Klasse `UIView` und ihre Unterklassen dienen dazu, den Bildschirm des iPhones zu füllen. In diesem Kapitel werden Sie von Grund auf in den Umgang mit Ansichten eingeführt. Sie lernen, wie Sie Ansichtshierarchien erstellen, untersuchen und zerlegen und wie Ansichten zusammenwirken. Außerdem erfahren Sie, welche Rolle die Geometrie beim Erstellen und Platzieren der Ansichten in der Schnittstelle spielt und wie Sie Ansichten animieren, sodass sie sich auf dem Bildschirm bewegen und umwandeln. Dieses Kapitel behandelt von den Grundlagen an alles, was Sie über die Arbeit mit Ansichten wissen müssen.

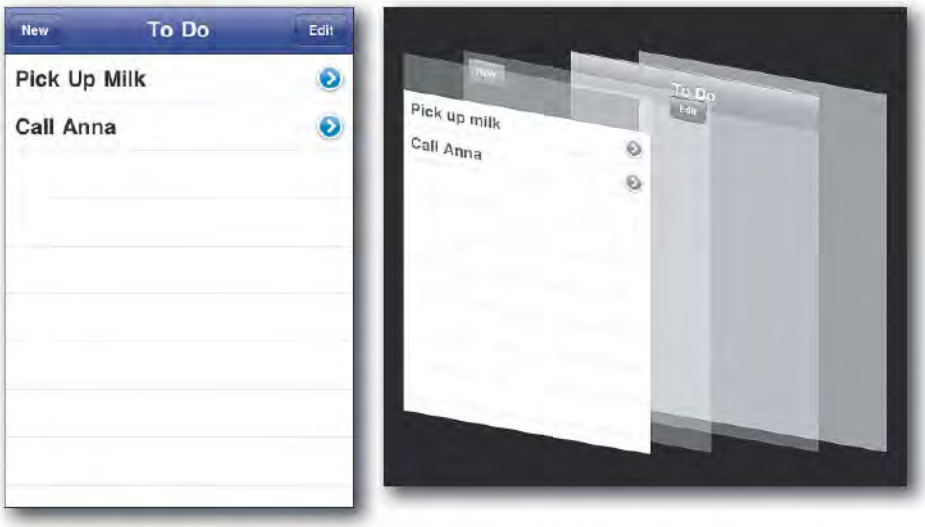
6.1 Ansichtshierarchien

Was Sie auf dem Bildschirm eines iPhones sehen, basiert auf einer baumartigen Hierarchie. Ansichten werden auf eine bestimmte hierarchische Art und Weise angeordnet, wobei das Hauptfenster den Ausgangspunkt darstellt. Alle Ansichten können Kinder, sogenannte Unteransichten, haben. Jede Ansicht, das Fenster eingeschlossen, hat eine geordnete Liste dieser Unteransichten. Ansichten können viele Unteransichten oder auch keine besitzen. Die Anwendung bestimmt, wie Ansichten angeordnet werden und welche zu welcher gehören.

Unteransichten erscheinen in ihrer Reihenfolge auf dem Bildschirm, immer von hinten nach vorne. Dies entspricht der Funktionsweise von den transparenten Folien bei einem Zeichentrickfilm. Es sind nur die Teile der Folien sichtbar, die gefärbt sind. Die freien Flächen ermöglichen die Sicht auf die Elemente hinter dieser Folie. Auch Ansichten können über leere und gefüllte Bereiche verfügen und lassen sich daher zu komplexen Darstellungen überlagern.

Abbildung 6.1 zeigt ein Beispiel für die Schichten in einem typischen Fenster. Hier besitzt das Fenster eine Hierarchie, die auf einem `UINavigationController` basiert. Die Elemente sind übereinandergelagert. Das Fenster (dargestellt durch das durchsichtige Element auf der rechten Seite) hat eine Navigationsleiste, die wiederum eine Unteransicht mit zwei Schaltflächen (eine links und eine rechts) besitzt. Außerdem verfügt das Fenster über eine Tabelle mit einer eigenen Unteransicht. Diese Elemente werden übereinandergelegt und bilden so die grafische Benutzeroberfläche.

Listing 6.1 zeigt die tatsächliche Ansichtshierarchie des Fensters aus *Abbildung 6.1*. Der Baum beginnt am oberen `UIWindow` und umfasst die Klassen für die einzelnen Kindansichten. Wenn Sie den Baum nach unten durchlaufen, stoßen Sie auf die Navigationsleiste (in Ebene 2) mit ihren beiden Schaltflächen (beide auf Ebene 3) sowie die Tabellenansicht (Ebene 4) mit ihren beiden Zeilen (jeweils Ebene 5). Einige der Elemente in der Liste sind private Klassen, die das SDK automatisch beim Stapeln von Ansichten hinzufügt. Beispielsweise wird `UILayoutContainerView` niemals direkt vom Entwickler verwendet, sondern gehört zur Implementierung des `UIWindow` durch das SDK.



▸ *Abbildung 6.1: Hierarchisch gegliederte Unteransichten fügen sich zu komplexen grafischen Benutzerschnittstellen zusammen.*

Das Einzige, was in diesem Listing fehlt, sind etwa ein Dutzend Zeilentrenner in der Tabelle, die ich hier aus Platzgründen weggelassen habe. Bei diesen Zeilentrennern handelt es sich jeweils um Instanzen von `UITableViewSeparatorView`. Sie gehören zur `UITableView` und werden normalerweise auf Ebene 5 angezeigt.

```
[ 0] UIWindow
—[ 1] UILayoutContainerView
——[ 2] UINavigationTransitionView
———[ 3] UIViewControllerWrapperView
————[ 4] UITableView
—————[ 5] UITableViewCell
```

```
————[ 6] UIView
————[ 7] UILabel
————[ 6] UIButton
————[ 7] UIImageView
————[ 6] UIView
———[ 5] UITableViewCell
————[ 6] UIView
————[ 7] UILabel
————[ 6] UIButton
————[ 7] UIImageView
————[ 6] UIView
———[ 5] UIImageView
———[ 5] UIImageView
—[ 2] UINavigationBar
——[ 3] UINavigationItemView
——[ 3] UINavigationButton
———[ 4] UIImageView
———[ 4] UIButtonLabel
——[ 3] UINavigationButton
———[ 4] UIImageView
———[ 4] UIButtonLabel
```

▸ *Listing 6.1: Ansichtshierarchie der Aufgabenliste*

6.2 Rezept: Die Ansichtshierarchie abrufen

Jede Ansicht kennt sowohl ihre Elternansicht (`[aView superview]`) als auch ihre Kindansichten (`[aView subviews]`). Um einen Ansichtsbaum wie in *Listing 6.1* zu erstellen, durchlaufen Sie rekursiv die Unteransichten einer Ansicht. Genau dies erledigt *Rezept 6.1*. Der Code baut eine grafische Baumdarstellung in der Konsole von Xcode auf, indem er sich die Klasse jeder einzelnen Ansicht merkt und bei jedem Übergang von einer Elternansicht zu deren Kindern die Einrückung verstärkt. Die Ergebnisse werden in einem veränderbaren String gespeichert und von der aufrufenden Methode zurückgegeben.

Der Baum in *Listing 6.1* wurde mit dem Code aus *Rezept 6.1* erstellt. Die Schnittstelle und das Rezept finden Sie auch im Beispielcode zu diesem Buch. Sie können diese Routine einsetzen, um das Ergebnis von *Listing 6.1* nachzuvollziehen, Sie können sie aber auch bei anderen Anwendungen einsetzen, um deren Hierarchien anzuzeigen.

```
// Rekursiver Durchlauf abwärts durch den Ansichtsbaum, wobei die
// Einrückungsebene für die Kinder jeweils erhöht wird
- (void) dumpView: (UIView *) aView atIndent: (int) indent
      into:(NSMutableString *) outstring
{
   for (int i = 0; i < indent; i++) [outstring appendString:@"-"];
   [outstring appendFormat:@"[%2d] %@\n", indent,
```

```
        [[aView class] description]];
    for (UIView *view in [aView subviews])
        [self dumpView:view atIndent:indent + 1 into:outstring];
}

// Baumrekursion beginnt bei der Wurzelansicht auf Ebene 0
- (NSString *) displayViews: (UIView *) aView
{
    NSMutableString *outstring = [[NSMutableString alloc] init];
    [self dumpView:aView atIndent:0 into:outstring];
    return [outstring autorelease];
}
```

▸ *Rezept 6.1: Einen Ansichtsbaum durchlaufen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.3 Rezept: Unteransichten abfragen

In Ansichten sind Arrays ihrer Kinder gespeichert, die Sie mit dem Aufruf `[aView subviews]` abrufen können. Auf dem Bildschirm werden die Kindansichten immer nach ihren Elternansichten gezeichnet, und zwar in der Reihenfolge, in der sie im Array der Unteransichten auftreten. Diese Reihenfolge spiegelt die Tatsache wider, dass die Unteransichten von hinten nach vorn gezeichnet werden. Ansichten, die weiter hinten im Array erscheinen, werden nach den Ansichten an einer früheren Position gezeichnet.

Die Methode `subviews` gibt nur die unmittelbaren Kindansichten einer gegebenen Ansicht zurück. Manchmal ist es jedoch sinnvoll, eine ausführlichere Liste der Unteransichten zu gewinnen, die auch die Enkelansichten einschließt. *Rezept 6.2* stellt die rekursive Funktion `allSubviews()` vor, die sämtliche Abkömmlinge einer Ansicht zurückgibt. Wenn Sie diese Funktion mit `view.window` aufrufen, erhalten Sie die vollständige Menge der Unteransichten in dem `UIWindow`, das die Ansicht beherbergt. Diese Liste ist sehr nützlich, wenn Sie nach einer bestimmten Ansicht suchen müssen, z. B. nach einem einzelnen Schieberegler oder einer Schaltfläche.

Es ist zwar nicht üblich, doch können iPhone-Anwendungen mehrere Fenster umfassen, die jeweils viele Ansichten aufweisen können. Eine umfassende Liste aller Anwendungsansichten erhalten Sie, indem Sie eine Iteration durch sämtliche verfügbaren Fenster durchführen. So geht die Funktion `allApplicationSubviews()` in *Rezept 6.2* vor. Ein Aufruf von `[[UIApplication sharedApplication]` gibt das Array der Anwendungsfenster zurück. Diese Funktion iteriert durch diese Fenster und fügt deren Unteransichten zu der Sammlung hinzu.

6.3 Rezept: Unteransichten abfragen

Eine Ansicht kennt nicht nur ihre Unteransichten, sondern weiß auch, zu welchem Fenster sie gehört, denn darauf zeigt ihre Eigenschaft `window`. *Rezept 6.2* enthält auch eine einfache Funktion namens `pathToView()`, die ein Array der übergeordneten Ansichten zurückgibt, und zwar vom Fenster bis zur fraglichen Ansicht. Hierzu ruft die Funktion wiederholt `superview` auf, bis sie das Fenster erreicht.

Die Vorfahren von Ansichten lassen sich auch auf andere Weise bestimmen. Die Methode `isDescendantOfView:` bestimmt, ob sich eine Ansicht innerhalb einer anderen befindet, selbst wenn diese keine direkt übergeordnete Ansicht ist. Als Ergebnis gibt die Methode einen einfachen booleschen Wert zurück. `YES` bedeutet, dass die Ansicht von der als Parameter übergebenen Ansicht abstammt.

```
// Gibt eine ausführliche Liste der Unteransichten einer Ansicht zurück
NSArray *allSubviews(UIView *aView)
{
   NSArray *results = [aView subviews];
   for (UIView *eachView in [aView subviews])
   {
      NSArray *riz = allSubviews(eachView);
      if (riz) results = [results arrayByAddingObjectsFromArray:riz];
   }
   return results;
}

// Gibt alle Ansichten innerhalb der Anwendung zurück
NSArray *allApplicationViews()
{
   NSArray *results = [[UIApplication sharedApplication] windows];
   for (UIWindow *window in [[UIApplication sharedApplication] windows])
   {
      NSArray *riz = allSubviews(window);
      if (riz) results = [results arrayByAddingObjectsFromArray: riz];
   }
   return results;
}

// Gibt ein Array der Elternansichten vom Fenster bis zur betreffenden
// Ansicht zurück.
NSArray *pathToView(UIView *aView)
{
   NSMutableArray *array = [NSMutableArray arrayWithObject:aView];
   UIView *view = aView;
   UIWindow *window = aView.window;
   while (view != window)
   {
```

```
        view = [view superview];
        [array insertObject:view atIndex:0];
    }
    return array;
}
```

▸ *Rezept 6.2: Hilfsfunktionen für Unteransichten*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.4 Unteransichten verwalten

Die Klasse `UIView` enthält viele Methoden zum Erstellen und Verwalten von Ansichten. Damit können Sie Ansichten zur Hierarchie hinzufügen und aus ihr entfernen sowie die Hierarchie umordnen und abfragen. Da die Hierarchie festlegt, was auf dem Bildschirm zu sehen ist, ändert sich bei einer Anpassung der Beziehungen zwischen den Sichten die Darstellung auf dem iPhone. Im Folgenden finden Sie einige Verfahrensweisen für typische Aufgaben der Ansichtsverwaltung.

6.4.1 Unteransichten hinzufügen

Um einer Ansicht neue Unteransichten hinzuzufügen, rufen Sie `[parentView addSubview: child]` auf. Neue Unteransichten erscheinen stets vorn auf dem Bildschirm, da das iPhone sie über allen bisherigen Ansichten hinzufügt. Um eine Unteransicht an einer bestimmten Stelle der Ansichtshierarchie einfügen zu können, bietet das SDK die folgenden drei Hilfsmethoden an:

- `insertSubview: atIndex:`
- `insertSubview: aboveSubview:`
- `insertSubview: belowSubview:`

Diese Methoden legen fest, wo die neue Ansicht landet. Die Einfügung kann relativ zu einer anderen Ansicht oder an einem bestimmten Index des Arrays der Unteransichten erfolgen. Die `above`- und `below`-Methoden fügen Unteransichten vor bzw. hinter einer gegebenen Kindansicht ein. Dabei werden vorhandene Ansichten nicht überschrieben, sondern nach vorn geschoben.

6.4.2 Unteransichten umordnen und entfernen

Bei der Benutzerinteraktion mit dem Bildschirm müssen Anwendungen häufig Ansichten umordnen und entfernen. Dazu gibt es im iPhone SDK verschiedene Möglichkeiten, um die Reihenfolge und den Inhalt von Ansichten zu ändern.

- Mit `[parentView exchangeSubviewAtIndex:i withSubviewAtIndex:j]` tauschen Sie die Positionen zweier Ansichten aus.
- Um Unteransichten in den Vordergrund oder Hintergrund zu verlegen, verwenden Sie `bringSubviewToFront:` bzw. `sendSubviewToBack`.
- Um eine Unteransicht zu entfernen, rufen Sie `[childView removeFromSuperview]` auf. Wird die Kindansicht zurzeit auf dem Bildschirm angezeigt, so verschwindet sie. Wenn Sie eine Unteransicht entfernen, erhält sie eine `release`-Nachricht, sodass ihr Speicher freigegeben werden kann, falls der Beibehaltungszähler auf null gefallen ist.

Beim Umordnen, Hinzufügen und Entfernen von Ansichten wird der Bildschirm automatisch neu gezeichnet, um die neue Ansichtsdarstellung zu zeigen.

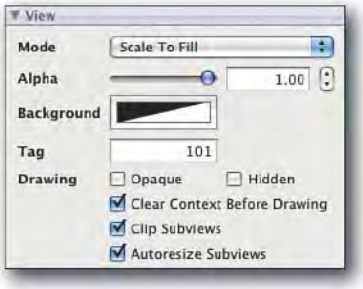
6.4.3 Ansichts-Callbacks

Bei einer Änderung der Ansichtshierarchie können Callbacks an die fraglichen Ansichten gesendet werden. Das iPhone SDK enthält sechs solcher Callback-Methoden, mit denen sich die Anwendung auf dem neuesten Stand über Ansichten halten kann, die verschoben werden oder ihre Elternansichten ändern.

- `didAddSubview:` wird nach dem erfolgreichen Aufruf von `addSubview:` an eine Ansicht gesendet. Dadurch können Unterklassen von `UIView` beim Hinzufügen von neuen Ansichten zusätzliche Vorgänge ausführen.
- `didMoveToSuperview:` informiert Ansichten darüber, dass sie einer neuen übergeordneten Ansicht zugeordnet wurden, damit die Ansicht auf irgendeine Weise auf ihre neue Elternansicht reagieren kann. Wenn die Ansicht von ihrer übergeordneten Ansicht entfernt wurde, ist die neue Elternansicht `nil`.
- `willMoveToSuperview:` wird gesendet, bevor eine Ansicht zu einer anderen Elternansicht verschoben wird.
- `didMoveToWindow:` ist das Gegenstück zu `didMoveToSuperview:` für den Fall, dass die Ansicht nicht zu einer neuen übergeordneten Ansicht, sondern zu einer neuen Fensterhierarchie verschoben wird.
- `willMoveToWindow:` wird wiederum gesendet, bevor die Verschiebung auftritt.
- `willRemoveSubview:` informiert eine Ansicht darüber, dass ihre Kindansicht entfernt wird.

6.5 Rezept: Ansichten mit Tags versehen und abrufen

Das iPhone SDK verfügt über eine integrierte Suchfunktion, mit der Sie Ansichten abrufen können, indem Sie sie mit Tags versehen. Tags sind einfach nur Zahlen, gewöhnlich positive Ganzzahlen, die eine Ansicht bezeichnen. Zugewiesen werden sie über die Eigenschaft `tag` einer Ansicht, z. B. als `myView.tag = 101`. Im Interface Builder können Sie das Tag einer Ansicht im Attribut-Informationsfeld festlegen. Wie Sie in *Abbildung 6.2* sehen, geben Sie das Tag im Abschnitt **View** an.



▶ *Abbildung 6.2: Das Tag für eine Ansicht legen Sie im Attribut-Informationsfenster des Interface Builder fest.*

Tags können Sie völlig willkürlich wählen, reserviert ist nur die 0 als Standardeinstellung für alle neu erstellen Ansichten. Die Entscheidung, wie die Tags angewendet werden und welche Werte sie annehmen, liegt ganz bei Ihnen. Sie können jegliche Instanzen mit Tags versehen, die Kinder von `UIView` sind, auch Fenster und Steuerelemente. Befinden sich auf dem Bildschirm also viele Schaltflächen und Umschalter, können Sie sie mithilfe von Tags unterscheiden, wenn die Benutzer sie auslösen. Fügen Sie zu Ihren Callback-Methoden eine einfache `switch`-Anweisung hinzu, damit sich die Methode das Tag ansieht und daraus ableitet, wie sie reagieren muss.

Apple selbst verwendet Tags für Unteransichten nur selten. Die einzigen Fälle, die ich bisher gesehen habe, sind die Schaltflächen in `UIAlertView`, die mit den Tags 1, 2 usw. bezeichnet sind. (Ich neige zu der Ansicht, dass die Tags nur aus Versehen dort zurückgeblieben sind.) Wenn Sie sich Sorgen über Konflikte mit Apple-Tags machen, beginnen Sie mit der Nummerierung bei 10 oder 100 oder irgendeiner anderen Zahl, die höher ist als die Werte, die Apple wahrscheinlich benutzt.

6.5.1 Tags zur Suche nach Ansichten verwenden

Mit Tags können Sie es vermeiden, Elemente der Benutzerschnittstelle in Ihrem Programm übergeben zu müssen, da Sie sie direkt von irgendeiner Elternansicht aus zugänglich machen. Die Methode `viewWithTag:` ruft eine mit Tags versehene Ansicht von der tiefer gelegenen Hierarchie ab. Die Suche erfolgt rekursiv, sodass das mit einem Tag versehene Element kein unmittelbares Kind der fraglichen Ansicht sein muss. Es ist möglich, dass Sie die Suche mit `[window viewWithTag:101]` vom Fenster aus durchführen und dabei eine Ansicht finden, die sich mehrere Äste tiefer im Hierarchiebaum befindet. Wenn ein Tag von mehreren Ansichten verwendet wird, gibt die Methode `viewWithTag:` das erste dieser Elemente zurück, das sie findet.

Das Problem bei `viewWithTag:` ist, dass diese Methode ein `UIView`-Objekt zurückgibt, sodass Sie es erst in den richtigen Typ umwandeln müssen, um es verwenden zu können. Nehmen wir an, dass Sie eine Beschriftung (`UILabel`) abrufen und deren Text festlegen möchten.

```
UILabel *label = (UILabel *)[self.view.window viewWithTag:101];
label.text = @"Hello World";
```

Es wäre weit einfacher, einen Aufruf zu verwenden, der ein bereits typisiertes Objekt zurückgibt, sodass Sie es unmittelbar verwenden können. Dies geschieht in den folgenden Aufrufen:

```
- (IBAction)updateTime:(id)sender
{
   // Setzt die Beschriftung auf die aktuelle Uhrzeit
   [self.view.window labelWithTag:LABEL_TAG].text =
      [[NSDate date] description];
}

- (IBAction)updateSwitch:(id)sender
{
   // Schaltet den Umschalter von der derzeitigen Einstellung um
   UISwitch *s = [self.view.window switchWithTag:SWITCH_TAG];
   [s setOn:!s.isOn];
}
```

Rezept 6.3 erweitert das Verhalten von `UIView`, um die neue Kategorie `TagExtensions`. Diese Kategorie fügt nur zwei typisierte Tag-Methoden für `UILabel` und `UISwitch` hinzu. Im Beispielcode zu diesem Buch ist dies zu einer vollständigen Suite von typisierten Tag-Hilfsmethoden ausgebaut. Aus Platzgründen habe ich hier auf die anderen Klassen verzichtet, doch sie folgen demselben Muster der Typumwandlung von `viewWithTag:`. Zugriff auf die vollständige Sammlung erhalten Sie, wenn Sie die `UIView-TagExtensions`-Dateien in Ihre Projekte aufnehmen.

```
@interface UIView (TagExtensions)
- (UILabel *) labelWithTag: (NSInteger) aTag;
- (UISwitch *) switchWithTag: (NSInteger) aTag;
@end

@implementation UIView (TagExtensions)
- (UILabel *) labelWithTag: (NSInteger) aTag
{
   return (UILabel *)[self viewWithTag:aTag];
}

- (UISwitch *) switchWithTag: (NSInteger) aTag
{
   return (UISwitch *)[self viewWithTag:aTag];
}
@end
```

▸ *Rezept 6.3: Mit Tags gekennzeichnete Ansichten mit korrekt umgewandelten Objekten abrufen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.6 Rezept: Ansichten benennen

Die Verwendung von Tags ist zwar eine solide Vorgehensweise zur Bezeichnung von Ansichten, doch manche Entwickler arbeiten lieber mit Namen als mit Zahlen. Dadurch erhält das Bezeichnungsschema für die Ansichten noch eine zusätzliche Bedeutung. Anstatt von der »Sicht mit dem Tag 101« sprechen Sie von einem Umschalter mit dem Namen »Ignition Switch« (»Auslöser«), was dessen Rolle beschreibt und für eine gewisse Form von Selbstdokumentation sorgt, die eine reine Zahl nicht bietet.

```
// Umschalter betätigen
UISwitch *s = [self.view switchNamed:@"Ignition Switch"];
[s setOn:!s.isOn];
```

Es ist relativ einfach, eine Klasse zu entwerfen, die Strings zu Ansichts-Tags zuordnet. Diese benutzerdefinierte Klasse muss ein Dictionary speichern, das Namen mit Tags verbindet, sodass Ansichten solche Namen registrieren und die Registrierung wieder aufheben können. *Rezept 6.4* zeigt, wie Sie einen solchen Ansichtsnamen-Manager erstellen können. Hier wird eine Singleton-Instanz genutzt (`[ViewIndexer sharedInstance]`), um das Tag/Namen-Dictionary zu speichern.

Die Klasse verlangt eindeutige Namen. Wenn ein Ansichtsname bereits registriert ist, schlägt eine weitere Registrierungsanforderung fehl. Wurde eine Ansicht bereits unter einem anderen Namen registriert, hebt die zweite Registrierung die erste auf. Es gibt natürlich einige Möglichkeiten, um das zu umgehen. Wenn Sie die Tags einer Ansicht ändern und sie dann erneut registrieren, kann der Indizierer nicht mehr erkennen, dass er diese Ansicht bereits registriert hat. Wenn Sie also diesen Ansatz verwenden, sollten Sie die Tags im Interface Builder angeben oder automatisch vom Registrierungsprozess festlegen lassen und die Tags ansonsten nicht mehr anfassen.

Falls Sie die Ansichten manuell erstellen, sollten Sie sie im selben Schritt registrieren und zur Ansichtshierarchie hinzufügen. Bei einer im Interface Builder erstellten Ansicht registrieren Sie die Namen in `viewDidLoad` mit den Tag-Nummern, die Sie im Attribute-Informationsfeld festgelegt haben.

```
- (void) viewDidLoad
{
   [[self.view viewWithTag:LABEL_TAG] registerName:@"my label"];
   [[self.view viewWithTag:SWITCH_TAG] registerName:@"my switch"];
}
```

Rezept 6.4 verbirgt die Klasse für den Ansichtsindizierer vor der Öffentlichkeit. Der Code verpackt die Aufrufe in einer `UIView`-Kategorie für Namenserweiterungen. Aus Platzgründen werden in dem Rezept keine Namensabrufe wie `labelNamed:` und `textFieldNamed:` gezeigt, aber sie sind im Beispielcode zu diesem Kapitel enthalten.

```
@interface ViewIndexer : NSObject {
   NSMutableDictionary *tagdict;
   NSInteger count;
}
@property (nonatomic, retain) NSMutableDictionary *tagdict;
@end
```

```
@implementation ViewIndexer
@synthesize tagdict;

static ViewIndexer *sharedInstance = nil;

+(ViewIndexer *) sharedInstance {
   if(!sharedInstance) sharedInstance = [[self alloc] init];
   return sharedInstance;
}

- (id) init
{
   if (!(self = [super init])) return self;
   self.tagdict = [NSMutableDictionary dictionary];
   count = 10000;
   return self;
}

- (void) dealloc
{
   self.tagdict = nil;
   [super dealloc];
}

// Generiert eine neue Nummer und erhöht den Zählerwert
- (NSInteger) pullNumber
{
   return count++;
}

// Prüft, ob der Name bereits im Dictionary vorhanden ist
- (BOOL) nameExists: (NSString *) aName
{
   return [self.tagdict objectForKey:aName] != nil;
}

// Ruft den ersten übereinstimmenden Namen für das Tag ab
- (NSString *) nameForTag: (NSInteger) aTag
{
   NSNumber *tag = [NSNumber numberWithInt:aTag];
   NSArray *names = [self.tagdict allKeysForObject:tag];
   if (!names) return nil;
   if ([names count] == 0) return nil;
   return [names objectAtIndex:0];
}
```

```
// Gibt das Tag für einen registrierten Namen zurück. Wird keines gefunden,
// erfolgt die Rückgabe von 0.
- (NSInteger) tagForName: (NSString *)aName
{
   NSNumber *tag = [self.tagdict objectForKey:aName];
   if (!tag) return 0;
   return [tag intValue];
}

// Aufheben der Registrierung setzt das Tag auf 0 zurück
- (BOOL) unregisterName: (NSString *) aName forView: (UIView *) aView
{
   NSNumber *tag = [self.tagdict objectForKey:aName];

   // Tag nicht gefunden
   if (!tag) return NO;

   // Tag stimmt nicht mit dem registrierten Namen überein
   if (aView.tag != [tag intValue]) return NO;

   aView.tag = 0;
   [self.tagdict removeObjectForKey:aName];
   return YES;
}

// Registriert einen neuen Namen. Namen können nicht zweimal registriert
// werden. (Zuerst muss die erste Registrierung aufgehoben werden.) Ist
// eine Ansicht bereits Registriert, wird die erste Registrierung
// aufgehoben und dann die zweite durchgeführt.
- (NSInteger) registerName:(NSString *)aName forView: (UIView *) aView
{
   // Sie können einen vorhandenen Namen nicht erneut registrieren
   if ([[ViewIndexer sharedInstance] nameExists:aName]) return 0;

   // Prüft, ob die Ansicht bereits benannt ist. Wenn ja, wird ihre
   // Registrierung aufgehoben.
   NSString *currentName = [self nameForTag:aView.tag];
   if (currentName) [self unregisterName:currentName forView:aView];

   // Registriert das vorhandene Tag oder generiert ein neues,
   // wenn aView.tag = 0 ist.
   if (!aView.tag) aView.tag = [[ViewIndexer sharedInstance]
      pullNumber];
   [self.tagdict setObject:[NSNumber numberWithInt:aView.tag]
```

```
        forKey: aName];
    return aView.tag;
}
@end

@implementation UIView (NameExtensions)
- (NSInteger) registerName: (NSString *) aName
{
    return [[ViewIndexer sharedInstance] registerName: aName
        forView: self];
}

- (BOOL) unregisterName: (NSString *) aName
{
    return [[ViewIndexer sharedInstance] unregisterName: aName
        forView:self];
}

- (UIView *) viewNamed: (NSString *) aName
{
    NSInteger tag = [[ViewIndexer sharedInstance] tagForName: aName];
        return [self viewWithTag: tag];
}
@end
```

► *Rezept 6.4: Einen Ansichtsnamen-Manager erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.7 Geometrie von Ansichten

Wie zu erwarten ist, spielt die Geometrie bei der Arbeit mit Ansichten eine wichtige Rolle. Sie definiert, wo die einzelnen Ansichten auf dem Bildschirm erscheinen, wie groß sie sind und welche Ausrichtung sie haben. Die Klasse `UIView` enthält zwei Eigenschaften zur Definition dieser Aspekte.

Jede Ansicht definiert ihre Ausdehnung mit einem Rahmen, der ihre Abgrenzungen angibt: die Position, die Breite und die Höhe. Wenn Sie den Rahmen einer Ansicht ändern, wird sie aktualisiert, damit sie in den neuen Rahmen passt. Bei einer Vergrößerung der Breite wird die Ansicht gestreckt, bei einer neuen Position verschoben. Der Rahmen legt die Grenzen der Ansicht auf dem Bildschirm

fest. Eine Ansicht muss nicht unbedingt die gleiche Größe wie der Bildschirm aufweisen, sondern kann größer oder kleiner sein. Es ist auch möglich, dass eine Ansicht größer oder kleiner als ihre Elternansicht ist.

Ansichten haben auch die Eigenschaft `transform`, die ihre Ausrichtung und jegliche darauf angewandten geometrischen Transformationen festlegt. So können Sie eine Ansicht beispielsweise dehnen oder strecken, indem Sie eine Transformation auf sie anwenden, oder sie aus der Vertikale kippen. Rahmen und Transformationen definieren zusammen die Geometrie einer Ansicht.

6.7.1 Rahmen

Für die Rechtecke der Rahmen wird eine `CGRect`-Struktur verwendet, die – wie das Präfix `CG` schon andeutet – im Core Graphics-Framework definiert ist. Ein `CGRect` besteht aus einem Ursprung (einem `CGPoint`, `x` und `y`) und einer Größe (einer `CGSize`, Breite und Höhe). Wenn Sie Ansichten erstellen, weisen Sie sie gewöhnlich zu und initialisieren sie wie im folgenden Beispiel mit einem Rahmen:

```
CGRect rect = CGRectMake(0.0f, 0.0f, 320.0f, 416.0f);
myView = [[UIView alloc] initWithFrame: rect];
```

Die Funktion `CGRectMake` erstellt ein neues Rechteck mit vier Parametern, nämlich den `x`- und `y`-Koordinaten des Ursprungs sowie der Breite und Höhe des Rechtecks. Neben `CGRectMake` gibt es noch mehrere andere Hilfsfunktionen, die Sie bei der Arbeit mit Rechtecken und Rahmen unterstützen.

- `NSStringFromCGRect(aCGRect)` konvertiert eine `CGRect`-Struktur in einen formatierten String. Diese Funktion macht es einfach, den Rahmen einer Ansicht beim Debugging zu protokollieren.
- `CGRectFromString(aString)` stellt ein Rechteck aus seiner Stringdarstellung wieder her. Dies ist nützlich, wenn Sie den Rahmen einer Ansicht als String in den Benutzervoreinstellungen gespeichert haben und diesen String wieder in ein `CGRect` zurückverwandeln wollen.
- Mit `CGRectInset(aRect, xinset, yinset)` können Sie ein kleineres oder größeres Rechteck mit demselben Zentrum wie das ursprüngliche erstellen. Verwenden Sie für kleinere Rechtecke einen positiven und für größere einen negativen `inset`-Wert.
- Mithilfe von `CGRectIntersectsRect(rect1, rect2)` können Sie erfahren, ob sich zwei Rechtecke schneiden. Verwenden Sie diese Funktion, wenn Sie wissen möchten, ob sich zwei Rechtecke auf dem Bildschirm überlappen.
- `CGRectCreateDictionaryRepresentation(aRect)` transformiert eine Rechteckstruktur in eine standardmäßige `CFDictionaryRef`, die auch als `(NSDictionary *)`-Instanz angesprochen werden kann. Um das Dictionary wieder zurück in ein Rechteck zu verwandeln, verwenden Sie `CGRectMakeWithDictionaryRepresentation(aDict, aRect)`.
- `CGRectZero` ist eine Konstante für ein Rechteck, das sich bei `(0,0)` befindet und dessen Breite und Höhe null betragen. Sie können diese Konstante verwenden, wenn Sie einen Rahmen erstellen müssen, aber noch nicht wissen, welche Größe und welche Position er zum Erstellungszeitpunkt haben wird.

Die CGRect-Struktur besteht aus zwei Unterstrukturen: CGPoint definiert den Ursprung des Rechtecks, CGSize seine Ausdehnung. Punkte sind Positionen, die in x- und y-Koordinaten definiert sind, Größen setzen sich aus Breite und Höhe zusammen. Mit CGPointMake(x, y) erstellen Sie Punkte, mit CGSizeMake(width, height) legen Sie Größen fest. Diese beiden Strukturen wirken zwar gleichartig (beide weisen je zwei Fließkommawerte auf), werden vom iPhone SDK aber unterschieden. Punkte sind Positionen, Größen sind Ausdehnungen. Sie können myFrame.size nicht auf einen Punkt setzen.

Wie Rechtecke können Sie auch Punkte und Größen in Strings umwandeln und umgekehrt. Dazu dienen die Funktionen NSStringFromCGPoint(), NSStringFromCGSize(), CGSizeFromString() und CGPointFromString(). Es ist auch möglich, Punkte und Größen in Dictionarys zu verwandeln und umgekehrt.

6.7.2 Transformationen

Im Rahmen seiner Core Graphics-Implementierung unterstützt das iPhone standardmäßige affine Transformationen. Damit können Sie Punkte aus einem Koordinatensystem in ein anderes umwandeln. Diese Funktionen werden sehr häufig bei 2D- und 3D-Animationen eingesetzt. Die Version im iPhone SDK verwendet eine 3 × 3-Matrix, um UIView-Transformationen zu definieren, was sie zu einer reinen 2D-Lösung macht. Mit affinen Transformationen können Sie Ansichten in Echtzeit in der Größe ändern, verschieben und drehen. Dazu legen Sie wie im folgenden Beispiel die Eigenschaft transform der Ansicht fest:

```
float angle = theta * (PI / 100);
CGAffineTransform transform = CGAffineTransformMakeRotation(angle);
myView.transform = transform;
```

Die Transformation wird stets in Bezug auf den Mittelpunkt der Ansicht durchgeführt. Wenn Sie also wie hier eine Drehung durchführen, erfolgt sie um den Mittelpunkt der Ansicht. Brauchen Sie eine Drehung um einen anderen Punkt, müssen Sie die Ansicht erst verschieben, dann drehen und wieder zurückverschieben.

Um Änderungen rückgängig zu machen, setzen Sie die Eigenschaft transform auf die Identitätstransformation. Dadurch wird die Ansicht auf die letzte Einstellung für den Rahmen zurückgesetzt.

```
myView.transform = CGAffineTransformIdentity;
```

6.7.3 Koordinatensysteme

Ansichten sind Wanderer zwischen den Welten: Ihre Rahmen sind im Koordinatensystem ihrer Elternansicht definiert, ihre Grenzen und Unteransichten aber in ihrem eigenen. Das iPhone SDK enthält verschiedene Werkzeuge, um sich zwischen diesen beiden Koordinatensystemen zu bewegen, solange die Ansicht nur im selben UIWindow bleibt. Um einen Punkt aus einer anderen Ansicht in das eigene Koordinatensystem zu konvertieren, verwenden Sie wie im folgenden Beispiel convertPoint: fromView::

```
myPoint = [myView convertPoint:somePoint fromView:otherView];
```

Wenn der ursprüngliche Punkt die Position eines Objekts angegeben hat, so tut er das auch nach wie vor, wobei sich die Koordinaten jetzt jedoch auf den Ursprung von `myView` beziehen. Um andersherum vorzugehen, transformieren Sie mit `convertPoint: toView:` einen Punkt in das Koordinatensystem einer anderen Ansicht. Ähnlich funktionieren `convertRect: toView:` und `convertRect: fromView:`. Hierbei werden aber keine `CGPoint`-, sondern `CGRect`-Strukturen umgewandelt.

6.8 Rezept: Mit Ansichtsrahmen arbeiten

Wenn Sie den Rahmen einer Ansicht ändern, aktualisieren Sie seine Größe (also seine Breite und Höhe) und seine Position. Beispielsweise können Sie einen Rahmen wie im folgenden Beispiel verschieben. Der Code erstellt eine Unteransicht bei `(0, 0)` und verschiebt sie um 30 Pixel nach unten auf `(0, 30)`.

```
CGRect initialRect = CGRectMake(0.0f, 0.0f, 320.0f, 50.0f);
myView = [[UIView alloc] initWithFrame:initialRect];
[topView addSubview:myView];
myView.frame = CGRectMake(0.0f, 30.0f, 320.0f, 50.0f);
```

Diese Vorgehensweise ist jedoch eher unüblich. Das iPhone SDK erwartet nicht, dass Sie eine Ansicht verschieben, indem Sie ihren Rahmen ändern. Stattdessen haben Sie die Möglichkeit, die Position einer Ansicht zu aktualisieren. Am besten geht das, indem Sie den Mittelpunkt der Ansicht festlegen. Dazu können Sie direkt auf die integrierte Ansichtseigenschaft `center` zugreifen:

```
myView.center = CGPointMake(160.0f, 55.0f);
```

Wahrscheinlich erwarten Sie jetzt, dass das SDK Ihnen auch einen Weg bietet, um eine Ansicht durch Änderung ihres Ursprungs zu verschieben, doch eine solche Möglichkeit besteht nicht. Sie müssen den Rahmen der Ansicht abrufen, dessen Ursprung auf den gewünschten Punkt setzen und dann den Rahmen aktualisieren. Das folgende Fragment erstellt jedoch die neue Eigenschaft `origin`, mit der Sie den Ursprung der Ansicht abrufen und ändern können.

```
@interface UIView (ViewGeometry)
@property CGPoint origin;
@end


@implementation UIView (ViewGeometry)
- (CGPoint) origin
{
   return self.frame.origin;
}

- (void) setOrigin: (CGPoint) aPoint
{
```

```
    CGRect newframe = self.frame;
    newframe.origin = aPoint;
    self.frame = newframe;
}
@end
```

Wenn Sie eine Ansicht verschieben, müssen Sie sich keine Sorgen über Dinge wie rechteckige Bereiche machen, die sichtbar sind oder verdeckt werden. Das iPhone kümmert sich darum, die Oberfläche neu zu zeichnen. Dadurch können Sie Ihre Ansichten wie greifbare Objekte behandeln und können die Probleme der Darstellung getrost Cocoa Touch überlassen.

6.8.1 Die Größe anpassen

Die Größe einer Ansicht wird durch ihren Rahmen und ihre Grenzen bestimmt. Wie Sie bereits wissen, definiert der Rahmen die Position einer Ansicht im Koordinatensystem der Elternansicht. Liegt der Ursprung eines Rahmens also bei `(0, 30)`, so erscheint die Ansicht in der übergeordneten Ansicht bündig an der linken Seite und oben um 30 Pixel verschoben. Die Grenzen definieren die Ansicht in ihrem eigenen Koordinatensystem. Der Ursprung für die Grenzen einer Ansicht, also `myView.bounds`, ist also immer `(0, 0)`, und ihre Größe entspricht der normalen Ausdehnung, also der Eigenschaft `size` des Rahmens.

Die Größe einer Ansicht auf dem Bildschirm ändern Sie, indem Sie ihren Rahmen oder ihre Grenze ändern. Technisch gesprochen, aktualisieren Sie die Größenkomponente dieser Strukturen. Wie beim Verschieben von Ursprüngen ist es einfach, eine eigene Hilfsmethode zu schreiben, um die Aufgabe direkt zu erledigen.

```
- (void) setSize: (CGSize) aSize
{
    CGRect newframe = self.frame;
    newframe.size = aSize;
    self.frame = newframe;
}
```

Wenn sich die Größe einer Ansicht ändert, wird die Ansicht selbst live auf dem Bildschirm aktualisiert. Je nachdem, wie die Elemente in der Ansicht und die Klasse der Ansicht definiert sind, können Unteransichten dabei abgeschnitten oder so verkleinert werden, dass sie passen. Es gibt keine allgemeine Regel für alle möglichen Situationen. Im Größen-Informationsfenster des Interface Builder finden Sie eine Reihe von Optionen zur Größenänderung, mit denen Sie festlegen können, wie die Unteransichten auf Änderungen des Rahmens ihrer Elternansicht reagieren. In Kapitel 4, *Benutzeroberflächen entwerfen*, erfahren Sie mehr darüber, wie Sie Elemente im Interface Builder gestalten.

Manchmal müssen Sie eine Ansicht in der Größe ändern, bevor Sie sie einer neuen Elternansicht hinzufügen. Stellen Sie sich z. B. eine Bildansicht vor, die Sie in einer Warnung platzieren möchten. Um die Ansicht passend zu machen, ohne ihr Seitenverhältnis zu ändern, können Sie eine Methode wie die folgende verwenden, die sicherstellt, dass Höhe und Breite proportional geändert werden.

```
- (void) fitInSize: (CGSize) aSize
{
   CGFloat scale;
   CGRect newframe = self.frame;

   if (newframe.size.height > aSize.height)
   {
      scale = aSize.height / newframe.size.height;
      newframe.size.width *= scale;
      newframe.size.height *= scale;
   }

   if (newframe.size.width >= aSize.width)
   {
      scale = aSize.width / newframe.size.width;
      newframe.size.width *= scale;
      newframe.size.height *= scale;
   }

   self.frame = newframe;
}
```

6.8.2 Die Mittelpunkte von CGRect-Objekten

Wie Sie bereits gelernt haben, verwenden `UIViews` zur Definition ihrer Rahmen `CGRect`-Strukturen, die aus einem Ursprung und einer Größe bestehen. Diese Strukturen enthalten aber keine Verweise auf den Mittelpunkt. Wenn Sie eine Ansicht an eine andere Stelle verschieben, wird zur Aktualisierung der Position aber die Eigenschaft `center` von `UIView` verwendet. Leider verwendet Core Graphics keine Mittelpunkte zur Definition von Rechtecken. Die Fähigkeiten von Core Graphics, was Mittelpunkte betrifft, beschränken sich darauf, den Mittelpunkt eines Rechtecks entlang der X- oder Y-Achse abzurufen.

Diese Lücke können Sie füllen, indem Sie eine Funktion erstellen, die zwischen den mit dem Ursprung definierten `CGRect`-Strukturen und den über den Mittelpunkt bestimmten `UIView`-Objekten vermittelt. Die folgende Funktion ruft den Mittelpunkt eines Rechtecks ab, indem sie aus den Mittelpunkten der X- und Y-Achse einen Punkt konstruiert. Sie nimmt ein Argument entgegen, nämlich das Rechteck, und gibt dessen Mittelpunkt zurück.

```
CGPoint CGRectGetCenter(CGRect rect)
{
   CGPoint pt;
   pt.x = CGRectGetMidX(rect);
   pt.y = CGRectGetMidY(rect);
   return pt;
}
```

Eine Funktion, die ein Rechteck anhand seines Mittelpunkts verschiebt und damit den Umgang mit `UIView`-Objekten nachahmt, kann ebenfalls nützlich sein. Nehmen wir an, Sie möchten eine Ansicht zu einer neuen Position verschieben, wobei sie aber innerhalb des Rahmens ihrer Elternansicht bleiben muss. Um vor der Verschiebung einen Test durchzuführen, können Sie eine Funktion wie die folgende verwenden, um den Ansichtsrahmen zu einem neuen Mittelpunkt zu verschieben. Dann können Sie den verschobenen Rahmen mit dem Elternrahmen vergleichen (mit `CGRectContainsRect()`), um sicherzustellen, dass die Ansicht nicht aus ihrem Container herausragt.

```
CGRect CGRectMoveToCenter(CGRect rect, CGPoint center)
{
   CGRect newrect = CGRectZero;
   newrect.origin.x = center.x-CGRectGetMidX(rect);
   newrect.origin.y = center.y-CGRectGetMidY(rect);
   newrect.size = rect.size;
   return newrect;
}
```

6.8.3 Weitere Hilfsmethoden

Wie Sie gesehen haben, ist es sehr bequem, neben dem Mittelpunkt auch den Ursprung und die Größe einer Ansicht bereitzustellen, um Core Graphics-Aufrufe in ihrer natürlichen Art und Weise durchführen zu können. Diese Idee können Sie noch weiter ausbauen und auch andere Eigenschaften einer Ansicht offenlegen, z. B. ihre Breite (`width`) und ihre Höhe (`height`) sowie grundlegende geometrische Aspekte wie die Punkte `left`, `right`, `top` und `bottom`.

In gewisser Weise hintertreiben Sie damit das Designmodell von Apple, denn dadurch legen Sie Elemente offen, die sich normalerweise in Strukturen befinden, ohne dass Sie die Strukturen selbst widerspiegeln. Andererseits kann aber auch gesagt werden, dass es sich bei diesen Elementen um echte Ansichtseigenschaften handelt. Sie zeigen grundlegende Merkmale der Ansicht auf und verdienen es daher, als Eigenschaften offengelegt zu werden.

Rezept 6.5 enthält eine komplette Kategorie von Hilfsfunktionen für Ansichtsrahmen. Es ist Ihre Entscheidung, ob Sie diese Eigenschaften verwenden möchten oder nicht.

```
@interface UIView (ViewGeometry)
@property CGPoint origin;
@property CGSize size;
@property (readonly) CGPoint bottomLeft;
@property (readonly) CGPoint bottomRight;
@property (readonly) CGPoint topRight;
@property CGFloat height;
@property CGFloat width;
@property CGFloat top;
@property CGFloat left;
@property CGFloat bottom;
@property CGFloat right;
```

```
- (void) moveBy: (CGPoint) delta;
- (void) scaleBy: (CGFloat) scaleFactor;
- (void) fitInSize: (CGSize) aSize;
@end

@implementation UIView (ViewGeometry)
// Ruft den Ursprung ab und legt ihn fest
- (CGPoint) origin
{
   return self.frame.origin;
}

- (void) setOrigin: (CGPoint) aPoint
{
   CGRect newframe = self.frame;
   newframe.origin = aPoint;
   self.frame = newframe;
}

// Ruft die Größe ab und legt sie fest
- (CGSize) size
{
   return self.frame.size;
}

- (void) setSize: (CGSize) aSize
{
   CGRect newframe = self.frame;
   newframe.size = aSize;
   self.frame = newframe;
}

// Fragt andere Rahmenpositionen ab
- (CGPoint) bottomRight
{
   CGFloat x = self.frame.origin.x + self.frame.size.width;
   CGFloat y = self.frame.origin.y + self.frame.size.height;
   return CGPointMake(x, y);
}

- (CGPoint) bottomLeft
{
   CGFloat x = self.frame.origin.x;
   CGFloat y = self.frame.origin.y + self.frame.size.height;
   return CGPointMake(x, y);
```

```
}

- (CGPoint) topRight
{
   CGFloat x = self.frame.origin.x + self.frame.size.width;
   CGFloat y = self.frame.origin.y;
   return CGPointMake(x, y);
}

// Ruft Höhe, Breite, obere, untere, linke und rechte Position ab
// und legt sie fest
- (CGFloat) height
{
   return self.frame.size.height;
}

- (void) setHeight: (CGFloat) newheight
{
   CGRect newframe = self.frame;
   newframe.size.height = newheight;
   self.frame = newframe;
}

- (CGFloat) width
{
   return self.frame.size.width;
}

- (void) setWidth: (CGFloat) newwidth
{
   CGRect newframe = self.frame;
   newframe.size.width = newwidth;
   self.frame = newframe;
}

- (CGFloat) top
{
   return self.frame.origin.y;
}

- (void) setTop: (CGFloat) newtop
{
   CGRect newframe = self.frame;
   newframe.origin.y = newtop;
   self.frame = newframe;
```

```
}

- (CGFloat) left
{
   return self.frame.origin.x;
}

- (void) setLeft: (CGFloat) newleft
{
   CGRect newframe = self.frame;
   newframe.origin.x = newleft;
   self.frame = newframe;
}

- (CGFloat) bottom
{
   return self.frame.origin.y + self.frame.size.height;
}

- (void) setBottom: (CGFloat) newbottom
{
   CGRect newframe = self.frame;
   newframe.origin.y = newbottom - self.frame.size.height;
   self.frame = newframe;
}

- (CGFloat) right
{
   return self.frame.origin.x + self.frame.size.width;
}

- (void) setRight: (CGFloat) newright
{
   CGFloat delta = newright - (self.frame.origin.x + self.frame.size.
width);
   CGRect newframe = self.frame;
   newframe.origin.x += delta ;
   self.frame = newframe;
}
@end
```

▸ *Rezept 6.5: Kategorie von Hilfsfunktionen für Ansichtsrahmen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.9 Eine Ansicht mit Grenzen zufällig verschieben

Wenn Sie eine Ansicht an einen zufälligen Punkt verschieben, müssen Sie verschiedene Aspekte bedenken. Sehr oft muss eine Ansicht komplett in den Container ihrer Elternansicht passen, damit nichts abgeschnitten wird. Häufig fügen Sie dem Container auch eine Grenze hinzu, damit die Ansicht die Kante des Elternelements nicht berührt. Wenn Sie mit der vorgefertigten SDK-Version der Klasse `UIView` arbeiten, müssen Sie außerdem zufällige Mittelpunkte verwenden, nicht zufällige Positionen, wie Sie weiter vorn in diesem Kapitel schon gesehen haben. Einfach einen Punkt irgendwo in der Elternansicht auszuwählen, wird diesen Bedingungen meistens nicht gerecht.

Um dieses Problem zu lösen, wird in *Rezept 6.6* eine Reihe von Abständen zwischen den Kanten der Kind- und der Elternansicht definiert.. Mit der Struktur `UIEdgeInset` aus dem UIKit des iPhone SDK werden die Grenzen der Ansicht definiert. Diese Struktur enthält vier Einspringwerte, die angeben, wie weit das Rechteck oben, links, unten und rechts gegenüber dem Elterncontainer nach innen versetzt ist.

```
typedef struct {
   CGFloat top, left, bottom, right;
} UIEdgeInsets;
```

Die im Rezept vorgestellte Methode verwendet die Funktion `UIEdgeInsetsInsetRect()`, um ein `CGRect`-Rechteck zu schrumpfen und daraus einen inneren Container zu erstellen, der hier `innerRect` heißt.

Anschließend wird der Container noch weiter verkleinert. Die Grenzen des Rechtecks werden um die Hälfte der Höhe und Breite des Kindelements nach innen gezogen. Dadurch bleibt um jeden Punkt des Unterrechtecks genug Platz für die Kindansicht, ohne dass diese Ansicht mit dem inneren, mit einem Rand versehenen Rechteck überlappt. Jeder beliebige Punkt innerhalb des Unterrechtecks ist ein gültiger Mittelpunkt für die Kindansicht.

```
- (CGPoint) randomCenterInView: (UIView *) aView withInsets: (UIEdgeIn-
sets) insets
{
   // Um den Einsprungwert und dann um die Größe der Unteransicht nach
   // innen verschieben
   CGRect innerRect = UIEdgeInsetsInsetRect([aView bounds], insets);
   CGRect subRect = CGRectInset(innerRect,
      self.frame.size.width / 2.0f, self.frame.size.height / 2.0f);
   // Einen zufälligen Punkt zurückgeben
```

```
    float rx = (float)(random() % (int)floor(subRect.size.width));
    float ry = (float)(random() % (int)floor(subRect.size.height));
    return CGPointMake(rx + subRect.origin.x, ry + subRect.origin.y);
}

- (CGPoint) randomCenterInView: (UIView *) aView
      withInset: (float) inset
{
    UIEdgeInsets insets = UIEdgeInsetsMake(inset, inset, inset, inset);
    return [self randomCenterInView:aView withInsets:insets];
}

- (void) moveToRandomLocationInView: (UIView *) aView animated: (BOOL)
    animated
{
    if (!animated)
    {
        self.center = [self randomCenterInView:aView withInset:5];
        return;
    }

    // Die Verschiebung wird mit einer Dauer von 0.3 Sekunden
    // animiert
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:0.3f];

    self.center = [self randomCenterInView:aView withInset:5];

    [UIView commitAnimations];
}
```

▸ *Rezept 6.6: Eine begrenzte Ansicht zufällig verschieben*

Die im Beispielcode verwendete Methode `moveToRandomLocationInView:animated:` verschiebt die Ansicht bei Bedarf animiert. Auf das Animieren von Ansichten gehe ich weiter hinten in diesem Kapitel genauer ein.

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.10 Rezept: Ansichten transformieren

Mit affinen Transformationen können Sie die Geometrie eines Objekts ändern, indem Sie es von einem Koordinatensystem auf ein anderes abbilden. Das iPhone SDK unterstützt standardmäßige affine 2D-Transformationen vollständig. Mit ihnen können Sie Ihre Ansichten skalieren, verschieben, drehen und scheren, wie immer Sie es wünschen oder wie Ihre Anwendung es erfordert.

Transformationen sind in Core Graphics definiert und bestehen aus Aufrufen wie `CGAffineTransformMakeRotation` und `CGAffineTransformScale`. Sie erstellen und ändern die 3×3-Transformationsmatrizen. Verwenden Sie den Aufruf `setTransform:` von `UIView`, sobald die Transformation erstellt ist, um zweidimensionale affine Transformationen auf `UIView`-Objekte anzuwenden.

Rezept 6.7 zeigt, wie Sie eine affine Transformation einer `UIView` erstellen und anwenden. Ich habe das Beispiel einfach gehalten. Es gibt einen `NSTimer`, der 30-mal pro Sekunde tickt. Bei jedem Tick wird eine Ansicht um ein Hundertstel von π gedreht und mit einer Kosinuskurve skaliert. Aus zwei Gründen verwende ich den Absolutbetrag des Kosinus: Die Ansicht ist so jederzeit sichtbar und bietet einen hübschen Federeffekt, wenn die Skalierung die Richtung ändert. Das erzeugt eine Drehung und eine ungedämpfte Federanimation.

Dies ist eines der Beispiele, die Sie am besten erstellen und anschauen, während Sie den Code lesen. Sie können dann besser sehen, wie die Methode `handleTimer:` mit den optischen Effekten in Beziehung steht, die Sie sehen.

Hinweis

In diesem Rezept wird die C-Standardbibliothek `math` verwendet, die sowohl die Kosinusfunktion als auch die Konstante `M_PI` enthält.

```
#import <math.h>
#define COOKBOOK_PURPLE_COLOR [UIColor colorWithRed:0.20392f
green:0.19607f \
blue:0.61176f alpha:1.0f]
#define BARBUTTON(TITLE, SELECTOR) [[[UIBarButtonItem alloc] \
initWithTitle:TITLE style:UIBarButtonItemStylePlain target:self \
action:SELECTOR] autorelease]

@interface TestBedViewController : UIViewController
{
   NSTimer *timer;
   int      theta;
}
@end

@implementation TestBedViewController
- (void) move: (NSTimer *) aTimer
```

```
{
    // Dreht die Ansicht bei jeder Iteration um 1/100 PI
    CGFloat angle = theta * (M_PI / 100.0f);
    CGAffineTransform transform = CGAffineTransformMakeRotation(angle);

    // Theta bewegt sich im Bereich von 0 und 2*PI
    theta = (theta + 1) % 200;

    // Skaliert die Ansicht spaßeshalber mit dem Absolutwert des Kosinus
    float degree = cos(angle);
    if (degree < 0.0) degree *= -1.0f;
    degree += 0.5f;

    // Führt zur Rotationstransformation eine Skalierung hinzu
    CGAffineTransform scaled = CGAffineTransformScale(transform,
        degree, degree);

    // Wendet die affine Transformation an
    [[self.view viewWithTag:999] setTransform:scaled];
}

- (void) start: (id) sender
{
    // Der Timer wird automatisch von der Ausführungsschleife beibehalten.
    // Sie können ihn starten und anhalten, ohne der Besitzer sein zu
    // müssen und ohne den Beibehaltungswert beachten zu müssen.
    timer = [NSTimer scheduledTimerWithTimeInterval:0.03f target:self
        selector:@selector(move:) userInfo:nil repeats:YES];
    [self move:nil];
    self.navigationItem.rightBarButtonItem = BARBUTTON(@"Stop",
        @selector(stop:));
}

- (void) stop: (id) sender
{
    [timer invalidate];
    timer = nil;
    self.navigationItem.rightBarButtonItem = BARBUTTON(@"Start",
        @selector(start:));
}

- (void) viewDidLoad
{
```

```
    self.navigationItem.rightBarButtonItem = BARBUTTON(@"Start",
       @selector(start:));
    self.navigationController.navigationBar.tintColor =

       COOKBOOK_PURPLE_COLOR;
    UIImageView *imgView = [[UIImageView alloc] initWithImage:[UIImage
       imageNamed:@"BflyCircle.png"]];
    imgView.tag = 999;
    imgView.center = CGPointMake(160.0f, 143.0f);
    [self.view addSubview:imgView];
    [imgView release];

    timer = nil;
    theta = 0;
}
@end
```

▸ *Rezept 6.7: Beispiel für die affine Transformation einer* `UIView`*-Ansicht*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.10.1 Ansichten im Querformat zentrieren

Verwenden Sie denselben Ansatz mit affinen Transformationen, um Ansichten im Querformat zu zentrieren. Das folgende Listing erstellt eine 480 × 320 Pixel große Ansicht, zentriert sie bei [160, 240] (in Hochformatkoordinaten) und dreht sie dann in ihre Position. Die Hälfte von π entspricht 90° und sorgt für eine Drehung ins Querformat nach rechts. Durch das Zentrieren bleibt die gesamte Ansicht auf dem Bildschirm. Alle Unteransichten einschließlich Textfelder, Beschriftungen, Schalter usw. drehen sich mit der Elternansicht in Position.

```
#import <math.h>
- (void)loadView
{
    contentView = [[UIView alloc] initWithFrame:
       CGRectMake(0.0f, 0.0f, 480.0f, 320.0f)];
    [contentView setCenter:CGPointMake(160.0f, 240.0f)];
    [contentView setBackgroundColor:[UIColor blackColor]];
    [contentView setTransform:CGAffineTransformMakeRotation(M_PI/2.0f)];
    self.view = contentView;
    [contentView release];
}
```

Meistens ist es weit einfacher, für Ereignisse der Neuausrichtung `UIViewController` zu verwenden, als Ansichten manuell zu drehen und darzustellen. Außerdem wird beim manuellen Drehen weder die Ausrichtung der Statusleiste noch die der Tastatur geändert. In Kapitel 4 werden Ansichtscontroller und Neuausrichtung ausführlich behandelt.

6.11 Anzeige- und Interaktionsaspekte

Neben dem physischen Bildschirmlayout stellt die Klasse `UIView` auch Eigenschaften bereit, die festlegen, wie Ansichten auf dem Bildschirm erscheinen und ob die Benutzer mit ihnen interagieren können. Für jede Ansicht gibt es einen Durchsichtigkeitsfaktor (Alpha), der von opak bis transparent reicht. Diesen Faktor können Sie mit `[myView setAlpha:value]` einstellen, wobei die Alphawerte zwischen 0,0 (völlig transparent) und 1,0 (völlig undurchsichtig) liefen. Damit haben Sie eine hervorragende Möglichkeit, um Ansichten zu verbergen und sie auf dem Bildschirm ein- und auszublenden.

Sie können dem Hintergrund jeder Ansicht eine Farbe zuweisen. Beispielsweise färben Sie eine Ansicht mit `[myView setBackgroundColor: [UIColor redColor]]` rot. Die Eigenschaft wirkt sich auf die verschiedenen Ansichtsklassen jeweils unterschiedlich aus, was davon abhängt, ob die Klassen über Unteransichten verfügen, die den Hintergrund verstellen. Einen transparenten Hintergrund erreichen Sie, indem Sie die Hintergrundfarbe der Ansicht auf farblos einstellen (d. h. `[UIColor clearColor]`).

Unabhängig davon, ob Sie den Hintergrund sehen können, hat jede Ansicht eine Eigenschaft für die Hintergrundfarbe. Eine gute Möglichkeit, um die tatsächliche Ausdehnung einer Ansicht zu erkennen, besteht darin, leuchtende, kontrastierende Hintergrundfarben zu verwenden. Wenn Sie erst mit der Entwicklung für das iPhone beginnen, erlangen Sie durch das Einfärben der Ansichten einen Begriff davon, was auf dem Bildschirm sichtbar ist und was nicht und wo sich die einzelnen Komponenten befinden.

Die Eigenschaft `userInteractionEnabled` legt fest, ob Benutzer eine gegebene Ansicht antippen und mit ihr interagieren können. Bei den meisten Ansichten weist diese Eigenschaft den Standardwert `YES` auf, bei `UIImageView` jedoch `NO`, was neue Entwickler zur Verzweiflung treiben kann. Häufig verwenden sie eine `UIImageView` als Hintergrund und verstehen nicht, warum ihre Schalter, Textfelder und Schaltflächen nicht funktionieren. Aktivieren Sie diese Eigenschaft für alle Ansichten, die die Benutzer antippen müssen – und deren Unteransichten wie Schaltflächen, Schalter, Auswahlfelder und andere Steuerelemente die Benutzer antippen müssen. Wenn Elemente nicht auf Berührungen reagieren, sollten Sie den Wert der Eigenschaft `userInteractionEnabled` für dieses Element und seine Elternelemente überprüfen.

Deaktivieren Sie diese Eigenschaft für alle Ansichten innerhalb des Interaktionsbereichs, die lediglich der Anzeige dienen. Um auf einer transparenten, bildschirmfüllenden Ansicht eine nicht interaktive Uhr anzuzeigen, schalten Sie die Interaktion ab. Dadurch werden Berührungen durch diese Ansicht hindurch an den darunterliegenden Interaktionsbereich der Anwendung übergeben.

6.12 UIView-Animationen

`UIView`-Animationen bieten einen der sonderbaren, aber großartigen Nebeneffekte bei der Arbeit mit dem iPhone als Entwicklungsplattform. Sie ermöglichen es Ihnen, die Änderungen bei der Aktualisierung von Ansichten zu verlangsamen, was zu weichen, animierten Übergängen führt, die für den Benutzer angenehmer sind. Und das Beste ist, dass all dieses geschieht, ohne dass Sie viel dazu tun müssen.

`UIView`-Animationen sind ideal dazu geeignet, eine optische Brücke zwischen dem aktuellen und dem geänderten Zustand einer Ansicht zu bauen. Damit heben Sie optische Änderungen hervor und verbinden diese Änderungen. Die folgenden Änderungen sind animierbar:

- Änderungen der Position – Verschieben einer Ansicht auf dem Bildschirm
- Änderungen der Größe – Aktualisierungen des Ansichtsrahmens
- Änderungen der Dehnung – Aktualisierungen der Dehnbereiche im Inhalt einer Ansicht
- Änderungen der Transparenz – Änderung des Alphawerts der Ansicht
- Änderungen des Status – Verborgen oder sichtbar
- Änderungen der Ansichtsreihenfolge – Änderungen daran, welche Ansicht vorn liegt
- Änderungen der Drehung sowie alle anderen affinen Transformationen, die Sie auf eine Ansicht anwenden

6.12.1 UIView-Animationsblöcke erstellen

`UIView`-Animationen funktionieren blockweise, also als eine vollständige Transaktion, die auf einmal ausgeführt wird. Starten Sie den Block mit `beginAnimations:context:`, und beenden Sie ihn mit `commitAnimations`. Diese Klassenmethoden senden Sie an `UIView` und nicht an einzelne Ansichten. Im Block zwischen diesen beiden Aufrufen definieren Sie, wie die Animation abläuft, und führen die tatsächlichen Aktualisierungen der Ansicht durch. Sie verwenden dabei die folgenden Animationssteuerungen:

- `beginAnimations:context:` Kennzeichnet den Start des Animationsblocks.
- `setAnimationCurve` Definiert die Art und Weise, wie die Animation beschleunigt und abgebremst wird. Verwenden Sie `ease-in/ease-out` (`UIViewAnimationCurveEaseInOut`), sofern Sie keinen zwingenden Grund haben, eine andere Kurve auszuwählen. Die anderen Kurventypen sind `ease in` (in die Animation beschleunigen), `linear` (keine Animationsbeschleunigung) und `ease out` (aus der Animation beschleunigen). `Ease-in/ease-out` bietet den natürlichsten Animationsstil.
- `setAnimationDuration` Gibt die Länge der Animation in Sekunden an. Das ist ein wirklich nützlicher Parameter. Sie können die Animation so lange dehnen, wie es erforderlich ist. Stellen Sie jedoch nicht die Geduld des Benutzers auf die Probe, sondern halten Sie die Dauer Ihrer Animationen unter einer oder zwei Sekunden. Zum Vergleich: Wenn die Tastatur auf den Bildschirm geschoben oder von ihm entfernt wird, dauert die Animation 0,3 Sekunden.
- `commitAnimations` Kennzeichnet das Ende des Animationsblocks.

Fügen Sie die eigentlichen Befehle für die Ansichtsänderungen nach dem Festlegen der Einzelheiten und vor dem Ende der Animation ein:

```
CGContextRef context = UIGraphicsGetCurrentContext();
[UIView beginAnimations:nil context:context];
[UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
[UIView setAnimationDuration:1.0];

// Hier stehen die Ansichtsänderungen
[contentView setAlpha:0.0f];

[UIView commitAnimations];
```

Dieses Listing zeigt `UIView`-Animationen im Einsatz. Hier werden eine Animationskurve und die Animationsdauer gesetzt (in diesem Fall eine Sekunde). Die eigentliche animierte Änderung ist eine Aktualisierung der Transparenz. Der Alphawert der Inhaltsansicht geht gegen null und macht sie unsichtbar. Die Ansicht verschwindet nicht einfach, sondern der Animationsblock verlangsamt die Änderung und blendet sie aus. Beachten Sie den Aufruf von `UIGraphicsGetCurrentContext()`, der den grafischen Kontext an der Spitze des aktuellen Ansichtsstacks zurückgibt. Ein Grafikkontext bietet eine virtuelle Verbindung zwischen den abstrakten Aufrufen zum Zeichnen und den Pixeln auf dem Bildschirm (bzw. innerhalb eines Bildes). Für dieses Argument können Sie im jüngsten SDK auch `nil` übergeben, ohne dass dies Schwierigkeiten nach sich zieht.

6.12.2 Animations-Callbacks

Ansichtsanimationen können einen optionalen Delegate über Statusänderungen informieren, vor allem darüber, dass eine Animation begonnen hat oder beendet wurde. Dies ist hilfreich, wenn Sie das Ende einer Animation erfassen müssen, um die nächste Animation in einer Folge zu starten. Um den Delegate festzulegen, verwenden Sie wie im folgenden Beispiel `setAnimationDelegate:`:

```
[UIView setAnimationDelegate:self];
```

Um einen Callback für das Ende der Animation einzurichten, geben Sie den an den Delegate gesendeten Selector an:

```
[UIView setAnimationDidStopSelector:
   @selector(animationDidStop:finished:context:)];
```

Weiter hinten in diesem Kapitel, in *Rezept 6.9*, in dem der Austausch einer Ansicht animiert wird, sehen Sie Animations-Callbacks im Einsatz.

6.13 Rezept: Ansichten ein- und ausblenden

Es kann vorkommen, dass Sie Informationen auf dem Bildschirm anzeigen möchten, die eine Ansicht überlagern, aber selbst nichts tun. Sie können z. B. eine Bestenliste, irgendwelche Anweisungen oder eine kontextsensitive Hilfe ausgeben. *Rezept 6.8* zeigt, wie Sie einen `UIView`-Animationsblock

verwenden, um eine Ansicht ein- und auszublenden. Dieses Rezept folgt dem ganz grundlegenden Animationsansatz. Es erstellt einen umgebenden Animationsblock und setzt dann die Eigenschaft `alpha` in einer einzigen Codezeile.

Eins jedoch tut dieses Rezept nicht: Es wartet nicht auf den Abschluss der Animation. Die Änderungen im Element der Leistenschaltfläche werden aufgerufen, sobald die Animationen bestätigt sind, was fast eine Sekunde vor deren Ende ist. Wenn Sie sehr schnell auf die Schaltfläche **Fade in/Fade Out** tippen (Sie können die Animationsdauer verlängern, um dies besser zu sehen), werden Sie feststellen, dass die neue Animation beginnt und die alte ersetzt, was zu einer optischen Unterbrechung führt.

Um dies zu korrigieren, können Sie einen Aufruf von `UIView` mit `setAnimationBeginsFromCurrentState:` hinzufügen und das Argument auf `YES` setzen. Dadurch weisen Sie das iPhone an, den aktuellen Status der laufenden Animation zu verwenden, um die nächste zu starten, wodurch der Sprung vermieden wird.

```
@implementation TestBedViewController

- (void) fadeOut: (id) sender
{
   CGContextRef context = UIGraphicsGetCurrentContext();
   [UIView beginAnimations:nil context:context];
   [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
   [UIView setAnimationDuration:1.0];
   [[self.view viewWithTag:999] setAlpha:0.0f];
   [UIView commitAnimations];
   self.navigationItem.rightBarButtonItem =

      BARBUTTON(@"Fade In", @selector(fadeIn:));
}

- (void) fadeIn: (id) sender
{
   CGContextRef context = UIGraphicsGetCurrentContext();
   [UIView beginAnimations:nil context:context];
   [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
   [UIView setAnimationDuration:1.0];
   [[self.view viewWithTag:999] setAlpha:1.0f];
   [UIView commitAnimations];
   self.navigationItem.rightBarButtonItem =

      BARBUTTON(@"Fade Out", @selector(fadeOut));
}

- (void) viewDidLoad
{

   self.navigationController.navigationBar.tintColor =
```

```
        COOKBOOK_PURPLE_COLOR;
    self.navigationItem.rightBarButtonItem =

        BARBUTTON(@"Fade Out", @selector(fadeOut));
}
@end
```

▸ *Rezept 6.8: Änderungen der Transparenz animieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.14 Rezept: Ansichten austauschen

Der `UIView`-Animationsblock beschränkt Sie nicht auf eine Änderung. *Rezept 6.9* verbindet Größenänderungen mit Änderungen der Transparenz, um eine überzeugendere Animation zu erstellen. Sie erreichen das, indem Sie mehrere Anweisungen gleichzeitig zum Animationsblock hinzufügen. Der Rezeptcode führt fünf Aktionen gleichzeitig durch. Er zoomt und blendet eine Ansicht ein, während er eine andere auszoomt und ausblendet. Anschließend tauscht er die beiden in der Arrayliste der Unteransichten aus.

Beachten Sie, wie die Methode `viewDidLoad` das Objekt `back` zur Animation vorbereitet, indem sie es verkleinert und transparent macht. Bei der ersten Ausführung der Methode `swap:` ist diese Ansicht bereit zum Einblenden und zur Änderung der Größe.

Anders als in *Rezept 6.8* wird hier mithilfe eines Delegates und eines Callbacks auf die Beendigung der Animation gewartet. Der Code blendet die Leistenschaltfläche aus, sobald der Benutzer darauf getippt hat, und zeigt sie erst wieder an, wenn die Animation abgeschlossen ist.

```
- (void) animationFinished: (id) sender
{
    self.navigationItem.rightBarButtonItem =

        BARBUTTON(@"Swap", @selector(swap:));
}

- (void) swap: (id) sender
{
    self.navigationItem.rightBarButtonItem = nil;

    UIView *frontObject = [[self.view subviews] objectAtIndex:2];
    UIView *backObject = [[self.view subviews] objectAtIndex:1];
```

```
    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];

    frontObject.alpha = 0.0f;
    backObject.alpha = 1.0f;
    frontObject.transform = CGAffineTransformMakeScale(0.25f, 0.25f);
    backObject.transform = CGAffineTransformIdentity;
    [self.view exchangeSubviewAtIndex:1 withSubviewAtIndex:2];

    [UIView setAnimationDelegate:self];
    [UIView setAnimationDidStopSelector:@selector(animationFinished:)];
    [UIView commitAnimations];
}

- (void) viewDidLoad
{
    UIView *backObject = [self.view viewWithTag:998];
    backObject.transform = CGAffineTransformMakeScale(0.25f, 0.25f);
    backObject.alpha = 0.0f;

    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
    self.navigationItem.rightBarButtonItem =
        BARBUTTON(@"Swap", @selector(swap:));
}
```

▸ *Rezept 6.9: Mehrere Ansichtenänderungen in einem Animationsblock kombinieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.15 Rezept: Ansichten wenden

Mit Übergängen (*Transitions*) können Sie Ihre `UIView`-Animationsblöcke erweitern, um noch mehr optisches Flair zu bieten. Die beiden Übergänge `UIViewAnimationTransitionFlipFromLeft` und `UIViewAnimationTransitionFlipFromRight` tun genau das, was ihre Namen andeuten. Damit können Sie Ansichten wie die Anwendungen *Wetter* und *Aktien* nach links oder nach rechts wenden. Wie das geht, zeigt *Rezept 6.10*.

Als Erstes fügen Sie den Übergang als Blockparameter hinzu. Weisen Sie den Übergang mit `setAnimationTransition:` zu dem umgebenden `UIView`-Animationsblock zu. Anschließend ändern Sie die Ansichtsreihenfolge innerhalb des Blocks. Dies geschieht am besten mit `exchangeSubviewAtIndex: withSubviewAtIndex:`. In *Rezept 6.10* wird mit diesen Techniken eine einfache Wendeansicht erstellt.

Der Code zeigt jedoch nicht, wie Sie die Ansichten einrichten. Der für das Umdrehen verwendete Übergang von `UIKit` erwartet mehr oder weniger einen schwarzen Hintergrund. Außerdem muss der Übergang auf einer Elternansicht stattfinden, wobei die beiden Unteransichten dieser Elternansicht ausgetauscht werden. Die in diesem Rezept verwendete Ansichtsstruktur sehen Sie in *Abbildung 6.3*.

Dort sehen Sie den schwarzen und den weißen Hintergrund, die beide denselben Rahmen verwenden. Der weiße Hintergrund enthält die beiden Kindansichten, die wiederum identische Rahmen aufweisen. Beim Wenden wird der weiße Hintergrund scheinbar umgedreht, wie *Abbildung 6.4* zeigt, um die zweite Kindansicht aufzudecken.

Verwechseln Sie `UIView`-Animationsblöcke nicht mit der Core Animation-Klasse `CATransition`. Leider können Sie Ihrer `UIView`-Animation keine `CATransition` zuweisen. Um eine `CATransition` zu verwenden, müssen Sie sie auf eine Ebene einer `UIView` anwenden, was als Nächstes gezeigt wird.

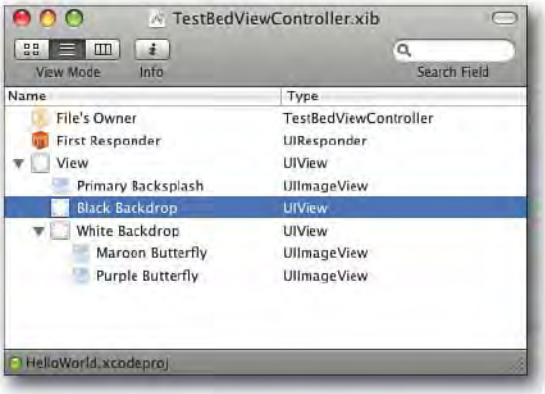
```
- (void) animationFinished: (id) sender
{
   self.navigationItem.rightBarButtonItem =
      BARBUTTON(@"Flip", @selector(flip:));
}

- (void) flip: (id) sender
{
    // blendet die "flip" Schaltfläche aus
    self.navigationItem.rightBarButtonItem = nil;

    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];

    UIView *whiteBackdrop = [self.view viewWithTag:100];

    // Wählt Links- oder Rechtsanimation
    if ([(UISegmentedControl *)self.navigationItem.titleView
    selectedSegmentIndex])
    {
       [UIView setAnimationTransition: UIViewAnimationTransitionFlip
         FromLeft
          forView:whiteBackdrop cache:YES];
```

```
    }
    else
    {
        [UIView setAnimationTransition: UIViewAnimationTransitionFlip
         FromRight
          forView:whiteBackdrop cache:YES];
    }

    NSInteger purple = [[whiteBackdrop subviews] indexOfObject:
      [whiteBackdrop viewWithTag:999]];
    NSInteger maroon = [[whiteBackdrop subviews] indexOfObject:
      [whiteBackdrop viewWithTag:998]];
    [whiteBackdrop exchangeSubviewAtIndex:purple
       withSubviewAtIndex:maroon];

    [UIView setAnimationDelegate:self];
    [UIView setAnimationDidStopSelector:@selector(animationFinished:)];
    [UIView commitAnimations];
}

- (void) viewDidLoad
{
    self.navigationController.navigationBar.tintColor =
       COOKBOOK_PURPLE_COLOR;
    self.navigationItem.rightBarButtonItem =
       BARBUTTON(@"Flip", @selector(flip:));

    // Erstellt eine Segmentleiste um die Animationsrichtung zu bestimmen
    UISegmentedControl *seg = [[[UISegmentedControl alloc]
       initWithItems:[@"Left Right" componentsSeparatedByString:@" "]]
          autorelease];
    seg.selectedSegmentIndex = 0;
    seg.segmentedControlStyle = UISegmentedControlStyleBar;
    self.navigationItem.titleView = seg;

}
```

▸ *Rezept 6.10: Übergänge in* `UIView`*-Animationsblöcken*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.



▸ *Abbildung 6.3: Für einen Wendeübergang brauchen Sie zwei Hintergründe.*

▸ *Abbildung 6.4: Für Animationen mit Wendeübergang müssen Sie einen schwarzen Hintergrund erstellen.*

6.16 Rezept: Core Animation-Übergänge verwenden

Neben `UIView`-Animationen unterstützt das iPhone im Rahmen des QuartzCore-Frameworks auch *Core Animation*. Die Core Animation-API bietet hochgradig flexible Animationslösungen für iPhone-Anwendungen. Vor allem enthält sie Übergänge für die gleichen Wechsel von einer Ansicht zur anderen, die Sie im vorherigen Rezept kennengelernt haben.

Core Animation-Übergänge erweitern Ihre Möglichkeiten für `UIView`-Animationen mit nur geringen Unterschieden bei der Implementierung. `CATransitions` wirken sich auf Ebenen statt auf Ansichten aus. Ebenen sind die Rendering-Oberflächen von Core Animation, die mit den einzelnen `UIViews` verknüpft sind. Wenn Sie mit Core Animation arbeiten, wenden Sie `CATransitions` auf die Standardebene einer Ansicht (`[myView layer]`) statt auf die Ansicht selbst an.

Bei diesen Übergängen setzen Sie Ihre Parameter nicht wie bei `UIView`-Animationen in der `UIView`. Stattdessen erstellen Sie ein Core Animation-Objekt, richten dessen Parameter ein und fügen dann den parametrisierten Übergang zur Ebene hinzu.

```
CATransition *animation = [CATransition animation];
animation.delegate = self;
animation.duration = 1.0f;
animation.timingFunction = UIViewAnimationCurveEaseInOut;
animation.type = kCATransitionMoveIn;
animation.subtype = kCATransitionFromTop;

// Hier werden Ansichten ausgetauscht oder entfernt

[myView.layer addAnimation:animation forKey:@"move in"];
```

Animationen haben sowohl einen *Typ* als auch einen *Untertyp*. Ersterer gibt die Art des verwendeten Übergangs an, Letzterer dessen Richtung. Zusammen legen der Typ und der Untertyp fest, wie sich die Ansichten verhalten sollen, wenn die Animation auf sie angewendet wird.

Core Animation-Übergänge unterscheiden sich von den in den vorstehenden Rezepten behandelten `UIViewAnimationTransitions`. Cocoa Touch bietet vier Arten solcher Animationen, die in *Rezept 6.11* vorgestellt werden. Zu den verfügbaren Typen gehören Überblendungen, das Wegschieben (eine Ansicht schiebt die andere vom Bildschirm), Aufdeckungen (eine Ansicht gleitet von einer anderen herunter) und Bedeckungen (eine Ansicht gleitet über eine andere). Bei den letzten drei Typen können Sie die Bewegungsrichtung für den Übergang durch Untertypen angeben. Es ist offensichtlich, dass Überblendungen keine Richtung haben und daher auch keine Untertypen aufweisen.

Da Core Animation ein Bestandteil des Quartz Core-Frameworks ist, müssen Sie dieses Framework zu Ihrem Projekt hinzufügen und `<QuartzCore/QuartzCore.h>` in den Code importieren, wenn Sie diese Merkmale verwenden möchten.

Hinweis

Core Animation von Apple enthält 2D- und 3D-Routinen, die auf Objective-C-Klassen beruhen. Diese Klassen bieten Grafikrendering und Animationen für iPhone- und Macintosh-Anwendungen. Core Animation umgeht viele Low-Level-Entwicklungsaufgaben, die z. B. mit OpenGL verbunden sind, sodass die Arbeit damit so einfach wie mit hierarchischen Ansichten ist.

```
- (void) animate: (id) sender
{
    // Die Animation einrichten
    CATransition *animation = [CATransition animation];
    animation.delegate = self;
    animation.duration = 1.0f;
    animation.timingFunction = UIViewAnimationCurveEaseInOut;
```

```
    switch ([(UISegmentedControl *)self.navigationItem.titleView
                selectedSegmentIndex])
    {
       case 0:
          animation.type = kCATransitionFade;
          break;
       case 1:
          animation.type = kCATransitionMoveIn;
          break;
       case 2:
          animation.type = kCATransitionPush;
          break;
       case 3:
          animation.type = kCATransitionReveal;
       default:
          break;
    }

    if (isLeft)
       animation.subtype = kCATransitionFromRight;
    else
       animation.subtype = kCATransitionFromLeft;

    // Die Animation durchführen
    UIView *whitebg = [self.view viewWithTag:10];
    NSInteger purple = [[whitebg subviews] indexOfObject:[whitebg
       viewWithTag:99]];
    NSInteger white = [[whitebg subviews] indexOfObject:[whitebg
       viewWithTag:100]];
    [whitebg exchangeSubviewAtIndex:purple withSubviewAtIndex:white];
    [[whitebg layer] addAnimation:animation forKey:@"animation"];

    // Benutzerinteraktionen erlauben oder verbieten (legt fest, ob
    // Berührungen "durch" die Deckschicht hindurchgehen können, um den
    // Schalter zu betätigen
    if (purple < white)
       [self.view viewWithTag:99].userInteractionEnabled = YES;
    else
       [self.view viewWithTag:99].userInteractionEnabled = NO;

    isLeft = !isLeft;

}
```

▸ *Rezept 6.11: Übergänge mit Core Animation animieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.17 Allgemeine Core Animation-Aufrufe

Das iPhone bietet eine *teilweise* Unterstützung für Core Animation-Aufrufe. Mit »teilweise« meine ich, dass viele Standardklassen fehlen, wobei sie jedoch mit fortschreitender Entwicklung des iPhone SDK nach und nach auftauchen. `CIFilter` von Core Image ist eine solche Klasse. Sie ist nicht in Cocoa Touch enthalten, obwohl die Klassen `CALayer` und `CATransition` Filter kennen. Wenn Sie bereit sind, mit diesen Beschränkungen zu arbeiten, können Sie in Ihren Programmen standardmäßige Core Animation-Aufrufe frei verwenden.

Rezept 6.12 zeigt nativen Core Animation-Code für das iPhone, der auf einem Beispiel von Lucas Newman beruht (*http://lucasnewman.com*). Bei der Ausführung verkleinert diese Methode den Inhalt einer `UIImageView` und blendet ihn aus.

Dieser Code ist gegenüber dem Beispiel für MacOS X, auf dem er beruht, praktisch unverändert. Komplexerer Core Animation-Code kann Herausforderungen für die Portierung darstellen, aber innerhalb des iPhones gewinnen Sie dadurch den Funktionsumfang, den Sie für einfache Spiegelungen, Schattierungen und Umwandlungen benötigen.

```
- (void) action: (id) sender
{
    self.navigationItem.rightBarButtonItem = nil;
    UIView *theView = [self.view viewWithTag:101];
    [CATransaction begin];
    [CATransaction setValue: [NSNumber numberWithFloat: 4.0f]
        forKey:kCATransactionAnimationDuration];

    // Herunterskalieren
    CABasicAnimation *shrinkAnimation = [CABasicAnimation
        animationWithKeyPath:@"transform.scale"];
    shrinkAnimation.delegate = self;
    shrinkAnimation.timingFunction = [CAMediaTimingFunction
        functionWithName:kCAMediaTimingFunctionEaseIn];
    shrinkAnimation.toValue = [NSNumber numberWithFloat:0.0];
    [[theView layer] addAnimation:shrinkAnimation
        forKey:@"shrinkAnimation"];

    // Ausblenden
    CABasicAnimation *fadeAnimation = [CABasicAnimation
        animationWithKeyPath:@"opacity"];
```

```
    fadeAnimation.toValue = [NSNumber numberWithFloat:0.0];
    fadeAnimation.timingFunction = [CAMediaTimingFunction
        functionWithName:kCAMediaTimingFunctionEaseIn];
    [[theView layer] addAnimation:fadeAnimation
        forKey:@"fadeAnimation"];

    // Lässt die Ansicht mit einer Keyframe-Animation mehrmals springen
    CAKeyframeAnimation *positionAnimation = [CAKeyframeAnimation
        animationWithKeyPath:@"position"];
    CGMutablePathRef positionPath =
        CGAutorelease(CGPathCreateMutable());

    CGPathMoveToPoint(positionPath, NULL,
        [theView layer].position.x, [theView layer].position.y);
    CGPathAddQuadCurveToPoint(positionPath, NULL,
        [theView layer].position.x, - [theView layer].position.y,
        [theView layer].position.x, [theView layer].position.y);
    CGPathAddQuadCurveToPoint(positionPath, NULL,
        [theView layer].position.x, - [theView layer].position.y *
        1.5, [theView layer].position.x, [theView layer].position.y);
    CGPathAddQuadCurveToPoint(positionPath, NULL,
        [theView layer].position.x, - [theView layer].position.y *
        1.25, [theView layer].position.x, [theView layer].position.y);

    positionAnimation.path = positionPath;
    positionAnimation.timingFunction = [CAMediaTimingFunction
        functionWithName:kCAMediaTimingFunctionEaseIn];

    // Fügt die Animation hinzu
    [[theView layer] addAnimation:positionAnimation
        forKey:@"positionAnimation"];

    [CATransaction commit];

}
```

► *Rezept 6.12: Standardmäßige Core Animation-Aufrufe auf dem iPhone verwenden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.18 Übergänge mit Umblättereffekt

In den beiden vorherigen Rezepten haben Sie zwei wichtige Dinge kennengelernt: `UIView`-Animationsübergänge und Core Animation-Übergänge. Mit diesen Ansätzen können Sie in Ihren Anwendungen den Wechsel von einer Ansicht zur nächsten animieren. Neben den beiden Wendeübergängen ermöglicht die Klasse `UIView` auch zwei Übergänge mit einem Umblättereffekt, nämlich `UIViewAnimationTransitionCurlUp` und `UIViewAnimationTransitionCurlDown`. Damit haben Sie eine neue Möglichkeit, Ansichten auszutauschen, nämlich so, dass die Ansicht nach oben (oder unten) umgeblättert wird, sodass die nächste erscheint. *Abbildung 6.5* zeigt diesen Vorgang.

▸ *Abbildung 6.5:* `UIView`*-Animationen mit Umblättereffekt verwenden*

Eine solche Animation erstellen und verwenden Sie genauso wie die integrierten Wendeübergänge. Sie wenden den Übergang auf einen Hintergrund mit den beiden Ansichten an, die Sie animieren möchten, und tauschen diese Ansichten aus. *Tabelle 6.1* zeigt, welche Übergänge auf dem iPhone möglich sind.

Übergangsname	Erklärung
`UIViewAnimationTransitionFlipFromLeft`	`UIView`-Übergang, der eine Ansicht von links nach rechts wendet und die alte Ansicht durch die neue ersetzt
`UIViewAnimationTransitionFlipFromRight`	`UIView`-Übergang, der eine Ansicht von rechts nach links wendet, wobei die alte Ansicht verborgen und die neue freigelegt wird

`UIViewAnimationTransitionCurlUp`	`UIView`-Übergang, der eine Ansicht von unten nach oben aufblättert, sodass die neue Ansicht sichtbar wird
`UIViewAnimationTransitionCurlDown`	`UIView`-Übergang, bei dem eine Ansicht von oben nach unten über die alte Ansicht geblättert wird
`kCATransitionFade`	Core Animation-Überblendung, bei der die neue Ansicht an Ort und Stelle ein- und die alte ausgeblendet wird
`kCATransitionMoveIn`	Core Animation-Übergang, bei der die neue Ansicht wie ein Blatt Papier über die alte geschoben wird. Hierbei gibt es die Stile `up`, `down`, `left` und `right`.
`kCATransitionPush`	Core Animation-Übergang, bei dem die neue Ansicht die alte wegschiebt. Hierbei gibt es die Stile `up`, `down`, `left` und `right`.
`kCATransitionReveal`	Core Animation-Übergang, bei dem die alte Ansicht weggezogen wird, um die neue darunter freizulegen. Hierbei gibt es die Stile `up`, `down`, `left` und `right`.

▶ *Tabelle 6.1: Übergänge in Cocoa Touch*

```
- (void) curl: (id) sender
{
    self.navigationItem.rightBarButtonItem = nil;

    CGContextRef context = UIGraphicsGetCurrentContext();
    [UIView beginAnimations:nil context:context];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];

    UIView *whiteBackdrop = [self.view viewWithTag:100];

    // Wählt die Animationsrichtung
    if ([(UISegmentedControl *)self.navigationItem.titleView
       selectedSegmentIndex])
    {
        [UIView setAnimationTransition: UIViewAnimationTransitionCurlUp
           forView:whiteBackdrop cache:YES];
    }
```

```
    else
    {
        [UIView setAnimationTransition: UIViewAnimationTransitionCurl
           Down
         forView:whiteBackdrop cache:YES];
    }

    // Die beiden Vordergrundansichten austauschen
    NSInteger purple = [[whiteBackdrop subviews]
       indexOfObject:[whiteBackdrop viewWithTag:999]];
    NSInteger maroon = [[whiteBackdrop subviews]
       indexOfObject:[whiteBackdrop viewWithTag:998]];
    [whiteBackdrop exchangeSubviewAtIndex:purple
       withSubviewAtIndex:maroon];

    [UIView setAnimationDelegate:self];
    [UIView setAnimationDidStopSelector:@selector(animationFinished:)];
    [UIView commitAnimations];
}
```

6.19 Rezept: Ansichten beim Erscheinen nachfedern lassen

Häufig verwendet Apple zwei Animationsblöcke, bei denen der zweite nach dem Abschluss des ersten aufgerufen wird, um eine Animation nachfedern zu lassen. Beispielsweise wird eine Ansicht etwas stärker vergrößert als notwendig und dann mit der zweiten Animation auf die endgültige Größe gebracht. Durch dieses »Nachfedern« werden Animationen lebendiger und wirken realer.

Wenn Sie eine Animation nach einer anderen aufrufen, müssen Sie sicherstellen, dass sich die beiden Animationen nicht überschneiden. Es gibt zwei »Standard«-Verfahren, um aufeinander folgende `UIView`-Animationsblöcke zu erstellen, ohne `CAKeyframeAnimation` zu verwenden (wobei Core Animation-Keyframe-Animationen aber den bevorzugten und einfacheren Ansatz darstellen, weshalb sie weiter hinten in diesem Kapitel ebenfalls vorgestellt werden).

Keine dieser beiden Möglichkeiten ist ideal. Sie stellen eher einen Programmieralbtraum dar, da die Steuerung ständig zwischen verschiedenen Methoden wechselt. Bei der einen dieser beiden Standardlösungen wird eine Verzögerung hinzugefügt, sodass die zweite Animation nicht vor dem Ende der ersten beginnt (`performSelector: withObject: afterDelay:`), bei der anderen wird ein Callback für einen Animations-Delegate zugewiesen (`animationDidStop: finished:context:`), um das Ende der ersten Animation abzuwarten, bevor die zweite gestartet wird.

Für die Programmierung ist es sehr viel einfacher, eine Animation einzurichten, die andere Animationen blockiert, solange sie noch nicht abgeschlossen ist. Genau dies geschieht in *Listing 6.2*. Dort wird die Klasse `UIView` um die neue Klassenmethode `commitModalAnimations` erweitert, die Sie anstelle von `commitAnimations` aufrufen. Dadurch entsteht eine neue Ausführungsschleife, die

bis zum Ende der Animation läuft, sodass `commitModalAnimations` die Steuerung nicht an die aufrufende Methode zurückgeben kann, bevor die Animation abgeschlossen ist. Mit dieser Erweiterung können Sie in Ihrem Code Blöcke hintereinander platzieren und müssen nichts weiter tun, um Überschneidungen zu vermeiden.

```
@interface UIView (ModalAnimationHelper)
+ (void) commitModalAnimations;
@end

@interface UIViewDelegate : NSObject
{
   CFRunLoopRef currentLoop;
}
@end

@implementation UIViewDelegate
-(id) initWithRunLoop: (CFRunLoopRef)runLoop
{
   if (self = [super init]) currentLoop = runLoop;
   return self;
}

-(void) animationFinished: (id) sender
{
   CFRunLoopStop(currentLoop);
}
@end

@implementation UIView (ModalAnimationHelper)
+ (void) commitModalAnimations
{
   CFRunLoopRef currentLoop = CFRunLoopGetCurrent();

   UIViewDelegate *uivdelegate = [[UIViewDelegate alloc]
      initWithRunLoop:currentLoop];
   [UIView setAnimationDelegate:uivdelegate];
   [UIView setAnimationDidStopSelector:@selector(animationFinished:)];
   [UIView commitAnimations];

   CFRunLoopRun();

   [uivdelegate release];
}
@end
```

▸ *Listing 6.2: Eine modale Animation mithilfe einer Ausführungsschleife erstellen*

6.19 Rezept: Ansichten beim Erscheinen nachfedern lassen

Mit diesem modalen Ansatz können Sie die nachfedernde Darstellung aus *Rezept 6.13* erstellen. Hier endet jeder Animationsblock mit dem modalen Commit. Die Ausführungsschleife einer Methode verhindert, dass der nächste Block startet, bevor der vorhergehende abgeschlossen ist.

```
- (void) animate: (id) sender
{
   // Die Leistenschaltfläche verbergen und die Ansicht anzeigen
   self.navigationItem.rightBarButtonItem = nil;
   [self.view viewWithTag:101].alpha = 1.0f;

   // Auf 115% der Normalgröße vorpreschen
   [UIView beginAnimations:nil context:UIGraphicsGetCurrentContext()];
   [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
   [UIView setAnimationDuration:0.4f];
   [self.view viewWithTag:101].transform =
      CGAffineTransformMakeScale(1.15f, 1.15f);
   [UIView commitModalAnimations];

   // Auf 100% zurückgehen
   [UIView beginAnimations:nil context:UIGraphicsGetCurrentContext()];
   [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
   [UIView setAnimationDuration:0.3f];
   [self.view viewWithTag:101].transform =
      CGAffineTransformMakeScale(1.0f, 1.0f);
   [UIView commitModalAnimations];

   // Eine Sekunde lang anhalten und die Darstellung bewundern
   [NSThread sleepUntilDate:[NSDate
      dateWithTimeIntervalSinceNow:1.0f]];

   // Langsam wieder verkleinern und die Ansicht ausblenden
   [UIView beginAnimations:nil context:UIGraphicsGetCurrentContext()];
   [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
   [UIView setAnimationDuration:1.0f];
   [self.view viewWithTag:101].transform =
      CGAffineTransformMakeScale(0.01f, 0.01f);
   [UIView commitModalAnimations];

   [self.view viewWithTag:101].alpha = 0.0f;

   // Leistenschaltfläche wiederherstellen
   self.navigationItem.rightBarButtonItem =
      BARBUTTON(@"Bounce", @selector(animate:));
}
```

► *Rezept 6.13: Ansichten nachfedern lassen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.20 Rezept: Bildansichten animieren

Die Klassse `UIImageView` ermöglicht Ihnen nicht nur, statische Bilder anzuzeigen, sondern enthält auch Animationen dafür. Nachdem Sie ein Array aus Bildern geladen haben, können Sie die `UIImageView`-Instanzen anweisen, die Bilder zu animieren. *Rezept 6.14* zeigt, wie das geht.

Sie beginnen damit, ein Array zu erstellen und mit einzelnen Bildern zu füllen, die Sie aus einer Datei laden, und weisen dieses Array dann der Eigenschaft `animationImages` einer `UIImageView`-Instanz zu. Setzen Sie `animationDuration` auf die Gesamtdauer der Schleife, um alle Bilder in dem Array anzuzeigen. Anschließend starten Sie die Animation, indem Sie die Nachricht `startAnimating` senden. (Es gibt auch die ergänzende Methode `stopAnimating`.)

Wenn Sie die animierte Bildansicht zu Ihrer Schnittstelle hinzufügen, können Sie sie an einer festen Stelle platzieren, aber auch so animieren wie jede andere `UIView`-Instanz.

```
NSMutableArray *bflies = [NSMutableArray array];
// Das Schmetterlingsbild laden
for (int i = 1; i <= 17; i++)
   [bflies addObject:[UIImage imageWithContentsOfFile:
      [[NSBundle mainBundle]
         pathForResource: [NSString stringWithFormat:@"bf_%d", i]
         ofType:@"png"]]];

// Die Ansicht erstellen
UIImageView *butterflyView = [[UIImageView alloc]
   initWithFrame:CGRectMake(40.0f, 300.0f, 60.0f, 60.0f)];

// Animationszellen und Dauer festlegen
butterflyView.animationImages = bflies;
butterflyView.animationDuration = 0.75f;

// Die Ansicht zur Elternansicht hinzufügen und freigeben
[self.view addSubview:butterflyView];

[butterflyView startAnimating];

[butterflyView release];
```

▸ *Rezept 6.14: `UIImageView`-Animationen verwenden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 6 und öffnen das Projekt zu diesem Rezept.

6.21 Zum guten Schluss: Spiegelungen zu Ansichten hinzufügen

Spiegelungen erhöhen den realistischen Eindruck von Objekten auf dem Bildschirm. Sie geben den üblichen Ansichten über einem Hintergrund zusätzliche optische Würze. Es ist nicht allzu schwer, Spiegelungen hervorzuheben, wobei der Schwierigkeitsgrad natürlich von Ihren Ansprüchen an die Ergebnisse abhängt.

Für die einfachsten Spiegelungen brauchen Sie nichts weiter als eine umgekehrte Kopie der ursprünglichen Ansicht und vielleicht noch eine Anpassung des Alphawerts für die Spiegelung, um ihr ein etwas ätherischeres Erscheinungsbild zu geben. In *Listing 6.3* sehen Sie eine grundlegende Spiegelung auf der Grundlage von Core Animation, für die die Ansicht in eine neue Ebene kopiert, über eine Skalierungstransformation umgedreht und mit einem bestimmten Abstand platziert wird. Das Ergebnis sehen Sie in *Abbbildung 6.6*.

Bei diesem Ansatz bleibt die Spiegelungsebene bei der Ansicht. Wenn Sie die Ansicht verschieben, folgt ihr die Spiegelung.

```
const CGFloat kReflectPercent = -0.25f;
const CGFloat kReflectOpacity = 0.3f;
const CGFloat kReflectDistance = 10.0f;

+ (void) addSimpleReflectionToView: (UIView *) theView
{
   CALayer *reflectionLayer = [CALayer layer];
   reflectionLayer.contents = [theView layer].contents;
   reflectionLayer.opacity = kReflectOpacity;
   reflectionLayer.frame = CGRectMake(0.0f, 0.0f,
      theView.frame.size.width, theView.frame.size.height *
      kReflectPercent);
   CATransform3D stransform = CATransform3DMakeScale(1.0f, -1.0f, 1.0f);
   CATransform3D transform = CATransform3DTranslate(stransform, 0.0f,
      -(kReflectDistance + theView.frame.size.height), 0.0f);
   reflectionLayer.transform = transform;
   reflectionLayer.sublayerTransform = reflectionLayer.transform;
   [[theView layer] addSublayer:reflectionLayer];
}
```

▸ *Listing 6.3: Spiegelungen erstellen*

▸ *Abbildung 6.6: Bei einer grundlegenden Core Graphics-Spiegelung werden Skalierung, Transparenz und ein kleiner vertikaler Versatz verwendet.*

6.21.1 Bessere Spiegelungen

Spiegelungen in voller Größe machen sich in einfachen Schnittstellen zwar sehr gut, aber eine bessere Spiegelung verschwimmt an ihrem unteren Rand, um ein eleganteres, Apple-artigeres Erscheinungsbild zu bekommen. Diese gedrehten, maskierten Spiegelungen, die Sie in *Abbildung 6.7* sehen, können Sie mit Core Graphics-Funktionen erstellen.

▸ *Abbildung 6.7: Eine Apple-artigere Spiegelung erhalten Sie, wenn Sie den unteren Teil maskieren.*

6.21 Zum guten Schluss: Spiegelungen zu Ansichten hinzufügen

Für diese Lösung ist zugegebenermaßen etwas mehr Aufwand nötig als für die einfache Spiegelung in *Listing 6.3*. Für die verschwimmende Spiegelung aus *Listing 6.4* wird der Inhalt der Ansicht in eine gekürzte Bitmap kopiert und mit einer Verlaufsmaske versehen. Das Ergebnis wird als `UIImage` zurückgegeben und als neue `UIImageView` der ursprünglichen Ansicht hinzugefügt. Auch bei diesem Ansatz mit der Unteransicht verbleibt die Spiegelung bei ihrem Elternelement.

Damit der Spiegelungseffekt funktioniert, müssen Sie unbedingt die Beschneidung der Ansichten deaktivieren. Setzen Sie die Eigenschaft `clipsToView` der Ansicht auf `NO`, damit die Elternansicht die Spiegelung nicht wegschneidet. Die Reflexion bleibt vollständig sichtbar, auch die Teile, die außerhalb der Grenzen der Elternansicht liegen.

```
+ (CGImageRef) createGradientImage: (CGSize)size
{
   CGFloat colors[] = {0.0, 1.0, 1.0, 1.0};

   // Verlauf im grauen Gerätefarbraum erstellen
   CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceGray();
   CGContextRef context = CGBitmapContextCreate(nil, size.width,
      size.height, 8, 0, colorSpace, kCGImageAlphaNone);
   CGGradientRef gradient =
   CGGradientCreateWithColorComponents(colorSpace, colors, NULL, 2);
   CGColorSpaceRelease(colorSpace);

   // Linearen Verlauf zeichnen
   CGPoint p1 = CGPointZero;
   CGPoint p2 = CGPointMake(0, size.height);
   CGContextDrawLinearGradient(context, gradient, p1, p2,
      kCGGradientDrawsAfterEndLocation);

   // Das CGImage zurückgeben
   CGImageRef theCGImage = CGBitmapContextCreateImage(context);
   CFRelease(gradient);
   CGContextRelease(context);
   return theCGImage;
}

// Einen verkleinerten Rahmen für die Spiegelung erstellen
+ (UIImage *) reflectionOfView: (UIView *)theView
   withPercent: (CGFloat) percent
{
   // Breite beibehalten, Höhe verringern
   CGSize size = CGSizeMake(theView.frame.size.width,
      theView.frame.size.height * percent);

   // Ansicht verkleinern
```

```
    UIGraphicsBeginImageContext(size);
    CGContextRef context = UIGraphicsGetCurrentContext();
    [theView.layer renderInContext:context];
    UIImage *partialimg =
        UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();

    // Maske erstellen
    CGImageRef mask = [ImageHelper createGradientImage:size];
    CGImageRef ref = CGImageCreateWithMask(partialimg.CGImage, mask);
    UIImage *theImage = [UIImage imageWithCGImage:ref];
    CGImageRelease(ref);
    CGImageRelease(mask);
    return theImage;
}

const CGFloat kReflectDistance = 10.0f;

+ (void) addReflectionToView: (UIView *) theView
{
    theView.clipsToBounds = NO;
    UIImageView *reflection = [[UIImageView alloc] initWithImage:
        [ImageHelper reflectionOfView:theView withPercent: 0.45f]];
    CGRect frame = reflection.frame;
    frame.origin = CGPointMake(0.0f, theView.frame.size.height +
        kReflectDistance);
    reflection.frame = frame;

    // Spiegelung als einfache Unteransicht hinzufügen
    [theView addSubview:reflection];
    [reflection release];

}
```

▸ *Listing 6.4: Spiegelungen in Core Graphics maskieren*

6.22 Zusammenfassung

`UIViews` stellen die Bildschirmkomponenten zur Verfügung, die Benutzer sehen und mit denen sie interagieren. Wie dieses Kapitel gezeigt hat, bieten sie selbst in ihrer grundlegendsten Form eine unglaubliche Flexibilität und Leistung. Sie haben erfahren, wie Sie Ansichten verwenden, um Elemente auf einem Bildschirm anzuzeigen, wie Sie Ansichten nach Tag oder Name abrufen und wie Sie bemerkenswerte Animationen verwenden. Die folgende Liste führt noch einmal einzelne Punkte der Rezepte auf, die Sie in diesem Kapitel gesehen haben und über die Sie nachdenken sollten, bevor Sie weiterlesen:

- Wenn Sie mehrere Ansichten auf dem Bildschirm haben, sollten Sie immer an die Hierarchie denken. Nutzen Sie Ihr Hierarchievokabular (`bringSubviewToFront:`, `sendSubviewToBack:`, `exchangeSubviewAtIndex:withSubviewAtIndex:`) für Ihre Ansichten, und bieten Sie Ihren Benutzern stets den richtigen optischen Kontext.
- Lassen Sie sich nicht durch den Konflikt zwischen `frame` in Core Graphics und `center` in `UIKit` behindern, sondern verwenden Sie Funktionen, um zwischen diesen Strukturen zu wechseln und die gewünschten Ergebnisse zu erzielen.
- Freunden Sie sich mit Tags an. Sie bieten auf vergleichbare Weise unmittelbaren Zugriff auf Ansichten wie die Symboltabelle eines Programms auf die Variablen.
- Die Verwendung nicht dokumentierter Animationen ist gefährlich, da solche Animationen nach einer Aktualisierung der Firmware möglicherweise nicht mehr so funktionieren, wie Sie es erwarten. Animationen mit einem Umblättereffekt gehören jedoch schon lange zu Mac OS X und zum iPhone.
- Animieren Sie alles. Animationen müssen nicht laut, aufdringlich oder geschmacklos sein. Dank der umfangreichen Unterstützung für Animationen auf dem iPhone können Sie Benutzeraufgaben sanft ineinander übergehen lasen. Solche unaufdringlichen, fließenden Übergänge machen das iPhone-Flair aus. Kurze, sanfte, klare Wechsel sind das A und O auf dem iPhone.

7 Mit Bildern arbeiten

Auf dem iPhone müssen wir genau zwischen Bildern und Ansichten unterscheiden. Im Gegensatz zu Ansichten sind Bilder nicht selbst auf dem Bildschirm vorhanden. Ansichten können zwar Bilder enthalten und anzeigen, sind aber keine Bilder, nicht einmal, wenn es sich um `UIImageView`-Objekte handelt. In diesem Kapitel erhalten Sie eine Einführung in Bilder, vor allem in die Klasse `UIImage`. Hier lernen Sie das grundlegende Know-how für den Umgang mit Bildern auf dem iPhone kennen. Sie erfahren, wie Sie Bilddaten in Ihren Anwendungen laden, speichern und bearbeiten, wie Sie Bilder zu Ansichten hinzufügen und wie Sie Ansichten in Bilder umwandeln. Außerdem wird erklärt, wie Sie Bilddaten verarbeiten, um besondere Effekte hervorzurufen, wie Sie Byte für Byte auf Bilder zugreifen und wie Sie mit der eingebauten Kamera des iPhones Fotos aufnehmen.

7.1 Rezept: Bilder finden und laden

Es gibt vier allgemeine Speicherorte für iPhone-Bilder, in denen Sie auf Bilddaten zugreifen und diese in Ihren Programmen darstellen können. Bei diesen Quellen handelt es sich um das Fotoalbum, das Anwendungsbundle, die Sandbox und das Internet:

- **Fotoalbum** Das Fotoalbum des iPhones enthält den »Film« der Kamera (bei Geräten, die damit ausgestattet sind) sowie Fotos, die vom Computer des Benutzers übertragen wurden. Benutzer können Bilder aus diesem Album über das interaktive Dialogfeld anfordern, das von der Klasse `UIImagePickerController` bereitgestellt wird. In diesem Dialogfeld können die Benutzer die gespeicherten Fotos durchsuchen und diejenigen auswählen, mit denen sie arbeiten möchten.
- **Anwendungsbundle** Das Bundle Ihrer Anwendung kann neben der ausführbaren Datei, der Datei `Info.plist` und anderen Ressourcen auch Bilder enthalten. Diese Bilder können Sie über ihren lokalen Dateipfad einlesen und in der Anwendung anzeigen lassen.

- **Sandbox** Eine Anwendung kann auch Bilder in die Sandbox schreiben und bei Bedarf wieder von dort lesen. Über die Sandbox können Sie Dateien in den Ordnern `Documents`, `Library` und `tmp` speichern. Jeden dieser Ordner kann Ihre Anwendung lesen. Neue Bilder erstellen Sie, indem Sie einen Dateipfad angeben. Rein technisch gesehen, ist es zwar auch möglich, in einigen Bereichen außerhalb der Sandbox zu lesen, doch hat Apple deutlich gemacht, dass der Zugriff auf diese Bereiche für App Store-Anwendungen unzulässig ist.
- **Internet** Mithilfe von URL-Ressourcen, die auf Dateien im Web zeigen, kann eine Anwendung Bilder aus dem Internet herunterladen. Damit dies funktioniert, braucht das iPhone eine aktive Webverbindung. Sobald sie besteht, sind die Daten eines im Netzwerk gespeicherten Bildes ebenso zugänglich wie lokale Daten.

7.1.1 Bilddateien lesen

Der Speicherort einer Bilddatei bestimmt, wie Sie die Daten lesen können. Man könnte meinen, mit einer Methode wie `imageWithContentsOfFile:` von `UIImage` ließen sich Dateien aller vier Typen laden, doch ist dies in der Praxis nicht möglich. Bilder im Fotoalbum und ihre Pfade sind (zumindest offiziell) vor dem direkten Zugriff durch Anwendungen verborgen. Nur Endbenutzer dürfen Bilder durchsuchen und auswählen und das gewünschte Bild damit der Anwendung zur Verfügung stellen. Es ist auch nicht möglich, Bilder direkt mit URLs zu initialisieren, obwohl es dafür eine einfache Notlösung gibt. Die folgenden Abschnitte zeigen, wie Sie Daten aus den verschiedenen Quellen laden.

Bilder aus dem Anwendungsbundle laden

Die Klasse `UIImage` enthält eine einfache Methode, um Bilder aus dem Anwendungsbundle zu laden. Rufen Sie wie im folgenden Beispiel `imageNamed:` mit einem Dateinamen samt Erweiterung auf:

```
myImage = [UIImage imageNamed:@"icon.png"];
```

Diese Methode sucht im Ordner der obersten Ebene des Anwendungsbundles nach einem Bild mit dem angegebenen Namen. Ist eines vorhanden, so wird es geladen und im Cache des iPhone-Systems zwischengespeichert. Das bedeutet, dass (theoretisch) dieser Cache die Speicherverwaltung des Bildes übernimmt.

In der Praxis kann die Methode `imageNamed:` jedoch nicht so ungehindert eingesetzt werden, da der Bildcache nicht sauber auf Speicherwarnungen reagiert und seine Objekte freigibt. Für einfache Anwendungen und für kleine Bilder, die in der Anwendung immer wieder verwendet werden, ist dies kein Problem. Bei umfangreichen Anwendungen jedoch, bei denen der Speicher sorgfältig zugewiesen und freigegeben werden muss und bei denen es nur wenig Reserven gibt, stellt sich ein großes Problem. Wegen dieser Schwierigkeiten mit dem integrierten Cache haben viele Entwickler ihren eigenen Bildercache entwickelt, wie schon der Beispielcode aus Kapitel 2, *Ein erstes Projekt erstellen*, gezeigt hat.

Nehmen Sie statt `imageNamed:` die Methode `imageWithContentsOfFile:`. Sie gibt das Bild zurück, das von dem als Argument angegebenen Pfad geladen wurde. Um einen Bildpfad aus dem Bundle abzurufen, fragen Sie die Klasse `NSBundle` nach dem Pfad für eine gegebene Ressource ab. Der folgende Code lädt das Bild `icon.png` von der obersten Ebene des Anwendungsbundles. Beachten Sie, dass Sie den Dateinamen und die Dateierweiterung als zwei getrennte Argumente angeben müssen.

```
NSString *path = [[NSBundle mainBundle]
   pathForResource:@"icon" ofType:@"png"];
myImage = [UIImage imageWithContentsOfFile:path];
```

Hinweis

Das iPhone unterstützt die Bildtypen PNG, JPG, THM, JPEG, TIF, TIFF, GIF, BMP, BMPF, ICO, CUR, XMB und PDF.

Bilder aus der Sandbox laden

Standardmäßig enthält jede Sandbox drei Ordner: `Documents`, `Library` und `tmp`. Von der Anwendung erstellte Daten wie Bilder befinden sich normalerweise im Ordner `Documents`. Dieser Ordner ist genau für den Zweck da, auf den sein Name hindeutet: zur Speicherung von Dokumenten und zum Zugriff darauf. Apple empfiehlt, hier die Datendateien aufzubewahren, die von Ihrem Programm erstellt oder genutzt werden.

Der Ordner `Library` dient zur Speicherung von Benutzervoreinstellungen und anderen Statusinformationen für Ihr Programm, während der Ordner `tmp` der Ort ist, an dem Sie flüchtige Dateien im laufenden Betrieb erstellen. Im Gegensatz zu den Dateien in `tmp` sind die in den Ordnern `Documents` und `Library` nicht flüchtig. Bei jeder Synchronisierung des iPhones sichert iTunes sämtliche Dateien in diesen beiden Ordnern. Die Dateien in `tmp` dagegen werden bei jedem Neustart verworfen.

Diese Verzeichnisse machen einen der Hauptunterschiede zwischen der Programmierung für den Macintosh und für das iPhone deutlich. Auf dem Mac können Sie sowohl die standardmäßigen als auch davon abweichende Speicherorte für Dateien wählen, doch das iPhone mit seiner Sandbox ist nach den Regeln von Apple sehr viel strenger gegliedert: Die Dateien erscheinen in genauer festgelegten Speicherorten. Wenn Sie auf dem Macintosh den Ordner `Documents` finden wollen, müssen Sie gewöhnlich den Benutzerordner durchsuchen. Der übliche Weg dazu sieht wie folgt aus:

```
NSArray *paths = NSSearchPathForDirectoriesInDomains(
   NSDocumentDirectory, NSUserDomainMask, YES);
return [paths lastObject];
```

Das iPhone unterliegt stärkeren Einschränkungen. Sie können sich darauf verlassen, dass Sie den obersten Sandbox-Ordner finden, wenn Sie eine Hilfsfunktion zum Aufruf des Benutzerordners einsetzen. Wenn Sie vom Ergebnis von `NSHomeDirectory()` eine Ebene tiefer gehen, können Sie sicher

sein, dass Sie Documents erreichen. Die folgende Funktion bietet eine praktische Möglichkeit, um den Pfad zum Dokumentenordner zurückzugeben:

```
NSString *documentsFolder()
{
   return [NSHomeDirectory()
      stringByAppendingPathComponent:@"Documents"];
}
```

Um ein Bild zu laden, hängen Sie seinen Dateinamen an den zurückgegebenen Pfad an und weisen UIImage an, ein neues Bild mit den betreffenden Inhalten zu erstellen. Der folgende Code lädt die Datei image.png von der obersten Ebene des Dokumentenordners und gibt eine mit diesen Daten initialisierte Instanz von UIImage zurück:

```
path = [documentsFolder() stringByAppendingPathComponent:@"image.png"];
return [UIImage imageWithContentsOfFile:path];
```

Bilder von URLs laden

Die Klasse UIImage kann Bilder aus NSData-Instanzen laden, aber nicht direkt aus URL-Strings oder NSURL-Objekten. Daher müssen Sie UIImage mit Daten versorgen, die Sie bereits von einem URL heruntergeladen haben. Der folgende Code lädt die jüngste US-Wetterkarte von *weather.com* herunter und erstellt dann mit diesen Daten ein neues Bild. Als Erstes wird ein NSURL-Objekt konstruiert und dann eine NSData-Instanz, die mit den Inhalten dieses URLs initialisiert wird. Die zurückgegebenen Daten helfen beim Aufbau der UIImage-Instanz.

```
NSURL *url = [NSURL URLWithString:
   @"http://image.weather.com/images/maps/current/curwx_600x405.jpg"];
UIImage *img = [UIImage imageWithData:
   [NSData dataWithContentsOfURL:url]];
```

Es ist einfach, eine Methode zu schreiben, die diesen Vorgang für Sie erledigt, sodass Sie nur noch einen URL-String angeben müssen, um ein UIImage abzurufen. Diese Methode übernimmt einen URL-String als einziges Argument und gibt das mithilfe dieser Ressource erstellte UIImage zurück.

```
+ (UIImage *) imageFromURLString: (NSString *) urlstring
{
   // Dies ist ein synchroner und blockierender Aufruf
   return [UIImage imageWithData:[NSData
      dataWithContentsOfURL:[NSURL URLWithString:urlstring]]];
}
```

Dies ist eine synchrone Methode, die sicherlich einige Nachteile aufweist. Wenn es keine Rückmeldung gibt, schlägt sie fehl, und sie hat auch keinen eingebauten Timeout. In Kapitel 13, *Netzwerke*, finden Sie ausführliche Erläuterungen über den Abruf von Ressourcen von URLs.

Daten aus dem Fotoalbum laden

Mit der Klasse `UIImagePickerController` können Benutzer Bilder aus dem iPhone-Fotoalbum auswählen. Sie bietet einen eigenständigen Ansichtscontroller, den Sie modal darstellen. Der Controller sendet eine Delegate-Nachricht über das Bild zurück, das der Benutzer ausgewählt hat.

7.1.2 Bilddateien laden

Rezept 7.1 dient zur Einführung in die Klasse `ImageHelper`, die überall in diesem Kapitel verwendet wird. Sie ist eine Hilfsklasse mit praktischen Routinen für den Umgang mit Bildern. All diese Routinen sind als Klassenmethoden implementiert, sodass Sie kein `ImageHelper`-Objekt zuweisen müssen. Fragen Sie einfach die Klasse ab, um die gewünschten Ergebnisse zu erhalten.

Die `ImageHelper`-Version von `imageNamed:` lädt Dateien mit der Methode `imageWithContentsOfFile:` von `UIImage` und umgeht dadurch die Cache-Probleme der systemeigenen `imageNamed:`-Methode. Die neue Methode durchsucht zunächst das Anwendungsbundle. Wenn sie die Datei dort nicht findet, führt sie eine zweite Suche im Dokumentenordner der Sandbox durch. In beiden Fällen handelt es sich um eine Tiefensuche, die sämtliche Unterordner einbezieht. Die Suche endet, wenn die erste Übereinstimmung gefunden wird oder wenn sich der Suchvorgang als ergebnislos erweist.

Die Methode `imageFromULRString:` in *Rezept 7.1* implementiert die Anforderung eines Bildes von einem URL, wie sie bereits weiter vorn in diesem Abschnitt besprochen wurde. Es wird nicht geprüft, ob das Gerät zurzeit mit dem Internet verbunden ist. Wenn Sie einen solchen Test benötigen, verwenden Sie ein permanentes Wi-Fi-Flag in `Info.plist` (siehe Anhang A, *Schlüssel in Info.plist*), und führen Sie eine Prüfung der Verbindung durch (siehe Kapitel 13, *Netzwerke*).

```
NSString *documentsFolder()
{
   // Gibt den Dokumentenordner der Sandbox zurück
   return [NSHomeDirectory()
      stringByAppendingPathComponent:@"Documents"];
}

NSString *bundleFolder()
{
   // Gibt den Ordner des Anwendungsbundles zurück
   return [[NSBundle mainBundle] bundlePath];
}

@implementation ImageHelper (Files)

+ (NSString *) pathForItemNamed: (NSString *) fname
   inFolder: (NSString *) path
{
   // Gibt den vollständigen Pfad für das benannte Element zurück
   NSString *file;
```

```
    NSDirectoryEnumerator *dirEnum =
        [[NSFileManager defaultManager] enumeratorAtPath:path];
    while (file = [dirEnum nextObject])
        if ([[file lastPathComponent] isEqualToString:fname])
            return [path stringByAppendingPathComponent:file];
    return nil;
}

// Durchsucht erst das Bundle und dann den Dokumentenordner
+ (UIImage *) imageNamed: (NSString *) aName
{
    // Gibt ein UIImage für das benannte Element zurück
    NSString *path = [ImageHelper pathForItemNamed:aName
        inFolder:bundleFolder()];
    path = path ? path : [ImageHelper pathForItemNamed:aName
        inFolder:documentsFolder()];
    if (!path) return nil;
    return [UIImage imageWithContentsOfFile:path];
}

+ (UIImage *) imageFromURLString: (NSString *) urlstring
{
    // Lädt das Bild herunter, das sich am URL befindet.
    // Dies ist eine blockierende Methode.
    NSURL *url = [NSURL URLWithString:urlstring];
    if (!url) return nil;
    return [UIImage imageWithData:
        [NSData dataWithContentsOfURL: url]];
}
```

▸ *Rezept 7.1: Bilder mit ImageHelper laden*

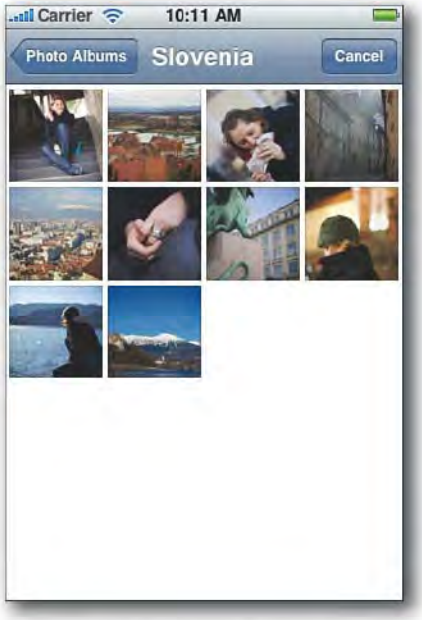
Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.2 Rezept: Auf Fotos im Fotoalbum zugreifen

Die Klasse `UIImagePickerController` bietet eine hochgradig spezialisierte Schnittstelle mit relativ wenigen öffentlichen Methoden und einigen bescheidenen Eigentümlichkeiten. Sie ist einzig und allein als modales Dialogfeld entworfen und umfasst einen eigenen Navigationscontroller. Wenn Sie sie in ein Ansichtsschema mit Navigationscontroller aufnehmen, wird eine zweite Navi-

gationsleiste unter der ersten angezeigt. Das bedeutet, dass Sie diese Klasse zwar mit einem Symbolbildschirm und als unabhängiges Ansichtssystem verwenden können, sie aber nicht in einen vorhandenen Navigationsstack einfügen können, wenn Sie eine richtige Ansicht bekommen möchten.



▶ *Abbildung 7.1: Apple stellt für Tests im Simulator verschiedene vorgefertigte Alben zur Verfügung, z. B. diese Bilder aus Slowenien.*

Rezept 7.2 zeigt die Bildauswahl im einfachsten Modus, in dem die Benutzer ein Bild aus einem der Alben auf dem Gerät auswählen können (siehe *Abbildung 7.1*). Richten Sie die Bildauswahl auf einen der drei folgenden zulässigen Quelltypen ein:

- `UIImagePickerControllerSourceTypePhotoLibrary` Alle auf das iPhone übertragenen Bilder sowie alle vom Benutzer mit der Kamera aufgenommenen Bilder.
- `UIImagePickerControllerSourceTypeSavedPhotoAlbum` Nur die vom Benutzer aufgenommenen Kamerabilder.
- `UIImagePickerControllerSourceTypeCamera` Hiermit können die Benutzer Bilder mit der eingebauten iPhone-Kamera aufnehmen.

7.2.1 Mit der Bildauswahl arbeiten

Rezept 7.2 folgt einem grundlegenden Arbeitsablauf: Album auswählen, Bild auswählen, das ausgewählte Bild anzeigen und wieder von vorn beginnen. Diese einfache Vorgehensweise funktioniert, da die Bildbearbeitungseigenschaft der Picker-Ansicht standardmäßig auf `NO` steht, sodass keine Bildbearbeitung stattfindet. Diese Eigenschaft – `allowsImageEditing` in SDKs vor Version 3.1 und `allowsEditing` in späteren Versionen – teilt der Bildauswahl mit, ob die Benutzer die Bilder verschieben und in der Größe ändern dürfen. Wenn sie deaktiviert ist, geht die Steuerung bei jeder Auswahl (also beim Antippen eines Bildes) über die abgeschlossene Auswahlmethode zurück zum `UIImagePickerControllerDelegate`-Objekt.

Die Delegierung für eine Picker-Ansicht muss den beiden Protokollen `UINavigationControllerDelegate` und `UIImagePickerControllerDelegate` gehorchen. Achten Sie darauf, diese Protokolle im Interface des Objekts zu deklarieren, das Sie als Picker-Delegierung verwenden wollen.

Das Rezept enthält nicht einen, sondern zwei Callbacks, nämlich eine 3.x- und eine 2.x-Version. Wenn Sie Ihre Software sowohl auf 2.x- als auch auf 3.x-Systemen bereitstellen wollen, um den Benutzerstamm auf das größtmögliche Publikum auszudehnen, muss Ihr Code auf Callbacks von beiden Versionen des Betriebssystems reagieren, da von der Verwendung des 2.x-Callbacks in Version 3.0 abgeraten wird.

7.2.2 Unterstützung für 2.x-Systeme hinzufügen

Für die einfache Bildauswahl ist die 2.x-Unterstützung trivial. Der 2.x- wird an den 3.x-Callback umgeleitet und übergibt zusammen mit dem ausgewählten Bild ein konstruiertes Dictionary. Wie Sie in *Rezept 7.3* sehen, wird diese Callback-Umleitung etwas komplizierter, wenn die Bildauswahl Bearbeitungsinformationen zurückgibt.

Es gibt eine grundlegende Garantie dafür, dass das von der Delegierungsmethode gesendete Bild nicht `nil` ist. Allerdings können Sie vor der Konstruktion des Dictionarys eine Überprüfung in die 2.x-Methode einbauen. Sollte der Benutzer den Vorgang abbrechen, empfängt die Delegierung einen `imagePickerControlDidCancel:`-Callback. Die Bildauswahl wird dann automatisch geschlossen und freigegeben.

Dieses Verhalten und die allgemeine Speichernutzung der Bildauswahl können Sie in Instruments beobachten. Dort sehen Sie, wie der Grad der Speichernutzung nach dem Abbruch wieder sinkt. Wenn Sie diesen Callback implementieren möchten (Apple nennt ihn »optional«, aber »erwartet«), müssen Sie dafür sorgen, den Controller manuell zu schließen und freizugeben.

Bei nicht trivialen Anwendungen müssen Sie die Speicherverwaltung in Ihrem Programm implementieren, sodass es bei der Verwendung der Bildauswahl auf Speicherwarnungen reagieren kann. Die integrierte Picker-Ansicht für Bilder geht in beiden Grundvarianten – Bildauswahl und Kameranutzung – schlampig mit dem Speicher um.

7.2.3 Unterstützung für 3.x-Systeme hinzufügen

Die Eigenschaft `allowsImageEditing` wurde in der Version 3.1 des SDK als unerwünscht gekennzeichnet. Zu der Zeit, als dieses Buch geschrieben wurde, konnte sie jedoch weiterhin in Anwendungen eingesetzt werden, was noch eine Zeit lang so bleiben wird, aber sicherlich nicht auf Dauer. In zukünftigen SDKs können unerwünschte Methoden ohne Vorwarnung wegfallen.

Wenn Sie Ihr Programm für verschiedene Firmware bereitstellen wollen, also sowohl für Versionen vor als auch nach 3.1, müssen Sie überprüfen, ob Ihre Instanzen der Bildauswahl auf `setAllowsImageEditing:` und `setAllowsEditing:` reagieren. Verwenden Sie zum Testen die Methode `respondsToSelector:` von `NSObject`.

Hinweis

Die Hilfskategorie `NSObject` unter *http://github.com/erica* löst dieses Problem, indem sie eine Liste der Selektoren durchsucht, bis sie einen findet, auf den ein Objekt antworten kann. Anwendungsbeispiele finden Sie im Begleitcode zu dieser Kategorie.

7.2.4 Videoauswahl

Trotz ihres Namens ist die Klasse `UIImagePickerController` nicht auf Bilder beschränkt, sondern kann zur Auswahl sowohl von Bildern als auch von Videos aus der Medienbibliothek auf dem Gerät eingerichtet werden. Einzelheiten über die Festlegung der Medientypen für die Auswahl finden Sie in Kapitel 15, *Audio, Video und Media Kit*. Dort erhalten Sie auch Hinweise zur Auswahl, Aufnahme und Bearbeitung von Videoressourcen.

```
#define SETIMAGE(X) [(UIImageView *)self.view setImage:X];
@interface TestBedViewController : UIViewController
   <UINavigationControllerDelegate, UIImagePickerControllerDelegate>
@end

@implementation TestBedViewController

// 3.x-Callback
- (void) imagePickerController:(UIImagePickerController *)picker
   didFinishPickingMediaWithInfo:(NSDictionary *)info
{
   SETIMAGE([info objectForKey:
      @"UIImagePickerControllerOriginalImage"]);
   [self dismissModalViewControllerAnimated:YES];
   [picker release];
}

// 2.x-Callback - wird zum 3.x-Callback umgeleitet
- (void)imagePickerController:(UIImagePickerController *)picker
      didFinishPickingImage:(UIImage *)image
      editingInfo:(NSDictionary *)editingInfo
{
   NSDictionary *dict = [NSDictionary dictionaryWithObject:image
      forKey:@"UIImagePickerControllerOriginalImage"];
   [self imagePickerController:picker
      didFinishPickingMediaWithInfo:dict];
}

// Optionale, aber "erwartete" Beendigung
- (void) imagePickerControllerDidCancel:
```

```
      (UIImagePickerController *)picker
{
   [self dismissModalViewControllerAnimated:YES];
   [picker release];
}

// Präsentiert die Bildauswahl
- (void) pickImage: (id) sender
{
   UIImagePickerController *ipc = [[UIImagePickerController alloc]
      init];
   ipc.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
   ipc.delegate = self;
   ipc.allowsImageEditing = NO; // benutze allowsEditing ab 3.1
   [self presentModalViewController:ipc animated:YES];
}

- (void) viewDidLoad
{
   self.navigationController.navigationBar.tintColor =
      COOKBOOK_PURPLE_COLOR;
  self.navigationItem.rightBarButtonItem = BARBUTTON(@"Pick",
      @selector(pickImage:));
   self.title = @"Image Picker";
}
@end
```

▸ *Rezept 7.2: Eine Bildauswahl mit* `UIImagePickerController`

Den Rezeptcode finden

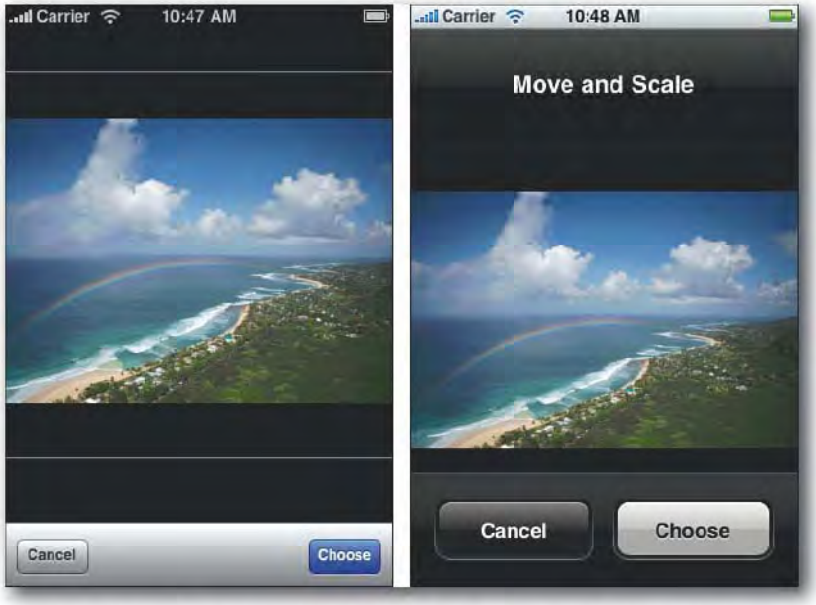
Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.3 Rezept: Bilder vom Kamerafilm auswählen und bearbeiten

Rezept 7.3 erweitert die Interaktionsmöglichkeiten der Bildauswahl um Bearbeitungsvorgänge des Benutzers. Um in einem `UIImagePickerController` die Bildbearbeitung zu ermöglichen, müssen Sie die Eigenschaft `allowsImageEditing` (Version 3.0 und früher) bzw. `allowsEditing` (Version 3.1 und höher) auf `YES` setzen. Dadurch kann der Benutzer die Bilder nach der Auswahl bzw. nach der Aufnahme mit der Kamera in der Größe verändern und positionieren. Diesen Editor sehen Sie auf dem

iPhone in Aktion, wenn Sie die Funktion zum Festlegen des Hintergrundbilds in den Einstellungen verwenden. *Abbildung 7.2* zeigt den Editor nach der Auswahl für Firmware der Versionen 3.x und 2.x.

In diesem Fenster kann der Benutzer die Bilder nach Wunsch verschieben und in der Größe ändern. Durch Kneifbewegungen wird das Bild vergrößert bzw. verkleinert, durch Ziehen wird der Ursprung verschoben.



▸ *Abbildung 7.2: Im interaktiven Bildeditor können die Benutzer Bilder verschieben und in der Größe ändern sowie die endgültige Darstellung auswählen. Links sehen Sie den Editor der Version 3.x, rechts die Version 2.x. Wie das linke Bild zeigt, erscheint die Bezeichnung »Move and Scale« nicht immer, nicht einmal, wenn sich das iPhone im Bearbeitungsmodus befindet.*

Wenn der Benutzer auf **Choose** tippt, geht die Steuerung zur Delegierung der Bildauswahl über, wo Ihr Programm sie aufgreift. Beim Antippen von **Cancel** geschieht etwas anderes: Die Steuerung kehrt zur Albumansicht zurück, sodass der Benutzer ein anderes Bild auswählen und von Neuem beginnen kann.

7.3.1 Bildbearbeitungsinformationen abrufen

Der 3.x-Callback gibt ein Dictionary mit Informationen über das ausgewählte Bild zurück. Ab der Firmware der Version 3.x enthält dieses Dictionary vier Schlüssel, die den Zugriff auf wichtige Dictionary-Daten gewähren:

- `UIImagePickerControllerMediaType` Gibt an, welche Art Medium der Benutzer ausgewählt hat, gewöhnlich `public.image`. Die Medientypen werden in der Headerdatei `UTCoreTypes.h` definiert, die zum Framework Mobile Core Services gehört und neu ab der Version 3.0 ist. Hauptsächlich werden Medientypen verwendet, um Elemente zur Zwischenablage des Systems hinzuzufügen.

- `UIImagePickerControllerCropRect` Gibt den vom Benutzer ausgewählten Abschnitt des Bildes in Form eines `NSValue`-Objekts, das ein `GCRect` enthält, zurück.
- `UIImagePickerControllerOriginalImage` Speichert eine `UIImage`-Instanz mit den Inhalten des ursprünglichen (nicht bearbeiteten) Bildes.
- `UIImagePickerControllerEditedImage` Stellt den vom Benutzer ausgewählten Ausschnitt der bearbeiteten Version des Bildes bereit. Das zurückgegebene `UIImage` ist klein und für die Größe des iPhone-Bildschirms dimensioniert.

Bei der Arbeit mit Firmware der Version 2.x gibt die Delegierungsmethode `imagePickerController: didFinishPickingImage: editingInfo:` als zweites Argument die bearbeitete Version des Bildes in der vom Benutzer gewählten Größe und Position zurück. Das dritte Argument, das Dictionary `editingInfo`, enthält eine Kopie des ursprünglichen Bildes und das Rechteck des Bildausschnitts. *Rezept 7.3* ist mit der 2.x-Firmware kompatibel, da hier das bearbeitete Bild dem Info-Dictionary hinzugefügt und an die Delegierungsmethode der Version 3.x übergeben wird.

Hinweis

Um den iPhone-Simulator mit Bildern zu füllen, wechseln Sie in das Benutzerdateisystem `~/Library/Applicatoin Support/iPhone Simulator/User`. Suchen Sie dort nach dem Ordner `Media/DCIM`, und kopieren Sie einen `100APPLE`-Ordner von einem physischen iPhone dort hinein. Dabei müssen Sie sowohl die JPG-Bilder als auch die THM-Vorschaubilder kopieren. Ab Version 3.x suchen Sie nach `Media/DCIM unterhalb von ~/Library/Application Support/iPhone Simulator/3.x`.

```
- (void)imagePickerController:(UIImagePickerController *)picker
   didFinishPickingMediaWithInfo:(NSDictionary *)info
{
   CFShow(info); // zeigt die info dictionary
   SETIMAGE([info objectForKey:
      @"UIImagePickerControllerEditedImage"]);
   [self dismissModalViewControllerAnimated:YES];
   [picker release];
}

// Sorgt für Kompatibilität mit Version 2.x
- (void)imagePickerController:(UIImagePickerController *)picker
      didFinishPickingImage:(UIImage *)image
      editingInfo:(NSDictionary *)editingInfo
{
   NSMutableDictionary *dict = [NSMutableDictionary
      dictionaryWithDictionary:editingInfo];
   [dict setObject:image
      forKey:@"UIImagePickerControllerEditedImage"];
```

```
    [self imagePickerController:picker
        didFinishPickingMediaWithInfo:dict];
}

- (void) pickImage: (id) sender
{
    // Stellt die Bildauswahl der Fotobibliothek dar
    UIImagePickerController *ipc = [[UIImagePickerController alloc]
        init];
    ipc.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
    ipc.delegate = self;
    ipc.allowsImageEditing = YES; // benutze allowsEditing ab 3.1
    [self presentModalViewController:ipc animated:YES];
}
```

► *Rezept 7.3: Benutzern die Bearbeitung der ausgewählten Bilder erlauben*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.4 Rezept: Fotos aufnehmen und ins Fotoalbum schreiben

In den Rezepten 7.2 und 7.3 haben Sie gesehen, wie Sie Bilder mit dem Bildauswahl-Controller auswählen und bearbeiten. In *Rezept 7.4* lernen Sie eine andere Möglichkeit kennen, nämlich die Aufnahme von Fotos mit der eingebauten Kamera des iPhones. Mit der Bildauswahl können die Benutzer ein Bild aufnehmen und entscheiden, ob sie es verwenden möchten. Da nicht alle iPhone-Modelle über eine Kamera verfügen (im Moment speziell der iPod Touch und das iPad), müssen Sie als Erstes prüfen, ob das System, auf dem die Kamera läuft, die Verwendung der Kamera zulässt. Der folgende Code sucht nach der Kamera und beschränkt den Zugriff auf die Auslöser-Schaltfläche (»Snap«).

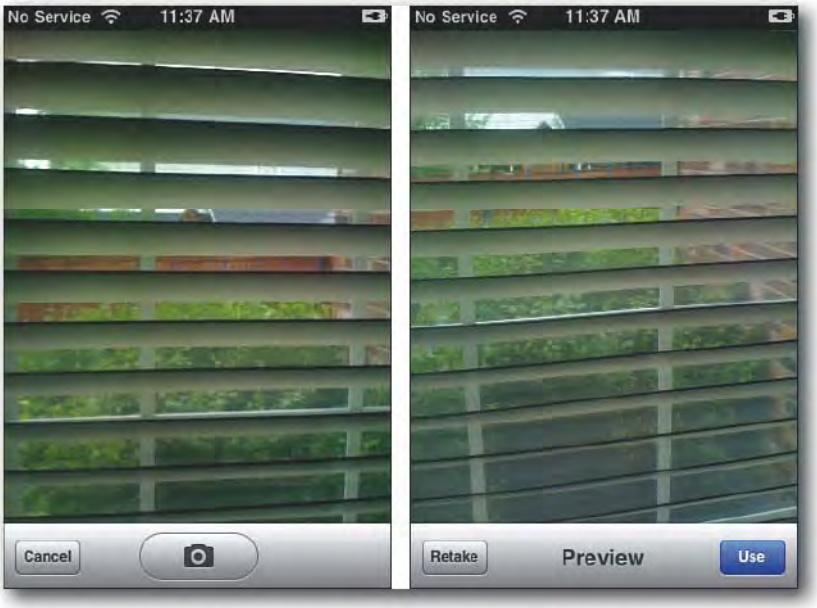
```
if ([UIImagePickerController isSourceTypeAvailable:
    UIImagePickerControllerSourceTypeCamera])
    self.navigationItem.rightBarButtonItem =
        BARBUTTON(@"Snap", @selector(snapImage));
else
    showAlert(CAMERA_NOT_AVAILABLE_STRING);
```

Wie bei den anderen Möglichkeiten können Sie auch hier festlegen, ob im Zuge der Aufnahme eine Bildbearbeitung zulässig sein soll oder nicht. Eine neue Funktion, die durch den Zugriff auf die Kamera möglich wird, ist der Vorschaubildschirm, der angezeigt wird, nachdem der Benutzer auf das Kamerasymbol getippt hat (siehe *Abbildung 7.3*). Auf dem Vorschaubildschirm hat der Benutzer

die Möglichkeit, das Foto erneut aufzunehmen oder es so zu belassen, wie es ist. Beim Antippen von **Use** (oder **Use Photo** unter 2.x) geht die Steuerung in die nächste Phase über. Ist die Bildbearbeitung aktiviert, kann der Benutzer sie als Nächstes vornehmen. Anderenfalls geht die Steuerung an die standardmäßige »Auswahl fertig«-Methode über.

Der Beispielcode zu diesem Rezept weist das zurückgegebene Bild der `UIImageView` zu, die den Hintergrund der Anwendung bildet. Beachten Sie, dass nur ein Teil des Bildes angezeigt wird, da das aufgenommene Foto viel größer ist als der iPhone-Bildschirm. Rezepte für eine Größenänderung solcher Bilder folgen weiter hinten in diesem Kapitel.

Durch den Aufruf von `UIImageWriteToSavedPhotosAlbum()` speichert der Code das aufgenommene Bild im Fotoalbum. Mit dieser Funktion können Sie alle Arten von Bildern speichern, nicht nur diejenigen von der eingebauten Kamera. Als zweites und drittes Argument geben Sie ein Callback-Ziel und einen Selektor an. Der Selektor muss drei Argumente entgegennehmen, wie *Rezept 7.4* zeigt: nämlich ein Bild, ein Fehler-Objekt und einen Zeiger auf Kontextinformationen. Von Anwendungen aufgenommene Fotos enthalten keine Geotagging-Informationen.



▸ *Abbildung 7.3: Nach dem Antippen der Auslöser-Schaltfläche (Kamerasymbol in der linken Abbildung) kann der Benutzer auf dem Vorschaubildschirm auswählen, ob er das Bild verwenden oder erneut aufnehmen möchte.*

```
- (void) snapImage: (id) sender
{
    // Stellt die Kameraoberfläche dar
    UIImagePickerController *ipc = [[UIImagePickerController alloc]
        init];
    ipc.sourceType = UIImagePickerControllerSourceTypeCamera;
    ipc.delegate = self;
```

```
    ipc.allowsImageEditing = NO; // benutze allowsEditing ab 3.1
    [self presentModalViewController:ipc animated:YES];
}

- (void)image:(UIImage *)image didFinishSavingWithError:
      (NSError *)error contextInfo:(void *)contextInfo;
{
    // Kümmert sich um den Abschluss des Schreibvorgangs für das Bild
    if (!error)
        showAlert(@"Image written to photo album");
    else
        showAlert(@"Error writing to photo album: %@",
            [error localizedDescription]);
}

- (void) imagePickerController:(UIImagePickerController *)picker
    didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    // Stellt das aufgenommene Bild wieder her
    UIImage *image = [info
        objectForKey:@"UIImagePickerControllerOriginalImage"];
    SETIMAGE(image);

    // Speichert das Bild im Album
    UIImageWriteToSavedPhotosAlbum(image, self,
        @selector(image:didFinishSavingWithError:contextInfo:), nil);
    [self dismissModalViewControllerAnimated:YES];
    [picker release];
}
```

▶ *Rezept 7.4: Bilder mit der eingebauten Kamera aufnehmen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.5 Rezept: Bilder im Dokumentenordner speichern

Jedes `UIImage` kann in JPEG- oder PNG-Daten umgewandelt werden, wobei die beiden integrierten `UIKit`-Funktionen `UIImageJPEGRepresentation()` und `UIImagePNGRepresentation()` das jeweils notwendige `NSData`-Objekt erstellen. Die JPEG-Version nimmt zwei Argumente an – das Bild und die Komprimierungsrate im Bereich von `0.0` (geringste Qualität, höchste Komprimierung) bis `1.0` (höchste Qualität, geringste Komprimierung) –, die PNG-Version nur eines, nämlich das Bild.

Um das Bild in eine Datei zu schreiben, verwenden Sie das von einer der beiden Funktionen zurückgegebene `NSData`-Objekt und rufen die Methode `writeToFile: atomically:` auf, die die gespeicherten Bilddaten in dem von Ihnen angegebenen Pfad speichert. Wenn Sie das zweite Argument auf `YES` setzen, wird die gesamte Datei geschrieben, bevor sie in dem Pfad platziert wird. Dadurch müssen Sie sich nicht um nur teilweise erfolgreiche Schreibvorgänge kümmern.

Rezept 7.5 verwendet einen Bildauswahl-Controller, um Elemente auszuwählen, die sich bereits in der iPhone-Bibliothek befinden. Der Code speichert das ausgewählte Element im Ordner `Documents` der Anwendungs-Sandbox. Die in diesem Rezept definierte Methode `findUniqueSavePath` gibt einen eindeutigen Namen zurück. Sie sucht so lange, bis sie einen Namen gefunden hat, der noch von keiner vorhandenen Datei verwendet wird. Diesen Namen nimmt die Delegierungsmethode der Picker-Ansicht, um das Bild zu speichern.

Am Ende des Callbacks wird eine Liste der Dateien auf der Debugging-Konsole ausgegeben. Dadurch können Sie erkennen, welche Elemente erstellt wurden, was sehr praktisch ist, wenn Sie dieses Rezept nicht auf dem Simulator, sondern auf einem iPhone-Gerät ausführen.

Das Schreiben von Dateien kann unterschiedlich schnell vonstatten gehen. Auf dem Simulator erfolgt dieser Vorgang sehr schnell, während er auf älteren iPhones der ersten Generation weit langsamer ablaufen kann, vor allem für Bilder in Originalgröße, die mit der Kamera aufgenommen wurden. Die Speicherung eines Fotos kann bis zu fünf oder sogar zehn Sekunden dauern, weshalb es angebracht ist, einen Hinweis auf den fortlaufenden Vorgang einzublenden, z. B. den aus *Rezept 7.11* in diesem Kapitel.

```
// Gibt einen eindeutigen Speicherpfad im Dokumentenordner zurück
- (NSString *) findUniqueSavePath
{
   int i = 1;
   NSString *path;
   do {
      // Iteriert, bis der Name nicht mit dem einer bestehenden Datei
      // übereinstimmt
      path = [NSString stringWithFormat:
         @"%@/Documents/IMAGE_%04d.PNG", NSHomeDirectory(), i++];
   } while ([[NSFileManager defaultManager] fileExistsAtPath:path]);
   return path;
}

- (void)imagePickerController:(UIImagePickerController *)picker
      didFinishPickingMediaWithInfo:(NSDictionary *)info
```

```
{
    // Ruft das ausgewählte Bild ab
    UIImage *image = [info objectForKey:
        @"UIImagePickerControllerOriginalImage"];
    [self dismissModalViewControllerAnimated:YES];
    [picker release];

    // Schreibt das Bild in eine Datei
    [UIImageJPEGRepresentation(image, 1.0f) writeToFile:
        [self findUniqueSavePath] atomically:YES];

    // Legt den Hintergrund fest
    SETIMAGE(image);

    // Zeigt den aktuellen Inhalt des Dokumentenordners
    CFShow([[NSFileManager defaultManager]
        directoryContentsAtPath:[NSHomeDirectory()
        stringByAppendingString:@"/Documents"]]);
}
```

▸ *Rezept 7.5: Bilder in Dateien speichern*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.6 Rezept: Bilder per E-Mail senden

Das neue Framework `Message UI` im SDK 3.0 ermöglicht den Benutzern, E-Mails direkt in Anwendungen zu verfassen. Diese Möglichkeit können Sie Ihrer Anwendung hinzufügen, indem Sie Instanzen von `MFMailComposeViewController` einrichten und initialisieren. *Rezept 7.6* zeigt, wie Sie eine entsprechende Ansicht erstellen und ihren Inhalt initialisieren.

Der E-Mail-Controller funktioniert ähnlich wie der Bildauswahlcontroller. Der Hauptansichtscontroller zeigt ihn als modalen Controller an und wartet auf Ergebnisse über einen Delegierungs-Callback. Achten Sie darauf, dass Sie das Protokoll `MFMailComposeViewControllerDelegate` deklarieren und den Callback implementieren, mit dem der Controller wieder entfernt wird. Dabei müssen Sie der Bildauswahl genügend Zeit einräumen, um herunterzufahren, bevor Sie den E-Mail-Controller anzeigen.

Die größtenteils optionalen Eigenschaften des E-Mail-Controllers verwenden Sie, um die E-Mail-Nachricht zusammenzustellen. Betreff und Text werden mit `setSubject:` und `setMessageBody:` definiert. Diese Methoden nehmen Strings als Argumente entgegen. Der Anhang erfordert jedoch etwas mehr Arbeit.

Um einen Anhang hinzuzufügen, müssen Sie alle Dateikomponenten bereitstellen, die der E-Mail-Client erwartet, also Daten (über ein `NSData`-Objekt), einen MIME-Typ (als String) und einen Dateinamen (ein weiterer String). Rufen Sie die Bilddaten mit der Funktion `UIImageJPEGRepresentation()` aus *Rezept 7.5* ab. Wie in jenem Rezept braucht diese Funktion auch hier etwas Zeit, häufig mehrere Sekunden. Gehen Sie also davon aus, dass es zu einer Verzögerung kommt, bevor die Nachrichtenansicht erscheint.

In diesem Beispiel wird der MIME-Typ `image/jpeg` verwendet. Wenn Sie andere Datentypen senden möchten, suchen Sie im Internet nach den entsprechenden MIME-Darstellungen. Der E-Mail-Client beim Empfänger verwendet den von Ihnen angegebenen Dateinamen, um die gesendeten Daten zu speichern. Dabei können Sie jeden beliebigen Namen nehmen.

```
- (void)mailComposeController:(MFMailComposeViewController*)controller
      didFinishWithResult:(MFMailComposeResult)result
      error:(NSError*)error
{
   // Entfernt den E-Mail-Controller, wenn der Benutzer fertig ist
   [self dismissModalViewControllerAnimated:YES];
}

- (void) emailImage: (UIImage *) image
{
   // Erfordert Version 3.0 oder höher. Legen Sie das zugrunde liegende
   // SDK entsprechend fest.
   if ([MFMailComposeViewController canSendMail])
   {
      // Passt die E-Mail an
      MFMailComposeViewController *mcvc =
         [[[MFMailComposeViewController alloc] init] autorelease];
      mcvc.mailComposeDelegate = self;
      [mcvc setSubject:@"Here's a great photo!"];
      NSString *body = @"<h1>Check this out</h1>\
         <p>I selected this image from the\
         <code><b>UIImagePickerController</b></code>.</p>";
      [mcvc setMessageBody:body isHTML:YES];
      [mcvc addAttachmentData:UIImageJPEGRepresentation(image, 1.0f)
         mimeType:@"image/jpeg" fileName:@"pickerimage.jpg"];

      // Zeigt den E-Mail-Controller an
      [self presentModalViewController:mcvc animated:YES];
   }
}
```

► *Rezept 7.6: Bilder per E-Mail senden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.7 Rezept: Fotos automatisch aufnehmen

Es gibt Gelegenheiten, bei denen die Kamera ohne Benutzereingriff einfach nur einen Schnappschuss machen soll. Denken Sie z. B. an ein Hilfsprogramm, das beim Fahrradfahren in regelmäßigen Zeitabständen automatisch ein Bild aufnimmt, oder an eine Anwendung, die eine Stop-Motion-Animation erstellt. *Rezept 7.7* zeigt, wie Sie so etwas mit den neuen Funktionen des SDK Version 3.1 mit der Kamera des `UIImagePickerController` erzielen.

Zwei Änderungen der API in Version 3.1 ermöglichen Aufnahmen dieser Art. Mit der Eigenschaft `showsCameraControls` können Sie die normale Kamera-Benutzeroberfläche ausblenden und stattdessen eine bildschirmfüllende Kameravorschau anzeigen. Setzen Sie diese Eigenschaft auf `NO`.

```
ipc.showsCameraControls = NO;
```

Um ein Bild programmgesteuert aufzunehmen statt über eine Benutzereingabe, rufen Sie die Methode `takePicture` auf. Dadurch beginnt der Aufnahmevorgang gerade so, als hätte der Benutzer auf den Auslöser gedrückt. Ist das Foto fertig, sendet der Picker den Callback `imagePickerController:didFinishPickingMediaWithInfo:` an seine Delegierung. Ein weiteres Foto können Sie erst nach dem Aufruf dieser Methode aufnehmen.

Der Code von *Rezept 7.7* nimmt eine Folge von drei Fotos auf, eines nach dem anderen. Jedes Bild wird im Fotoalbum gespeichert, danach wird das nächste geschossen. Alle Bilder sind Fotos mit hoher Auflösung, die jeweils zwei oder drei Megabyte an Speicher einnehmen. Um die Zeitabstände zwischen den Aufnahmen zu vergrößern, können Sie einfach einen Timer einfügen.

Wenn Sie das iPhone in einem Dock verwenden oder über einen längeren Zeitraum hinweg Fotos aufnehmen, müssen Sie wie folgt den Leerlauftimer von `UIApplication` deaktivieren. Der folgende Code sorgt dafür, dass das Gerät nicht in den Ruhezustand verfällt, auch wenn es für längere Zeit keine Benutzerinteraktion registriert hat.

```
[UIApplication sharedApplication].idleTimerDisabled = YES;
```

Hinweis

Kombinieren Sie die zeitgesteuerte Fotoaufnahme aus *Rezept 7.7* mit dem Twitpic-Uploader aus Rezept 13.11, um ein ansonsten nicht genutztes iPhone als Überwachungskamera einzusetzen.

```
@implementation TestBedViewController
- (void)image:(UIImage *)image
   didFinishSavingWithError:(NSError *)error
   contextInfo:(void *)contextInfo;
{
   // Regiert auf das Ergebnis der Dateispeicherung
   if (!error)
      NSLog(@"Image written to photo album");
   else
      NSLog(@"Error writing to photo album: %@",
         [error localizedDescription]);

   // Nimmt drei Fotos auf und hält dann an
   if (count++ == 3)
   {
      [self dismissModalViewControllerAnimated:YES];
      [ipc release];
      ipc = nil;
   }
   else [ipc takePicture];
}

- (void) imagePickerController:(UIImagePickerController *)picker
   didFinishPickingMediaWithInfo:(NSDictionary *)info
{
   // Speichert den Schnappschuss im Fotoalbum
   UIImage *image = [info objectForKey:
      @"UIImagePickerControllerOriginalImage"];
   UIImageWriteToSavedPhotosAlbum(image, self,
      @selector(image:didFinishSavingWithError:contextInfo:), nil);
}

- (void) snapImage: (id) sender
{
   count = 0; // wird 3 Fotos machen

   // Initialisiert die Bildauswahl
   ipc = [[UIImagePickerController alloc] init];
   ipc.sourceType = UIImagePickerControllerSourceTypeCamera;
   ipc.delegate = self;
   ipc.allowsEditing = NO;
   ipc.showsCameraControls = NO;
   [self presentModalViewController:ipc animated:YES];
   // Wartet auf die Einrichtung der Kamera und nimmt dann ein Bild auf
   [NSTimer scheduledTimerWithTimeInterval:2.0f target:ipc
```

```
        selector:@selector(takePicture) userInfo:nil repeats:NO];
}

- (void) viewDidLoad
{
    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
    if ([UIImagePickerController isSourceTypeAvailable:
        UIImagePickerControllerSourceTypeCamera])
        self.navigationItem.rightBarButtonItem =
            BARBUTTON(@"Snap", @selector(snapImage:));
    else
        showAlert(@"This demo relies on camera access.");
    self.title = @"Image Picker";
}
@end
```

► *Rezept 7.7: Fotos automatisch aufnehmen. Für dieses Rezept benötigen Sie iPhone SDK 3.1*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.8 Rezept: Eine benutzerdefinierte Kameraeinblendung verwenden

Mit der Firmware der Version 3.1 können Sie jetzt benutzerdefinierte Einblendungen zu Kameraoberflächen hinzufügen. Damit können Sie eine grafische Benutzeroberfläche erstellen, die über der aktuellen Kameravorschau schwimmt und Schaltflächen sowie andere Steuerelemente zur Aufnahme von Bildern und zum Beenden des Controllers enthält. *Abbildung 7.4* zeigt eine rudimentäre Einblendung dieser Art mit zwei Schaltflächen: einer zur Aufnahme eines Fotos und einer zum Entfernen des Bildauswahl-Controllers (bei Letzterer handelt es sich um den Kreis mit dem X).

Der hellgraue Balken hinter der Schaltfläche **Snap** wurde bei der Gestaltung der Einblendung in Interface Builder hinzugefügt. In *Abbildung 7.4* befindet sich dieser Balken teilweise im Bildbereich und teilweise im schwarzen Steuerbereich, der für Ihre eigenen Zwecke leer gelassen wurde.

Die Einblendung richten Sie ein, indem Sie der Eigenschaft `cameraOverlayView` des Pickers eine Ansicht zuweisen und die normalen Steuerelemente ausblenden. Wenn Sie den Picker anzeigen, erscheint die benutzerdefinierte Einblendung, nicht die eingebaute.

Ein weiteres Merkmal der Version 3.1, die Eigenschaft `cameraViewTransform`, bietet eine Möglichkeit, um die Darstellung der Kameraansicht zu ändern. In *Rezept 7.8* wird diese Eigenschaft verwendet, um die Vorschau rotieren zu lassen, während ein Bild gespeichert wird. Im normalen Gebrauch wäre diese Eigenschaft sehr praktisch für Videokonferenzen, sollte Apple jemals eine vorn montierte Kamera in einem iPhone oder iPod anbringen.

Rezept 7.8 stellt diese beiden Merkmale vor und zeigt, wie Sie sie in Ihren iPhone-Anwendungen nutzen können.

▶ *Abbildung 7.4: Fotos mit einer benutzerdefinierten Bildauswahl-Einblendung aufnehmen. Für dieses Rezept benötigen Sie iPhone SDK 3.1*

```
@implementation TestBedViewController
- (void)image:(UIImage *)image didFinishSavingWithError:
   (NSError *)error contextInfo:(void *)contextInfo;
{
   // Reagiert auf erfolgreiche Dateispeicherung
   if (!error)
      NSLog(@"Image written to photo album");
   else
      NSLog(@"Error writing to photo album: %@",
         [error localizedDescription]);

   // Stellt den Standard des Picker-Controllers wieder her
   overlay.alpha = 1.0f;
   [timer invalidate];
   ipc.cameraViewTransform = CGAffineTransformIdentity;
}
```

```
- (void) imagePickerController:(UIImagePickerController *)picker
   didFinishPickingMediaWithInfo:(NSDictionary *)info
{
   // Ruft das Bild ab und speichert es
   UIImage *image = [info objectForKey:
      @"UIImagePickerControllerOriginalImage"];
   UIImageWriteToSavedPhotosAlbum(image, self,
      @selector(image:didFinishSavingWithError:contextInfo:), nil);
}

- (void) rotate
{
   // Dreht die Kameraansicht
   ipc.cameraViewTransform =
      CGAffineTransformMakeRotation(2.0f*M_PI*((float)count/100.0f));
   count = (count + 10) % 100;
}

- (void) snap: (id) sender
{
   // Bereitet die Aufnahme eines Fotos vor
   overlay.alpha = 0.0f;
   [ipc takePicture];
   count = 0;
   timer = [NSTimer scheduledTimerWithTimeInterval:0.1f
      target:self selector:@selector(rotate) userInfo:nil
      repeats:YES];
}

- (void) dismiss: (id) sender
{
   // Entfernt die Bildauswahl-Oberfläche
   [self dismissModalViewControllerAnimated:YES];
   [ipc release];
   ipc = nil;
}

- (void) takePics: (id) sender
{
   // Erstellt die Bildauswahl-Oberfläche und zeigt sie an
   ipc = [[UIImagePickerController alloc] init];
   ipc.sourceType = UIImagePickerControllerSourceTypeCamera;
   ipc.delegate = self;
   ipc.allowsEditing = NO;
   ipc.showsCameraControls = NO;
```

```
    ipc.cameraOverlayView = overlay;
    [self presentModalViewController:ipc animated:YES];
}

- (void) viewDidLoad
{

    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
    if ([UIImagePickerController
        isSourceTypeAvailable:
        UIImagePickerControllerSourceTypeCamera])
            self.navigationItem.rightBarButtonItem =
                BARBUTTON(@"Camera", @selector(takePics:));
    else
        showAlert(@"This demo relies on camera access.");
    self.title = @"Image Picker";
}
@end
```

▸ *Rezept 7.8: Benutzerdefinierte Kameraeinblendungen und -transformationen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.9 Rezept: Bilder in einer scrollbaren Ansicht anzeigen

Bei der Anzeige von Bildern geht es vor allem um den Arbeitsspeicher, weshalb Sie die Darstellung von großen und von kleinen Bildern als zwei getrennte Probleme betrachten sollten. Die Klasse `UIWebView` kann speicherintensive Daten leicht handhaben. Mit einer Methode wie der folgenden können Sie z. B. ein großes Bild in eine Webansicht laden. Dieser Ansatz eignet sich sehr gut für sperrige PDF-Bilder. `UIWebView` ist ein Komplettpaket für die Bilddarstellung einschließlich Rollmöglichkeit und Größenänderung.

```
- (void) loadImageIntoWebView: (NSString *) path
{
    // Passt das Bild automatisch in die Ansicht ein
    self.webView.scalesPageToFit = YES;
```

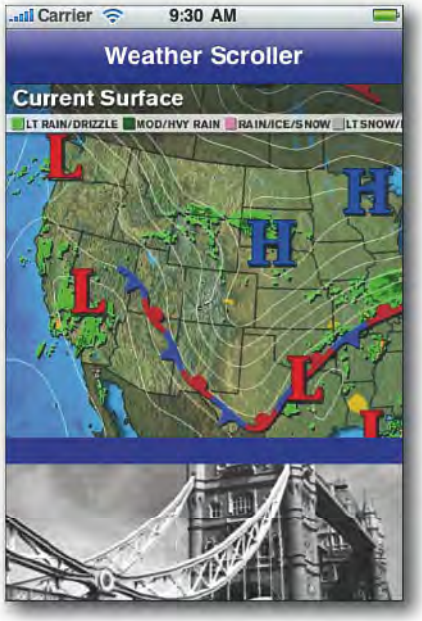
```
    // Lädt das Bild durch Erstellen einer Anforderung
    NSURL *fileURL = [NSURL fileURLWithPath:path];
    NSURLRequest *request = [NSURLRequest requestWithURL:fileURL];
    [self.webView loadRequest:request];
}
```

Kleinere Bilder – etwa von einem halben Megabyte Größe in komprimiertem Zustand – können Sie direkt in `UIImageViews` laden und zur Oberfläche hinzufügen. Aufgrund von Speichereinschränkungen empfiehlt Apple, bei `UIImage`-Bildern nicht über 1024 × 1024 Pixel hinauszugehen.

Das Problem bei einfachen Bildansichten besteht darin, dass sie statisch sind. Anders als Webansichten reagieren sie nicht auf Roll- und Kneifgesten. Dieses Manko können Sie beheben, indem Sie sie in eine `UIScrollView` einbetten. In solchen Rollansichten sind die genannten Möglichkeiten zur Benutzerinteraktion gegeben.

Rezept 7.9 zeigt, wie Sie dabei vorgehen. Der Code fügt der Oberfläche eine Rollansicht und dieser eine Wetterkarte hinzu, wie *Abbildung 7.9* zeigt. Anschließend wird aufgrund der Bildgröße der Mindest-Zoomfaktor berechnet, der erforderlich ist, um das Bild in der Ansicht vollständig zu zeigen. Dieser Wert wird der Eigenschaft `minimumZoomScale` der Ansicht zugewiesen. Der maximale Zoomfaktor wird willkürlich auf den dreifachen Wert der Bildgröße gesetzt. Diese Einstellungen erlauben dem Benutzer eine vollständige Interaktion mit dem Bild und schränken sie gleichzeitig auf einen sinnvollen Bereich ein.

Die in dem Rezept gezeigte Delegierungsmethode gibt an, welche Ansicht auf den Zoom reagieren soll. In diesem Rezept ist dies die einzelne Bildansicht in der Rollansicht. Rollansichten wissen von vornherein nichts über die Unteransichten, die Sie ihnen hinzufügen. Durch die Definition dieser Delegierungsmethode binden Sie den Zoom an Ihr Bild.



▸ *Abbildung 7.5: Diese aktuelle Wetterkarte wird von einem URL im Web heruntergeladen und mit einer Rollansicht überlagert, die es dem Benutzer erlaubt, die Größe des Bildes zu ändern und sich durch die Anzeige zu bewegen.*

Hinweis

Wie bei anderen Ansichten können Sie auch die Eigenschaften einer `UIImageView` in Interface Builder (IB) ändern. Im Informationsbereich von IB können Sie die Alpha-Einstellung, die Größe, die Position usw. der Ansicht festlegen. Dabei gibt es jedoch eine Eigentümlichkeit. Wenn Sie eine Bildansicht als Hauptansicht verwenden, hindert IB Sie daran, Unteransichten hinzuzufügen. In diesem Fall müssen Sie eine andere Ansicht als Hauptansicht erstellen und die Bildansicht entsprechend bearbeiten. Anschließend löschen Sie die Hauptansicht. Ziehen Sie die bearbeitete Bildansicht bei gedrückter Ctrl-Taste von Ihrer Anwendungsdelegierung, und weisen Sie sie dem Outlet der Ansicht zu.

```
@implementation TestBedViewController
@synthesize weathermap;

- (UIView *)viewForZoomingInScrollView:(UIScrollView *)scrollView
{
   return [self.view viewWithTag:201];
}

- (void) viewDidLoad
{
   self.navigationController.navigationBar.tintColor =
      COOKBOOK_PURPLE_COLOR;
   self.weathermap = URLIMAGE(MAP_URL);
   self.title = @"Weather Scroller";
   // Erstellt die Rollansicht und legt deren Inhaltsgröße und
   // Delegierung fest
   UIScrollView *sv = [[[UIScrollView alloc]
      initWithFrame:CGRectMake(0.0f, 0.0f, 320.0f, 284.0f)]
      autorelease];
   sv.contentSize = self.weathermap.size;
   sv.delegate = self;

   // Erstellt eine Bildansicht und fügt sie der Rollansicht hinzu

   UIImageView *iv = [[[UIImageView alloc]
      initWithImage:self.weathermap] autorelease];
   iv.userInteractionEnabled = YES;
   iv.tag = 201;
```

```
    // Berechnet die Zoomfaktoren und legt sie fest
    float minzoomx = sv.frame.size.width/self.weathermap.size.width;
    float minzoomy = sv.frame.size.height/self.weathermap.size.height;
    sv.minimumZoomScale = MIN(minzoomx, minzoomy);
    sv.maximumZoomScale = 3.0f;

    // Fügt die Unteransichten hinzu
    [sv addSubview:iv];
    [self.view addSubview:sv];
}
@end
```

▸ *Rezept 7.9: Ein Bild in eine Rollansicht einbetten*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.10 Rezept: Eine mehrseitige Rollansicht mit mehreren Bildern erstellen

Die Möglichkeiten von Rollansichten sind mit dem Zoomen nicht erschöpft. Mit der Seiteneigenschaft von `UIScrollView` können Sie Bilder (und andere Ansichten) in einer Rollansicht platzieren und sie Bild für Bild durchblättern. Entscheidend ist hierbei, dass jedes geladene Bild bei einer horizontalen Darstellung genau die Breite des Rollansichtsrahmens aufweist bzw. bei einer vertikalen Darstellung genau die Höhe.

Setzen Sie die Eigenschaft `pagingEnabled` auf `YES`. Dadurch können die Benutzer Bild für Bild durch die Ansicht blättern. *Rezept 7.10* zeigt, wie Sie dies tun. Dadurch erhalten Sie eine seitenweise Darstellung mehrerer Bildansichten. Mit diesem Ansatz können Sie auch andere Ansichten darstellen, bei denen es sich nicht um Bilder handelt.

Hinweis

Zoom zu einer mehrseitigen Ansicht hinzuzufügen, ist schwieriger als das hier gezeigte einfache Rollverhalten. Dieses Problem wurde geschickt und umfassend von Joe Hewitt gelöst, dem Entwickler der Facebook-Anwendung für das iPhone. Sein Open-Source-Projekt three20 (*http://github.com/joehewitt/three20*) bietet Interaktionsmöglichkeiten für Fotoalben innerhalb einer mehrseitigen Rollansicht, darunter auch einen Zoom. Dieses Projekt ist eine Fundgrube an Hilfsklassen für nützliche und schöne Ansichten.

```
- (void) viewDidLoad
{
   self.navigationController.navigationBar.tintColor =
      COOKBOOK_PURPLE_COLOR;
   self.title = @"Image Scroller";

   // Erstellt die Rollansicht und legt die Inhaltsgröße und die
   // Delegierung fest
   UIScrollView *sv = [[[UIScrollView alloc]
      initWithFrame:CGRectMake(0.0f, 0.0f, 320.0f, BASEHEIGHT)]
      autorelease];
   sv.contentSize = CGSizeMake(NPAGES * 320.0f, sv.frame.size.height);
   sv.pagingEnabled = YES;
   sv.delegate = self;

   // Lädt alle Seiten
   for (int i = 0; i < NPAGES; i++)
   {
      NSString *filename = [NSString stringWithFormat:@"image%d.png",
                            i+1];
      UIImageView *iv = [[UIImageView alloc] initWithImage:
                         [UIImage imageNamed:filename]];
      iv.frame = CGRectMake(i * 320.0f, 0.0f, 320.0f, BASEHEIGHT);
      [sv addSubview:iv];
      [iv release];
   }

   [self.view addSubview:sv];
}
```

► *Rezept 7.10: Eine mehrseitige Bilddarstellung anlegen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.11 Rezept: Neue Bilder erstellen

Bilder können Sie nicht nur aus einer Datei oder aus dem Web laden, sondern mit Cocoa Touch auch im laufenden Betrieb erstellen. Hierzu werden die `UIKit`-Funktionen mit standardmäßigen Quartz 2D-Grafiken kombiniert, um neue `UIImage`-Instanzen anzulegen.

Wieso sollten Sie so etwas tun? Dafür gibt es viele Gründe. Beispielsweise können Sie damit ein Vorschaubild erstellen, indem Sie ein großes Bild schrumpfen und daraus ein neues Bild machen. Sie können auch programmgesteuert ein beschriftetes Spielfeld zeichnen, einen halbtransparenten Hintergrund für benutzerdefinierte Warnhinweise anlegen, bestehende Bilder um Effekte wie die in Kapitel 6, *Ansichten und Animationen zusammenstellen*, erwähnte Reflexion bereichern oder ein Bild auf irgendeine andere Weise anpassen. Bei jedem dieser Vorgänge erstellen Sie im Code ein neues Bild, das entweder von einem anderen abgeleitet ist oder vollständig aus neuen Elementen besteht.

Mit Cocoa Touch können Sie neue Bilder auf einfache Weise erstellen. Wie der folgende Code zeigt, legen Sie einfach einen neuen Bildkontext an, zeichnen darin und wandeln den Kontext dann in ein `UIImage`-Objekt um.

```
UIGraphicsBeginImageContext(CGSizeMake(40.0f, 40.0f));
CGContextRef context = UIGraphicsGetCurrentContext();

// Hier zeichnen Sie im Kontext

UIImage *theImage = UIGraphicsGetImageFromCurrentImageContext();
UIGraphicsEndImageContext();
```

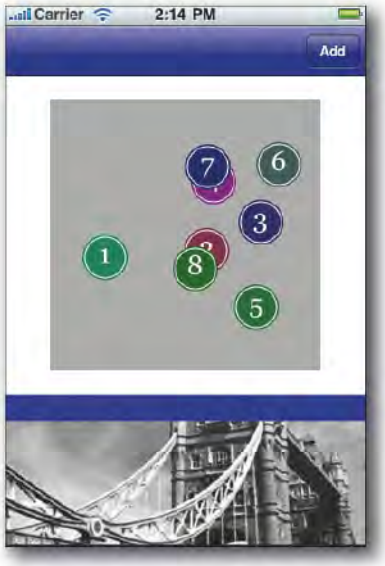
Die Zeichenbefehle, die Sie verwenden, können `UIKit`-Aufrufe (wie `drawAtPoint:` oder `drawInRect:`) und Quartz-Aufrufe in Core Graphics wie die aus *Rezept 7.11* umfassen. Der Code aus *Rezept 7.11* erstellt neue Bildansichten und füllt sie mit Bildern, die von Grund auf neu gezeichnet werden. Wie *Abbildung 7.6* zeigt, handelt es sich bei diesen Bildern um Kreise mit Zufallsfarben und einer Nummerierung. Die Zahlen werden direkt in die Bilder gezeichnet und nicht mit einem zusätzlichen `UILabel` hinzugefügt.

```
// Zeichnet zentrierten Text in den Kontext
void centerText(CGContextRef context, NSString *fontname,
   float textsize, NSString *text, CGPoint point, UIColor *color)
{
   CGContextSaveGState(context);
   CGContextSelectFont(context, [fontname UTF8String], 24.0f,
      kCGEncodingMacRoman);

   // Ruft die Textbreite ab, ohne etwas zu zeichnen
   CGContextSaveGState(context);
   CGContextSetTextDrawingMode(context, kCGTextInvisible);
   CGContextShowTextAtPoint(context, 0.0f, 0.0f, [text UTF8String],
      text.length);
   CGPoint endpoint = CGContextGetTextPosition(context);
```

```
    CFShow(NSStringFromCGPoint(endpoint));
    CGContextRestoreGState(context);

    // Fragt die Größe ab, um die Höhe wiederherzustellen. Die Breite ist
    // weniger zuverlässig.
    CGSize stringSize = [text sizeWithFont:
        [UIFont fontWithName:fontname size:textsize]];

    // Zeichnet den Text
    CGContextSetShouldAntialias(context, true);
    CGContextSetTextDrawingMode(context, kCGTextFill);
    CGContextSetFillColorWithColor(context, [color CGColor]);
    CGContextSetTextMatrix (context,
        CGAffineTransformMake(1, 0, 0, -1, 0, 0));
    CGContextShowTextAtPoint(context, point.x - endpoint.x / 2.0f,
        point.y + stringSize.height / 3.0f, [text UTF8String],
        text.length);
    CGContextRestoreGState(context);
}

- (UIImage *) createImageWithColor: (UIColor *) color
{
    // Erstellt einen neuen Bildkontext der Größe 40x40
    UIGraphicsBeginImageContext(CGSizeMake(40.0f, 40.0f));
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Zeichnet einen gefüllten Kreis
    CGContextSetFillColorWithColor(context, [color CGColor]);
    CGContextAddEllipseInRect(context,
        CGRectMake(0.0f, 0.0f, 40.0f, 40.0f));
    CGContextFillPath(context);
    CGContextClip(context);

    // Versieht den Kreis mit einer Nummer
    CGContextSetFillColorWithColor(context,
        [[UIColor whiteColor] CGColor]);
    NSString *numstring = [NSString stringWithFormat:@"%d", count++];
    centerText(context, @"Georgia", 18.0f, numstring,
        CGPointMake(20.0f, 20.0f), [UIColor whiteColor]);

    // Umrandet den Kreis mit einer 2 Pixel nach innen gerückten Linie
    CGContextSetStrokeColorWithColor(context,
        [[UIColor whiteColor] CGColor]);
    CGContextAddEllipseInRect(context,
        CGRectMake(2.0f, 2.0f, 36.0f, 36.0f));
    CGContextStrokePath(context);
```

```
    // Gibt das Bild zurück
    UIImage *theImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return theImage;
}

- (void) add: (id) sender
{
    // Färbt den Kreis mit einer Zufallsfarbe
    CGFloat red = (random() % 128) / 256.0f;
    CGFloat green = (random() % 128) / 256.0f;
    CGFloat blue = (random() % 128) / 256.0f;
    UIColor *color = [UIColor colorWithRed:red green:green
        blue:blue alpha:1.0f];

    // Fordert das neue Bild an und platziert es in einer UIImageView
    UIImage *newimage = [self createImageWithColor:color];
    UIImageView *newview = [[UIImageView alloc]
        initWithImage:newimage];

    // Platziert die Bildansicht zufällig
    newview.center = [newview randomCenterInView:
        [self.view viewWithTag:101] withInset:0];
    [[self.view viewWithTag:101] addSubview:newview];
    [newview release];

}
```

▶ *Rezept 7.11: Neue* `UIImage`*-Instanzen erstellen*



▶ *Abbildung 7.6: Alle Kreise sind vollständig mit Core Graphics/ Quartz-Aufrufen erstellte Bilder*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.12 Rezept: Vorschaubilder aus Bildern erstellen

In jeder Anwendung, in der Bilder verwendet werden, spielen Vorschaubilder eine wichtige Rolle. Häufig müssen Sie die Größe eines Bildes ändern, damit es in einem begrenzten Raum Platz hat. Natürlich könnten Sie eine `UIImageView` mit dem vollständigen Original laden und dann ihren Rahmen verkleinern, aber es lässt sich eine Menge Arbeitsspeicher einsparen, wenn Sie das Bild mit weniger Bytes neu zeichnen. Für Vorschaubilder können Sie drei verschiedene Ansätze verwenden, die alle in *Rezept 7.12* vorgeführt werden:

- Sie können die Größe des Bildes unter Beibehaltung der Seitenverhältnisse ändern. Je nach den Proportionen des Originalbildes müssen Sie überschüssige Gebiete mit horizontalen oder vertikalen Balken versehen und das Bild an den betreffenden Stellen mit transparenten Pixeln maskieren.
- Sie können einen Teil des Bildes ausschneiden, sodass es in den vorhandenen Platz passt. Bei dem Beispiel in *Abbildung 7.7* wird ein zentriertes Teilbild angezeigt. Elemente außerhalb des Bildbereichs werden abgeschnitten.
- Sie können das Bild stauchen, indem Sie seine Höhe und Breite an den vorhandenen Platz anpassen. Alle Pixel werden verwendet, das Bild wird jedoch entweder horizontal oder vertikal beschnitten. Dies ähnelt der Vollbildwiedergabe von Filmen auf Fernsehern ohne Breitbild, bei denen an den Seitenrändern Einzelheiten verloren gehen.

▶ *Abbildung 7.7: Diese Screenshots zeigen die drei Möglichkeiten, um Vorschaubilder zu erstellen. Beim Einpassen (links) wird das ursprüngliche Seitenverhältnis beibehalten und das Bild an den Rändern nach Bedarf mit horizontalen oder vertikalen Balken versehen. Beim Zentrieren (Mitte) verwenden Sie die ursprünglichen Bildpixel und schneiden die Mitte aus. Beim Stauchen (rechts) verwenden Sie jeden verfügbaren Pixel und schneiden nur die Teile ab, die außerhalb des Rahmens fallen.*

7.12 Rezept: Vorschaubilder aus Bildern erstellen

Rezept 7.12 zeigt, wie Sie diese drei Arten von Vorschaubildern erstellen. Den Methoden in diesem Code übergeben Sie ein Bild und eine Größe, woraufhin sie ein neues eingepasstes, zentriertes oder gestauchtes Vorschaubild zurückgeben.

```
// Berechnet eine Größe, die unter Beibehaltung des ursprünglichen
// Seitenverhältnisses in eine andere Größe passt
+ (CGSize) fitSize: (CGSize)thisSize inSize: (CGSize) aSize
{
   CGFloat scale;
   CGSize newsize = thisSize;

   if (newsize.height && (newsize.height > aSize.height))
   {
      scale = aSize.height / newsize.height;
      newsize.width *= scale;
      newsize.height *= scale;
   }

   if (newsize.width && (newsize.width >= aSize.width))
   {
      scale = aSize.width / newsize.width;
      newsize.width *= scale;
      newsize.height *= scale;
   }

   return newsize;
}

// Ändert die Größe proportional und passt das Bild vollständig und ohne
// Beschnitt in die Ansicht ein
+ (UIImage *) image: (UIImage *) image fitInSize: (CGSize) viewsize
{
   // Berechnet die Einpassgröße
   CGSize size = [ImageHelper fitSize:image.size inSize:viewsize];

   UIGraphicsBeginImageContext(viewsize);

   // Berechnet eine möglicherweise erforderliche Maskierung zum Einpassen
   float dwidth = (viewsize.width - size.width) / 2.0f;
   float dheight = (viewsize.height - size.height) / 2.0f;

   CGRect rect = CGRectMake(dwidth, dheight, size.width, size.height);
   [image drawInRect:rect];
```

```
    UIImage *newimg = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();

    return newimg;
}

// Keine Größenänderung, aber möglicherweise Beschnitt
+ (UIImage *) image: (UIImage *) image centerInSize: (CGSize) viewsize
{
    CGSize size = image.size;

    UIGraphicsBeginImageContext(viewsize);

    // Berechnet den Versatz, damit der Bildmittelpunkt im Mittelpunkt der
    // Ansicht liegt
    float dwidth = (viewsize.width - size.width) / 2.0f;
    float dheight = (viewsize.height - size.height) / 2.0f;

    CGRect rect = CGRectMake(dwidth, dheight, size.width, size.height);
    [image drawInRect:rect];

    UIImage *newimg = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();

    return newimg;
}
// Füllt jeden Pixel der Ansicht ohne schwarze Ränder, ändert die Größe
// und schneidet bei Bedarf Überschuss ab
+ (UIImage *) image: (UIImage *) image fillSize: (CGSize) viewsize
{
    CGSize size = image.size;

    // Wählt den geringstmöglichen Verkleinerungsfaktor
    CGFloat scalex = viewsize.width / size.width;
    CGFloat scaley = viewsize.height / size.height;
    CGFloat scale = MAX(scalex, scaley);

    UIGraphicsBeginImageContext(viewsize);

    CGFloat width = size.width * scale;
    CGFloat height = size.height * scale;

    // Zentriert das verkleinerte Bild
    float dwidth = ((viewsize.width - width) / 2.0f);
    float dheight = ((viewsize.height - height) / 2.0f);
```

```
    CGRect rect = CGRectMake(dwidth, dheight,
       size.width * scale,
       size.height * scale);
    [image drawInRect:rect];

    UIImage *newimg = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();

    return newimg;
}
```

► *Rezept 7.12: Vorschaubilder erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.

7.13 Die Orientierung von Fotos korrigieren

Jedes mit einer Digitalkamera geschossene Foto kann mit einem internen Orientierungswert versehen sein, der angibt, wie die Kamera bei der Aufnahme gehalten wurde. Wenn der Benutzer die Kamera beispielsweise nach links oder rechts gedreht hat, um ein Bild im Breitformat aufzunehmen, können die mit dem Foto gespeicherten EXIF-Metadaten eine Eigenschaft enthalten, die diese Orientierung angibt. Die Klasse `UIImage` liest diese Metadaten zusammen mit dem Bild und verwendet sie, um ihre Eigenschaft `imageOrientation` festzulegen.

Cocoa Touch kann mit acht Arten von `UIImageOrientation`-Werten umgehen, die den Richtungen oben, unten, links und rechts sowie jeweils den gespiegelten Versionen dieser Ausrichtungen entsprechen:

- `UIImageOrientationUp`
- `UIImageOrientationLeft`
- `UIImageOrientationRight`
- `UIImageOrientationDown`
- `UIImageOrientationUpMirrored`
- `UIImageOrientationLeftMirrored`
- `UIImageOrientationRightMirrored`
- `UIImageOrientationDownMirrored`

Gespiegelte Bilder werden gewöhnlich mit Webcams aufgenommen, deren Software das Bild automatisch umkehrt. Wenn Sie sich eine Webcam-Liveaufnahme von sich selbst ansehen, erscheint Ihnen das gespiegelte Bild vertrauter.

Dieser Umstand kann sehr wichtig sein, wenn Sie Bilder aus Dateien ohne Rücksicht auf die Orientierung in Bytereihenfolge laden. Ein Bild mit einer alternativen Ausrichtung kann dann gekippt, horizontal oder vertikal gespiegelt in eine Bitmap geladen werden. Durch die Korrektur der Bildorientierung stellen Sie sicher, dass das angezeigte Bild dem entspricht, was der Fotograf gesehen hat.

Listing 7.1 zeigt, wie Sie eine nicht gedrehte Version eines `UIImage` zurückgeben. Hierzu wird die Eigenschaft `imageOrientation` abgerufen und das Bild in einen Grafikkontext gezeichnet, der so transformiert wurde, dass seine Eigenschaften mit den ursprünglichen Eigenschaften bei der Aufnahme übereinstimmen. Diesen Vorgang müssen Sie nur selten durchführen, nämlich wenn Sie direkt mit Bits umgehen. Um die meisten Orientierungsprobleme kümmert sich schon die Klasse `UIImage`.

```
// Hilfsmakros für die Orientierung
#define MIRRORED ((image.imageOrientation ==
   UIImageOrientationUpMirrored) || (image.imageOrientation ==
   UIImageOrientationLeftMirrored) || (image.imageOrientation ==
   UIImageOrientationRightMirrored) || (image.imageOrientation ==
   UIImageOrientationDownMirrored))
#define ROTATED90 ((image.imageOrientation ==
   UIImageOrientationLeft) || (image.imageOrientation ==
   UIImageOrientationLeftMirrored) || (image.imageOrientation ==
   UIImageOrientationRight) || (image.imageOrientation ==
   UIImageOrientationRightMirrored))

// Gibt eine nicht gedrehte Version des Bildes zurück
+ (UIImage *) doUnrotateImage: (UIImage *) image
{
   CGSize size = image.size;
   if (ROTATED90) size = CGSizeMake(image.size.height,
                                    image.size.width);

   UIGraphicsBeginImageContext(size);
   CGContextRef context = UIGraphicsGetCurrentContext();
   CGAffineTransform transform = CGAffineTransformIdentity;

   // Dreht das Bild nach Bedarf
   switch(image.imageOrientation)
   {
      case UIImageOrientationLeft:
      case UIImageOrientationRightMirrored:
         transform = CGAffineTransformRotate(transform,
                                             M_PI / 2.0f);
```

```
            transform = CGAffineTransformTranslate(transform,
                                                   0.0f, -size.width);
            size = CGSizeMake(size.height, size.width);
            CGContextConcatCTM(context, transform);
            break;
        case UIImageOrientationRight:
        case UIImageOrientationLeftMirrored:
            transform = CGAffineTransformRotate(transform,
                -M_PI / 2.0f);
            transform = CGAffineTransformTranslate(transform,
                -size.height, 0.0f);
            size = CGSizeMake(size.height, size.width);
            CGContextConcatCTM(context, transform);
            break;
        case UIImageOrientationDown:
        case UIImageOrientationDownMirrored:
            transform = CGAffineTransformRotate(transform, M_PI);
            transform = CGAffineTransformTranslate(transform,
                -size.width, -size.height);
            CGContextConcatCTM(context, transform);
            break;
        default:
            break;
    }

    if (MIRRORED)
    {
        // Hebt die Spiegelung auf
        transform = CGAffineTransformMakeTranslation(size.width, 0.0f);
        transform = CGAffineTransformScale(transform, -1.0f, 1.0f);
        CGContextConcatCTM(context, transform);
    }

    // Zeichnet das Bild in den transformierten Kontext und gibt es zurück
    [image drawAtPoint:CGPointMake(0.0f, 0.0f)];
    UIImage *newimg = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return newimg;
}
```

▶ *Listing 7.1: Die Drehung von* `UIImage`*-Instanzen unterbinden*

7.13.1 Testbilder hinzufügen

Sie können ganz einfach mit der eingebauten iPhone-Kamera eine Reihe von Testbildern mit den vier Hauptorientierungen (links, rechts, oben, unten) aufnehmen und zum Simulator hinzufügen, indem Sie sie in den Ordner `Library/Application Support/iPhone Simulator/User/Media/DCIM/100APPLE/` in Ihrem Benutzerordner kopieren. Den Unterordner `DCIM/100APPLE` müssen Sie dabei selbst erstellen.

7.14 Screenshots aufnehmen

Wie *Listing 7.2* zeigt, können Sie Ansichten in Bildkontexten zeichnen und diese Kontexte in `UIImage`-Instanzen umwandeln. Dieser Code verwendet den Aufruf `renderInContext` von Core Graphics für `CALayer`-Instanzen. Dadurch ergibt sich ein Screenshot nicht nur der betreffenden Ansicht, sondern aller Ansichten, die sie besitzt.

Natürlich gibt es hierbei Einschränkungen. Sie können keine Screenshots des gesamten Fensters aufnehmen (die Statusleiste wird fehlen) und auch keine von Videos und von der Kameravorschau. Auch OpenGLES-Ansichten werden unter Umständen nicht erfasst.

```
+ (UIImage *) imageFromView: (UIView *) theView
{
    // Zeichnet den Inhalt einer Ansicht in einen Bildkontext
    UIGraphicsBeginImageContext(theView.frame.size);
    CGContextRef context = UIGraphicsGetCurrentContext();
    [theView.layer renderInContext:context];
    UIImage *theImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return theImage;
}
```

► *Listing 7.2: Screenshots von einer Ansicht aufnehmen*

7.15 Rezept: Direkt mit Bitmaps arbeiten

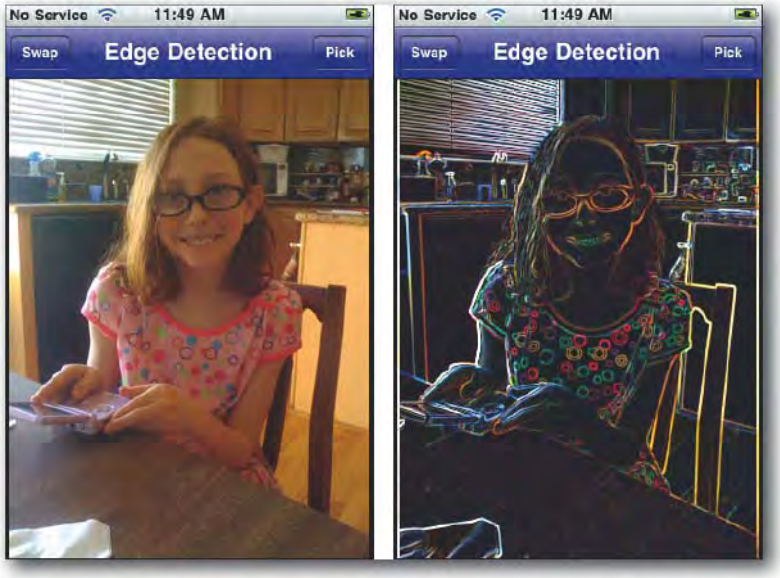
Cocoa Touch enthält zwar hervorragende, auflösungsunabhängige Werkzeuge für die Arbeit mit Bildern, doch manchmal müssen Sie sich auch zu den Bits vorwagen, aus denen sich ein Bild zusammensetzt, und Bit für Bit auf die Daten zugreifen. Beispielsweise kann es sein, dass Sie eine Kantenerkennung durchführen oder Weichzeichnerroutinen anwenden möchten. Solche Funktionen berechnen ihre Ergebnisse, indem sie Matrizenoperationen auf die tatsächlichen Bytewerte anwenden.

Abbildung 7.8 zeigt das Ergebnis der Anwendung einer Canny-Kantenerkennung auf ein iPhone-Bild. Der Canny-Operator in seiner einfachsten Form ist einer der ersten Algorithmen, die in Seminaren zur Bildverarbeitung gelehrt werden. Die Version, die verwendet wurde, um das hier gezeigte Bild hervorzurufen, verwendet eine hartkodierte 3×3-Maske.

7.15.1 In einem Bitmapkontext zeichnen

Um mit der Bildverarbeitung zu beginnen, zeichnen Sie ein Bild in einen Bitmapkontext und rufen die Bytes dann als `char *`-Puffer ab. Dies geschieht im folgenden Code. Nachdem das Bild gezeichnet wurde, werden die Bits aus dem Kontext abgerufen.

```
+ (unsigned char *) bitmapFromImage: (UIImage *) image
{
   // Erstellt Bitmapdaten für das gegebene Bild
   CGContextRef context = CreateARGBBitmapContext(image.size);
   if (context == NULL) return NULL;

   CGRect rect = CGRectMake(0.0f, 0.0f,
      image.size.width, image.size.height);
   CGContextDrawImage(context, rect, image.CGImage);
   unsigned char *data = CGBitmapContextGetData (context);
   CGContextRelease(context);
   return data;
}
```



▸ *Abbildung 7.8: Bei der Anwendung einer Kantenerkennung auf ein Bild werden die Bereiche umrandet, bei denen die Bytewerte den größten Änderungen unterliegen.*

```
CGContextRef CreateARGBBitmapContext (CGSize size)
{
   // Erstellt den neuen Farbraum
   CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
   if (colorSpace == NULL)
```

```
    {
        fprintf(stderr, "Error allocating color space\n");
        return NULL;
    }

    // Weist Speicher für die Bitmapdaten zu
    void *bitmapData = malloc(size.width * size.height * 4);
    if (bitmapData == NULL)
    {
        fprintf (stderr, "Error: Memory not allocated!");
        CGColorSpaceRelease(colorSpace);
        return NULL;
    }

    // Erstellt einen Kontext mit 8 Bit pro Kanal
    CGContextRef context = CGBitmapContextCreate (bitmapData,
        size.width, size.height, 8, size.width * 4, colorSpace,
        kCGImageAlphaPremultipliedFirst);
    CGColorSpaceRelease(colorSpace );
    if (context == NULL)
    {
        fprintf (stderr, "Error: Context not created!");
        free (bitmapData);
        return NULL;
    }

    return context;
}
```

Sobald die Bildbytes verfügbar sind, können Sie direkt darauf zugreifen. Die folgenden Funktionen geben unter Verwendung der Breite `w` Offsets für jeden Punkt `(x,y)` innerhalb einer ARGB-Bitmap zurück. Die Höhe wird für diese Berechnungen nicht gebraucht. Anhand der Breite der einzelnen Zeilen können Sie in einem eigentlich eindimensionalen Puffer einen Punkt in einem zweidimensionalen Raum bestimmen. Beachten Sie, wie die Daten miteinander verknüpft sind. Jede 4-Byte-Sequenz enthält einen Wert für Alpha, Rot, Grün und Blau, die jeweils von 0 (0 %) bis 255 (100 %) reichen können. Um die ARGB-Werte zu erhalten, müssen Sie diese Angaben in Fließkommawerte umwandeln und durch 255,0 teilen.

```
NSUInteger alphaOffset(NSUInteger x, NSUInteger y, NSUInteger w)
    {return y * w * 4 + x * 4 + 0;}
NSUInteger redOffset(NSUInteger x, NSUInteger y, NSUInteger w)
    {return y * w * 4 + x * 4 + 1;}
NSUInteger greenOffset(NSUInteger x, NSUInteger y, NSUInteger w)
    {return y * w * 4 + x * 4 + 2;}
NSUInteger blueOffset(NSUInteger x, NSUInteger y, NSUInteger w)
    {return y * w * 4 + x * 4 + 3;}
```

7.15.2 Bildbearbeitung durchführen

Es ist jetzt ziemlich einfach, auf die Bytes eines Bildes zuzugreifen und einige Bildbearbeitungsfunktionen darauf anzuwenden. Die folgende Routine verwendet den bereits erwähnten Canny-Algorithmus für die Kantenerkennung. Sie berechnet das Ergebnis für vertikale und horizontale Kanten für jeden einzelnen Farbkanal, fasst die beiden Ergebnisse jeweils zu einer Summe zusammen und skaliert diese, sodass sie in das Intervall [0, 255] fällt. Der Alpha-Wert bleibt erhalten.

```
+ (UIImage *) convolveImageWithEdgeDetection: (UIImage *) image
{
   // Dimensionen
   int theheight = (int) image.size.height;
   int thewidth = (int) image.size.width;

   // Ruft die Eingabe ab und erstellt Ausgabebits
   unsigned char *inbits = (unsigned char *)[ImageHelper
      bitmapFromImage:image];
   unsigned char *outbits = (unsigned char *)malloc(theheight *
      thewidth * 4);

   int radius = 1;

   // Iteriert durch alle verfügbaren Pixel (lässt eine Grenze im
   // Abstand des Radius übrig)
   for (int y = radius; y < (theheight - radius); y++)
      for (int x = radius; x < (thewidth - radius); x++)
      {
         int sumr1 = 0, sumr2 = 0;
         int sumg1 = 0, sumg2 = 0;
         int sumb1 = 0, sumb2 = 0;

         // Grundlegende Canny-Kantenerkennung
         int matrix1[9] = {-1, 0, 1, -2, 0, 2, -1, 0, 1};
         int matrix2[9] = {-1, -2, -1, 0, 0, 0, 1, 2, 1};
         int offset = 0;
         for (int j = -radius; j <= radius; j++)
            for (int i = -radius; i <= radius; i++)
            {
               sumr1 += inbits[redOffset(x+i, y+j, thewidth)] *
                  matrix1[offset];
               sumr2 += inbits[redOffset(x+i, y+j, thewidth)] *
                  matrix2[offset];

               sumg1 += inbits[greenOffset(x+i, y+j, thewidth)] *
                  matrix1[offset];
```

```
                sumg2 += inbits[greenOffset(x+i, y+j, thewidth)] *
                    matrix2[offset];

                sumb1 += inbits[blueOffset(x+i, y+j, thewidth)] *
                    matrix1[offset];
                sumb2 += inbits[blueOffset(x+i, y+j, thewidth)] *
                    matrix2[offset];

                offset++;
            }
        // Weist die Ausgabebits zu
        int sumr = MIN(((ABS(sumr1) + ABS(sumr2)) / 2), 255);
        int sumg = MIN(((ABS(sumg1) + ABS(sumg2)) / 2), 255);
        int sumb = MIN(((ABS(sumb1) + ABS(sumb2)) / 2), 255);
        outbits[redOffset(x, y, thewidth)] = (unsigned char) sumr;
        outbits[greenOffset(x, y, thewidth)] = (unsigned char)
            sumg;
        outbits[blueOffset(x, y, thewidth)] = (unsigned char) sumb;
        outbits[alphaOffset(x, y, thewidth)] =
            (unsigned char) inbits[alphaOffset(x, y, thewidth)];
    }

    // Gibt die ursprüngliche Bitmap frei. imageWithBits gibt die
    // Ausgabebits frei.
    free(inbits);
    return [ImageHelper imageWithBits:outbits withSize:image.size];
}
```

7.15.3 Einschränkungen bei der Bildbearbeitung

Das iPhone ist kein Rechenzentrum. Routinen wie die hier vorgestellte können Anwendungen erheblich verlangsamen, weshalb Sie sie mit Bedacht verwenden müssen. *Rezept 7.13* zeigt, wie Sie die Bildbearbeitung an die eingeschränkte Kapazität des iPhones anpassen. Der Code folgt den drei Hauptregeln der iPhone-Programmierung:

- Geben Sie dem Benutzer aussagekräftige Meldungen, wenn es zu unvermeidbaren Verzögerungen kommt.
- Führen Sie prozessorintensive Funktionen in einem zweiten Thread durch.
- Führen Sie Aktualisierungen der grafischen Benutzeroberfläche nur im Hauptthread durch.

Den Ablauf dieser Lösung sehen Sie in *Abbildung 7.9*. Ein Schwebefenster (Head Up Display, HUD) mit der Meldung »Please wait« (»Bitte warten«) erscheint mit einer sich drehenden Aktivitätsanzeige. Es bleibt sichtbar, bis ein zweiter Verarbeitungsthread abgeschlossen wird.

Wenn die gesamte Verarbeitung im Hauptthread stattfände, könnte diese Aktivitätsanzeige nicht richtig dargestellt erden. Umfangreiche Verarbeitungsvorgänge blockieren die Aktualisierung der grafischen Benutzeroberfläche, was dazu führen würde, dass das Erscheinen der `UIAlertView` mit dem Schwebefenster verzögert wird, bis die Verarbeitung abgeschlossen ist. Das widerspricht der Regel, »aussagekräftige Meldungen« zu geben, die bei der Entwicklung von iPhone-Anwendungen so wichtig sind. Darum müssen Sie hier einen Ansatz mit zwei Threads verfolgen.

Die Routine `process` ist so entworfen, dass sie auf einem eigenen Thread läuft. Sie verfügt über einen eigenen `NSAutorelease`-Pool und wird von der Methode ausgelöst, die die Beendigung der Auswahl signalisiert. All dies geschieht zwar im Bildkontext, doch werden in der Methode keinerlei Elemente der grafischen Benutzeroberfläche geändert. Die Aufgabe dieser Methode besteht nur darin, ein Bild in einen Raum von 320 × 416 Pixel zu zeichnen und dann die Canny-Kantenerkennung darauf anzuwenden.

Wenn der Thread mit der Verarbeitung fertig ist, ruft er eine `finish`-Methode im Hauptthread auf. Diese Methode wiederum räumt die grafische Benutzeroberfläche auf, indem sie das Schwebefenster entfernt, eine Austausch-Schaltfläche (**Swap**) hinzufügt und das angezeigte Bild festlegt.

```
@implementation TestBedViewController
@synthesize original;
@synthesize processed;
#define SETIMAGE(X) [UIImageView*)self.view setImage:X];
// Erlaubt dem Benutzer, zwischen dem ursprünglichen und dem verarbeiteten
// Bild zu wechseln
- (void) swap
{
   // SETIMAGE funktioniert ab Version 2.2
   if ([(UIImageView *)self.view image] == self.original)
      SETIMAGE(self.processed)
   else
      SETIMAGE(self.original);
}

// Räumt die Benutzeroberfläche im Hauptthread auf
- (void) finish
{
   SETIMAGE(self.processed);
   self.navigationItem.leftBarButtonItem = BARBUTTON(@"Swap",
      @selector(swap));
   [ModalHUD dismiss];
}

// Führt die rechenintensive Verarbeitung in einem zweiten Thread durch
- (void) process
{
```

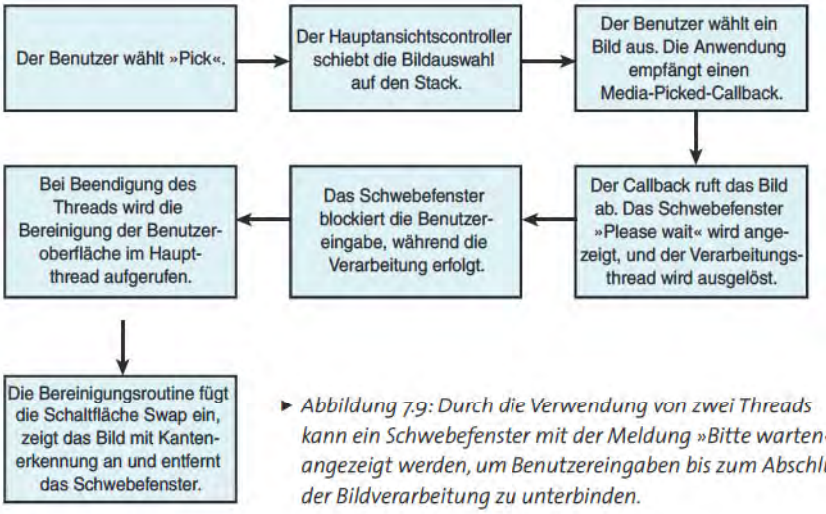
```
    NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
    CGSize coreSize = CGSizeMake(320.0f, 416.0f);

    // Skaliert das Bild
    UIGraphicsBeginImageContext(coreSize);
    [self.original drawInRect:[ImageHelper frameSize:self.original.size
        inSize:coreSize]];
    UIImage *newimg = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    self.original = newimg;

    // Berechnet die Kantenerkennung
    self.processed = [ImageHelper convolveImageWithEdgeDetection:
        self.original];

    // Räumt im Hauptthread auf
    [self performSelectorOnMainThread:@selector(finish) withObject:nil
        waitUntilDone:NO];
    [pool release];
}

// Zeigt das Schwebefenster an und startet den Verarbeitungsthread
- (void)imagePickerController:(UIImagePickerController *)picker
        didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    [ModalHUD showHUD:@"Processing\nPlease wait."];
    self.original = [info objectForKey:
        @"UIImagePickerControllerOriginalImage"];
    [self dismissModalViewControllerAnimated:YES];
    [picker release];

    [NSThread detachNewThreadSelector:@selector(process)
        toTarget:self withObject:nil];
}

// Ermöglicht 2.x-Kompatibilität
- (void)imagePickerController:(UIImagePickerController *)picker
        didFinishPickingImage:(UIImage *)image
        editingInfo:(NSDictionary *)editingInfo
{
    NSDictionary *dict = [NSDictionary dictionaryWithObject:image
        forKey:@"UIImagePickerControllerOriginalImage"];
    [self imagePickerController:picker
        didFinishPickingMediaWithInfo:dict];
}
```

```
// Ermöglicht dem Benutzer, ein neues Bild zur Verarbeitung auszuwählen
- (void) pickImage: (id) sender
{
   UIImagePickerController *ipc = [[UIImagePickerController alloc]
      init];
   ipc.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
   ipc.delegate = self;
   ipc.allowsImageEditing = NO; // .allowsEditing in 3.1
   [self presentModalViewController:ipc animated:YES];
}

// Initialisiert Titel und Leistenschaltfläche
- (void) viewDidLoad
{
   self.navigationController.navigationBar.tintColor =
      COOKBOOK_PURPLE_COLOR;
   self.navigationItem.rightBarButtonItem = BARBUTTON(@"Pick",
      @selector(pickImage));
   self.title = @"Edge Detection";
}
@end
```

► *Rezept 7.13: Eine für das iPhone geeignete Benutzeroberfläche für die Bildverarbeitung anzeigen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 7 und öffnen das Projekt zu diesem Rezept.



► *Abbildung 7.9: Durch die Verwendung von zwei Threads kann ein Schwebefenster mit der Meldung »Bitte warten« angezeigt werden, um Benutzereingaben bis zum Abschluss der Bildverarbeitung zu unterbinden.*

7.16 Ein letzter Punkt: Graustufenbilder

Wie in *Rezept 7.12* angedeutet wurde, können Sie auch sehr leicht eine Graustufenversion eines Bildes erstellen, nicht einfach nur Schwarz-Weiß-Masken. *Listing 7.3* zeigt eine allgemeine Hilfsmethode, die ein Bild in einen Graustufen-Farbraum zeichnet. Anders als die zuvor gezeigte Funktion `CreateMaskImage()` muss bei dieser Methode das `UIImage`-Koordinatensystem nicht mit dem von Quartz übereinstimmen, sodass der Kontext hier nicht umgekehrt wird. Die Methode zeichnet einfach das Bild in den Kontext und gibt die Graustufenversion zurück.

Wenn Sie diese Funktion mit der Screenshot-Funktion `renderInContext:` kombinieren, die weiter vorn in diesem Kapitel beschrieben wurde, können Sie einen »interaktiven« Hintergrund erstellen, der die aktuelle grafische Benutzeroberfläche simuliert. Damit können Sie einen grafischen Kontext bereitstellen, der die Aufmerksamkeit des Benutzers auf einen fortlaufenden Vorgang richtet, z. B. einen Dateidownload. Dies ist eine kreative Alternative zu der üblichen Abdunkelung des Bildschirms.

```
+ (UIImage *) grayscaleImage: (UIImage *) image
{
   CGSize size = image.size;
   CGRect rect = CGRectMake(0.0f, 0.0f, image.size.width,
      image.size.height);

   // Erstellt einen monochromen oder Graustufen-Farbraum
   CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceGray();
   CGContextRef context = CGBitmapContextCreate(nil, size.width,
      size.height, 8, 0, colorSpace, kCGImageAlphaNone);
   CGColorSpaceRelease(colorSpace);

   // Zeichnet das Bild im Graustufenkontext
   CGContextDrawImage(context, rect, [image CGImage]);
   CGImageRef grayscale = CGBitmapContextCreateImage(context);
   CGContextRelease(context);

   // Ruft das Bild ab
   UIImage *img = [UIImage imageWithCGImage:grayscale];
   CFRelease(grayscale);
   return img;
}
```

▸ *Listing 7.3: Die Graustufenversion eines Bildes zurückgeben*

7.17 Zusammenfassung

In diesem Kapitel haben Sie viele Möglichkeiten kennengelernt, um mit Bildern umzugehen – um sie auszuwählen, zu lesen, zu ändern und zu speichern. In den Rezepten haben Sie erfahren, wie Sie die eingebaute Bildauswahl des iPhones nutzen und wie Sie Fotos mit der Kamera aufnehmen. Außerdem haben Sie etwas darüber gelesen, wie Sie Bilder zur Klasse `UIScrollView` hinzufügen und als E-Mail-Anhänge senden. Bevor wir zum nächsten Kapitel übergehen, finden Sie an dieser Stelle noch einige Anmerkungen zu den vorgestellten Rezepten:

- Die integrierte Bildauswahl geht sehr schludrig mit dem Speicher um. Richten Sie sich beim Programmieren nach dieser bedauerlichen Tatsache.
- Geben Sie den Benutzern bei Verzögerungen aufgrund einer intensiven Verarbeitung stets Rückmeldung. Die meisten Operationen zur Bildverarbeitung sind langsam. Da der Simulator eine bessere Leistung zeigt als ein echtes iPhone, müssen Sie Ihre Anwendungen nicht nur auf dem Simulator, sondern auch auf einem physischen Gerät testen, und Sie müssen die Benutzer mit einem Mechanismus wie dem in diesem Kapitel verwendeten Schwebefenster wissen lassen, dass ein Vorgang eine gewisse Zeit in Anspruch nehmen kann.
- Die Möglichkeit, Bilder direkt aus einem Programm heraus als E-Mail-Anhänge zu senden, ist ein hervorragendes neues Merkmal der Version 3.0. Bevor Sie versuchen, den entsprechenden Controller zu verwenden, sollten Sie jedoch prüfen, ob das Gerät für E-Mail geeignet ist. Denken Sie auch daran, dass das Senden von Bildern sehr, sehr lange dauern kann.
- Mehrseitige Rollansichten sind eine praktische Grundlage für grafische Benutzeroberflächen. Verwenden Sie sie, um mehrere Bilder anzuzeigen oder um rollbare Oberflächen mit mehreren Bildschirmen darzustellen.
- Für Vorschaubilder ist weit weniger Arbeitsspeicher erforderlich als für das Laden sämtlicher Bilder auf einmal. Neben den in diesem Kapitel vorgestellten Routinen für die Verkleinerung von Bildern zu Vorschaubildern können Sie auch Versionen der Bilder in Symbolgröße im Voraus berechnen.

8 Gesten und Berührungen

Berührungen sind der Dreh- und Angelpunkt der Interaktion auf dem iPhone und die wichtigste Art und Weise, auf die die Benutzer einer Anwendung mitteilen können, was sie tun wollen. Dabei sind Berührungen nicht auf die Betätigung von Schaltflächen oder Tastaturen beschränkt. Sie können Anwendungen entwerfen und schreiben, die unmittelbar auf Antippen und andere Gesten reagieren. In diesem Kapitel lernen Sie die direkte Bearbeitung von Oberflächen kennen, die weit über die Verwendung vorgefertigter Steuerelemente hinausgeht. Sie erfahren hier auch, wie Sie Gesten unterscheiden und interpretieren und wie Sie mit den eingebauten Mehrfachberührungssensoren des iPhones arbeiten. Nach dem Durcharbeiten dieses Kapitels haben Sie viel Wissen über die verschiedenen Möglichkeiten erworben, um in Ihren eigenen Anwendungen eine Steuerung durch Gesten umzusetzen.

8.1 Berührungen

Cocoa Touch implementiert die direkte Bearbeitung auf die einfachste mögliche Weise, nämlich durch das Senden von Berührungsereignissen an die aktuelle Ansicht. Als iPhone-Entwickler müssen Sie der Ansicht mitteilen, wie sie auf die verschiedenen Berührungen reagieren soll.

Berührungen übermitteln Informationen: wo die Berührung stattgefunden hat (sowohl die jetzige als auch die vorhergehende Position), welche Phase einer Berührung erfolgt (bei Desktop-Anwendungen sind das allgemein das Drücken der Maustaste, das Verschieben der Maus und das Freigeben der Maustaste, was bei der direkten Bearbeitung dem Heben, Bewegen und Senken des Fingers entspricht), wie oft getippt wurde (Einzeltipp/Doppeltipp) und wann die Berührung stattfand (Zeitstempel). Berührungen und die damit verbundenen Informationen werden in `UITouch`-Objekten gespeichert. Jedes dieser Objekte steht für ein einzelnes Berührungsereignis. Die Anwendung empfängt diese Ereignisse in der Ansichtsklasse, und dort müssen Sie sie verarbeiten und darauf reagieren.

Dies mag der Intuition widersprechen. Wahrscheinlich erwarten Sie, das Aussehen einer Oberfläche (die Ansicht) von der Reaktion auf die Berührungen (dem Controller) zu trennen. Auf dem iPhone folgt die direkte Interaktion durch Berührungen einem ziemlich primitiven Entwurfsmuster, das nur wenig oder gar nicht mit dem Model-View-Controller-Prinzip (MVC) übereinstimmt. Es gilt die Regel, dass Sie in der Klasse `UIView` programmieren und nicht in `UIViewController`. Dies ist ein sehr wichtiger Umstand. Der Versuch, eine maschinennahe Gestensteuerung in der falschen Klasse zu implementieren, hat schon viele Einsteiger in die iPhone-Entwicklung ins Stolpern gebracht.

Bei der maschinennahen Berührungsinteraktion sind die Interpretation der Gesten und die grafische Darstellung eng miteinander verzahnt. Dieser Aufbau hat auch seine Vorteile. Ansichtscontroller können mehrere Ansichten besitzen, die jeweils anders auf Berührungen reagieren. Würde der Ansichtscontroller alle Berührungen direkt handhaben, müssten die Antwortroutinen die jeweils für eine Ansicht geeignete Reaktion auswählen. Solcher Code würde sehr schnell unübersichtlich. Durch die Programmierung in der Ansicht wird die Implementierung vereinfacht. Ein zweiter Vorteil besteht darin, dass die Programmierung in der Ansicht es ermöglicht, benutzerdefinierte Oberflächenobjekte zu erstellen, die vollständig eigenständig sind.

In den folgenden Abschnitten und Rezepten erfahren Sie, wie Berührungen funktionieren, wie Sie sie in Ihren Anwendungen einsetzen können und wie Sie eine Verbindung zwischen dem herstellen, was der Benutzer sieht und dem, was er auf dem Bildschirm tut.

8.1.1 Phasen

Jede Berührung hat einen Lebenszyklus, in dem sie in bis zu fünf Phasen eintreten kann:

- `UITouchPhaseBegan` Beginnt, wenn der Benutzer den Bildschirm berührt.
- `UITouchPhaseMoved` Bedeutet, dass der Finger über den Bildschirm bewegt wird.
- `UITouchPhaseStationary` Zeigt an, dass der Finger auf der Bildschirmoberfläche bleibt, dass es aber seit dem letzten Ereignis keine Bewegung gegeben hat.
- `UITouchPhaseEnded` Wird ausgelöst, wenn der Finger von der Oberfläche genommen wird.
- `UITouchPhaseCancelled` Tritt auf, wenn das iPhone-Betriebssystem aufhört, die Berührung nachzuverfolgen. Dies geschieht gewöhnlich bei einer Systemunterbrechung, z. B. wenn die Anwendung nicht mehr aktiv ist oder wenn die Ansicht aus dem Fenster entfernt wird.

Als Ganzes bilden diese fünf Phasen die Interaktionssprache für ein Berührungsereignis. Sie beschreiben alle möglichen Arten, die der Verlauf einer Berührung in einer Oberfläche nehmen kann, und ermöglichen damit die Steuerung der Oberfläche. Es ist Ihre Aufgabe als Entwickler, diese Phasen zu interpretieren und Reaktionen darauf vorzusehen. Dazu implementieren Sie eine Folge von Ansichtsmethoden.

8.1.2 Berührungen und Ansichtsmethoden

Alle Member und Kindelemente der Klasse `UIResponder`, darunter auch `UIView`, reagieren auf Berührungen, wobei jede Klasse entscheidet, ob und wie sie reagiert. Wenn ja, so implementiert sie ein angepasstes Verhalten, wenn ein Benutzer die Ansicht oder das Fenster mit einem oder mehreren Fingern berührt.

Vordefinierte Callback-Methoden kümmern sich um den Start, die Bewegung und das Ende der Berührungen auf dem Bildschirm. Diese im Folgenden angegebenen Methoden entsprechen den zuvor erwähnten Phasen, wobei jedoch `UITouchPhaseStationary` keinen Callback auslöst.

- `touchesBegan:withEvent:` Wird in der Startphase des Ereignisses aufgerufen, wenn der Benutzer damit beginnt, den Bildschirm zu berühren.
- `touchesMoved:withEvent:` Kümmert sich um die Bewegung der Finger im Zeitverlauf.
- `touchesEnded:withEvent:` Schließt den Berührungsvorgang ab, wenn die Finger vom Bildschirm genommen werden. Dies ist der geeignete Zeitpunkt, um Arbeiten abzuschließen, die während der Bewegungsphase ausgeführt wurden.
- `touchesCancelled:withEvent:` Wird aufgerufen, wenn Cocoa Touch auf eine Systemunterbrechung des fortlaufenden Berührungsereignisses reagieren muss.

All dies sind `UIResponder`-Methoden, die gewöhnlich in einer Unterklasse von `UIView` implementiert werden. Sämtliche Ansichten erben die grundlegenden, nicht funktionalen Versionen dieser Methoden. Wenn Sie Ihrer Anwendung ein Berührungsverhalten hinzufügen möchten, müssen Sie diese Methoden überschreiben und eine benutzerdefinierte Version schreiben, die so reagiert, wie es die Anwendung erfordert.

In den Rezepten in diesem Kapitel werden einige dieser Methoden implementiert, aber nicht alle. Für die Entwicklung in der Praxis sollten Sie noch ein Ereignis für abgebrochene Berührungen ergänzen, um den Fall zu handhaben, dass ein Telefonanruf eingeht und die Berührungssequenz dadurch abgebrochen wird. Apple empfiehlt, in den Unterklassen von `UIView` alle vier Methoden zu überschreiben.

Hinweis

Ansichten weisen einen sogenannten »exklusiven Berührungsmodus« auf, der verhindert, dass Berührungen an andere Ansichten übergehen. Wenn diese Eigenschaft aktiviert ist, blockiert sie den Empfang von Berührungsereignissen durch andere Ansichten. Die Hauptansicht handhabt alle Berührungsereignisse exklusiv.

8.1.3 Ansichten berühren

Befinden sich mehrere Ansichten auf dem Bildschirm, entscheidet das iPhone automatisch, welche davon der Benutzer berührt hat, und übergibt die Berührungsereignisse an diese Ansicht. Dadurch können Sie konkrete Oberflächen mit direkter Bearbeitung schreiben, auf denen die Benutzer die dargestellten Objekte berühren, ziehen und auf andere Weise bearbeiten können.

Dass einer Ansicht eine Berührung übergeben wird, heißt noch lange nicht, dass die Ansicht auch darauf reagieren muss. Jede Ansicht kann entscheiden, ob sie eine Berührung verarbeitet oder an die darunter liegenden Ansichten weitergibt. In den folgenden Rezepten sehen Sie, wie Sie mit geschickten Verfahren entscheiden, welche Ansicht reagieren soll, vor allem dann, wenn es um unregelmäßig geformte und teilweise transparente Objekte geht.

8.1.4 Mehrfachberührung

Auf dem iPhone sind Oberflächen möglich, die auf Einzel- sowie auf Mehrfachberührungen reagieren. Grafische Benutzeroberflächen mit Einzelberührung können immer nur eine Berührung auf einmal handhaben. Dadurch sind Sie als Entickler der Verantwortung enthoben, zu bestimmen, welche Berührung Sie nachverfolgen. Die Berührung, die Sie empfangen, ist auch diejenige, mit der Sie arbeiten müssen. Sie schauen sich ihre Daten an, reagieren darauf und warten auf das nächste Ereignis.

Bei Mehrfachberührungen – wenn Sie also auf mehrere Berührungen des Bildschirms auf einmal reagieren – empfangen Sie einen Satz von Berührungen. Es liegt an Ihnen, diese Berührungen zu ordnen und auf den Gesamtsatz zu reagieren. Dabei verfolgen Sie die einzelnen Berührungen getrennt nach und beobachten, wie sie sich mit der Zeit ändern, was eine breitere Palette an Benutzeraktivitäten ermöglicht. In diesem Kapitel finden Sie Rezepte für die Interaktion sowohl mit Einzel- als auch mit Mehrfachberührungen.

8.2 Rezept: Eine einfache Oberfläche mit Direktbearbeitung hinzufügen

Bei der Direktbearbeitung verschiebt sich der Schwerpunkt des Designs von der Klasse `UIViewController` zu `UIView`. Die Ansicht, oder genauer gesagt, der `UIResponder`, ist der Kern der Entwicklung von Oberflächen zur Direktbearbeitung. Um berührungsgesteuerte Oberflächen zu erstellen, passen Sie Methoden an, die von der Klasse `UIResponder` abgeleitet sind.

Rezept 8.1 zeigt Berührungen in Aktion. In diesem Beispiel wird ein Kind von `UIImageView` namens `DragView` erstellt und um Reaktionen auf Berührungen ergänzt. Da es eine Bildansicht ist, müssen Sie die Benutzerinteraktion aktivieren, indem Sie `setUserInteractionEnabled` auf `YES` setzen. Diese Eigenschaft betrifft sowohl die Ansicht selbst als auch alle ihre Kinder.

In diesem Rezept wird der Mittelpunkt der Ansicht so angepasst, dass er der Berührung folgt. Wenn der Benutzer zum ersten Mal auf irgendeine `DragView`-Ansicht tippt, speichert das Objekt die Startposition als Versatz vom Ursprung der Ansicht. Wenn der Benutzer zieht, wird die Ansicht zusammen mit dem Finger verschoben, wobei sie stets denselben Versatz vom Ursprung beibehält, sodass die Bewegung natürlich erscheint. Die Verschiebung erfolgt durch Aktualisierung des Objektmittelpunkts. *Rezept 8.1* berechnet den x- und y-Versatz und ändert die Position des Mittelpunkts der Ansicht nach jeder Fingerbewegung um diese Offset-Werte.

Wenn die Ansicht berührt wird, springt sie in den Vordergrund. Dies geschieht aufgrund eines Aufrufs in der Methode `touchesMoved:withEvent:`. Der Code weist die übergeordnete Ansicht von `DragView` an, Letztere nach vorn zu bringen. Dadurch erscheint das aktive Element stets im Vordergrund der Oberfläche.

8.2 Rezept: Eine einfache Oberfläche mit Direktbearbeitung hinzufügen

In diesem Rezept werden keine Methoden für das Ende oder den Abbruch der Berührung implementiert, da der Schwerpunkt hier auf der Bewegung von Objekten auf dem Bildschirm liegt. Wenn der Benutzer nichts mehr auf dem Bildschirm tut, hat die Klasse keine weitere Arbeit mehr zu verrichten.

```
@interface DragView : UIImageView
{
   CGPoint startLocation;
}
@end

@implementation DragView
- (id) initWithImage: (UIImage *) anImage
{
   if (self = [super initWithImage:anImage])
      self.userInteractionEnabled = YES;
   return self;
}

- (void) touchesBegan:(NSSet*)touches withEvent:(UIEvent*)event
{
   // Berechnet und speichert den Offset und holt die Ansicht ggf. in den
   // Vordergrund
   CGPoint pt = [[touches anyObject] locationInView:self];
   startLocation = pt;
   [[self superview] bringSubviewToFront:self];
}

- (void) touchesMoved:(NSSet*)touches withEvent:(UIEvent*)event
{
   // Berechnet den Offset
   CGPoint pt = [[touches anyObject] locationInView:self];
   float dx = pt.x - startLocation.x;
   float dy = pt.y - startLocation.y;
   CGPoint newcenter = CGPointMake(self.center.x + dx,
      self.center.y + dy);

   // Legt die neue Position fest
   self.center = newcenter;
}
@end
```

► *Rezept 8.1: Eine ziehbare Ansicht erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

8.3 Rezept: Die Bewegung einschränken

Das Problem bei dem einfachen Ansatz von *Rezept 8.1* besteht darin, dass es durchaus möglich ist, eine Ansicht so weit vom Bildschirm zu ziehen, dass der Benutzer sie nicht mehr sehen und nicht mehr zurückholen kann. In diesem Rezept sind die Bewegungen völlig uneingeschränkt. Es wird nicht geprüft, ob das Objekt sichtbar und berührbar bleibt. In *Rezept 8.2* wird dieses Problem dadurch gelöst, dass die Bewegung einer Ansicht auf die Elternansicht beschränkt wird.

Hierzu wird die Bewegung in jeder Richtung eingeschränkt und die Überprüfung auf getrennte x- und y-Bedingungen aufgeteilt. Dadurch kann die Ansicht auch dann noch bewegt werden, wenn sie in einer Richtung die Grenze erreicht hat. Kommt die Ansicht z. B. am rechten Rand der Elternansicht an, kann sie immer noch nach oben und unten verschoben werden.

Abbildung 8.1 zeigt die Oberfläche. Die Blumen sind auf das schwarze Rechteck in der Mitte beschränkt und können nicht aus der Ansicht herausgezogen werden. Der Code ist allgemein und lässt sich an Begrenzungen und Kindansichten jeder Größe anpassen.

▸ *Abbildung 8.1: Die Bewegung der Blumen ist auf das schwarze Rechteck eingeschränkt.*

```
- (void) touchesMoved:(NSSet*)touches withEvent:(UIEvent*)event
{
    // Berechnet den Offset
    CGPoint pt = [[touches anyObject] locationInView:self];
    float dx = pt.x - startLocation.x;
    float dy = pt.y - startLocation.y;
    CGPoint newcenter = CGPointMake(self.center.x + dx,
        self.center.y + dy);

    // Beschränkt die Bewegung auf die Grenzen der Elternansicht
    float halfx = CGRectGetMidX(self.bounds);
    newcenter.x = MAX(halfx, newcenter.x);
    newcenter.x = MIN(self.superview.bounds.size.width - halfx,
        newcenter.x);

    float halfy = CGRectGetMidY(self.bounds);
    newcenter.y = MAX(halfy, newcenter.y);
    newcenter.y = MIN(self.superview.bounds.size.height - halfy,
        newcenter.y);

    // Legt die neue Position fest
    self.center = newcenter;
}
```

► *Rezept 8.1: Eingeschränkte Bewegung*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

8.4 Rezept: Ansichten auf Berührungen testen

Die wenigsten Ansichtselemente für die Direktbearbeitung sind rechteckig. Dies erschwert die Erkennung von Berührungen, da das Rechteck der Ansicht nicht mit der Berührungsfläche übereinstimmt. *Abbildung 8.2* veranschaulicht dieses Problem. Der Screenshot auf der rechten Seite zeigt die Oberfläche mit den berührungsgesteuerten Unteransichten. Links sehen Sie die Unteransichten mit ihren tatsächlichen Grenzen. Die hellgrauen Flächen um die Kreise befinden sich innerhalb der Grenzen, aber Berührungen in diesen Bereichen sollten von der betreffende Ansicht ignoriert werden.

▸ *Abbildung 8.2: Die Anwendung sollte Berührungen in den grauen Bereichen um die Kreise ignorieren (links). In der eigentlichen Oberfläche (rechts) werden Alpha-Werte von null verwendet, um die nicht verwendeten Teile der Ansichten auszublenden.*

Das iPhone erkennt Berührungen innerhalb des gesamten Rahmens einer Ansicht, also nicht nur in der eigentlichen Präsentation, sondern auch in den nicht gezeichneten Flächen wie den Ecken der Rahmen außerhalb der Kreise von *Abbildung 8.2*. Das bedeutet, dass Sie eine Form von Berührungstest hinzufügen müssen, da ansonsten der Versuch, durch einen durchsichtigen Teil eines `UIView`-Frames hindurch auf eine andere Ansicht zu tippen, zu unerwarteten Ergebnissen führt.

Die tatsächlichen Grenzen einer Ansicht können Sie anzeigen, indem Sie eine Hintergrundfarbe für sie festlegen:

```
dragger.backgroundColor = [UIColor lightGrayColor];
```

Dadurch werden die Hintergründe aus *Abbildung 8.2* (links) sichtbar, ohne die eigentliche Grafik zu ändern. In diesem Fall besteht die Grafik aus einem zentrierten Kreis mit transparentem Hintergrund. Sofern Sie keinen Test hinzufügen, werden alle Berührungen in jeglichen Bereichen dieses Rahmens von der betreffenden Sicht erfasst. Die Aktivierung von Hintergrundfarben ist eine bequeme Hilfe beim Debugging, um sich die wahre Ausdehnung der Ansichten deutlich zu machen. Allerdings dürfen Sie nicht vergessen, die Zuweisung der Hintergrundfarbe im endgültigen Code wieder auszukommentieren.

Rezept 8.3 fügt einen einfachen Test zu den Ansichten hinzu, um zu bestimmen, ob die Berührung innerhalb des Kreises stattfand. Dieser Test überschreibt die Standardmethode `pointInside:withEvent:` von `UIView`. Sie gibt entweder `YES` zurück (der Ort der Berührung liegt im Innern der Ansicht) oder `NO` (der Ort liegt außerhalb). Dieser Test stützt sich auf elementare Geometrie und

prüft, ob die Berührung innerhalb des Kreisradius erfolgt ist. Für Ihre eigenen Ansichten müssen Sie jeweils einen geeigneten Test vorsehen. Wie Sie in *Rezept 8.4* sehen, lassen sich diese Tests erweitern, um eine sehr viel feinere Steuerung zu ermöglichen.

```
- (BOOL) pointInside:(CGPoint)point withEvent:(UIEvent *)event
{
    CGPoint pt;
    float HALFSIDE = SIDELENGTH / 2.0f;

    // Normalisiert die Koordinaten mit Mittelpunkt der Ansicht
    pt.x = (point.x - HALFSIDE) / HALFSIDE;
    pt.y = (point.y - HALFSIDE) / HALFSIDE;

    // x^2 + y^2 = Radius
    float xsquared = pt.x * pt.x;
    float ysquared = pt.y * pt.y;

    // Bei Radius <= 1 liegt der Punkt innerhalb des Kreises
    if ((xsquared + ysquared) <= 1.0) return YES;
    return NO;
}
```

► *Rezept 8.3: Berührungstest für kreisförmige Ansichten*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

8.5 Rezept: Tests anhand einer Bitmap

Leider weisen die wenigsten Ansichten eine der einfachen Geometrien auf, die den Berührungstest aus *Rezept 8.3* so einfach machen. Beispielsweise haben die Blumen aus *Abbildung 8.1* unregelmäßige Grenzen und schwankende Transparenzwerte. Bei komplizierten Grafiken können Sie die Tests anhand einer Bitmap durchführen, die Byte für Byte den Inhalt einer Bildansicht darstellt. Dadurch können Sie prüfen, ob die Berührung auf einen undurchsichtigen Bestandteil der Ansicht fällt oder an die darunter liegenden Ansichten weitergeleitet werden soll.

In *Rezept 8.4* wird aus einer `UIImageView` eine Bild-Bitmap gewonnen. Voraussetzung ist, dass das verwendete Bild eine pixelweise Darstellung der betreffenden Ansicht gibt. Wenn Sie diese Ansicht verformen (normalerweise durch eine Veränderung der Rahmengröße oder durch eine Transformation) müssen Sie auch die Berechnungen entsprechend anpassen. Die Überprüfung wird einfacher,

wenn die Grafik proportional zur Ansicht ist. Sie können den betreffenden Pixel abrufen, seinen Alpha-Wert prüfen und bestimmen, ob die Berührung einen undurchsichtigen Teil der Ansicht getroffen hat oder nicht.

In diesem Beispiel wird die Grenze bei 85 gezogen, was einem Alpha-Wert von 33 % (85/255) entspricht. Die Methode `pointInside:` sieht jeden Pixel mit einem Alpha-Wert unter 33 % als transparent an. Dies ist jedoch eine willkürliche Entscheidung. Sie können jeden Alpha-Wert (oder eine andere Eigenschaft) nehmen, der sich für Ihre Benutzeroberfläche eignet.

```
// Gibt den Offset für den Alpha-Pixel bei (x,y) zurück.
// Dies gilt für RGBA-Bitmapdaten mit 4 Bytes pro Pixel.
NSUInteger alphaOffset(NSUInteger x, NSUInteger y, NSUInteger w)
   {return y * w * 4 + x * 4;}

// Gibt die Bitmap für das bereitgestellte Bild zurück
unsigned char *getBitmapFromImage (UIImage *image)
{

   CGColorSpaceRef colorSpace = CGColorSpaceCreateDeviceRGB();
   if (colorSpace == NULL)
   {
      fprintf(stderr, "Error allocating color space\n");
      return NULL;
   }

   CGSize size = image.size;
   void *bitmapData = calloc(size.width * size.height * 4, 1);
   if (bitmapData == NULL)
   {
      fprintf (stderr, "Error: Memory not allocated!");
      CGColorSpaceRelease(colorSpace);
      return NULL;
   }

   CGContextRef context = CGBitmapContextCreate (bitmapData,
      size.width, size.height, 8, size.width * 4, colorSpace,
      kCGImageAlphaPremultipliedFirst);
   CGColorSpaceRelease(colorSpace );
   if (context == NULL)
   {
      fprintf (stderr, "Error: Context not created!");
      free (bitmapData);
      return NULL:
   }

   CGRect rect = CGRectMake(0.0f, 0.0f, size.width, size.height);
```

```
    CGContextDrawImage(context, rect, image.CGImage);
    unsigned char *data = CGBitmapContextGetData(context);
    CGContextRelease(context);

    return data;
}

@interface DragView : UIImageView
{
    CGPoint startLocation;
    unsigned char *bytes;
}
@end

@implementation DragView
- (id) initWithImage: (UIImage *) anImage
{
    if (self = [super initWithImage:anImage])
    {
        self.userInteractionEnabled = YES;
        bytes = getBitmapFromImage(anImage);
    }
    return self;
}

- (void) dealloc
{
    free(bytes);
    [super dealloc];
}

// Trifft die Berührung die Ansicht?
- (BOOL) pointInside:(CGPoint)point withEvent:(UIEvent *)event
{
    if (!CGRectContainsPoint(self.bounds, point)) return NO;
    return (bytes[alphaOffset(point.x, point.y,
        self.image.size.width)] > 85);
}

- (void) touchesBegan:(NSSet*)touches withEvent:(UIEvent*)event
{
    // Berechnet und speichert den Offset und bringt die Ansicht ggf. in den
    // Vordergrund
    CGPoint pt = [[touches anyObject] locationInView:self];
```

```
    startLocation = pt;
    [[self superview] bringSubviewToFront:self];
}

- (void) touchesMoved:(NSSet*)touches withEvent:(UIEvent*)event
{
    // Berechnet den Offset
    CGPoint pt = [[touches anyObject] locationInView:self];
    float dx = pt.x - startLocation.x;
    float dy = pt.y - startLocation.y;
    CGPoint newcenter = CGPointMake(self.center.x + dx,
        self.center.y + dy);

    // Beschränkt die Bewegung auf die Grenzen der Elternansicht
    float halfx = CGRectGetMidX(self.bounds);
    newcenter.x = MAX(halfx, newcenter.x);
    newcenter.x = MIN(self.superview.bounds.size.width - halfx,
        newcenter.x);

    float halfy = CGRectGetMidY(self.bounds);
    newcenter.y = MAX(halfy, newcenter.y);
    newcenter.y = MIN(self.superview.bounds.size.height - halfy,
        newcenter.y);

    // Legt die neue Position fest
    self.center = newcenter;
}
@end
```

▸ *Rezept 8.4: Berührungen anhand einer Bitmap von Alpha-Werten testen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

8.6 Rezept: Oberflächen zur Direktbearbeitung dauerhaft machen

Die Dauerhaftigkeit (Persistenz) ist ein wichtiger Aspekt des iPhone-Designs. Apple empfiehlt dringend, ein Programm beim Start zu einem Zustand zurückkehren zu lassen, der möglichst genau dem entspricht, in dem der Benutzer es beim letzten Mal verlassen hat. Die einfachste Möglich-

keit, um eine Oberfläche zur Direktbearbeitung mit Dauerhaftigkeit zu versehen, besteht darin, eine Darstellung der Daten auf dem Bildschirm zu speichern, wenn die Anwendung beendet wird, und diesen Status beim Start wiederherzustellen.

8.6.1 Den Status speichern

Jede Ansicht kennt ihre Position, da Sie ihren Rahmen oder ihren Mittelpunkt abfragen können. Dadurch können Sie sehr leicht die Positionen der einzelnen Blumen auf dem Bildschirm abrufen und speichern. Der Blumentyp (grün, rosa oder blau) muss separat davon gehandhabt werden. Damit jede Ansicht melden kann, welche Art Blume sie enthält, muss die Klasse `DragView` auch diesen Wert speichern. Wenn Sie eine String-Instanzvariable hinzufügen, kann die Ansicht den Namen des verwendeten Bildes zurückgeben. Dazu erweitern Sie wie folgt das `DragView`-Interface:

```
@interface DragView : UIImageView
{
   CGPoint startLocation;
   NSString *whichFlower;
}
@property (retain) NSString *whichFlower;
@end
```

Durch die Ergänzung um diese zusätzliche Eigenschaft kann der Ansichtscontroller, dem die Blumen gehören, in seiner Datei der Voreinstellungswerte eine Liste sowohl der Farben als auch der Positionen ablegen. Im folgenden Code werden beide Werte durch eine einfache Schleife aus einer ziehbaren Ansicht abgerufen und gespeichert:

```
- (void) updateDefaults
{
   NSMutableArray *colors = [[NSMutableArray alloc] init];
   NSMutableArray *locs = [[NSMutableArray alloc] init];

   for (DragView *dv in [[self.view viewWithTag:201] subviews])
   {
      [colors addObject:dv.whichFlower];
      [locs addObject:NSStringFromCGRect(dv.frame)];
   }

   [[NSUserDefaults standardUserDefaults] setObject:colors
      forKey:@"colors"];
   [[NSUserDefaults standardUserDefaults] setObject:locs
      forKey:@"locs"];
   [[NSUserDefaults standardUserDefaults] synchronize];

   [colors release];
   [locs release];
}
```

Wie Sie sehen, verhalten sich die Voreinstellungswerte wie ein Dictionary. Sie müssen einem Objekt nur einen Schlüssel zuweisen, und das iPhone aktualisiert die Voreinstellungsdatei, die zur ID Ihrer Anwendung gehört. Die Voreinstellungswerte werden innerhalb der Sandbox Ihrer Anwendung in `Library/Preferences` gespeichert. Beim Aufruf der Synchronisierungsfunktion werden sie unmittelbar aktualisiert, ohne dass erst auf die Beendigung des Programms gewartet wird.

Die Funktion `NSStringFromCGRect()` bietet eine kompakte Möglichkeit, um Rahmeninformationen als String zu speichern. Um das Rechteck wiederherzustellen, verwenden Sie `CGRectFromString()`. Jeder dieser Aufrufe nimmt ein Argument entgegen, nämlich ein `CGRect`-Objekt im ersten Fall und ein `NSString`-Objekt im zweiten. Das `UIKit`-Framework enthält Funktionen, die Punkte und Größen sowie Rechtecke in Strings übersetzen und umgekehrt.

Die Methode `updateDefaults`, die den aktuellen Status auf der Festplatte speichert, sollte innerhalb der Methode `applicationWillTerminate:` der Anwendungsdelegierung aufgerufen werden, und zwar unmittelbar vor der Beendigung des Programms. Die Voreinstellungswerte werden gespeichert, um den letzten Status der Anwendung aufzubewahren.

```
- (void) applicationWillTerminate: (UIApplication *) application
{

    // speichert die Einstellungen beim Beenden
    [self.tbvc updateDefaults];
}
```

8.6.2 Den Status wiederherstellen

Um Ansichten wieder ins Leben zurückzurufen, müssen Sie sie in der Methode `loadView` oder `viewDidLoad` des Ansichtscontrollers neu erstellen. (Wenn Sie nicht mit den Ansichten selbst arbeiten, kann die Dauerhaftigkeit auch in der `init`-Methode des Ansichtscontrollers verankert sein.) Die Methoden müssen sämtliche Informationen über den vorhergehenden Status finden und die Oberfläche entsprechend aktualisieren.

Bei der Abfrage der Benutzervoreinstellungen prüft *Rezept 8.5*, ob die Statusdaten verfügbar sind (dies ist z. B. nicht der Fall, wenn der zurückgegebene Wert `nil` ist). Wenn Statusdaten fehlen, erstellt die Methode zufällige Blumen an zufälligen Positionen.

Hinweis

Bei der Arbeit mit umfangreichen Datenquellen können Sie das gespeicherte Objektarray in der `init`-Methode des `UIViewControllers` initialisieren und füllen und die Objekte anschließend in `loadView` oder `viewDidLoad` zeichnen. Verwenden Sie nach Möglichkeit mehrere Threads, wenn Sie mit vielen Objekten zu tun haben, um zu vermeiden, dass im Hauptthread zu viel Verarbeitung stattfindet, da das Programm sonst durch blockierende Aktualisierung der Benutzeroberfläche langsam reagiert oder einzufrieren scheint.

```
- (void) loadFlowersInView: (UIView *) backdrop
{
   // Versucht, die vorherigen Farben und Positionen zu lesen
   NSMutableArray *colors = [[NSUserDefaults standardUserDefaults]
      objectForKey:@"colors"];
   NSMutableArray *locs = [[NSUserDefaults standardUserDefaults]
      objectForKey:@"locs"];

   // Fügt die Blumen an zufälligen Positionen auf dem Bildschirm ein
   for (int i = 0; i < MAXFLOWERS; i++)
   {
      NSString *whichFlower = [[NSArray
         arrayWithObjects:@"blueFlower.png", @"pinkFlower.png",
         @"orangeFlower.png", nil] objectAtIndex:(random() % 3)];

      if (colors && ([colors count] == MAXFLOWERS)) whichFlower =
         [colors objectAtIndex:i];

      DragView *dragger = [[DragView alloc] initWithImage:[UIImage
         imageNamed:whichFlower]];
      dragger.center = randomPoint();
      dragger.userInteractionEnabled = YES;
      dragger.whichFlower = whichFlower;
      if (locs && ([locs count] == MAXFLOWERS)) dragger.frame =
         CGRectFromString([locs objectAtIndex:i]);

      [backdrop addSubview:dragger];
      [dragger release];
   }
}
```

▸ *Rezept 8.5: Den vorherigen Status abrufen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

8.7 Rezept: Dauerhaftigkeit durch Archivierung

In *Rezept 8.5* wurde Dauerhaftigkeit mithilfe des Benutzervoreinstellungssystems erreicht. Dieser Code speichert Beschreibungen der Ansichten auf dem Bildschirm und baut diese Ansichten anhand der abgerufenen Beschreibungen wieder auf. In *Rezept 8.6* gehen wir einen Schritt weiter. Statt Beschreibungen werden die Objekte selbst archiviert – oder zumindest so viel von ihnen, wie nötig ist, um sie beim Start zu rekonstruieren.

Die beiden Klassen `NSKeyedArchiver` und `NSKeyedUnarchiver` bilden eine elegante Lösung für die Archivierung von Objekten in Daten für den späteren Abruf. Diese Archivklassen stellen eine API für Objektdauerhaftigkeit dar, mit der sie Objekte aus einer Anwendungssitzung in der nächsten wiederherstellen können. In dem Beispiel sehen Sie den einfachsten möglichen Ansatz zur Archivierung. Hierbei wird ein einzelnes Wurzelobjekt gespeichert, in diesem Fall ein Array aus `DragViews`, also der Blumen.

Um eine Klasse für archivierbare Objekte zu erstellen, müssen Sie zwei Methoden definieren. Die erste, `encodeWithCoder:`, speichert alle Informationen, die nötig sind, um ein Objekt wiederherzustellen, in diesem Fall also den Rahmen und die Blume einer Ansicht. Beide werden als `NSString`-Objekte abgelegt. Die zweite Methode heißt `initWithCoder:`. Sie ruft diese Informationen ab und initialisiert die Objekte mit den gespeicherten Informationen. Im Folgenden sehen Sie die beiden Methoden, wie sie für die Klasse `DragView` definiert sind, sodass Sie damit Objekte dieser Klasse kodieren und wieder aus dem Archiv abrufen können.

```
- (void) encodeWithCoder: (NSCoder *)coder
{
   [coder encodeCGRect:self.frame forKey:@"viewFrame"];
   [coder encodeObject:self.whichFlower forKey:@"flowerType"];
}

- (id) initWithCoder: (NSCoder *)coder
{
   [super initWithFrame:CGRectZero];
   self.frame = [coder decodeCGRectForKey:@"viewFrame"];
   self.whichFlower = [coder decodeObjectForKey:@"flowerType"];
   self.image = [UIImage imageNamed:self.whichFlower];
   self.userInteractionEnabled = YES;
   return self;
}
```

Alle Elemente werden mit dem Namen eines Schlüssels gespeichert, anhand dessen Sie die gespeicherten Daten in jeder beliebigen Reihenfolge wieder abrufen können. Besondere `UIKit`-Erweiterungen ergänzen die Klasse `NSCoder` um Speichermethoden für Punkte, Größen, Rechtecke, affine Transformationen und einspringende Kanten. In diesem Beispiel wird die Rechteckmethode zum Kodieren und Dekodieren des Ansichtsrahmens genutzt.

Die Daten werden in einer Datei abgelegt, wobei Sie den Archivpfad zu dieser Datei angeben. In diesem Beispiel werden die Daten in der Datei `flowers.archive` gespeichert, die sich im Dokumentenordner der Sandbox befindet:

```
#define DATAPATH [NSString stringWithFormat: \
   @"%@/Documents/flowers.archive", NSHomeDirectory()]
```

Aber wie führen Sie die Archivierung und den Abruf für diese Oberfläche durch? *Rezept 8.6* zeigt die genauen Aufrufe, die in diesem Fall im Ansichtscontroller implementiert werden. Hier gibt es zwei benutzerdefinierte Methoden, von denen die eine die `DragViews` erfasst und in einer Datei archiviert und die andere die Ansichten aus dieser Datei abruft.

Beachten Sie, dass die zweite Methode einen booleschen Wert zurückgibt, der anzeigt, ob die Ansicht korrekt gelesen werden konnte. Falls dies nicht der Fall ist, kann das daran liegen, dass die Daten beschädigt sind oder dass die Anwendung zum ersten Mal läuft. Auf jeden Fall wird das Anwendungsfenster mit neuen Daten gefüllt. Dazu erstellt eine Ausweichmethode einen neuen Satz von Unteransichten.

```
- (void) archiveInterface
{
   NSArray *flowers = [[self.view viewWithTag:201] subviews];
   [NSKeyedArchiver archiveRootObject:flowers toFile:DATAPATH];
}

- (BOOL) unarchiveInterfaceInView: (UIView *) backdrop
{
   NSArray *flowers = [NSKeyedUnarchiver
      unarchiveObjectWithFile:DATAPATH];
   if (!flowers) return NO;

   for (UIView *aView in flowers)
      [backdrop addSubview:aView];
   return YES;
}
```

► *Rezept 8.6: Oberflächen archivieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

8.8 Rezept: Möglichkeiten zum Widerrufen hinzufügen

Möglichkeiten zum Widerrufen oder Rückgängigmachen (Undo) sind eine weitere wichtige Komponente von Oberflächen zur Direktbearbeitung. Bei einer einfachen Oberfläche ist dazu ein wenig mehr erforderlich, als jedes Objekt in die vorhergehende Position auf dem Bildschirm zurückzuversetzen. Cocoa Touch enthält die Klasse `NSUndoManager`, mit der sich Benutzeraktionen umkehren lassen.

8.8.1 Einen Undo-Manager erstellen

Definieren Sie den Undo-Manager so zentral wie möglich. Sie sollten für jeden Hauptansichtscontroller nur eine einzige Instanz dieser Klasse verwenden und diese mit den zugehörigen Kindansichten in der Oberfläche teilen. Die Methoden `viewDidLoad` und `loadView` sind geeignete Orte für die Zuweisung eines Undo-Managers.

```
// Initialisiert den Undo-Manager für diese Anwendung
self.undoManager = [[NSUndoManager alloc] init];
[self.undoManager setLevelsOfUndo:999];
[self.undoManager release];
```

Der Manager kann eine beliebige Anzahl von Widerrufsaktionen speichern, wobei Sie festlegen, wie tief dieser Stack geht. Die einzelnen Aktionen können komplex sein, sodass Gruppen von Widerrufstätigkeiten erforderlich sind, aber auch – wie in dem hier gezeigten Beispiel – sehr einfach. Hier geschieht beim Widerrufen nur eines: Eine Ansicht wird an die vorherige Position zurückgeschoben.

8.8.2 Widerrufsmöglichkeiten in Kindansichten

Alle Kinder der Klasse `UIResponder` finden den nächstgelegenen Undo-Manager in der Responder-Kette. Wenn Sie also `DragView`-Instanzen in eine Ansicht ziehen, deren Controller über einen Undo-Manager verfügt, weiß jede dieser `DragViews` dank Ihrer Eigenschaft `undoManager` über den zugehörigen Undo-Manager Bescheid. Dies ist sehr komfortabel, denn wenn Sie Widerrufsmöglichkeiten in den Hauptansichtscontroller einfügen, stehen diese Möglichkeiten ohne Zusatzaufwand auch in den Kindansichten zur Verfügung.

8.8.3 Mit Navigationsleisten arbeiten

Wenn Sie in irgendeiner Weise mit Navigationsleisten arbeiten, sollten die Kindansichten einen Zeiger auf ihren Ansichtscontroller speichern, damit sie mit den Schaltflächen auf der Navigationsleiste umgehen können. Eine **Undo**-Schaltfläche soll nur dann erscheinen, wenn auf dem Undo-Stack auch tatsächlich Elemente zur Verfügung stehen.

```
@interface DragView : UIImageView
{
    CGPoint startLocation;
```

```
    NSString *whichFlower;
    UIViewController *viewController;
}
@property (retain) NSString *whichFlower;
@property (assign) UIViewController *viewController;
@end
```

Sobald ein Undo-Element zum Manager hinzugefügt wird, soll wie in diesem Beispiel eine **Undo**-Schaltfläche erscheinen. Sie ruft die Methode `undo` des Managers auf, die wiederum das Ziel, die Aktion und den Objektsatz verwendet, die oben auf dem Undo-Stack stehen, um den eigentlichen Widerruf durchzuführen. Hat der Undo-Manager keine Elemente mehr, die rückgängig gemacht werden können, soll die **Undo**-Schaltfläche wieder ausgeblendet werden.

```
- (void) checkUndoAndUpdateNavBar
{
    while ([self.undoManager isUndoing]);

    // Zeigt die Undo-Schaltfläche bei leerem Undo-Stack nicht an
    if (!self.undoManager.canUndo)
        self.navigationItem.leftBarButtonItem = nil;
    else
        self.navigationItem.leftBarButtonItem =
            BARBUTTON(@"Undo", @selector(undo));
}

- (void) undo
{
    [self.undoManager undo];
}
```

Diese Methode wartet darauf, dass der Undo-Manager alle aktuellen Widerrufsaktionen abschließt, bevor sie die Navigationsleiste aktualisiert.

8.8.4 Undo-Elemente registrieren

Der einfachste Aufruf, um eine rückgängig zu machende Aktion (ein Undo-Element) zu registrieren, speichert die Positon des Objekts beim Beginn einer Berührungssequenz und gibt an, dass das Objekt bei einem Widerruf auf diese Ausgangsposition zurückgesetzt werden soll. Dieser Aufruf erfolgt in der Kindansicht, nicht im Ansichtscontroller. Hierbei wird der Undo-Manager angewiesen, wie er ein Objekt auf seine vorherigen Attribute zurücksetzen soll.

```
[self.undoManager registerUndoWithTarget:self
    selector:@selector(resetPosition)
    object:NSStringFromCGPoint(self.center)];
```

Eine alternative und zu bevorzugende Möglichkeit besteht darin, anstelle eines Ziels und eines Selektors einen Aufruf zu verwenden. Dieser Aufruf speichert eine Nachricht für das Zurücksetzen des Status, also eine Möglichkeit, wie er zum vorherigen Zustand zurückkehren kann. Bevor Sie den Status des Objekts ändern, führen Sie folgende Vorbereitungen durch:

```
[[self.undoManager prepareWithInvocationTarget:self]
   setPosition:self.center];
```

Für Aufrufe können Sie Methoden mit beliebiger Anzahl von Argumenten und beliebigen Argumenttypen verwenden. Der hier gezeigte Aufruf vereinfacht eine mögliche Wiederherstellung einer widerrufenen Aktion (Redo), weshalb dies die bevorzugte Vorgehensweise ist.

Es gibt verschiedene Möglichkeiten für die Undo-Registrierung in einer Oberfläche zur Direktbearbeitung. Die einfachste Lösung, die Sie auch in *Rezept 8.7* sehen, besteht darin, eine Set/Unset-Methode von `touchesBegan:withEvent:` aus aufzurufen.

Beachten Sie, dass die Ergebnisse eines Widerrufs unvorhersehbar sein können, wenn ein Benutzer ein Objekt berührt, aber wieder loslässt, ohne es zu bewegen. Sie sollten in die Routinen für das Berührungsende einen Test aufnehmen, um sicherzustellen, dass ein Objekt auch tatsächlich bewegt wurde. Wenn nicht, entfernen Sie das letzte Element vom Undo-Stack, indem Sie `undo` ausgeben.

Rezept 8.7 zeigt den eigentlichen Widerrufscode. Die Methode `setPosition:` bietet dem Undo-Manager eine Möglichkeit zum Setzen und zum Zurücksetzen der Positionen. Bei der Registrierung speichert sie die Position einer Ansicht im Undo-Stack; bei einem Widerruf schiebt sie die Ansicht zurück auf diese Position und zeigt dabei grafisch die Verbindung zwischen dem alten und dem neuen Wert an. In diesem Rezept gibt es zwar keine Möglichkeiten für eine Wiederherstellung (die finden Sie in *Rezept 8.8*), doch die Methode `setPosition:` ist wiederherstellungskompatibel. Wird sie vom Undo-Manager aufgerufen, wird der wiederholte Aufruf von `prepareWithInvocationTarget:` zum Redo-Stack hinzugefügt.

Der verzögerte Selektor in dieser Methode, `checkUndoAndUpdateNavBar:`, wird ausgelöst, nachdem die Verschiebung abgeschlossen ist. Dadurch kann die Methode `setPosition:` abgeschlossen werden, bevor der Undo-Stack geprüft wird.

Der Stack verringert seinen Zähler erst, wenn die registrierte Methode die Steuerung zurückgibt. Wenn Sie die Methode direkt aufrufen, wird die **Undo**-Schaltfläche im Navigationscontroller selbst dann nicht entfernt, wenn es keine weiteren Aktionen mehr gibt, die rückgängig gemacht werden könnten. Die `while`-Schleife, die `isDoing` überprüft, würde dann niemals verlassen, und `setPosition:` würde niemals die Steuerung zurückgeben.

```
- (void) setPosition: (CGPoint) pos
{
   [[self.undoManager prepareWithInvocationTarget:self]
      setPosition:self.center];

   [self.viewController
      performSelector:@selector(checkUndoAndUpdateNavBar)
      withObject:nil afterDelay:0.2f];
```

```
    [UIView beginAnimations:@"" context:nil];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:0.1f];

    self.center = pos; // animiere

    [UIView commitAnimations];
}

- (void) touchesBegan:(NSSet*)touches withEvent:(UIEvent*)event
{
    [self setPosition:self.center];

    // Berechnet und speichert den Offset und bringt die Ansicht ggf. in den
    // Vordergrund
    CGPoint pt = [[touches anyObject] locationInView:self];
    startLocation = pt;
    [[self superview] bringSubviewToFront:self];
}
```

► *Rezept 8.7: Eine benutzerdefinierte Undo-Routine erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

8.9 Rezept: Schüttelgesteuerte Widerrufsmöglichkeiten hinzufügen

Seit dem SDK 3.0 gibt es die neue, kuriose Funktion der schüttelgesteuerten Widerrufsunterstützung, die automatisch ein Menü zum Widerrufen und Wiederherstellen einblendet. Wenn der Benutzer sein iPhone schüttelt, erscheint das mit dem aktuellen Undo-Manager verbundene Menü. Hier kann der Benutzer die vorhergehende Aktion widerrufen oder eine widerrufene Aktion wiederherstellen. Dieses Menü sehen Sie in *Abbildung 8.3*.

Bearbeitung durch Schütteln ist ein witziges Konzept, in Anwendungen aber nicht unbedingt praxistauglich. Den Benutzern beizubringen, ihr iPhone zu schütteln, anstatt auf eine **Undo**-Schaltfläche zu tippen, erweist sich in der Praxis als eine Hürde. Selbst wenn sich die Benutzer daran gewöhnen, ist es doch umständlich, das iPhone zu schütteln, um Aktionen rückgängig zu machen. Wenn Sie diese Funktion in Ihren Anwendungen einsetzen möchten, sollten Sie sie nur als Erweiterung und nicht als Ersatz für die bestehenden Widerrufsmöglichkeiten ansehen.

▸ *Abbildung 8.3: Durch Schütteln wird ein Widerrufs- und Wiederherstellungsmenü eingeblendet.*

Möglichkeiten für die Bearbeitung durch Schütteln bereitzustellen, erfordert nur wenige Arbeitsschritte. Die folgendenden Abschnitte zeigen die einzelnen Elemente, die Sie ändern müssen, um Ihre Anwendungen um dieses Merkmal zu ergänzen.

8.9.1 Einen Aktionsnamen für Widerruf und Wiederherstellung angeben

Aktionsnamen stellen die Wörter bereit, die in dem Menü aus *Abbildung 8.3* hinter **Undo** bzw. **Redo** erscheinen. In diesem Beispiel lautet der Aktionsname »Movement«, also »Bewegung«, sodass als Menüoption für den Widerruf **Undo Movement** (Bewegung widerrufen) angezeigt wird. Um diesen Namen anzugeben, müssen Sie die Methode `setPosition:` erweitern, indem Sie die folgende Zeile nach der Vorbereitung des Aufrufziels einfügen:

```
if (![self.undoManager isUndoing])
   [self.undoManager setActionName:@"movement"];
```

Hinweis

Beachten Sie, das es bis zur derzeit aktuellen iPhone OS Version 3.1 keine sprachlich lokalisierten Varianten des Widerrufs- und Wiederherstellungsmenüs gibt. Sie müssen also zumindest im Moment noch davon ausgehen, das auch bei deutscher Spracheinstellung des Gerätes immer **Undo** ... erscheint. Vermeiden Sie daher Aktionsnamen wie »Bewegung«, da in diesem Fall **Undo Bewegung** erscheinen würde. Notfalls verzichten Sie gänzlich auf das Widerrufs- und Wiederherstellungsmenü.

8.9.2 Unterstützung für die schüttelgesteuerte Bearbeitung hinzufügen

Fügen Sie der Methode `applicationDidFinishLaunching:` der Anwendungsdelegierung die folgende Zeile hinzu. Dadurch, dass Sie die Eigenschaft `applicationSupportsShakeToEdit` ausdrücklich festlegen, aktivieren Sie die schüttelgesteuerte Bearbeitung für die gesamte Anwendung.

```
application.applicationSupportsShakeToEdit = YES;
```

8.9.3 First Responder festlegen

Damit ein Ansichtscontroller Widerruf und Wiederherstellung erledigen kann, muss er stets als Erster reagieren, also als »First Responder« festgelegt sein. Da eine Anwendung mehrere Clients für Undo-Manager haben kann, muss sie die einzelnen Undo-Manager jeweils einem bestimmten Ansichtscontroller zuweisen. Nur der First Responder empfängt Widerrufs- und Wiederherstellungsaufrufe.

Da sich der Undo-Manager gewöhnlich in einer `UIViewController`-Instanz befindet, müssen Sie die Routinen von *Rezept 8.8* Ihrem Ansichtscontroller hinzufügen. Dadurch machen Sie ihn zum First Responder, wenn er auf dem Bildschirm erscheint, und sorgen dafür, dass er seinen Undo-Manager verwendet.

```
- (BOOL)canBecomeFirstResponder {
    return YES;
}

- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];
    [self becomeFirstResponder];
}

- (void)viewWillDisappear:(BOOL)animated {
    [super viewWillDisappear:animated];
    [self resignFirstResponder];
}
```

► *Rezept 8.8: Den First Responder für die schüttelgesteuerte Bearbeitung festlegen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

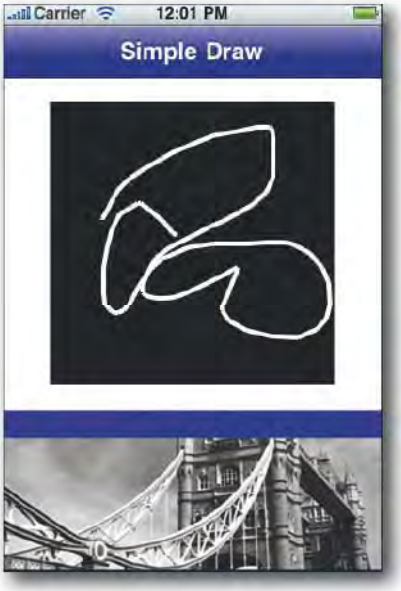
Hinweis

Um die Schüttelgestik mit dem iPhone Simulator zu verwenden, nutzen Sie **Hardware ► Schüttelgeste** ⌘ + ⇧ + Z des iPhone Simulator Menüs.

8.10 Rezept: Auf dem Bildschirm zeichnen

Die `UIView` ist nicht nur für Gesten verantwortlich, sondern auch für das direkte Zeichnen auf dem Bildschirm. Ihre Methode `drawRect:` bietet eine maschinennahe Möglichkeit zum direkten Zeichnen, wobei Sie mithilfe von Quartz 2D-Aufrufen beliebige Elemente erstellen und anzeigen können. Wenn Sie diese Möglichkeit mit Gesten kombinieren, können Sie konkrete, bearbeitbare Oberflächen erstellen.

Rezept 8.9 kombiniert Gesten mit `drawRect`, um ein Zeichnen durch Berührung zu ermöglichen. Wenn ein Benutzer den Bildschirm berührt, erfasst die Klasse `TouchView` eine Reihe von Punkten. Bei jeder Fingerbewegung ruft die Methode `touchesMoved:withEvent:` ihrerseits `setNeedsDisplay` auf, die wiederum einen Aufruf von `drawRect:` auslöst. In letzterer Methode zeichnet die Ansicht aus den erfassten Punkten eine Folge von Liniensegmenten, um auf dem Bildschirm einen Pfad anzuzeigen. *Abbildung 8.4* zeigt die Oberfläche mit einem vom Benutzer gezeichneten Pfad.



▸ *Abbildung 8.4: Für ein einfaches iPhone-Zeichenprogramm ist nicht viel mehr zu tun, als Berührungen entlang eines Pfades zu erfassen und diesen Pfad mit Quartz 2D-Aufrufen zu zeichnen.*

```
@interface TouchView : UIView
{
   NSMutableArray *points;
}
@property (retain) NSMutableArray *points;
@end

@implementation TouchView
@synthesize points;

- (BOOL) isMultipleTouchEnabled {return NO;}

// Beginnt ein neues Array
- (void) touchesBegan:(NSSet *) touches withEvent:(UIEvent *) event
```

```
{
    self.points = [NSMutableArray array];
    CGPoint pt = [[touches anyObject] locationInView:self];
    [self.points addObject:[NSValue valueWithCGPoint:pt]];
}

// Fügt jeden Punkt zum Array hinzu
- (void) touchesMoved:(NSSet *) touches withEvent:(UIEvent *) event
{
    CGPoint pt = [[touches anyObject] locationInView:self];
    [self.points addObject:[NSValue valueWithCGPoint:pt]];
    [self setNeedsDisplay];
}

// Zeichnet alle Punkte
- (void) drawRect: (CGRect) rect
{
    if (!self.points) return;
    if (self.points.count < 2) return;

    [[UIColor whiteColor] set];

    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetLineWidth(context, 4.0f);


    for (int i = 0; i < (self.points.count - 1); i++)
    {
        CGPoint pt1 = POINT(i);
        CGPoint pt2 = POINT(i+1);
        CGContextMoveToPoint(context, pt1.x, pt1.y);
        CGContextAddLineToPoint(context, pt2.x, pt2.y);
        CGContextStrokePath(context);
    }
}
@end
```

▸ *Rezept 8.9: Berührungsgesteuertes Zeichnen in einer* `UIView`

Den Rezeptcode finden

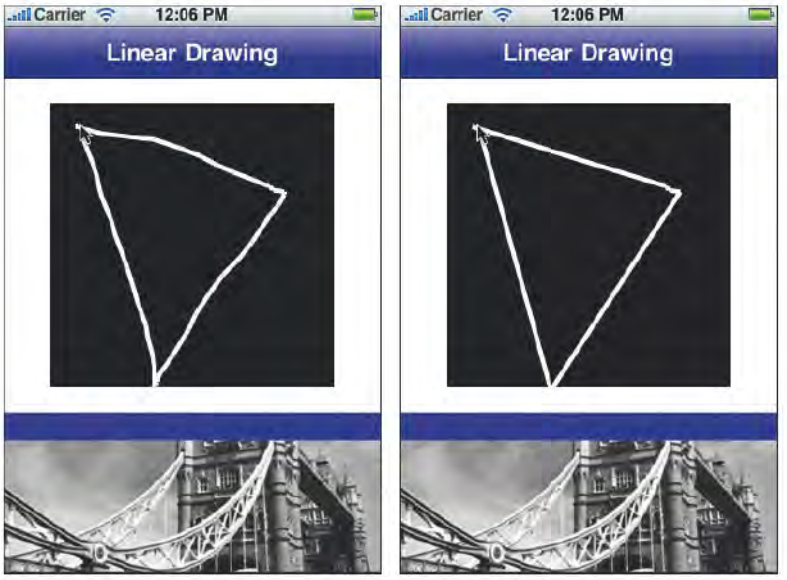
Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

8.11 Rezept: Linien berechnen

Wenn die Benutzereingabe hauptsächlich aus Berührungen besteht, kann angewandte Geometrie helfen, diese Gesten zu interpretieren. In diesem und dem folgenden Rezept wird die Benutzereingabe durch Berechnungen gefiltert, um kleinere Datenmengen zu erhalten, die für die Anwendung geeigneter sind. *Rezept 8.10* erfasst das gleiche Berührungsarray wie das in *Rezept 8.9*. Nach Abschluss der Geste, wenn also der Finger wieder vom Bildschirm gehoben wird, analysiert der Code das Array und erstellt einen minimierten Satz von Liniensegmenten aus den freihändig gezeichneten Punkten.

Durch eine verringerte Punktmenge erreichen Sie zweierlei. Erstens entsteht dadurch eine gerade, besser aussehende Darstellung. Die rechte Grafik in *Abbildung 8.5* wirkt sehr viel sauberer gezeichnet als die linke. Zweitens lässt sich die reduzierte Punktmenge besser interpretieren. Die Liniensegmente mit den sechs Punkten aus der rechten Hälfte von *Abbildung 8.5* sind weit einfacher zu analysieren als die über 50 Punkte im linken Teil.

Die zusätzlichen Liniensegmente an der oberen rechten Ecke des Dreiecks sind auf ein »Ausrutschen« des Fingers zurückzuführen. Die Interpretation der Benutzerabsicht aus einer Freihandgeste kann ein ziemlich schwieriges Problem sein. Für einen Menschen ist es zwar offensichtlich, dass der Benutzer ein Dreieck zeichnen wollte, doch Rechenalgorithmen sind niemals perfekt. Wenn Sie Gesten interpretieren müssen, ist eine gewisse großzügige Anpassung erforderlich.



► *Abbildung 8.5: Mit Rechenverfahren lassen sich Benutzereingaben interpretieren. Hier verringert ein Algorithmus zur Linienerkennung die Anzahl der Eingabepunkte und überführt die Absicht des Benutzers dadurch in eine bessere geometrische Darstellung.*

In *Rezept 8.10* wird jeweils eine Menge von drei Punkten auf einmal analysiert, wobei jeweils der erste und der dritte Punkt jedes dieser Triplets als Vektor mit dem zweiten Punkt als Ursprung aufgefasst wird. Anschließend wird das Punktprodukt dieser Vektoren berechnet. Das Ergebnis ist der

Kosinus des Winkels zwischen den beiden Vektoren. Liegen die Punkte auf einer Linie, d. h., beträgt der Winkel also ungefähr 180°, verwirft der Algorithmus den mittleren Punkt.

Der Kosinus von 180° beträgt -1. Der Code verwirft alle Punkte, deren Vektorkosinus unter -0,75 liegt. Wenn Sie die Toleranz erhöhen (indem Sie den Kosinusgrenzwert auf -0,6 oder -0,5 setzen), erhalten Sie ein geraderes Ergebnis, wobei Sie aber vom Benutzer beabsichtigte Richtungswechsel verlieren können. Besteht das Ziel darin, Dreiecke, Vierecke oder andere einfache Polygone zu erkennen, kann die Toleranz ziemlich grob eingestellt werden. Für genauere Linienzeichnungen müssen Sie die Toleranz enger fassen, um vom Benutzer absichtlich gezeichnete Details erfassen zu können.

```
// Gibt das Punktprodukt zweier normalisierter Vektoren zurück
float dotproduct (CGPoint v1, CGPoint v2)
{
   float dot = (v1.x * v2.x) + (v1.y * v2.y);
   float a = ABS(sqrt(v1.x * v1.x + v1.y * v1.y));
   float b = ABS(sqrt(v2.x * v2.x + v2.y * v2.y));
   dot /= (a * b);

   return dot;
}

// Entfernt alle Zwischenpunkte, die ungefähr auf einer Linie liegen
- (void) touchesEnded:(NSSet *) touches withEvent:(UIEvent *) event
{
   if (!self.points) return;
   if (self.points.count < 3) return;

   // Erstellt das gefilterte Array
   NSMutableArray *newpoints = [NSMutableArray array];
   [newpoints addObject:[self.points objectAtIndex:0]];
   CGPoint p1 = POINT(0);

   // Fügt nur die Punkte hinzu, bei denen ein Richtungswechsel
   // stattfindet
   for (int i = 1; i < (self.points.count - 1); i++)
   {
      CGPoint p2 = POINT(i);
      CGPoint p3 = POINT(i+1);

      // Erstellt Vektoren mit p2 als Ursprung
      CGPoint v1 = CGPointMake(p1.x - p2.x, p1.y - p2.y);
      CGPoint v2 = CGPointMake(p3.x - p2.x, p3.y - p2.y);
      float dot = dotproduct(v1, v2);

      // Der Winkel zwischen kolinearen Vektoren muss so genau wie möglich
      // 180° betragen
```

```
        if (dot < -0.75f) continue;
        p1 = p2;
        [newpoints addObject:[self.points objectAtIndex:i]];
    }

    // Fügt den Endpunkt hinzu
    if ([newpoints lastObject] != [self.points lastObject])
        [newpoints addObject:[self.points lastObject]];

    // Gibt die Anzahl der Punkte vorher und nachher aus
    NSLog(@"%@",[NSString stringWithFormat:@"%d points to %d points",
        self.points.count, newpoints.count]);

    // Aktualisiert das Bild mit den gefilterten Punkten und zeichnet es
    self.points = newpoints;
    [self setNeedsDisplay];
}
```

► *Rezept 8.10: Liniensegmente aus Freihandgesten erstellen*

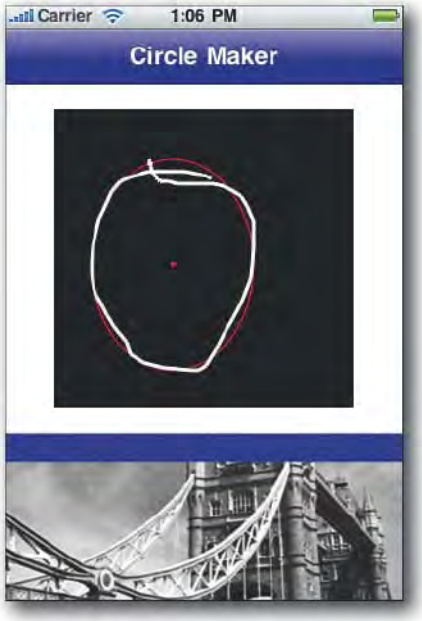
Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

8.12 Rezept: Kreise erkennen

Auf einer Oberfläche mit Direktbearbeitung wie auf dem iPhone können die meisten Menschen sicherlich einfach dadurch arbeiten, indem sie auf Objekte auf dem Bildschirm tippen. Trotzdem ist die Erkennung von Kreisen eine der gefragtesten Möglichkeiten. Entwickler lieben es, die Benutzer Kreise auf dem Bildschirm zeichnen zu lassen. Da ich Lösungen anbieten möchte, nach denen meine Leser gefragt haben, stelle ich in *Rezept 8.11* eine relativ einfache Kreiserkennung vor, deren Ergebnis Sie in *Abbildung 8.6* sehen.

In dieser Implementierung wird ein zweiphasiger Test durchgeführt. Der erste ist ein Konvergenztest. Der Anfangs- und Endpunkt der Kreislinie müssen nah genug zusammenfallen. Hier ist eine hohe Toleranz erforderlich, da die Benutzer den Anfangspunkt nur schwer treffen können, wenn sie die gemalte Linie nicht sehen. Hier ist ein großzügiger Abstand von 60 Pixeln zwischen den beiden Punkten zulässig, was etwa einem Drittel der Ansichtsgröße entspricht.



▸ *Abbildung 8.6: Der Punkt und die schwach sichtbare Ellipse sind die Hauptmerkmale des erkannten Kreises.*

Mit dem zweiten Test wird die Bewegung um einen Mittelpunkt untersucht. Die durchlaufenen Kreisbögen werden summiert, was in einem idealen Kreis 360° ergeben muss. In diesem Beispiel sind Bewegungen zulässig, die sich diesem Wert bis auf 45° annähern.

Sind beide Tests bestanden, erstellt der Algorithmus das kleinste einschließende Rechteck und zentriert es auf den geometrischen Mittelwert der Punkte, aus denen sich die ursprüngliche Geste zusammensetzt. Das Ergebnis wird der Instanzvariable `circle` zugewiesen. Das Erkennungssystem ist nicht perfekt (beim Ausprobieren des Beispielcodes können Sie versuchen, es zu überlisten), aber eignet sich gut genug, um in vielen iPhone-Anwendungen zu erkennen, ob der Benutzer einen Kreis gezeichnet hat.

```
// Ermittelt am Ende der Berührung, ob ein Kreis gezeichnet wurde
- (void) touchesEnded:(NSSet *) touches withEvent:(UIEvent *) event
{
   if (!self.points) return;
   if (self.points.count < 3) return;

   // Test 1: Anfangs- und Endpunkt dürfen nicht mehr als 60 Pixel
   // voneinander entfernt liegen
   CGRect tcircle;
   if (distance(POINT(0), POINT(self.points.count - 1)) < 60.0f)
   tcircle = [self centeredRectangle];

   // Test 2: Summiert die Winkel der zurückgelegten Strecke. Der Wert muss
   // sich auf maximal 45° (pi/4) an 360° (2 pi) annähern.
   CGPoint center = CGPointMake(CGRectGetMidX(tcircle),
      CGRectGetMidY(tcircle));
```

```
    float distance = ABS(acos(dotproduct(centerPoint(POINT(0), center),
       centerPoint(POINT(1), center))));
    for (int i = 1; i < (self.points.count - 1); i++)
       distance += ABS(acos(dotproduct(centerPoint(POINT(i), center),
          centerPoint(POINT(i+1), center))));
    if ((ABS(distance - 2 * M_PI) < (M_PI / 4.0f))) circle = tcircle;

    [self setNeedsDisplay];
}
```

► *Rezept 8.11: Kreise erkennen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

8.13 Rezept: Mehrfachberührungen erkennen

Wenn Sie die Multitouch-Interaktion in Ihren `UIView`-Ansichten aktivieren, kann das iPhone mehr als eine Fingerberührung auf einmal erfassen und darauf reagieren. Setzen Sie dazu die `UIView`-Eigenschaft `multipleTouchEnabled` auf `YES`, oder überschreiben Sie `isMultipleTouchEnabled` für die Ansicht. Bei aktivierter Multitouch-Funktion gibt jeder Berührungs-Callback eine Menge von Berührungen zurück. Ist die Anzahl dieser Menge größer als 1, haben Sie es mit einer Mehrfachberührung zu tun.

Theoretisch kann das iPhone eine beliebige Anzahl von gleichzeitigen Berührungen erfassen, doch ist die Multitouch-Interaktion auf fünf Fingerberührungen auf einmal beschränkt. Selbst das ist schon weit mehr, als die meisten Entwickler brauchen, denn es gibt nicht viele sinnvolle Gesten, die man mit fünf Fingern auf einmal durchführen kann. Das gilt vor allem dann, wenn Sie das iPhone mit der einen Hand festhalten und mit der anderen den Bildschirm berühren.

Berührungen sind nicht gruppiert. Wenn Sie den Bildschirm beispielsweise mit jeweils einem Finger jeder Hand berühren, ist es nicht möglich zu bestimmen, welche Berührung zu welcher Hand gehört. Die Berührungsreihenfolge ist willkürlich. Gruppierte Berührungen können zwar die Fingerreihenfolge während des Lebenszyklus eines einzelnen Berührungsereignisses (Finger senken, bewegen und heben) beibehalten, doch die Reihenfolge kann schon bei der nächsten Berührung anders sein. Wenn Sie die Berührungen nach linker und rechter Hand unterscheiden wollen, müssen Sie ein Berührungs-Dictionary aufstellen, das nach Berührungsobjekten indiziert ist.

Vielleicht ist es tröstlich zu wissen, dass die Erkennung zusätzlicher Fingerberührungen eingebaut ist, falls Sie so etwas benötigen. Leider neigt der Bildschirm dazu, bei drei oder mehr Berührungen

auf einmal eine oder mehrere der Spuren zu verlieren. Es ist sehr schwierig, fließende Gesten programmgesteuert nachzuverfolgen, wenn mehr als zwei Finger daran beteiligt sind.

Rezept 8.12 fügt einer `UIView` Multitouch-Unterstützung hinzu (über die Methode `isMultipleTouchEnabled`) und zeichnet Linien zwischen den einzelnen Berührungspunkten auf dem Bildschirm. Wenn Sie die Eingabe auf zwei Berührungen beschränken, ist das Ergebnis von brauchbarer Regelmäßigkeit und zeigt die Verbindungslinie zwischen den beiden Fingern stetig an. Sobald eine dritte Berührung hinzukommt, beginnen die Linien zu flackern. Das liegt daran, dass das iPhone nicht alle Berührungen stetig erfassen kann.

Leider ist die Multitouch-Erkennung nicht annähernd so stabil und vorhersehbar wie die Erfassung von Einzelberührungen. Dies können Sie bereits in diesem Rezept und noch deutlicher in *Rezept 8.13* erkennen. Die Multitouch-Interaktion ist zwar möglich und zugegeben eine bemerkenswerte Technologie, aber aufgrund ihrer Einschränkungen sollten Sie sie nur vorsichtig einsetzen und ausführlich testen, bevor Sie sie in Praxisanwendungen bereitstellen.

```
@implementation TouchView
@synthesize points;

- (BOOL) isMultipleTouchEnabled {return YES;}

- (void) touchesBegan:(NSSet *) touches withEvent: (UIEvent *) event
{
   self.points = [touches allObjects];
   [self setNeedsDisplay];
}

- (void) touchesMoved:(NSSet *) touches withEvent: (UIEvent *) event
{
   self.points = [touches allObjects];
   [self setNeedsDisplay];
}

- (void) drawRect: (CGRect) rect
{
    if (!self.points) return;
    if (self.points.count < 2) return;

    CGContextRef context = UIGraphicsGetCurrentContext();
    CGContextSetLineWidth(context, 4.0f);
    [[UIColor redColor] set];

    // Zeichnet Kreise an den einzelnen Punkten
    CGPoint pt1;
    for (int i = 0; i < self.points.count; i++)
    {
```

```
        pt1 = POINT(i);
        CGRect rect =
           CGRectMake(pt1.x - 20.0f, pt1.y - 20.0f, 40.0f, 40.0f);
        CGContextFillEllipseInRect(context, rect);
    }

    // Zeichnet Linien zwischen den einzelnen Punkten
    CGContextMoveToPoint(context, pt1.x, pt1.y);
    pt1 = POINT(0);

    for (int i = 1; i < self.points.count; i++)
    {
        pt1 = POINT(i % self.points.count);
        CGPoint pt2 = POINT((i + 1) % self.points.count);
        CGContextAddLineToPoint(context, pt2.x, pt2.y);
    }
    CGContextStrokePath(context);
}
@end
```

► *Rezept 8.12: Grundlegende Unterstützung für Mehrfachberührungen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

Hinweis

Auf der Entwicklerwebsite von Apple finden Sie viele Quellen zu Core Graphics und Quartz 2D. Viele dieser Foren, Mailinglisten und Quellcodebeispiele sind zwar nicht iPhone-spezifisch, bieten aber doch unschätzbare Hilfe zur Erweiterung Ihrer iPhone-Core-Graphics-Kenntnisse.

8.14 Rezept: Gesten erkennen

Standardmäßige Apple iPhone-Anwendungen unterstützen verschiedene Gesten, die zu einer grundlegenden Sprache für die Berührungsinteraktion geworden sind. Benutzer können auf den Bildschirm tippen, doppelt tippen, etwas wegstreichen und ziehen, und die Apple-Anwendungen interpretieren diese Gesten entsprechend. Leider stellt Apple erst ab dem SDK 3.2 für das iPad eine

öffentliche API für die Hauptarbeit bereit, weshalb Sie die Gesten selbst interpretieren müssen. *Rezept 8.13* zeigt ein System zum Erkennen von Gesten, das auf Benutzereingaben wartet und diese anschließend auswertet.

Die Unterscheidung von Gesten ist nicht trivial, vor allem wenn Sie es auch noch mit Mehrfachberührungen zu tun haben. Wie *Rezept 8.12* gezeigt hat, sind die Berührungssensoren des iPhones im Multitouch-Modus weniger zuverlässig. Beim Ziehen mit zwei Fingern etwa kann das System zwischen der Erkennung zweier und eines einzigen Fingers hin- und herspringen.

Für den Umgang mit diesen Inkonsistenzen bietet *Rezept 8.13* eine zweifache Lösung. Erstens versucht der Code die unmittelbarste Lösung zu finden, um so schnell wie möglich eine passende bekannte Geste zu einer Eingabe zu finden. Bei einer Übereinstimmung wird ein »Fertig«-Flag gesetzt, sodass die erste passende Geste gewinnt. Zweitens betrachtet der Code eine Übereinstimmung als ungültig, wenn die Benutzereingabe über einen sinnvollen Grenzwert hinaus fortgesetzt wird. Beispielsweise ist ein Antippen gewöhnlich kurz und sollte keine 20 oder gar 30 `UITouch`-Instanzen umfassen. Die folgende Liste enthält die Gesten, die *Rezept 8.13* erkennt, und gibt an, wie sie interpretiert werden:

- **Wegstreichen** Wegstreichvorgänge sind kurze Ein-Finger-Gesten in einer einzigen Hauptrichtung, also nach unten, nach oben, nach links oder rechts, wobei die Bewegung nicht zu stark von dieser Hauptrichtung abweichen darf. Der Code prüft, ob die Bewegung mindestens 16 Pixel weit in x- oder y-Richtung verläuft, ohne dabei mehr als 8 Pixel in eine andere Richtung zu streuen.
- **Kneifen** Bei einer öffnenden oder schließenden Kneifbewegung muss der Benutzer zwei Finger in einer einzigen Geste aufeinander zubewegen oder voneinander entfernen. Damit der Code diese Geste erkennt, muss die Bewegung mindestens über 8 Pixel hinweg erfolgen.
- **Antippen** Das Antippen sollte im Idealfall nur eine einzige Berührung des Bildschirms darstellen, doch können auch zusätzliche Callbacks registriert werden. *Rezept 8.13* setzt ein Limit von drei Punkten für das Antippen mit einem Finger und von zehn Punkten bei zwei Fingern. Diese große Toleranz ist wirklich erforderlich. Bei empirischen Tests haben sich die in diesem Rezept verwendeten Grenzwerte gezeigt. Dabei haben die Benutzer mit einem oder zwei Fingern auf den Bildschirm getippt, wobei der Code die dadurch hervorgerufenen `UITouch`-Instanzen gezählt hat.
- **Doppeltes Antippen** Jedes Berührungsobjekt führt einen Antippzähler, mit dem Sie bestimmen können, ob der Benutzer ein- oder zweimal auf den Bildschirm getippt hat. Ein Doppeltipp wird jedoch nur gezählt, wenn zuvor ein Einzeltipp verarbeitet wurde. Beachten Sie dieses Verhalten, wenn Sie zwischen Einzel- und Doppeltippen unterscheiden müssen.
- **Ziehen** Für die Zwecke dieses Beispiels handelt es sich beim Ziehen um alle Einzelberührungsereignisse, die weder Einzeltipp, Doppeltipp noch Durchstreichung sind.

```
@interface TouchView : UIView
{
   BOOL multitouch;
   BOOL finished;
   CGPoint startPoint;
   NSUInteger touchtype;
   NSUInteger pointCount;
   UIViewController *vc;
}
@property (assign) UIViewController *vc;
@end

@implementation TouchView
@synthesize vc;

#define SWIPE_DRAG_MIN 16
#define DRAGLIMIT_MAX 8
#define POINT_TOLERANCE 16
#define MIN_PINCH 8

- (BOOL) isMultipleTouchEnabled {return YES;}

- (void) touchesBegan:(NSSet *) touches withEvent: (UIEvent *) event
{
   finished = NO;
   startPoint = [[touches anyObject] locationInView:self];
   multitouch = (touches.count > 1);
   pointCount = 1;
}

- (void) touchesMoved:(NSSet *) touches withEvent: (UIEvent *) event
{
   pointCount++;
   if (finished) return;

   // Handhabt Mehrfachberührungen
   if (touches.count > 1)
   {
      // Erfasst Berührungen
      UITouch *touch1 = [[touches allObjects] objectAtIndex:0];
      UITouch *touch2 = [[touches allObjects] objectAtIndex:1];

      // Findet den aktuellen und vorherigen Punkt
      CGPoint cpoint1 = [touch1 locationInView:self];
      CGPoint ppoint1 = [touch1 previousLocationInView:self];
```

```
        CGPoint cpoint2 = [touch2 locationInView:self];
        CGPoint ppoint2 = [touch2 previousLocationInView:self];

        // Berechnet den Abstand zwischen den Punkten
        CGFloat cdist = distance(cpoint1, cpoint2);
        CGFloat pdist = distance(ppoint1, ppoint2);

        multitouch = YES;

        // Die Kneifbewegung muss einen Mindestabstand übersteigen
        if (ABS(cdist - pdist) < MIN_PINCH) return;

        if (cdist < pdist)
           touchtype = UITouchPinchIn;
        else
           touchtype = UITouchPinchOut;

        finished = YES;
        return;
     }
     else
     {
        // Überprüft Einzelbewegung auf Wegstreichung
        CGPoint cpoint = [[touches anyObject] locationInView:self];
        float dx = DX(cpoint, startPoint);
        float dy = DY(cpoint, startPoint);
        multitouch = NO;

        finished = YES;
        if ((dx > SWIPE_DRAG_MIN) && (ABS(dy) < DRAGLIMIT_MAX))
           touchtype = UITouchSwipeLeft;
        else if ((-dx > SWIPE_DRAG_MIN) && (ABS(dy) < DRAGLIMIT_MAX))
           touchtype = UITouchSwipeRight;
        else if ((dy > SWIPE_DRAG_MIN) && (ABS(dx) < DRAGLIMIT_MAX))
           touchtype = UITouchSwipeUp;
        else if ((-dy > SWIPE_DRAG_MIN) && (ABS(dx) < DRAGLIMIT_MAX))
           touchtype = UITouchSwipeDown;
        else
           finished = NO;
     }
}

- (void) touchesEnded:(NSSet *) touches withEvent: (UIEvent *) event
{
   // Geste wurde nicht als Wegstreichung erkannt
```

```
if (!finished && !multitouch)
{
   // Einzel- oder Doppeltipp
   if (pointCount < 3)
   {
      if ([[touches anyObject] tapCount] == 1)
         touchtype = UITouchTap;
      else
         touchtype = UITouchDoubleTap;
   }
   else
      touchtype = UITouchDrag;
}

// Gehen die Punkte über die einer Durchstreichung hinaus?
if (finished && !multitouch)
{
   if (pointCount > POINT_TOLERANCE) touchtype = UITouchDrag;
}

// Ist dies wirklich ein Einzel- oder Doppeltipp?
if (multitouch || (touches.count > 1))
{
   // Die Toleranz ist *sehr* hoch
   if (pointCount < 10)
   {
      if ([[touches anyObject] tapCount] == 1)
         touchtype = UITouchMultitouchTap;
      else
         touchtype = UITouchMultitouchDoubleTap;
   }
}

NSString *whichItem = nil;
if (touchtype == UITouchUnknown)
   whichItem = @"Unknown";
else if (touchtype == UITouchTap)
   whichItem = @"Tap";
else if (touchtype == UITouchDoubleTap)
   whichItem = @"Double Tap";
else if (touchtype == UITouchDrag)
   whichItem = @"Drag";
else if (touchtype == UITouchMultitouchTap)
   whichItem = @"Multitouch Tap";
else if (touchtype == UITouchMultitouchDoubleTap)
```

```
        whichItem = @"Multitouch Double Tap";
    else if (touchtype == UITouchSwipeLeft)
        whichItem = @"Swipe Left";
    else if (touchtype == UITouchSwipeRight)
        whichItem = @"Swipe Right";
    else if (touchtype == UITouchSwipeUp)
        whichItem = @"Swipe Up";
    else if (touchtype == UITouchSwipeDown)
        whichItem = @"Swipe Down";
    else if (touchtype == UITouchPinchIn)
        whichItem = @"Pinch In";
    else if (touchtype == UITouchPinchOut)
        whichItem = @"Pinch Out";

    [self.vc performSelector:@selector(updateState:withPoints:)
        withObject:whichItem
        withObject:[NSNumber numberWithInt:pointCount]];
}
@end
```

▸ *Rezept 8.13: Gesten interpretieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 8 und öffnen das Projekt zu diesem Rezept.

8.15 Ein letzter Punkt: Interaktives Zoomen und Drehen

Wie die Rezepte in diesem Kapitel gezeigt haben, kann das iPhone auf vielfältige Weise reagieren, wenn Sie gewillt sind, sich stärker auf die Mathematik einzulassen. *Listing 8.1* zeigt, was alles möglich ist, wenn Sie die in diesem Kapitel verwendete Klasse `DragView` mit Beispielcode von Apple für eine berührbare, interaktive Ansicht kombinieren, die auf Einzel- und Doppeltipps hin Objekte verschiebt, dreht und zoomt.

Diese Implementierung, deren Merkmale auf Apple zurückgehen und für deren Fehler ich allein verantwortlich bin, speichert zu Beginn einer Berührung eine Menge von Punkten. Anschließend führt sie analog zum Fortschreiten der Berührung inkrementelle affine Transformationen durch, vergleicht die Berührungspunkte mit den Anfangspositionen und aktualisiert die Ansichtstransformation in Echtzeit.

Dies ist eine komplizierte Vorgehensweise für die Direktbearbeitung, aber die Ergebnisse sind überwältigend. Die Klasse reagiert unmittelbar auf die Benutzereingaben, um die berührten Ansichten entsprechend umzuformen.

```
@implementation DragView

// Bereitet das Ziehen der Ansicht vor
- (id) initWithImage: (UIImage *) anImage
{
   if (self = [super initWithImage:anImage])
   {
      self.userInteractionEnabled = YES;
      self.multipleTouchEnabled = YES;
      self.exclusiveTouch = NO;
      originalSize = anImage.size;
      originalTransform = CGAffineTransformIdentity;
      touchBeginPoints = CFDictionaryCreateMutable(NULL, 0,
         NULL, NULL);
   }
   return self;
}

// Erstellt eine inkrementelle Transformation, die der aktuellen
// Berührungsmenge entspricht
- (CGAffineTransform)incrementalTransformWithTouches:(NSSet *)touches
{
   // Sortiert die Berührungen nach Speicheradressen
   NSArray *sortedTouches = [[touches allObjects]
      sortedArrayUsingSelector:@selector(compareAddress:)];
   NSInteger numTouches = [sortedTouches count];

   // Gibt es keine Berührungen, wird die Identitätstransformation
   // zurückgegeben.
   if (numTouches == 0) return CGAffineTransformIdentity;

   // Behandelt Einzelberührung als Translation
   if (numTouches == 1) {
      UITouch *touch = [sortedTouches objectAtIndex:0];
      CGPoint beginPoint = *(CGPoint *)
         CFDictionaryGetValue(touchBeginPoints, touch);
      CGPoint currentPoint = [touch locationInView:self.superview];
      return CGAffineTransformMakeTranslation(currentPoint.x -
         beginPoint.x, currentPoint.y - beginPoint.y);
   }

   // Bei zwei oder mehr Berührungen werden die ersten beiden verarbeitet
   UITouch *touch1 = [sortedTouches objectAtIndex:0];
   UITouch *touch2 = [sortedTouches objectAtIndex:1];
```

```
    CGPoint beginPoint1 = *(CGPoint *)
        CFDictionaryGetValue(touchBeginPoints, touch1);
    CGPoint currentPoint1 = [touch1 locationInView:self.superview];
    CGPoint beginPoint2 = *(CGPoint *)
        CFDictionaryGetValue(touchBeginPoints, touch2);
    CGPoint currentPoint2 = [touch2 locationInView:self.superview];

    double layerX = self.center.x;
    double layerY = self.center.y;

    double x1 = beginPoint1.x - layerX;
    double y1 = beginPoint1.y - layerY;
    double x2 = beginPoint2.x - layerX;
    double y2 = beginPoint2.y - layerY;
    double x3 = currentPoint1.x - layerX;
    double y3 = currentPoint1.y - layerY;
    double x4 = currentPoint2.x - layerX;
    double y4 = currentPoint2.y - layerY;

    // Löst folgendes Gleichungssystem:
    // [a b t1, -b a t2, 0 0 1] * [x1, y1, 1] = [x3, y3, 1]
    // [a b t1, -b a t2, 0 0 1] * [x2, y2, 1] = [x4, y4, 1]

    double D = (y1-y2)*(y1-y2) + (x1-x2)*(x1-x2);
    if (D < 0.1) {
        return CGAffineTransformMakeTranslation(x3-x1, y3-y1);
    }

    double a = (y1-y2)*(y3-y4) + (x1-x2)*(x3-x4);
    double b = (y1-y2)*(x3-x4) - (x1-x2)*(y3-y4);
    double tx = (y1*x2 - x1*y2)*(y4-y3) - (x1*x2 + y1*y2)*(x3+x4) +
        x3*(y2*y2 + x2*x2) + x4*(y1*y1 + x1*x1);
    double ty = (x1*x2 + y1*y2)*(-y4-y3) + (y1*x2 - x1*y2)*(x3-x4) +
        y3*(y2*y2 + x2*x2) + y4*(y1*y1 + x1*x1);

    return CGAffineTransformMake(a/D, -b/D, b/D, a/D, tx/D, ty/D);
}

// Speichert die Punkte zwischen, an denen die einzelnen Berührungen
// begannen
- (void)cacheBeginPointForTouches:(NSSet *)touches
{
    for (UITouch *touch in touches) {
        CGPoint *point = (CGPoint *)
            CFDictionaryGetValue(touchBeginPoints, touch);
```

```
        if (point == NULL) {
            point = (CGPoint *)malloc(sizeof(CGPoint));
            CFDictionarySetValue(touchBeginPoints, touch, point);
        }
        *point = [touch locationInView:self.superview];
    }
}

// Löscht Berührungen aus dem Cache
- (void)removeTouchesFromCache:(NSSet *)touches
{
    for (UITouch *touch in touches) {
        CGPoint *point = (CGPoint *)
              CFDictionaryGetValue(touchBeginPoints, touch);
        if (point != NULL) {
            free((void *)CFDictionaryGetValue(touchBeginPoints,
                touch));
            CFDictionaryRemoveValue(touchBeginPoints, touch);
        }
    }
}

// Beschränkt den Zoom auf einen Max- und einen Min-Wert
- (void) setConstrainedTransform: (CGAffineTransform) aTransform
{
    self.transform = aTransform;
    CGAffineTransform concat;
    CGSize asize = self.frame.size;

    if (asize.width > MAXZOOM * originalSize.width)
    {
        concat = CGAffineTransformConcat(self.transform,
            CGAffineTransformMakeScale((MAXZOOM * originalSize.width /
            asize.width), 1.0f));
        self.transform = concat;
    }
    else if (asize.width < MINZOOM * originalSize.width)
    {
        concat = CGAffineTransformConcat(self.transform,
            CGAffineTransformMakeScale((MINZOOM * originalSize.width /
            asize.width), 1.0f));
        self.transform = concat;
    }
    if (asize.height > MAXZOOM * originalSize.height)
    {
```

```
        concat = CGAffineTransformConcat(self.transform,
            CGAffineTransformMakeScale(1.0f, (MAXZOOM *
            originalSize.height / asize.height)));
        self.transform = concat;
    }
    else if (asize.height < MINZOOM * originalSize.height)
    {
        concat = CGAffineTransformConcat(self.transform,
            CGAffineTransformMakeScale(1.0f, (MINZOOM *
            originalSize.height / asize.height)));
        self.transform = concat;
    }
}

// Wendet Berührungen an, um Transformationen zu erstellen
- (void)updateOriginalTransformForTouches:(NSSet *)touches
{
    if ([touches count] > 0) {
        CGAffineTransform incrementalTransform = [self
            incrementalTransformWithTouches:touches];
        [self setConstrainedTransform:
            CGAffineTransformConcat(originalTransform,
            incrementalTransform)];
        originalTransform = self.transform;
    }
}

// Speichert zu Beginn den Berührungsanfangspunkt und legt die erste
// Transformation fest
- (void)touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
    [[self superview] bringSubviewToFront:self];
    NSMutableSet *currentTouches = [[[event touchesForView:self]
        mutableCopy] autorelease];
    [currentTouches minusSet:touches];
    if ([currentTouches count] > 0) {
        [self updateOriginalTransformForTouches:currentTouches];
        [self cacheBeginPointForTouches:currentTouches];
    }
    [self cacheBeginPointForTouches:touches];
}

// Aktualisiert während der Bewegung die Transformation, damit sie mit
// den Berührungen übereinstimmt
```

```
- (void)touchesMoved:(NSSet *)touches withEvent:(UIEvent *)event
{
   CGAffineTransform incrementalTransform = [self
      incrementalTransformWithTouches:[event touchesForView:self]];
   [self setConstrainedTransform:
      CGAffineTransformConcat(originalTransform,
      incrementalTransform)];
}

// Fertigstellung durch Entfernen der Berührungen und Handhabung der
// Doppeltippanforderungen
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
   [self updateOriginalTransformForTouches:[event
      touchesForView:self]];
   [self removeTouchesFromCache:touches];

   for (UITouch *touch in touches) {
      if (touch.tapCount >= 2) {
         [self.superview bringSubviewToFront:self];
      }
   }

   NSMutableSet *remainingTouches = [[[event touchesForView:self]
      mutableCopy] autorelease];
   [remainingTouches minusSet:touches];
   [self cacheBeginPointForTouches:remainingTouches];
}

// Leitet Abbruchberührung an touchesEnded weiter
- (void)touchesCancelled:(NSSet *)touches withEvent:(UIEvent *)event
{
   [self touchesEnded:touches withEvent:event];
}

- (void)dealloc {
   if (touchBeginPoints) CFRelease(touchBeginPoints);
   [super dealloc];
}
@end
```

► *Listing 8.1: Ansichten in der Größe verändern und drehen*

8.16 Zusammenfassung

`UIViews` stellen die Komponenten bereit, die der Benutzer auf dem Bildschirm sieht. Durch Gesten können Benutzer über die Klasse `UITouch` mit diesen Ansichten arbeiten. In diesem Kapitel haben Sie gesehen, dass berührungsgesteuerte Oberflächen selbst in ihrer einfachsten Form auf eine leicht zu implementierende Weise eine Vielfalt an leistungsstarken Möglichkeiten bieten. Sie haben gelernt, wie Sie Ansichten auf dem Bildschirm verschieben und wie Sie diese Bewegung einschränken können. Außerdem wissen Sie jetzt, wie Sie Ansichten auf Berührungen testen, um festzustellen, ob sie darauf reagieren sollen oder nicht. In mehreren Rezepten wurden die Dauerhaftigkeit und Widerrufsmöglichkeiten für Oberflächen mit Direktbearbeitung behandelt. Sie haben auch gesehen, wie Sie in einer Ansicht zeichnen und wie Sie Berührungen verarbeiten, um die verschiedenen Gesten zu erkennen und darauf zu reagieren. Bevor Sie weiterlesen, sollten Sie sich über folgende Aspekte der Rezepte in diesem Kapitel im Klaren sein:

- Seien Sie konkret. Das iPhone verfügt über einen perfekten Berührungsbildschirm. Warum sollten Sie den Benutzern nicht erlauben, Elemente mit den Fingern zu verschieben? Das lässt die Anwendung realer wirken und betont die interaktive Natur der Plattform.
- Die meisten Benutzer haben fünf Finger an jeder Hand. Beschränken Sie die Oberfläche nicht auf Berührungen durch einen Finger, wenn es sinnvoll ist, auch Mehrfachberührungen einzusetzen.
- Solide Grundkenntnisse in Quartz-Grafiken und Core Animation sind von Vorteil. Mit `drawRect:` können Sie jede Art von benutzerdefinierter `UIView`-Darstellung erstellen, die Sie brauchen, z. B. Text, Bézier-Kurven, Kritzeleien usw.
- Forschen Sie selbst nach! In diesem Kapitel konnten die Möglichkeiten zur Direktbearbeitung in Anwendungen nur gestreift werden. Verwenden Sie den dargebotenen Stoff als Ausgangspunkt, um alle Möglichkeiten der Klasse `UITouch` kennenzulernen.

9 Steuerelemente erstellen und verwenden

Die Klasse `UIControl` bildet die Grundlage für viele interaktive Elemente des iPhones, darunter Schaltflächen, Textfelder, Schieberegler und Schalter. Diese Bildschirmobjekte haben mehr gemeinsam als die ihres Vorgängers. Alle Steuerelemente verwenden ähnliche Ansätze für Layout und Target-Action. Dieses Kapitel behandelt Steuerelemente und ihre Anwendung. Sie erfahren, wie Steuerelemente auf verschiedenste Weisen erstellt und angepasst werden können. In diesem Kapitel finden Sie eine große Vielzahl von Rezepten zu Steuerelementen, zunächst einfache, dann immer ausgeklügeltere, die Sie in Ihren Programmen verwenden können.

9.1 Die Klasse UIControl

Die Steuerelemente auf dem iPhone gehören zu einer Bibliothek vorgefertigter Bildschirmobjekte für die Benutzerinteraktion. Dazu gehören Schaltflächen und Textfelder, Schieberegler und Schalter sowie andere von Apple bereitgestellte Objekte. Die Rolle eines Steuerelements besteht darin, Benutzeraktionen in Callbacks umzuwandeln. Die Benutzer berühren und betätigen die Steuerelemente und kommunizieren dadurch mit der Anwendung.

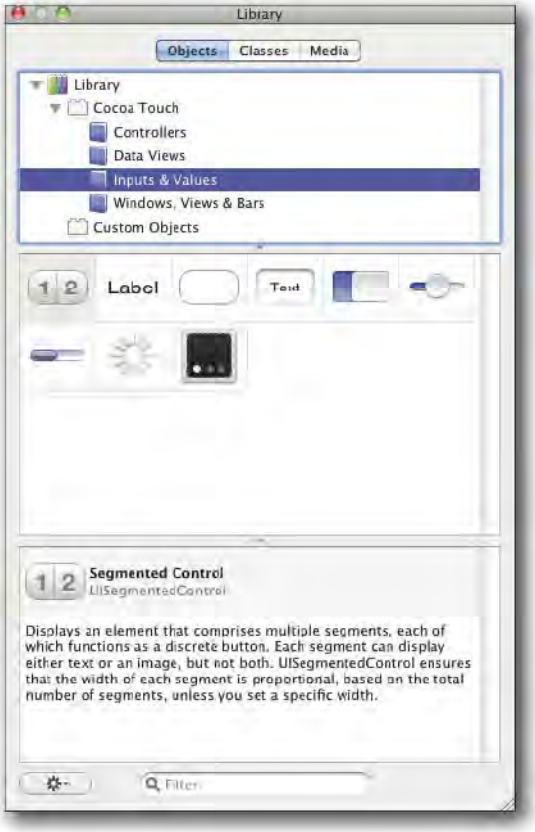
Die Klasse `UIControl` bildet die Wurzel des Baums der Steuerelementklassen. Alle Steuerelemente definieren eine grafische Schnittstelle und implementieren Möglichkeiten, um Nachrichten auszugeben, wenn die Benutzer diese Schnittstellen betätigen. Diese Nachrichten werden mithilfe von Target-Action gesendet. Wenn Sie ein neues Steuerelement definieren, geben Sie an, welche Nachricht gesendet wird, wann dies geschieht und wer sie empfängt.

9.1.1 Arten von Steuerelementen

Zu den Mitgliedern der `UIControl`-Familie gehören Schaltflächen, unterteilte Steuerelemente, Schalter, Schieberegler, Seitensteuerelemente und Textfelder, die Sie alle in der Objektbibliothek von Interface Builder (**Tools ▸ Library ▸ Objects**) im Abschnitt **Inputs & Values** finden (siehe *Abbildung 9.1*). Die Steuerelementobjekte gehören dabei zu den Eingaben (**Inputs**), während die Beschriftung, die Fortschritts- und die Aktivitätsanzeige zu den Werten (**Values**) gehören.

9.1.2 Steuerelementereignisse

Steuerelemente reagieren hauptsächlich auf drei Arten von Ereignissen: Berührungen, Wertänderungen und Bearbeitungen. *Tabelle 9.1* führt sämtliche verfügbaren Ereignistypen für Steuerelemente auf.



▸ *Abbildung 9.1: Interface Builder gruppiert Steuerelemente zusammen im Abschnitt* **Inputs & Values** *der Objektbibliothek.*

Ereignis	Typ	Verwendung
`UIControlEvent TouchDown`	Berührung	Ein Touch-Down-Ereignis irgendwo in den Grenzen des Steuerelements
`UIControlEvent TouchUpInside`	Berührung	Ein Touch-Up-Ereignis irgendwo in den Grenzen des Steuerelements. Dies ist der gebräuchlichste Ereignistyp für Schaltflächen.
`UIControlEvent TouchUpOutside`	Berührung	Ein Touch-Up-Ereignis außerhalb der Grenzen des Steuerelements
`UIControlEvent TouchDragEnter` `UIControlEvent TouchDragExit`	Berührung	Ziehvorgänge, die in die Grenzen des Steuerelements hinein- oder von dort herausführen
`UIControlEvent TouchDragInside` `UIControlEvent TouchDragOutside`	Berührung	Ziehvorgänge, die vollständig innerhalb oder vollständig außerhalb der Grenzen des Steuerelements ablaufen
`UIControlEvent TouchDownRepeat`	Berührung	Wiederholte Touch-Down-Ereignisse mit einem `tapCount` größer 1, also ein Doppeltippen
`UIControlEvent TouchCancel`	Berührung	Ein Systemereignis, das die aktuelle Berührung abbricht. Weitere Einzelheiten über Phasen und Lebenszyklen von Berührungen finden Sie in Kapitel 8, *Gesten und Berührungen*.
`UIControlEvent AllTouchEvents`	Berührung	Eine Maske, mit der Sie jegliche Berührungsereignisse erfassen können
`UIControlEvent ValueChanged`	Wert	Ein vom Benutzer ausgelöstes Ereignis, bei dem der Wert eines Steuerelements geändert wird, z. B. durch Verschieben eines Reglers oder Betätigen eines Schalters
`UIControlEvent EditingDidBegin` `UIControlEvent EditingDidEnd`	Bearbeitung	Berührungen innerhalb oder außerhalb eines `UITextField`. Eine Berührung innerhalb beginnt eine Bearbeitungssitzung, eine Berührung außerhalb beendet sie.
`UIControlEvent EditingChanged`	Bearbeitung	Eine Bearbeitung, bei der der Inhalt eines `UITextField` geändert wird

`UIControlEvent EditingDidEndOnExit`	Bearbeitung	Ein Ereignis, das eine Bearbeitungssitzung beendet. Dabei muss es sich nicht unbedingt um eine Berührung außerhalb der Grenzen des Textfeldes handeln.
`UIControlEvent AllEditingEvents`	Bearbeitung	Eine Maske für sämtliche Bearbeitungsereignisse
`UIControlEvent ApplicationReserved`	Anwendung	Ein anwendungsspezifischer Ereignisbereich (selten verwendet, falls überhaupt)
`UIControlEvent SystemReserved`	System	Ein systemspezifischer Ereignisbereich (selten verwendet, falls überhaupt)
`UIControlEvent AllEvents`	Berührung, Wert, Bearbeitung, Anwendung, System	Eine Maske für alle Berührungs-, Wertänderungs-, Bearbeitungs-, Anwendungs- und Systemereignisse

▸ *Tabelle 9.1: Ereignistypen für* `UIControl`

Ereignistypen werden meistens nach den folgenden Regeln eingesetzt: Für Schaltflächen werden Berührungsereignisse verwendet, wobei `UIControlEventTouchUPInside` für fast die gesamte Schaltflächeninteraktion da ist. Wertänderungsereignisse (also `UIControlEventValueChanged`) entsprechen den vom Benutzer ausgelösten Änderungen von unterteilten Steuerelementen, Schaltern, Schiebereglern und Seitensteuerelementen. Wenn der Benutzer einen Schalter oder Regler betätigt oder auf diese Objekte tippt, ändert sich der Wert des Steuerelements. `UITextField`-Objekte lösen Bearbeitungsereignisse aus. Die Benutzer verursachen diese Ereignisse, indem sie in das Textfeld tippen (oder außerhalb) oder den Inhalt des Feldes ändern.

Wie alle anderen Bestandteile der grafischen Benutzeroberfläche des iPhones können Sie auch Steuerelemente in Interface Builder (IB) gestalten oder direkt in Xcode erstellen. In diesem Kapitel geht es um den IB-Ansatz, wobei der Schwerpunkt jedoch auf dem Code liegt. Wenn Sie das IB-Layout einmal beherrschen, ist das Vorgehen unabhängig von dem betreffenden Element meistens ziemlich ähnlich. Sie platzieren ein Objekt auf der Oberfläche, passen es mit den Informationsfeldern an und verbinden es mit anderen IB-Objekten.

9.2 Schaltflächen

Mit `UIButton`-Instanzen können Sie einfache Schaltflächen erstellen. Die Benutzer können darauf tippen, um via Target-Action-Programmierung einen Callback auszulösen. Dabei geben Sie an, wie die Schaltfläche aussehen soll, welche Grafik dafür verwendet wird und welcher Text darauf zu sehen ist.

Auf dem iPhone haben Sie zwei verschiedene Möglichkeiten, um Schaltflächen zu erstellen: Sie können einen vorgefertigten Schaltflächentyp verwenden oder einen eigenen gestalten. Das derzeitige

iPhone SDK enthält die folgenden vorgefertigten Typen. Wie Sie sehen, sind diese Schaltflächentypen ziemlich speziell gehalten. Sie sind im SDK vor allem zum Nutzen von Apple enthalten und nicht zu Ihrem eigenen Zwecken. Trotzdem können Sie sie in Ihren Programmen bei Bedarf verwenden. *Abbildung 9.2* zeigt die verschiedenen Schaltflächen.

▸ *Abbildung 9.2: Das iPhone-SDK enthält fünf vorgefertigte Schaltflächentypen, die Sie in Interface Builder oder direkt in Ihrer Anwendung verwenden können. Von links nach rechts handelt es sich um das Einblenddreieck, eine helle und eine dunkle Info-Schaltfläche, die Schaltfläche zum Hinzufügen von Kontakten und ein abgerundetes Rechteck.*

- **Einblenddreieck** Hierbei handelt es sich um den gleichen blauen Kreis mit Pfeilspitze, den Sie auch sehen, wenn Sie Zellen einer Tabelle mit einem Hilfselement für weitere Einzelheiten versehen. Einbblenddreiecke werden in Tabellen verwendet und führen zu einem Bildschirm mit weiteren Einzelheiten zu der zurzeit ausgewählten Zelle.
- **Info-Schaltfläche hell bzw. dunkel** Diese beiden Schaltflächen zeigen ein kleines *i* in einem Kreis, wie Sie es auch bei einem Dashboard-Widget auf dem Macintosh finden. Sie werden bei den Anwendungen *Aktien* und *Wetter* dazu verwendet, in der Ansicht von einer Seite auf die nächste zu wechseln.
- **Kontakt hinzufügen** Dieser blaue Kreis mit einem weißen Pluszeichen in der Mitte ist in der Anwendung *Mail* dazu da, der Empfängerliste neue E-Mail-Adressen aus der *Kontakt* Anwendung hinzuzufügen.
- **Abgerundetes Rechteck** Diese Schaltfläche erstellt ein einfaches abgerundetes Rechteck auf dem Bildschirm, das den Text der Schaltfläche umrandet. In ihrem Standardzustand ist dies keine besonders schöne Schaltfläche (sie sieht nicht unbedingt nach Apple aus), aber sie ist einfach zu programmieren und in Anwendungen zu nutzen.

Um eine vorgefertigte Schaltfläche zu verwenden, weisen Sie sie zu, legen ihren Rahmen fest und fügen ein Zielobjekt hinzu. Machen Sie sich keine Gedanken über die individuelle Gestaltung oder das allgemeine Erscheinungsbild der Schaltfläche. Das SDK kümmert sich darum. Das folgende Listing zeigt, wie Sie eine einfache, abgerundete, rechteckige Schaltfläche erstellen können. Beachten Sie, dass `buttonWithType:` ein automatisch freigegebenes Objekt zurückgibt.

```
UIButton *button = [UIButton buttonWithType:UIButtonTypeRoundedRect];
[button setFrame:CGRectMake(0.0f, 0.0f, 80.0f, 30.0f)];
[button setCenter:CGPointMake(160.0f, 208.0f)];
[button setTitle:@"Beep" forState:UIControlStateNormal];
[button addTarget:self action:@selector(playSound)
   forControlEvents:UIControlEventTouchUpInside];
[contentView addSubview:button];
```

Um eine der anderen Arten von Standardschaltflächen anzulegen, lassen Sie die Titelzeile weg. Das abgerundete Rechteck ist der einzige der vorgefertigten Schaltflächentypen, der eine Überschrift verwendet.

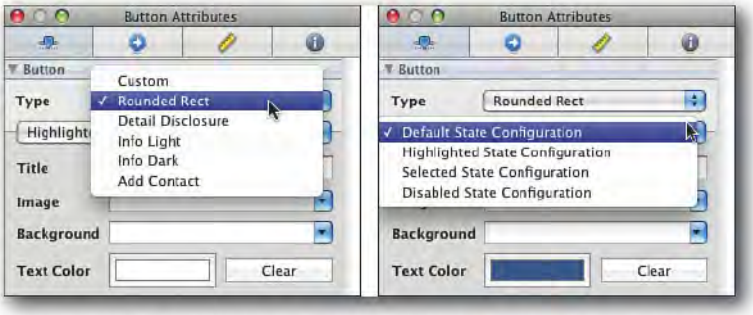
Die meisten Schaltflächen verwenden einen `TouchUpInside`-Trigger, bei dem die Berührung innerhalb der Grenzen der Schaltflächen endet. Die Standards für die Benutzeroberflächen von iPhone-Anwendungen erlauben es dem Benutzer, die Betätigung einer Schaltfläche abzubrechen, indem er den Finger von der Schaltfläche hinwegzieht, bevor er ihn wieder anhebt. Das Ereignis `UIControlEventTouchUpInside` spiegelt diesen Standard wider.

Bei der Verwendung von vorgefertigten Schaltflächen *müssen* Sie den Vorschriften zur Verwendung dieser Steuerelemente aus den Richtlinien von Apple für Benutzeroberflächen folgen. Wenn Sie beispielsweise ein Einblenddreieck verwenden, um den Benutzer zu einer Informationsseite zu führen, kann dies dazu führen, dass Ihre Anwendung im App Store abgelehnt wird. Eine Verwendung mag als sinnvolle Extrapolation der Rolle einer Schaltfläche erscheinen, aber wenn sie nicht wortwörtlich den Vorstellungen entspricht, die Apple von der Nutzung des betreffenden Steuerelements hat, kann das dazu führen, dass die Anwendung die Beurteilung nicht besteht. Um solche Probleme zu vermeiden, sollten Sie nach Möglichkeit abgerundete Rechtecke und selbst gestaltete Schaltflächen verwenden.

9.3 Schaltflächen in Interface Builder hinzufügen

Schaltflächen erscheinen in der Bibliothek von Interface Builder als `Button`-Objekte vom Typ `Rounded Rect`. Um sie zu verwenden, müssen Sie sie auf Ihre Oberfläche ziehen. Anschließend können Sie sie mithilfe des Attribut-Informationsfeldes ⌘ + 1 in einen anderen Schaltflächentyp ändern. Oben im Informationsfeld erscheint ein Einblendmenü mit Schaltflächentypen, wie Sie in *Abbildung 9.3* sehen. Wählen Sie aus diesem Menü den gewünschten Typ aus.

Wenn auf der Schaltfläche eine Beschriftung erscheinen soll (wie das Wort **Button** in *Abbildung 9.2*), geben Sie diesen Text in das Feld **Title** ein. Aus den Einblendmenüs **Image** und **Background** können Sie ein Vordergrund- und ein Hintergrundbild für die Schaltfläche auswählen.



▶ *Abbildung 9.3: Wählen Sie aus dem Einblendmenü* **Type** *im Attribut-Informationsfeld den Schaltflächentyp aus (links). Änderungen im Abschnitt* **Button** *wirken sich auf die aktuelle Konfiguration aus (rechts).*

Jede Schaltfläche weist vier Konfigurationseinstellungen auf, wie Sie in *Abbildung 9.3* (rechts) sehen. Die vier Schaltflächenzustände sind *Default* (Standard – die Schaltfläche befindet sich im normalen Zustand), *Highlighted* (hervorgehoben – der Benutzer berührt gerade die Schaltfläche), *Selected* (ausgewählt – die »Ein«-Version von Schaltflächen, die als Umschalter ausgeführt sind) und *Disabled* (deaktiviert – die Schaltfläche steht für den Benutzer nicht zur Verfügung).

Änderungen im Abschnitt **Button Attributes ▸ Button ▸ Configuration** (dem abgedunkelten Rechteck unterhalb des Konfigurationseinblendmenüs) wirken sich auf die zurzeit ausgewählte Konfiguration aus. Damit können Sie z. B. für den Standard- und den deaktivierten Zustand einer Schaltfläche jeweils eine andere Textfarbe auswählen.

Um eine Vorschau auf die einzelnen Zustände zu erhalten, aktivieren Sie die drei Markierungsfelder in **Button Attributes ▸ control ▸ Content**. Die Optionen **Highlighted**, **Selected** und **Enabled** zeigen Ihnen jeweils den entsprechenden Schaltflächenzustand. Nach der Vorschau und vor der Kompilierung müssen Sie die Schaltfläche wieder in den Zustand zurückversetzen, die sie bei der ersten Ausführung der Anwendung haben soll.

9.3.1 Grafik

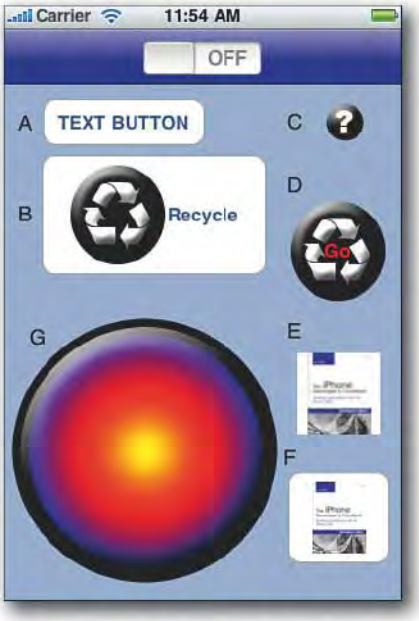
Neben den vorgefertigten Typen (Einblenddreieck, Info-Schaltfläche und Schaltfläche zum Hinzufügen von Kontakten) können Sie auch Schaltflächen mit eigener Grafik verwenden. *Abbildung 9.4* zeigt eine Auswahl verschiedener Schaltflächen, die mit den Schaltflächenklassen `Rounded Rect` und `Custom` erstellt wurden.

In dieser Abbildung sehen Sie auch, dass Sie bei `Rounded Rect`-Schaltflächen nicht auf reinen Text beschränkt sind wie bei Schaltfläche A. Sie können auch ein Bild mit dem Text kombinieren (B) oder anstelle von Text verwenden (F). Es ist auch möglich, den Hintergrund einer Schaltfläche in Form eines abgerundeten Rechtecks durch eine eigene Grafik zu ersetzen (E), was aber meistens nur wenig sinnvoll ist.

Benutzerdefinierte Schaltflächen haben kein vorgegebenes Erscheinungsbild. Sie können ihnen jede beliebige Größe geben (C und G) und Text mit dem Attribut-Informationsfeld hinzufügen (D). Was aus *Abbildung 9.4* nicht unmittelbar hervorgeht, ist die Tatsache, dass die drei genannten Schaltflächen auch Beispiele für andere Designentscheidungen sind.

Schaltfläche D verwendet dieselbe Grafik wie Schaltfläche B. Da es sich um eine benutzerdefinierte Schaltfläche handelt, wird der Text zentriert und nicht auf einem gewölbten Hintergrund platziert. Abgesehen davon gibt es keinen großen Unterschied zwischen dem Layout von B und D. Für die Schaltflächen wird die standardmäßige Hervorhebung verwendet, die Interface Builder und die Klasse `UIButton` bieten.

Schaltfläche C wurde so entworfen, dass sie bei Berührung hervorgehoben wird. Dank ihrer relativ geringen Größe können Sie hier **Button Attributes ▸ Button ▸ Shows Touch On Highlight** verwenden. Wird die Schaltfläche berührt, so erscheint ein leuchtender Hof mit einer Größe von etwa 55 × 55 Pixel um sie herum. Dieser optische Effekt ist bei Schaltflächen größer als 40 × 40 Pixel nicht mehr wirkungsvoll.



▸ *Abbildung 9.4: Diese Beispiele zeigen verschiedene Möglichkeiten für Schaltflächengrafiken sowohl für* `Rounded Rect` *- als auch für benutzerdefinierte Schaltflächen.*

Der statische Screenshot kann auch nicht zeigen, dass die Schaltfläche G ein anderes Bild anzeigt, wenn der Benutzer darauf tippt. Wenn Sie in **Button Attributes ▸ Button ▸ Highlighted State Configuration** ein zweites Bild festlegen, ändert die Schaltfläche bei Berührung ihr Aussehen. Beispiel G zeigt dann die gleiche Schaltfläche, aber so, als sei sie in eine Fassung eingedrückt.

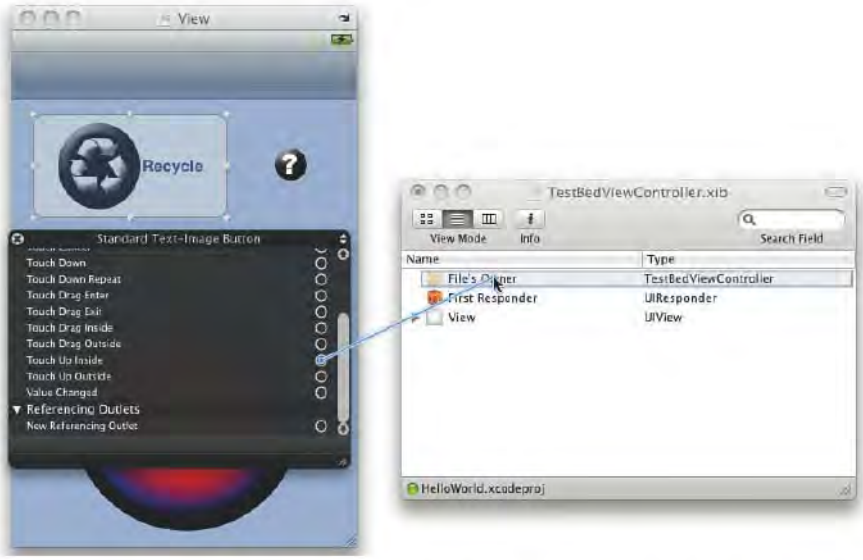
9.3.2 Schaltflächen mit Aktionen verbinden

Wenn Sie eine Schaltfläche bei gedrückter Ctrl-Taste (bzw. rechter Maustaste) auf ein IB-Objekt wie den Ansichtscontroller `File's Owner` ziehen, zeigt Interface Builder ein Einblendmenü von möglichen Aktionen an, die aus den verfügbaren `IBActions` des Zielobjekts abgerufen werden. Durch die Verbindung mit einer Aktion erstellen Sie ein Target-Action-Paar für das `TouchUpInside`-Ereignis der Schaltfläche.

Wie *Abbildung 9.5* zeigt, können Sie alternativ auch mit der Ctrl-Taste (bzw. mit der rechten Maustaste) auf die Schaltfläche klicken, bis zum Eintrag `Touch Up Inside` nach unten scrollen und dessen unausgefüllten Punkt zu dem Ziel ziehen, mit dem Sie ihn verbinden möchten. Wiederum erscheint das Einblendmenü mit den verfügbaren Aktionen. Wählen Sie die gewünschte aus, um die Definition des Target-Action-Callbacks abzuschließen.

9.3.3 Schaltflächen, die keine sind

In Interface Builder finden Sie auch Schaltflächen, die wie Ansichten aussehen, sich wie Ansichten verhalten, aber gar keine Ansichten sind. Leistenschaltflächenelemente (`UIBarButtonItem`) speichern die Eigenschaften der Schaltflächen in Symbol- und Navigationsleisten, sind aber selbst keine Schaltflächen. Näheres zu Leistenschaltflächenelementen finden Sie in Kapitel 5, *Mit Ansichtscontrollern arbeiten*.



▸ *Abbildung 9.5: Wenn Sie in Interface Builder mit der* `Ctrl` *-Taste (bzw. mit der rechten Maustaste) auf ein* `UIControl` *klicken, wird ein Menü der Ereignisse eingeblendet, die Sie mit einem Ziel verbinden können. Die verfügbaren Aktionen erscheinen in einem Einblendmenü, nachdem Sie durch Ziehen die Verbindung hergestellt haben.*

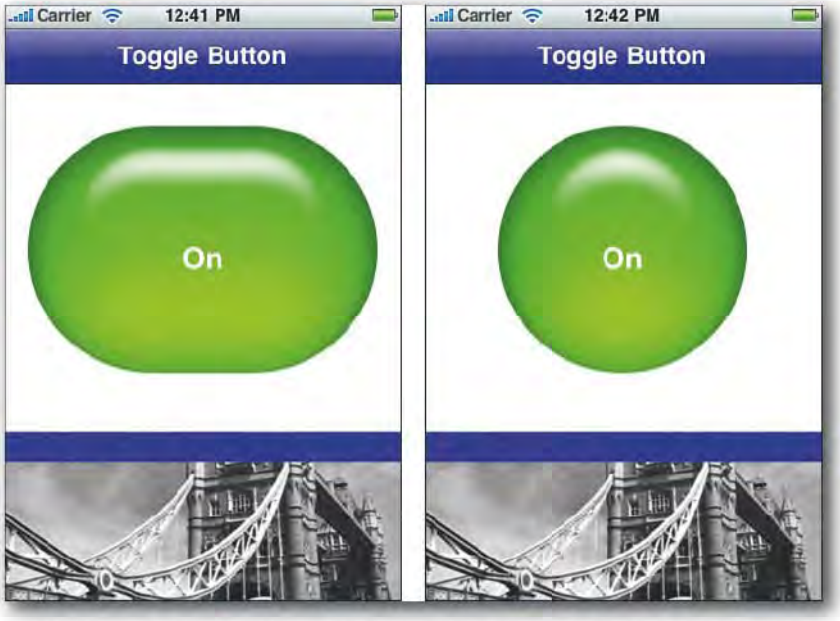
9.4 Eigene Schaltflächen in Xcode erstellen

Wenn Sie `UIButtonTypeCustom` als Stil verwenden, liegt die Gestaltung der Schaltfläche ganz bei Ihnen. Die Anzahl der Bilder hängt davon ab, wie die Schaltfläche nach Ihrem Willen funktionieren soll. Für eine einfache anzutippende Schaltfläche können Sie ein Hintergrundbild hinzufügen und die Farbe der Beschriftung ändern lassen, wenn die Schaltfläche betätigt wird. Bei einem Schalter können Sie vier Bilder verwenden: für die »Aus«-Stellung im normalen Zustand, für die »Aus«-Stellung bei Hervorhebung (also beim Antippen) und zwei weitere für die »Ein«-Stellung. Einzelheiten der Interaktion wählen und gestalten Sie selbst.

Rezept 9.1 erstellt eine Schaltfläche, die zwischen An und Aus wechseln kann, und demonstriert die erforderlichen Einzelheiten für die Erstellung benutzerdefinierter Schaltflächen. Beim Antippen wechselt die Schaltfläche ihre Farbe von grün (ein) zu rot (aus) bzw. umgekehrt. Dies ermöglicht es Ihren Benutzern (falls sie nicht gerade farbenblind sind), den aktuellen Zustand sofort zu erkennen. *Abbildung 9.6* (links) zeigt die mit diesem Rezept erstellte Schaltfläche.

Die `UIImage`-Aufrufe für dehnbare Bilder in diesem Rezept spielen bei der Schaltflächenerstellung eine wichtige Rolle. `StretchableImage` ermöglicht es Ihnen, Schaltflächen von beliebiger Breite zu erstellen und somit Kreisformen in ovale Schaltflächen umzuwandeln. Dabei legen Sie den Rand der Taste an beiden Seiten fest (also die Stellen, an denen die Grafik nicht gedehnt werden darf). In

diesem Fall ist der Rand 110 Pixel breit. Wenn Sie die Schaltflächenbreite von den in diesem Rezept verwendeten 330 Pixel auf 220 ändern, wird die Schaltfläche in der Mitte nicht mehr gedehnt, wie Sie in *Abbildung 9.6* (rechts) sehen.



▶ *Abbildung 9.6: Mit der* `UIImage`*-Dehnung ändern Sie die Größe der Grafik für willkürliche Schaltflächenbreiten. Geben Sie die linke Randbreite an, um festzulegen, wo die Dehnung beginnen kann.*

Hinweis

Die `UIView`-Eigenschaft `contentStretch` dehnt die jeweilige Ansicht. Das in der Eigenschaft definierte Rechteck definiert den Teil der Ansicht, der gedehnt werden kann. Dabei werden die Seitenlängen auf Werte im Bereich zwischen 0,0 und 1,0 normalisiert. Wenn Sie nur den Mittelteil einer Ansicht dehnen wollen, können Sie das Rechteck daher auf `(0.25, 0.25, 0.5, 0.5)` setzen. Bei der Verwendung der Eigenschaft `contentStretch` bleiben die Ränder wie in *Abbildung 9.6* gezeigt erhalten.

```
- (void) toggleButton: (UIButton *) button
{
   if (isOn = !isOn)
   {
      [button setTitle:@"On" forState:UIControlStateNormal];
      [button setTitle:@"On" forState:UIControlStateHighlighted];
      [button setBackgroundImage:baseGreen
         forState:UIControlStateNormal];
```

```
        [button setBackgroundImage:altGreen
            forState:UIControlStateHighlighted];
    }
    else
    {
        [button setTitle:@"Off" forState:UIControlStateNormal];
        [button setTitle:@"Off" forState:UIControlStateHighlighted];
        [button setBackgroundImage:baseRed
            forState:UIControlStateNormal];
        [button setBackgroundImage:altRed
            forState:UIControlStateHighlighted];
    }
}

- (void) viewDidLoad
{
    self.title = @"Toggle Button";

    baseGreen = [[[UIImage imageNamed:@"green.png"]
        stretchableImageWithLeftCapWidth:110.0f topCapHeight:0.0f]
        retain];
    baseRed = [[[UIImage imageNamed:@"red.png"]
        stretchableImageWithLeftCapWidth:110.0f topCapHeight:0.0f]
        retain];
    altGreen = [[[UIImage imageNamed:@"green2.png"]
        stretchableImageWithLeftCapWidth:110.0f topCapHeight:0.0f]
        retain];
    altGreen = [[[UIImage imageNamed:@"red2.png"]
        stretchableImageWithLeftCapWidth:110.0f topCapHeight:0.0f]
        retain];

    // Erstellt eine Schaltfläche in der Größe der Grafik
    UIButton *button = [UIButton buttonWithType:UIButtonTypeCustom];
    button.frame = CGRectMake(0.0f, 0.0f, 300.0f, 233.0f);
    button.center = CGPointMake(160.0f, 140.0f);

    // Legt die Ausrichtungseigenschaften der Schaltfläche fest
    button.contentVerticalAlignment =
        UIControlContentVerticalAlignmentCenter;
    button.contentHorizontalAlignment =
        UIControlContentHorizontalAlignmentCenter;

    // Legt Schriftart und Farbe fest
    [button setTitleColor:[UIColor whiteColor]
        forState:UIControlStateNormal];
```

```
    [button setTitleColor:[UIColor lightGrayColor]
       forState:UIControlStateHighlighted];
    button.titleLabel.font = [UIFont boldSystemFontOfSize:24.0f];

    // Fügt eine Aktion hinzu
    [button addTarget:self action:@selector(toggleButton)
       forControlEvents: UIControlEventTouchUpInside];

    // Zur Nachverfolgung der beiden Zustände
    isOn = NO;
    [self toggleButton:button];

    // Platziert die Schaltfläche in der Ansicht. Die Schaltfläche wird
    // automatisch freigegeben.
    [self.view addSubview:button];

}
```

▸ *Rezept 9.1: Einen* `UIButton`*-Schalter erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.5 Schaltflächentext mit mehreren Zeilen

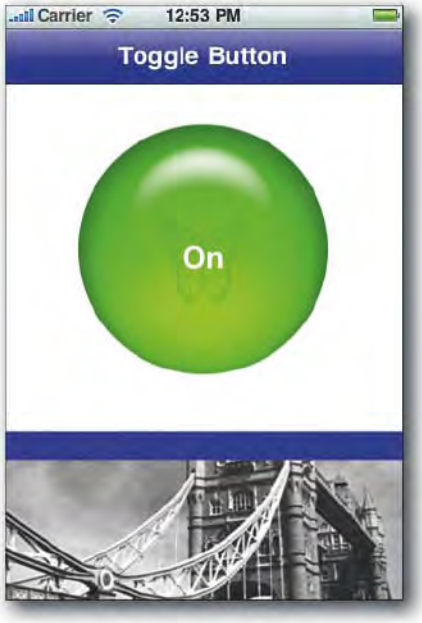
Im SDK 3.0 besteht jetzt die neue Möglichkeit, über die Eigenschaft `titelLabel` auf die Beschriftung von `UIButton`-Schaltflächen zuzugreifen. Dadurch können Sie die Titelattribute direkt bearbeiten, darunter auch die Schriftart und den Zeilenumbruchmodus. Im folgenden Beispiel ist die Schriftgröße sehr hoch eingestellt worden (hauptsächlich, um dafür zu sorgen, dass der Text zur korrekten Anzeige umbrochen werden muss), außerdem sind Zeilenumbruch und Zentrierung aktiviert.

```
button.titleLabel.font = [UIFont boldSystemFontOfSize:36.0f];
[button setTitle:@"Lorem Ipsum Dolor Sit" forState:
   UIControlStateNormal];
button.titleLabel.textAlignment = UITextAlignmentCenter;
button.titleLabel.lineBreakMode = UILineBreakModeWordWrap;
```

Beachten Sie, dass Standardbeschriftungen von einem Ende der Schaltfläche zum anderen verlaufen können. Dadurch kann es jedoch sein, dass sich der Text weiter erstreckt, als Sie es wollen, eventuell sogar über die Ränder der Schaltflächengrafik hinaus. Um dieses Problem zu vermeiden, können Sie im Zeilenumbruchmodus einen Wagenrücklauf vorsehen, indem Sie ein Literal für einen Zeilenwechsel einfügen (also `\n`). Dadurch steuern Sie, wie viel Text in den einzelnen Zeilen der Schaltflächenbeschriftung steht.

9.6 Schaltflächen mit animierten Elementen versehen

Bei der Arbeit mit Schaltflächen können Sie zusätzliche Grafiken davor oder dahinter anbringen. Dazu verwenden Sie die standardmäßige `UIView`-Hierarchie, wobei Sie sicherstellen müssen, dass Sie die Benutzerinteraktion für alle Ansichten deaktivieren, die sich mit der Schaltfläche überschneiden könnten (`setUserInteractionEnabled:NO`). *Abbildung 9.7* zeigt eine Kombination aus einer halbtransparenten Schaltflächengrafik mit einer animierten `UIImageView` dahinter. Der Inhalt der Bildansicht scheint für den Betrachter durch. Auf diese Weise können Sie die Schaltflächen um Animationen bereichern.



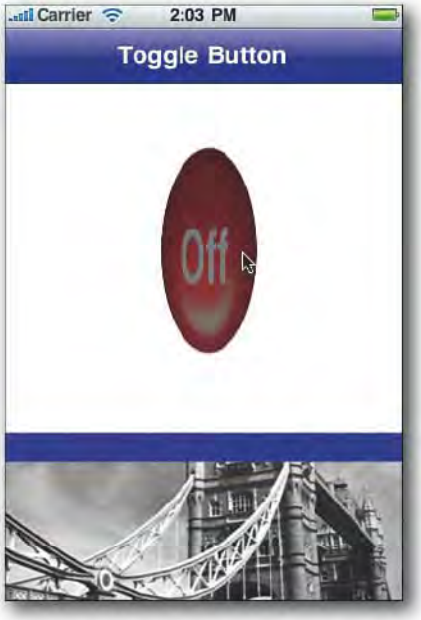
▸ *Abbildung 9.7: Kombinieren Sie ein halbtransparentes Schaltflächendesign mit animierten* `UIImageViews`*, um beeindruckende Oberflächenelemente zu entwerfen. Bei diesem Rezept flattert der Schmetterling »innerhalb« der Schaltfläche.*

9.7 Rezept: Schaltflächenreaktionen animieren

Bei `UIControl`-Instanzen geht es nicht nur um Rahmen und Target-Action. Alle Steuerelemente erben von der Klasse `UIView`. Das bedeutet, dass Sie bei der Arbeit mit Steuerelementen `UIView`-Animationsblöcke genauso verwenden können wie bei Standardansichten. *Rezept 9.2* erstellt einen An/Aus-Schalter, der mithilfe von `UIViewAnimationTransitionFlipFromLeft` gewendet wird, um die Schaltfläche bei einem Wechsel des Zustands zu drehen.

Anders als in *Rezept 9.1* tauscht dieser Code keine Grafiken aus, sondern Schaltflächen. Es gibt hier zwei Schaltflächen: eine für An und eine für Aus, die beide auf einer leeren Ansicht liegen. Da beide Schaltflächen eine durchsichtige Elternansicht haben, können Sie den Wechsel auf genau diese beiden Schaltflächen anwenden, ohne dass sich dies auf die Benutzerschnittstelle an sich auswirkt. Lassen Sie den leeren Hintergrund weg, so dreht sich das gesamte Fenster – was keine empfehlenswerte Oberfläche abgäbe.

Da dieses Rezept die gleiche halbtransparente Grafik verwendet wie das vorherige, ist es wichtig, dass immer nur eine der beiden Schaltflächen auf dem Bildschirm erscheint. Zu diesem Zweck wird die aktuelle Schaltfläche aus der leeren Oberansicht ausgeblendet (der Alpha-Wert wird auf 0,0 gesetzt), während sie im Animationsblock ist. Die Schaltfläche mit dem entgegengesetzten Zustand tritt stattdessen an ihre Stelle. *Abbildung 8.3* zeigt die wechselnde Schaltfläche mitten in diesem Vorgang.



▶ *Abbildung 9.8: Verwenden Sie* `UIView`*-Animationsblöcke, um zwischen Steuerelementzuständen zu wechseln. Hier dreht sich eine Schaltfläche, um zwischen An und Aus zu wechseln.*

```
- (IBAction) flip: (UIButton *) button
{
    // Verbirgt die auszublendende Ansicht
    [self.view viewWithTag:BUTTON1].alpha = 1.0f;
    [self.view viewWithTag:BUTTON2].alpha = 1.0f;
    [button setAlpha:0.0f];

    // Entscheidet, welche Animation verwendet wird
    UIViewAnimationTransition trans;
    trans = (button.tag == BUTTON1) ?
       UIViewAnimationTransitionFlipFromLeft :
       UIViewAnimationTransitionFlipFromRight;

    // Animiert die Drehung
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:1.0f];
    [UIView setAnimationTransition:trans forView:[self.view
```

```
        viewWithTag:CLEARVIEW] cache:YES];
    [[self.view viewWithTag:CLEARVIEW] exchangeSubviewAtIndex:0
        withSubviewAtIndex:1];
    [UIView commitAnimations];
}
```

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.8 Mit Schaltern arbeiten

Ein `UISwitch`-Objekt ist ein einfacher An/Aus-Schalter, mit dem der Benutzer einen booleschen Wert festlegt. Das Schalterobjekt enthält eine einzige (bearbeitbare) Werteigenschaft namens `on`, die je nach Zustand des Steuerelements `YES` oder `NO` zurückgibt. Diesen Schalterwert können Sie im Programm aktualisieren, indem Sie den Eigenschaftswert direkt ändern oder `setOn:animated:` aufrufen, womit sich die Änderung animieren lässt.

Interface Builder (IB) bietet relativ wenige Möglichkeiten zur Arbeit mit Schaltern. Sie können sie aktivieren und den Anfangswert festlegen, aber darüber hinaus lässt sich nicht viel anpassen. Wenn ein Benutzer einen Schalter umlegt, ruft er damit ein Wertänderungsereignis hervor. *Rezept 9.3* nutzt dieses Verhalten, um einen `IBAction`-Callback auszulösen. Wird der Schalter aktualisiert, aktiviert oder deaktiviert er eine zugehörige Schaltfläche.

Wie immer bei der Arbeit in IB müssen Sie die Outlets und Aktionen in **Library ▸ Classes** definiert haben, bevor Sie Verbindungen einrichten. Der Schalter sollte bei `Value Changed` ausgelöst werden und die Aktion `doSwitch:` an `File's Owner` senden, also den Hauptansichtscontroller. Der Controller legt dann die Aktivierungseigenschaft für die Schaltfläche fest. Leider können Sie den Schalter in IB nicht direkt mit der Schaltfläche verbinden, um den Schalterwert mit der Aktivierungseigenschaft zu verknüpfen. Wenn Sie ein langjähriger IB-Benutzer sind, werden Sie sich an Zeiten erinnern können, in denen solche Verbindungen erlaubt waren.

Dieses Rezept baut auf den Steueranimationen aus *Rezept 9.2* sowie auf den modalen Animationen auf, die in Kapitel 6, *Ansichten und Animationen zusammenstellen*, eingeführt wurden. Wird der Schalter aktiviert, ruft er eine oder mehrere Animationsanforderungen auf, die die Schaltfläche in den aktiven oder inaktiven Zustand versetzen.

Hinweis

Geben Sie `UISwitch`-Instanzen nicht den Namen `switch`. Dies ist ein reserviertes Wort in C, das für bedingte Anweisungen verwendet wird. Dieser einfache Lapsus hat sich schon als Fallgrube für so manchen iPhone-Entwickler erwiesen.

```
@implementation TestBedViewController
- (void) expand: (NSNumber *) aFactor
{
   // Vergrößert die Schaltfläche um den gegebenen Faktor
   dangerButton.transform =
   CGAffineTransformMakeScale(aFactor.intValue, aFactor.intValue);
}

- (void) rotate
{
   // Dreht die Schaltfläche um 90°
   dangerButton.transform =
      CGAffineTransformRotate(dangerButton.transform, M_PI_2);
}

- (void) updateAlpha: (NSNumber *) level
{
   // Setzt die Transparenz der Schaltfläche auf den gegebenen Wert
   dangerButton.alpha = level.floatValue;
}

- (IBAction) doSwitch: (UISwitch *) aSwitch
{
   dangerButton.enabled = aSwitch.isOn;

   // Passt den Alpha-Wert der Schaltfläche an den aktivierten bzw.
   // deaktivierten Zustand an
   NSNumber *aLevel = NUMBER((dangerButton.enabled) ? 1.0f : 0.25f);
   [UIView modalAnimationWithTarget:self
      selector:@selector(updateAlpha)
      object:aLevel duration:0.3f];
   dangerButton.transform = CGAffineTransformIdentity;

   if (!dangerButton.enabled) return;

   // Fügt eine kleine Animation ein, wenn der Schalter die Schaltfläche
   // aktiviert, um die Änderung deutlich zu machen
   [UIView modalAnimationWithTarget:self
      selector:@selector(expand)
      object:NUMBER(2.0f) duration:0.3f];
   [UIView modalAnimationWithTarget:self
      selector:@selector(expand)
      object:NUMBER(1.0f) duration:0.3f];
   for (int i = 0; i < 4; i++)
      [UIView modalAnimationWithTarget:self
```

```
        selector:@selector(rotate)
        object:nil duration:0.3f];
}

- (void) boom
{
   // Zeigt die Warnung "Boom" an, wenn der Benutzer auf die Gefahr-
   // Schaltfläche tippt
   UIAlertView *av = [[[UIAlertView alloc] initWithTitle:@"Boom"
      message:nil delegate:nil
      cancelButtonTitle:@"OK"otherButtonTitlesnil] autorelease];
   [av show];
}

- (void) viewDidLoad
{
   // Initialisiert die Gefahr-Schaltfläche mit halbtransparentem
   // Erscheinungsbild
   dangerButton.alpha = 0.25f;
   [dangerButton addTarget:self action:@selector(boom)
      forControlEvents:UIControlEventTouchUpInside];
}
@end
```

► *Rezept 9.3: Eine Schaltfläche über einen Schalter aktivieren und deaktivieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.9 Rezept: Benutzerdefinierte Schieberegler hinzufügen

`UISlider`-Instanzen bilden Steuerelemente, bei denen der Benutzer einen Wert auswählen kann, indem er einen Stellknopf zwischen dem linken und dem rechten Ausschlag verschiebt. `UISlider`-Regler kennen Sie aus der Musikanwendung iPod, wo diese Klasse für die Einstellung der Lautstärke verwendet wird.

Der Wertebereich eines solchen Reglers reicht standardmäßig von 0,0 bis 1,0, was Sie aber im Attribute-Informationsfeld von Interface Builder oder durch Festlegen der Eigenschaften `minimumValueImage` und `maximumValueImage` leicht ändern können. Wenn Sie die Enden des Steuerelements grafisch gestalten möchten, können Sie ein zusammengehöriges Bilderpaar (`minimnumValueImage` und

`maximumValueImage`) hinzufügen, um den Mindest- und Höchstwert deutlicher zu machen. Beispielsweise können Sie bei einem Temperaturregler auf der einen Seite einen Schneemann und auf der anderen eine dampfende Tasse Kaffee anzeigen.

Die Eigenschaft `continuous` des Reglers legt fest, ob das Steuerelement stetig Wertaktualisierungen sendet, während der Benutzer den Stellknopf verschiebt. Wenn Sie diese Eigenschaft auf `NO` setzen (der Standardwert lautet `YES`), sendet der Regler erst dann ein Aktionsereignis, wenn der Benutzer den Stellknopf loslässt.

9.9.1 UISlider anpassen

In der Klasse `UISlider` können Sie nicht nur Bilder für den Mindest- und den Höchstwert hinzufügen, sondern auch für den Stellknopf ein beliebiges Bild festlegen, indem Sie `setThumbImage:forState:` aufrufen. In *Rezept 9.4* wird diese Möglichkeit genutzt, um während des Verschiebevorgangs dynamisch Stellknopfbilder zu erstellen, wie Sie in *Abbildung 9.9* sehen. Die Sprechblase mit dem Wert, die über dem Finger des Benutzers erscheint, ist ein Teil des angepassten Stellknopfes. In diesem Element erhält der Benutzer unmittelbare Rückmeldung über seine Aktion, und zwar sowohl in Textform (die Zahl innerhalb der Blase) als auch grafisch (die Farbe der Sprechblase spiegelt den Reglerwert wider und verändert sich beim Ziehen von Schwarz in Weiß).

▸ *Abbildung 9.9: Mit Core Graphics/Quartz-Aufrufen können Sie das Bild des Stellknopfs entsprechend dem aktuellen Reglerwert abdunkeln und aufhellen. Der Text innerhalb der Sprechblase zeigt den Wert an.*

Diese Art von Rückmeldung kann auf Daten beliebiger Art basieren. Sie können ebenso einfach Werte Ihrer Onboard-Sensoren abgreifen oder Aufrufe ins Internet richten, wie Sie die Fingerbewegung auf dem Regler heranziehen können. Unabhängig davon, welches Aktualisierungssystem Sie verwenden,

sind dynamische Aktualisierungen selbstverständlich grafikintensiv – aber so teuer, wie Sie vielleicht fürchten, ist es nicht. Die Core Graphics-Aufrufe sind schnell, und die Speicheranforderungen für die verkleinerten Vorschaubilder sind minimal.

Im folgenden Rezept werden dem Regler zwei Stellknopfbilder zugewiesen. Die Sprechblase erscheint nur dann, wenn der Regler gerade betätigt wird, also wenn er den Status `UIControlStateHighlighted` ausweist. Im normalen Zustand – `UIControlStateNormal` – wird nur der kleine, rechteckige Stellknopf angezeigt. Die Benutzer können auf diesen Stellknopf tippen, um die aktuelle Einstellung abzulesen. Die kontextspezifische Sprechblase ahmt die Buchstabenhervorhebung der standardmäßigen iPhone-Tastatur nach.

Um die Änderungen in der Darstellung widerzuspiegeln, aktualisiert der Regler seinen Rahmen zu Beginn und Ende jeder Geste. Wird der Regler berührt (`UIControlEventTouchDown`), so wird der Rahmen zum 60 Pixel höheren `thumbFrame`. Die zusätzliche Höhe bietet genügend Platz, um bei der Betätigung des Reglers den erweiterten Stellknopf zu zeigen.

Nimmt der Benutzer den Finger vom Bildschirm (`UIControlEventTouchUpInside` oder `UIControlEventTouchUpOutside`), kehrt der Regler zur vorherigen Größe zurück, also zu `baseFrame`. Dadurch wird der Platz für andere Objekte auf dem Bildschirm wiedergewonnen, sodass der Regler nur dann aktiviert wird, wenn der Benutzer unmittelbar auf ihn tippt.

9.9.2 Die Berechnung wirtschaftlicher gestalten

In diesem Rezept wird der vorhergehende Reglerwert gespeichert, um die Rechenlast auf dem iPhone zu verringern. Der Stellknopf wird nur dann mit einem neuen benutzerdefinierten Bild aktualisiert, wenn sich der Reglerwert um mindestens 0,1, also 10 %, geändert hat. Diese Prüfung können Sie auch weglassen und das Rezept mit kontinuierlicher Aktualisierung ausführen. Bei Tests erfolgte die Aktualisierung dann selbst auf einem iPod touch der ersten Generation in annehmbarer Geschwindigkeit. Außerdem vermeiden Sie dadurch Probleme am Ende des Einstellbereichs, wenn der Regler bei 0,9 hängen bleibt und sich nicht korrekt auf 1,0 aktualisieren lassen will. In diesem Rezept ist für diese Situation eine hartkodierte Lösung für Werte über 0,98 vorgesehen, die eine Aktualisierung erzwingt.

```
@implementation TestBedViewController

// Zeichnet zentrierten Text in den Kontext
void centerText(CGContextRef context,
   NSString *fontname, float textsize,
   NSString *text, CGPoint point, UIColor *color)
{
   CGContextSaveGState(context);
   CGContextSelectFont(context, [fontname UTF8String], textsize,
      kCGEncodingMacRoman);

   // Ruft die Textbreite ab, ohne etwas zu zeichnen
   CGContextSaveGState(context);
   CGContextSetTextDrawingMode(context, kCGTextInvisible);
```

```
    CGContextShowTextAtPoint(context, 0.0f, 0.0f, [text UTF8String],
       text.length);
    CGPoint endpoint = CGContextGetTextPosition(context);
    CGContextRestoreGState(context);

    // Fragt die Größe ab, um die Höhe zu gewinnen. Die Breite ist weniger
    // zuverlässig.
    CGSize stringSize = [text sizeWithFont:[UIFont
       fontWithName:fontname size:textsize]];

    // Zeichnet den Text
    [color setFill];
    CGContextSetShouldAntialias(context, true);
    CGContextSetTextDrawingMode(context, kCGTextFill);
    CGContextSetTextMatrix (context, CGAffineTransformMake(1, 0, 0, -1,
       0, 0));
    CGContextShowTextAtPoint(context, point.x - endpoint.x / 2.0f,
       point.y + stringSize.height / 4.0f, [text UTF8String],
       text.length);
    CGContextRestoreGState(context);
}

// Erstellt eine Stellknopfgrafik mit einem numerischen Graustufenwert
- (UIImage *) createImageWithLevel: (float) aLevel
{
    UIGraphicsBeginImageContext(CGSizeMake(40.0f, 100.0f));
    CGContextRef context = UIGraphicsGetCurrentContext();

    float INSET_AMT = 1.5f;

    // Erstellt ein gefülltes Rechteck für den Stellknopf
    [[UIColor darkGrayColor] setFill];
    CGContextAddRect(context, CGRectMake(INSET_AMT, 40.0f + INSET_AMT,
       40.0f - 2.0f * INSET_AMT, 20.0f - 2.0f * INSET_AMT));
    CGContextFillPath(context);

    // Umrandet den Stellknopf
    [[UIColor whiteColor] setStroke];
    CGContextSetLineWidth(context, 2.0f);
    CGContextAddRect(context, CGRectMake(2.0f * INSET_AMT,
       40.0f + 2.0f * INSET_AMT, 40.0f - 4.0f * INSET_AMT,
       20.0f - 4.0f * INSET_AMT));
    CGContextStrokePath(context);
```

```
    // Erstellt eine gefüllte Ellipse für die Wertanzeige
    [[UIColor colorWithWhite:aLevel alpha:1.0f] setFill];
    CGContextAddEllipseInRect(context, CGRectMake(0.0f, 0.0f, 40.0f,
        40.0f));
    CGContextFillPath(context);

    // Fügt eine Zahl als Beschriftung hinzu
    NSString *numstring = [NSString stringWithFormat:@"%0.1f", aLevel];
    UIColor *textColor = (aLevel > 0.5f) ? [UIColor blackColor] :
        [UIColor whiteColor];
    centerText(context, @"Georgia", 20.0f, numstring,
        CGPointMake(20.0f, 20.0f), textColor);

    // Umrandet die Kreisform der Wertanzeige
    [[UIColor grayColor] setStroke];
    CGContextSetLineWidth(context, 3.0f);
    CGContextAddEllipseInRect(context, CGRectMake(INSET_AMT, INSET_AMT,
        40.0f - 2.0f * INSET_AMT, 40.0f - 2.0f * INSET_AMT));
    CGContextStrokePath(context);

    // Erstellt das Bild und gibt es zurück
    UIImage *theImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return theImage;
}

// Gibt ein einfaches Stellknopfbild ohne Blase zurück
UIImage *createSimpleThumb()
{
    float INSET_AMT = 1.5f;
    UIGraphicsBeginImageContext(CGSizeMake(40.0f, 100.0f));
    CGContextRef context = UIGraphicsGetCurrentContext();

    // Erstellt ein gefülltes Rechteck für den Stellknopf
    [[UIColor darkGrayColor] setFill];
    CGContextAddRect(context, CGRectMake(INSET_AMT, 40.0f + INSET_AMT,
        40.0f - 2.0f * INSET_AMT, 20.0f - 2.0f * INSET_AMT));
    CGContextFillPath(context);

    // Umrandet den Stellknopf
    [[UIColor whiteColor] setStroke];
    CGContextSetLineWidth(context, 2.0f);
    CGContextAddRect(context, CGRectMake(2.0f * INSET_AMT,
        40.0f + 2.0f * INSET_AMT, 40.0f - 4.0f * INSET_AMT,
        20.0f - 4.0f * INSET_AMT));
```

```
    CGContextStrokePath(context);

    UIImage *theImage = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();
    return theImage;
}

// Aktualisiert die Stellknopfbilder bei Bedarf
- (void) updateThumb: (UISlider *) aSlider
{
    // Aktualisiert den Stellknopf nur bei signifikanten Änderungen
    if ((aSlider.value < 0.98) &&
        (ABS(aSlider.value - previousValue) < 0.1f)) return;

    // Erstellt ein neues Stellknopfbild für den hervorgehobenen Zustand
    UIImage *customimg = [self createImageWithLevel:aSlider.value];
    [aSlider setThumbImage: simpleThumbImage forState:
        UIControlStateNormal];
    [aSlider setThumbImage: customimg forState:
        UIControlStateHighlighted];
    previousValue = aSlider.value;
}

// Erweitert den Regler für den größeren Stellknopf
- (void) startDrag: (UISlider *) aSlider
{
    aSlider.frame = thumbFrame;
    aSlider.center = CGPointMake(160.0f, 140.0f);
}

// Verkleinert den Rahmen bei der Freigabe wieder auf die Normalgröße
- (void) endDrag: (UISlider *) aSlider
{
    aSlider.frame = baseFrame;
    aSlider.center = CGPointMake(160.0f, 140.0f);
}

- (void) viewDidLoad
{
    self.title = @"Custom Slider";

    // Initialisiert die Reglereinstellungen
    previousValue = -99.0f;
    simpleThumbImage = [createSimpleThumb() retain];
    thumbFrame = CGRectMake(0.0f, 0.0f, 280.0f, 100.0f);
```

```
    baseFrame = CGRectMake(0.0f, 0.0f, 280.0f, 40.0f);

    // Erstellt den Regler
    UISlider *slider = [[UISlider alloc] initWithFrame:baseFrame];
    slider.center = CGPointMake(160.0f, 140.0f);
    slider.value = 0.0f;

    // Erstellt die Callbacks für das Berühren, Bewegen und Loslassen des
    // Reglers
    [slider addTarget:self action:@selector(startDrag)
       forControlEvents:UIControlEventTouchDown];
    [slider addTarget:self action:@selector(updateThumb)
       forControlEvents:UIControlEventValueChanged];
    [slider addTarget:self action:@selector(endDrag)
       forControlEvents:UIControlEventTouchUpInside |
    UIControlEventTouchUpOutside];

    // Stellt den Regler dar
    [self.view addSubview:slider];
    [self performSelector:@selector(updateThumb) withObject:slider
       afterDelay:0.1f];
}
@end
```

▸ *Rezept 9.4: Dynamische Stellknöpfe für Schieberegler erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.10 Rezept: Ein doppelt antippbares Steuerelement mit Unterteilungen erstellen

Die Klasse `UISegmentedControl` stellt eine Oberfläche mit mehreren Schaltflächen dar, in der die Benutzer eine Option aus einer Gruppe auswählen können. Dabei gibt es zwei mögliche Arten der Verwendung. Im normalen Modus als Optionsschalter bleibt eine einmal ausgewählte Schaltfläche aktiviert. Die Benutzer können auf die anderen Schaltflächen tippen, aber wenn sie erneut auf die bereits getroffene Auswahl tippen, rufen sie kein neues Ereignis hervor. Der alternative Momentan-Modus erlaubt den Benutzern, so oft auf eine Schaltfläche zu tippen, wie sie wollen, speichert aber keine Statusinformationen über das zurzeit ausgewählte Element. Es wird nichts hervorgehoben, um die aktuelle Auswahl anzuzeigen.

In *Rezept 9.5* werden diese beiden Ansätze kombiniert. Die Benutzer können die zurzeit ausgewählte Option erkennen, diese Auswahl aber auf Wunsch auch erneut treffen. Das entspricht nicht der üblichen Funktionsweise von unterteilten Steuerelementen. Es gibt jedoch Fälle, in denen die Neuauswahl zu einem neuen Ergebnis führen soll (wie im Momentan-Modus), in denen die Benutzer aber auch gleichzeitig auf die aktuelle Auswahl hingewiesen werden sollten (wie im Optionsschalter-Modus).

Leider funktionieren die »offensichtlichen« Lösungen für dieses Verhalten nicht: Sie können keine Target-Action-Paare hinzufügen, die `UIControlEventTouchUpInside` erkennen, denn das einzige Steuerereignis, das Instanzen von `UISegmentedControl` hervorrufen, ist `UIControlEventValueChanged`. (Das können Sie leicht überprüfen, indem Sie ein Target-Action-Paar für Berührungsereignisse hinzufügen.)

An dieser Stelle kommen Unterklassen ins Spiel. Es ist recht einfach, eine neue Klasse von `UISegmentedControl` abzuleiten, die auf das zweite Antippen reagiert. *Rezept 9.5* definiert diese Klasse. Der Code erkennt eine Berührung und funktioniert unabhängig von den internen Berührungs-Handlern des unterteilten Steuerelements, die von `UIControl` abgeleitet sind.

Der Wechsel von einem Segment zu einem anderen bleibt davon unberührt. Die einzelnen Unterteilungen werden weiterhin aktualisiert und hin und her geschaltet, wenn die Benutzer darauf tippen. Anders als in der Elternklasse haben Berührungen eines bereits berührten Segments jedoch eine Auswirkung. In diesem Fall fordern Sie die Methode `performSegmentAction` der Delegierung des betreffenden Objekts an.

Versuchen Sie nicht, Target-Action-Paare so zu den Controllern von unterteilten Steuerelementen hinzuzufügen, wie Sie es gewohnt sind. Da alle `TouchDown`-Ereignisse erkannt werden, würden Target-Action-Paare für Wertänderungsereignisse einen zweiten Callback hervorrufen und beim Wechsel der Segmente zweimal ausgelöst werden. Implementieren Sie stattdessen den Delegierungs-Callback, und lassen Sie die Objektdelegierung die Aktualisierungen handhaben.

```
@class DoubleTapSegmentedControl;

@protocol DoubleTapSegmentedControlDelegate <NSObject>
- (void) performSegmentAction: (DoubleTapSegmentedControl *) aDTSC;
@end

@interface DoubleTapSegmentedControl : UISegmentedControl
{
   id <DoubleTapSegmentedControlDelegate> delegate;
}
@property (nonatomic, retain) id delegate;
@end

@implementation DoubleTapSegmentedControl
@synthesize delegate;
```

```
- (void) touchesBegan:(NSSet *)touches withEvent:(UIEvent *)event
{
   [super touchesBegan:touches withEvent:event];
   if (self.delegate)
      [self.delegate performSegmentAction:self];
}
@end
```

▸ *Rezept 9.5: Eine Unterklasse für unterteilte Steuerelemente erstellen, die auf wiederholtes Antippen reagiert*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.11 Unterklassen von UIControl erstellen

Apple bietet zwar verschiedene vorgefertigte Steuerelemente an, die Sie direkt in Ihren Anwendungen einsetzen können, aber Sie sollten sich nicht auf die von Apple gelieferten Elemente beschränken. *Rezept 9.6* zeigt, wie Sie Unterklassen von `UIControl` erstellen können, um eigene Steuerelemente zu entwickeln. In diesem Beispiel wird ein Drehregler angelegt, der denen auf älteren iPod-Modellen ähnelt.

Bei Drehreglern ist die bewegungsgesteuerte Eingabe nicht beschränkt. Die Benutzer können mit den Fingern kreisende Bewegungen im Uhrzeigersinn oder entgegengesetzt dazu machen, wobei der Wert des Objekts entsprechend erhöht oder verringert wird. Jede vollständige Drehung des Reglers, also jeder Durchlauf um 360°, entspricht einer Wertänderung um den Betrag 1,0. Drehungen im Uhrzeigersinn sind positive Änderungen, Drehungen entgegengesetzt zum Uhrzeigersinn negative. Die Werte werden bei jeder Berührung summiert, es ist jedoch auch möglich, den Wert zurückzusetzen. Dazu weisen Sie der Eigenschaft `value` des Steuerelements einfach wieder den Wert 0,0 zu. Obwohl bei vielen Steuerelementen Werte verwendet werden, gehört diese Eigenschaft nicht zu den Standardbestandteilen von `UIControl`-Instanzen.

Das Rezept berechnet die vom Benutzer hervorgerufenen Änderungen mithilfe von Vektoren, deren Ursprung in der Mitte des Steuerelements liegt. Der Code addiert die Winkeldifferenzen, während der Benutzer seine Finger kreisen lässt, und aktualisiert den aktuellen Wert entsprechend. Bei drei Umkreisungen des Reglers wird beispielsweise je nach Bewegungsrichtung 3 zum aktuellen Wert addiert oder von ihm subtrahiert.

9.11.1 Berührungen nachverfolgen

Für den Umgang mit Berührungen verwenden `UIControl`-Instanzen einen Satz eingebetteter Methoden, mit denen die Berührungen während der Betätigung des Steuerelements nachverfolgt werden:

- `beginTrackingWithTouch:withEvent:` Wird aufgerufen, wenn eine Berührung innerhalb der Grenzen des Steuerelements auftritt.
- `continueTrackingWithTouch:withEvent:` Folgt der Berührung mit wiederholten Aufrufen, solange die Berührung innerhalb der Grenzen des Steuerelements verbleibt.
- `endTrackingWithTouch:withEvent:` Handhabt die letzte Berührung des Ereignisses.
- `cancelTrackingWithEvent:` Handhabt den Abbruch einer Berührung.

Ihre eigene Steuerelementlogik fügen Sie hinzu, indem Sie einige oder alle diese Methoden in einer Unterklasse von `UIControl` implementieren. In *Rezept 9.6* werden Versionen der `begin`- und `continue`-Methode verwendet, um eine Berührung zu lokalisieren und nachzuverfolgen, bis der Benutzer den Finger hebt oder die Berührung die Grenzen des Steuerelements verlässt.

9.11.2 Ereignisse ausgeben

Um die von den Ereignissen ausgelösten Änderungen mitzuteilen, werden in Steuerelementen Target-Action-Paare verwendet. Wenn Sie ein neues Steuerelement erstellen, müssen Sie entscheiden, welche Arten von Ereignissen Ihr Objekt hervorrufen soll, und Code zum Auslösen dieser Ereignisse hinzufügen.

Ihrem benutzerdefinierten Steuerelement können Sie Verteilernachrichten hinzufügen, indem Sie `sendActionsForControlEvents:` aufrufen. Diese Methode ermöglicht es Ihnen, ein Ereignis an das angegebene Ziel zu senden, in diesem Fall `UIControlEventValueChanged`. Steuerelemente übertragen diese Aktualisierungen, indem sie das `UIApplication`-Singleton ansprechen. Apple weist darauf hin, dass die Anwendung als zentraler Verteilerpunkt für sämtliche Nachrichten dient.

Hinweis

Der einfache Drehregler aus *Rezept 9.6* verfolgt Drehbewegungen nach, aber sonst nichts. Bei dem alten Drehregler des iPods gab es fünf Antipppunkte: im inneren Kreis und an den Viertelstrichen. Die Unterstützung für das Antippen und die zugehörigen Schaltflächenereignisse (`UIControlEventTouchUpInside`) hinzuzufügen, bleibt Ihnen als Übungsaufgabe überlassen.

```
@implementation ScrollWheel
@synthesize value;
@synthesize theta;

- (id) initWithFrame: (CGRect) aFrame
```

```
{
   if (self = [super initWithFrame:aFrame])
   {
      // Dieses Steuerelement verwendet einen festen Rahmen von
      //  200 x 200 Pixeln
      self.frame = CGRectMake(0.0f, 0.0f, 200.0f, 200.0f);
      self.center = CGPointMake(CGRectGetMidX(aFrame),
         CGRectGetMidY(aFrame));

      // Fügt die Drehreglergrafik hinzu
      UIImageView *iv = [[UIImageView alloc] initWithImage:[UIImage
         imageNamed:@"wheel.png"]];
      [self addSubview:iv];
      [iv release];
   }

   return self;
}

- (id) init
{
   return [self initWithFrame:CGRectZero];
}

+ (id) scrollWheel
{
   return [[[self alloc] init] autorelease];
}

- (BOOL)beginTrackingWithTouch:(UITouch *)touch
      withEvent:(UIEvent *)event
{
   CGPoint p = [touch locationInView:self];
   CGPoint cp = CGPointMake(self.bounds.size.width / 2.0f,
      self.bounds.size.height / 2.0f);
   // self.value = 0.0f; // Für separate Ereigniswerte entkommentieren

   // Die erste Berührung muss den grauen Teil des Reglers treffen
   if (!pointInsideRadius(p, cp.x, cp)) return NO;
   if (pointInsideRadius(p, 30.0f, cp)) return NO;

   // Legt den ursprünglichen Winkel fest
   self.theta = getangle([touch locationInView:self], cp);
   return YES;
}
```

```
- (BOOL)continueTrackingWithTouch:(UITouch *)touch
      withEvent:(UIEvent *)event
{
   CGPoint p = [touch locationInView:self];
   CGPoint cp = CGPointMake(self.bounds.size.width / 2.0f,
      self.bounds.size.height / 2.0f);

   // Prüft, ob die Berührung zu weit außerhalb stattfindet, wobei der
   // Rahmen 50 Pixel breit ist.
   // Ein Fingerstrich innerhalb dieses Radius wird als Berührung gewertet.
   if (!pointInsideRadius(p, cp.x + 50.0f, cp)) return NO;

   float newtheta = getangle([touch locationInView:self], cp);
   float dtheta = newtheta - self.theta;

   // Korrektur für Randbedingungen
   int ntimes = 0;
   while ((ABS(dtheta) > 300.0f) && (ntimes++ < 4))
      if (dtheta > 0.0f) dtheta -= 360.0f; else dtheta += 360.0f;

   // Aktualisiert den bestehenden Wert
   self.value -= dtheta / 360.0f;
   self.theta = newtheta;

   // Sendet eine Meldung über eine Wertänderung
   [self sendActionsForControlEvents:UIControlEventValueChanged];

   return YES;
}
@end
```

▸ *Rezept 9.6: Einen Drehregler erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.12 Rezept: UITextField-Tastaturen entfernen

Die am häufigsten gestellte Frage zu `UITextField`-Steuerelementen lautet: »Wie entferne ich die Tastatur?« Es gibt keine eingebaute Möglichkeit, automatisch herauszufinden, wann dies geschehen muss. Wenn Benutzer die Bearbeitung der Inhalte eines `UITextField`s beenden, sollte sich die Tastatur ausblenden.

Zum Glück erfordert es nur wenig Arbeit, auf das Ende der Bearbeitung zu reagieren. Durch Beobachtung der Eingabetaste können Sie den Status als First Responder aufgeben. Dies schiebt die Tastatur außer Sichtweite, wie *Rezept 9.7* zeigt. Beachten Sie dabei folgende wichtige Aspekte:

- Optional können Sie für die Eingabetaste den Typ `UIReturnKeyDone` festlegen. Dies erledigen Sie im Attribut-Informationsfeld von Interface Builder oder indem Sie die Eigenschaft `returnKeyType` des Textfeldes festlegen. Die Verwendung eines »Done«-Typs für die Eingabetaste zeigt dem Benutzer an, wie er die Bearbeitung beenden kann. *Abbildung 9.10* zeigt eine Tastatur mit einer mit **Done** (Fertig) bezeichneten Eingabetaste.

▸ *Abbildung 9.10: Wenn Sie als Name der Eingabeschaltfläche* **Done** *festlegen (links), teilen Sie dem Benutzer mit, wie er die Bearbeitung des Feldes abschließen kann. Diese Einstellung nehmen Sie direkt im Code oder im Attribut-Informationsfeld von Interface Builder vor (rechts), um das Aussehen und Verhalten des Textfeldes zu ändern.*

- Setzen Sie die Eigenschaft `delegate` des Textfeldes im Code auf Ihren Ansichtscontroller. In Interface Builder gibt es keine Möglichkeit, diese Zuweisung grafisch durchzuführen. Stellen Sie sicher, dass der Ansichtscontroller das Protokoll `UITextFieldDelegate` implementiert.

- Implementieren Sie `textFieldShouldReturn:`. Diese Methode erfasst jedes Antippen der Eingabetaste – unabhängig davon, wie sie bezeichnet ist. Verwenden Sie die Methode, um den First-Responder-Status aufzugeben. Dadurch wird die Tastatur verborgen, bis der Benutzer ein anderes Textfeld oder eine andere Textansicht berührt.

Hinweis

Beim Antippen der Eingabetaste können Sie nicht nur die Tastatur entfernen, sondern auch mithilfe von `textFieldShouldReturn:` eine Aktion durchführen.

Ihr Programmcode muss jeden dieser Aspekte berücksichtigen, um eine reibungslose Interaktion für Ihre `UITextField`-Instanzen hervorzubringen.

9.12.1 Texteigenschaften

Textfelder implementieren das Protokoll `UiTextInputTraits` mit sieben Eigenschaften, mit denen Sie die Handhabung von Texteingaben in den Feldern festlegen können. Damit können Sie folgende Aspekte steuern:

- `autocapitalizationType` Legt den Stil der automatischen Großschreibung fest. Verfügbar sind die Großschreibung aller Wörter im Satz (`UITextAutocapitalizationTypeSentence` – im Deutschen nicht gebräuchlich), die Großschreibung einzelner Wörter (`UITextAutocapitalizationTypeWords`), die Schreibung komplett in Großbuchstaben (`UITextAutocapitalizationTypeAllCharacters`) und der Verzicht auf die Großschreibung (`UITextAutocapitalizationTypeNone`). Vermeiden Sie die Großschreibung bei der Eingabe von Kontonamen. Bei der Eingabe von Eigennamen und Postadressen können Sie die Großschreibung einzelner Wörter verwenden.
- `autocorrectionType` Gibt an, ob der Text die automatische Korrekturfunktion des iPhones durchlaufen soll (siehe die Blase in *Abbildung 9.10*). Bei aktivierter Autokorrektur (`UITextAutocorrectionTypeYes`) schlägt das iPhone dem Benutzer Ersatzwörter vor.
- `enablesReturnKeyAutomatically` Hiermit steuern Sie, ob die Eingabetaste aktiviert wird, wenn es in einem Eingabefeld oder einer Ansicht keinen Text gibt. Wenn Sie diese Eigenschaft auf `YES` setzen, wird die Eingabetaste eingeblendet, sobald der Benutzer mindestens ein Zeichen eingibt.
- `keyboardAppearance` Hiermit können Sie zwischen zwei Darstellungsformen der Tastatur wählen: dem Standardstil und einem Stil für die Verwendung in einem Meldungsfeld.
- `keyboardType` Hiermit können Sie auswählen, was für eine Tastatur als Erste angezeigt wird, wenn ein Benutzer ein Feld oder eine Textansicht bearbeitet. Verfügbare Typen sind `UIKeyboardTypeDefault`, `UIKeyboardTypeASCIICapable`, `UIKeyboardTypeNumbersAndPunctuation`, `UIKeyboardTypeURL`, `UIKeyboardTypeNumberPad`, `UIKeyboardTypePhonePad`, `UIKeyboardTypeNamePhonePad` und `UIKeyboardTypeEmailAddress`.

- `returnKeyType` Gibt an, welcher Text auf der Eingabeschaltfläche der Tastatur angezeigt wird. Zur Auswahl stehen neben dem Standardwert RETURN auch GO, GOOGLE, JOIN, NEXT, ROUTE, SEARCH, SEND, YAHOO, DONE und EMERGENCY CALL.
- `secureTextEntry` Schaltet eine Funktion ein und aus, durch die der Text verborgen wird, um eine sichere Texteingabe zu ermöglichen. Ist diese Eigenschaft aktiviert, können Sie das zuletzt eingegebene Zeichen sehen, während alle anderen nur als Punkte dargestellt werden. Schalten Sie diese Funktion für Passwortfelder ein.

9.12.2 Andere Textfeldeigenschaften

Neben den Textaspekten weisen Textfelder noch verschiedene andere Eigenschaften auf, die ihre Darstellung steuern. Bei leerem Textfeld wird der Wert von `placeholder` als Eingabeaufforderung in Form von hellgrauem Text angezeigt. Verwenden Sie diesen Platzhalter, um Nutzungshinweise wie »Benutzername« oder »E-Mail-Adresse« anzugeben.

Bei Textfeldern können Sie mit `borderStyle` auch die Art des Rahmens festlegen, der um den Textbereich angezeigt wird. Zur Auswahl stehen eine einfache Linie, eine Abschrägung und ein abgerundetes Rechteck. Am besten lassen sie sich in Interface Builder erkennen, wo Sie im Attribut-Informationsfeld zwischen den einzelnen Darstellungen umschalten können.

Die Löschschaltfläche eines Textfeldes erscheint als X auf der rechten Seite des Eingabebereichs. Mit `clearButtonMode` geben Sie an, ob und wann diese Schaltfläche erscheinen soll: immer, niemals, beim Bearbeiten oder wenn keine Bearbeitung stattfindet.

```
@interface TestBedViewController : UIViewController <UITextFieldDelegate>
@end

@implementation TestBedViewController
- (BOOL)textFieldShouldReturn:(UITextField *)textField
{
   [textField resignFirstResponder];
   return YES;
}

- (void) viewDidLoad
{
   self.title = @"Keyboard Dismissal";

   // Passt ein Textfeld aus Interface Builder an
   UITextField *tf = (UITextField *)[self.view viewWithTag:101];
   tf.delegate = self;
```

```
    // Erstellt manuell ein Textfeld
    tf = [[UITextField alloc] initWithFrame:CGRectMake(0.0f, 0.0f,
       100.0f, 30.0f)];
    tf.center = CGPointMake(160.0f, 120.0f);
    tf.borderStyle = UITextBorderStyleRoundedRect;
    tf.autocorrectionType = UITextAutocorrectionTypeNo;
    tf.placeholder = @"Name";
    tf.returnKeyType = UIReturnKeyDone;
    tf.clearButtonMode = UITextFieldViewModeWhileEditing;
    tf.delegate = self;
    [self.view addSubview:tf];
    [tf release];
}
@end
```

▸ *Rezept 9.7: Textfeld-Tastaturen mit einer* **Done**-*Schaltfläche entfernen*

Den Rezeptcode finden

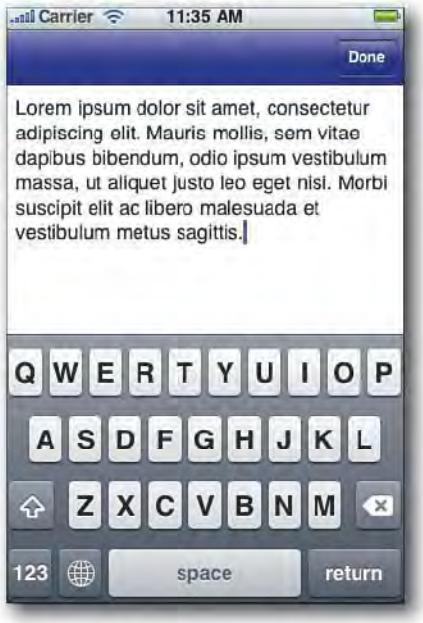
Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.13 Rezept: UITextView-Tastaturen entfernen

Beim Entfernen von Tastaturen erfordern `UITextView`-Instanzen einen anderen Ansatz als `UITextField`-Instanzen. Benutzer sollten in der Lage sein, die Eingabetaste in der Textansicht zu betätigen, um Zeilenumbrüche einzufügen, ohne die Tastatur zu entfernen. Ergänzen Sie stattdessen die allgemeine Oberfläche um eine **Done**-Schaltfläche, wenn die Textansicht aktiviert wird, wie in *Abbildung 9.11* zu sehen ist. Verwenden Sie diese Schaltfläche, um den First-Responder-Status aufzugeben, wenn der Benutzer seine Eingaben beendet.

Um die Aktivität der Textansicht wahrzunehmen, muss Ihr Ansichtscontroller das Protokoll `UITextViewDelegate` implementieren und als Delegierung der Textansicht festgelegt werden. Die Delegierungsmethode `textViewDidBeginEditing:` wird immer dann ausgelöst, wenn ein Benutzer die Ansicht berührt. Dass Sie diesen Vorgang feststellen können, ermöglicht es Ihnen, die **Done**-Schaltfläche entweder hinzuzufügen oder zu aktivieren. Benutzer können auf **Done** tippen, wenn sie die Bearbeitung beendet haben. Die **Done**-Schaltfläche bietet eine klar erkennbare Möglichkeit, die Bearbeitung zu beenden und die Tastatur zu entfernen.

Rezept 9.8 stellt dar, wie die Schaltfläche für das Navigationselement in den Aufruf der Delegierungsmethode eingefügt und wie sie entfernt werden kann, wenn der Benutzer mit der Bearbeitung fertig ist. Bringen Sie die **Done**-Schaltfläche zum Vorschein, wenn die Ansicht aktiviert wird, und blenden Sie sie aus, wenn die Ansicht den First-Responder-Status verliert.



▸ *Abbildung 9.11: Fügen Sie der Navigationsleiste eine* **Done** *-Schaltfläche hinzu, wenn der Benutzer beginnt, in einer Textansicht zu arbeiten. Dies bietet Benutzern die unübersehbare Möglichkeit, die Bearbeitung zu beenden und die Tastatur zu entfernen.*

```
@interface TestBedViewController : UIViewController <UITextViewDelegate>
@end

@implementation TestBedViewController

// Zeigt bei Beginn der Bearbeitung eine Done-Schaltfläche an
- (void) textViewDidBeginEditing: (UITextView *) textView
{
   self.navigationItem.rightBarButtonItem = BARBUTTON(@"Done",
      @selector(doneEditing));
}

// Entfernt die Done-Schaltfläche und die Tastatur
- (void) doneEditing: (id) sender
{
   [self.view resignFirstResponder];
   self.navigationItem.rightBarButtonItem = nil;
}

- (void) viewDidLoad
{
   [(UITextView *)self.view setDelegate:self];
   [(UITextView *)self.view setFont:[UIFont systemFontOfSize:16.0f]];
}
@end
```

▸ *Rezept 9.8: Eine* **Done***-Schaltfläche zu aktiven* `UITextView`*-Sitzungen hinzufügen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.14 Den Texteditor verbessern

In *Rezept 9.8* haben Sie gesehen, wie Sie Benutzerinteraktionen in einer Textansicht erfassen. Diese Möglichkeit wird in *Rezept 9.9* um eine Reihe entscheidender Funktionen erweitert, um den Texteditor zu verbessern. All diese Funktionen können Sie sehr einfach auch in Ihren eigenen Programmen implementieren.

Als Erstes fügen wir im Ansichtscontroller eine Widerrufs-Unterstützung hinzu. Wenn die Benutzer ihr iPhone schütteln, laden sie den Widerrufs- bzw. Wiederherstellungs-Editor, der in Kapitel 8 vorgestellt wurde. `UITextView`-Objekte sind schon im Lieferzustand widerrufsfähig. Die integrierte Widerrufsunterstützung erkennt Aktionen wie Markieren, Ausschneiden, Kopien und Einfügen. Der Undo-Manager versteht diese Aktionen, sodass dem Benutzer Optionen wie **Undo Paste** (Einfügen widerrufen), **Redo Cut** (Ausschneiden wiederherstellen) usw. angezeigt werden können. Der Ansichtscontroller muss dazu lediglich einen Undo-Manager instanziieren. Den Rest der Arbeit erledigen die eingebauten Objekte.

Zweitens wird die Ansicht dauerhaft gemacht. Die Inhalte werden mit der Methode `performArchive` in einer Datei archiviert. Die Anwendungsdelegierung ruft diese Methode kurz vor Beendigung der Anwendung auf.

```
- (void) applicationWillTerminate: (UIApplication *) application
{
    // Aktualisiert beim Aufruf der ARchivierungsmethode des Test-
    // Ansichtscontrollers die Voreinstellungen
    [self.tbvc performArchive];
}
```

Beim Start werden eventuell in dieser Datei vorhandene Daten gelesen, um die Textansichtsinstanzen zu initialisieren.

Außerdem wird die Größe der Textansicht automatisch angepasst, sobald die Tastatur erscheint, damit Letztere keinen Text überdeckt. Das ist vor allem dann wichtig, wenn Sie das Ende eines langen Texteintrags bearbeiten möchten. Da die Textansicht so verkleinert wird, dass sie vollständig oberhalb der Tastatur erscheint, haben die Benutzer Zugriff auf den gesamten Text.

Damit dies geschehen kann, lauscht der Code von *Rezept 9.9* auf zwei Standardbenachrichtigungen, die kurz vor dem Ein- und Ausblenden der Tastatur gesendet werden. Der Code enthält auch Beobachter, die auf den Tastaturstatus reagieren und die Höhe der Textansicht in Übereinstimmung mit der Tastaturdarstellung anpassen.

```
@interface TestBedViewController : UIViewController <UITextViewDelegate>
{
   NSUndoManager *undoManager;
   IBOutlet UITextView *textView;
}
@property (retain) NSUndoManager *undoManager;
@end

@implementation TestBedViewController
@synthesize undoManager;

- (void) performArchive
{
   [[textView text] writeToFile:DATAPATH atomically:YES
      encoding:NSUTF8StringEncoding error:nil];
}

// Zeigt bei Beginn der Bearbeitung eine Done-Schaltfläche an
- (void) textViewDidBeginEditing: (UITextView *) aTextView
{
   self.navigationItem.rightBarButtonItem = BARBUTTON(@"Done",
      @selector(doneEditing));
}

// Entfernt die Done-Schaltfläche und die Tastatur
- (void) doneEditing: (id) sender
{
   [textView resignFirstResponder];
   self.navigationItem.rightBarButtonItem = nil;
}

// Bereitet die Größenänderung der Tastatur vor. Mit freundlicher
// Genehmigung von August Joki.
- (void)keyboardWillShow:(NSNotification *)notification
{
   NSDictionary *userInfo = [notification userInfo];
   CGRect bounds;
   [(NSValue *)[userInfo objectForKey:UIKeyboardBoundsUserInfoKey]
      getValue:&bounds];

   // Ändert die Größe der Textansicht
   CGRect aFrame = textView.frame;
   aFrame.size.height -= bounds.size.height;
   textView.frame = aFrame;
```

```
}

// Erweitert die Textansicht beim Entfernen der Tastatur
- (void)keyboardWillHide:(NSNotification *)notification
{
   // Ändert die Größe der Textansicht
   CGRect aFrame = CGRectMake(0.0f, 0.0f, 320.0f, 416.0f);
   textView.frame = aFrame;
}

- (void) viewDidLoad
{
   // Initialisiert die Textansicht
   textView.delegate = self;
   textView.font = [UIFont systemFontOfSize:16.0f];
   textView.text = [NSString stringWithContentsOfFile:DATAPATH];

   // Bereitet den Undo-Manager vor
   [[UIApplication sharedApplication]
      setApplicationSupportsShakeToEdit:YES];
   self.undoManager = [[NSUndoManager alloc] init];
   [self.undoManager setLevelsOfUndo:99];
   [self.undoManager release];

   // Lauscht auf die Tastatur
   [[NSNotificationCenter defaultCenter] addObserver:self
      selector:@selector(keyboardWillShow)
      name:UIKeyboardWillShowNotification object:nil];
   [[NSNotificationCenter defaultCenter] addObserver:self
      selector:@selector(keyboardWillHide)
      name:UIKeyboardWillHideNotification object:nil];
}

- (void) dealloc
{
   // Räumt auf
   [[NSNotificationCenter defaultCenter] removeObserver:self];
   self.undoManager = nil;
   [super dealloc];
}
@end
```

► *Rezept 9.9: Widerrufsunterstützung, Dauerhaftigkeit und automatische Größenänderung der Textansicht*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.15 Rezept: Texteingaben filtern

In manchen Fällen müssen Sie sicherstellen, dass die Benutzer nur eine bestimmte Teilmenge von Zeichen eingeben. Beispielsweise kann es sein, dass Sie ein rein numerisches Textfeld brauchen, in dem keine Buchstaben zulässig sind. Zwar können Sie die endgültige Eingabe mithilfe von Prädikaten mit einem regulären Ausdruck vergleichen (der Operator `MATCH` der Klasse `NSPredicate` unterstützt Regex-Werte), doch ist es zur Datenfilterung einfacher, jedes neu eingegebene Zeichen anhand eines Satzes zulässiger Zeichen zu überprüfen.

Eine `UITextField`-Delegierung kann diese Zeichen erfassen, während sie eingegeben werden, und jeweils entscheiden, ob sie in das aktive Textfeld eingefügt werden sollen. Die optionale Delegierungsmethode `textField:shouldChangeCharactersInRange:replacementString:` gibt entweder `YES` zurück, was bedeutet, dass die neu eingegebenen Zeichen zulässig sind, oder `NO`, um sie abzulehnen. In der Praxis wird diese Methode Zeichen für Zeichen verwendet und nach jedem Antippen der Tastatur aufgerufen. Angesichts der neuen Unterstützung für die Zwischenablage in Version 3.0 kann der untersuchte String theoretisch länger sein, wenn Text in ein Textfeld kopiert wird.

Rezept 9.10 sucht nach unzulässigen Zeichen im neuen String. Wenn es sie gibt, wird die Eingabe verworfen, sodass das Textfeld unverändert bleibt. Wenn Sie also einen Text aus sowohl zulässigen wie auch unzulässigen Zeichen in das Feld einfügen, wird die gesamte Eingabe verworfen.

Dieses Rezept deckt vier verschiedene Situationen ab: reine Buchstabeneingabe, reine Zahleneingabe, Zahleneingabe mit zulässigem Dezimalpunkt und eine Mischung aus alphanumerischen Zeichen. Dieses Beispiel können Sie an jeden gewünschten Satz zulässiger Zeichen anpassen.

Für den dritten Eingabetyp – Zahlen mit Dezimalpunkt – wird ein kleiner Trick eingesetzt, um dafür zu sorgen, dass nur ein einziger Dezimalpunkt verwendet wird. Sobald der Code einen Punkt im Textfeld findet, ändert er die Menge der zulässigen Zeichen von einem Satz mit Punkt zu einem Satz ohne Punkt. Es ist zwar möglich, dieses Hindernis durch Einfügen aus der Zwischenablage zu umgehen, aber es ist unwahrscheinlich, dass ein Benutzer so etwas tut.

```
#define ALPHA @"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz "
#define NUMBERS @"0123456789"
#define ALPHANUM \
   @"ABCDEFGHIJKLMNOPQRSTUVWXYZabcdefghijklmnopqrstuvwxyz0123456789 "
#define NUMBERSPERIOD @"0123456789."

@implementation TestBedViewController
- (BOOL)textField:(UITextField *)textField
```

```
      shouldChangeCharactersInRange:(NSRange)range
      replacementString:(NSString *)string
{
   NSCharacterSet *cs;

   switch (SEGMENT)
   {
      case 0:
         cs = [[NSCharacterSet characterSetWithCharactersInString:
            ALPHA] invertedSet];
         break;
      case 1:
         cs = [[NSCharacterSet characterSetWithCharactersInString:
            NUMBERS] invertedSet];
         break;
      case 2:
         cs = [[NSCharacterSet characterSetWithCharactersInString:
            NUMBERS] invertedSet];
         if ([textField.text rangeOfString:@"."].location ==
            NSNotFound)
            cs = [[NSCharacterSet
               characterSetWithCharactersInString:NUMBERSPERIOD]
               invertedSet];
         break;
      case 3:
         cs = [[NSCharacterSet characterSetWithCharactersInString:
            ALPHANUM] invertedSet];
         break;
      default:
         break;
   }
   NSString *filtered = [[string componentsSeparatedByCharactersInSet:
      cs] componentsJoinedByString:@""];
   BOOL basicTest = [string isEqualToString:filtered];
   return basicTest;
}

- (void) segmentChanged: (UISegmentedControl *) seg
{
   [(UITextField *)[self.view viewWithTag:101] setText:@""];
}

- (void) viewDidLoad
{
   // In Interface Builder definiertes Textfeld
```

```
    [(UITextField *)[self.view viewWithTag:101] setDelegate:self];

    // Fügt ein unterteiltes Steuerelement mit Eingabeoptionen hinzu
    UISegmentedControl *seg = [[UISegmentedControl alloc]
        initWithItems:[@"ABC 123 2.3 A2C"
        componentsSeparatedByString:@" "]];
    seg.segmentedControlStyle = UISegmentedControlStyleBar;
    seg.selectedSegmentIndex = 0;
    [seg addTarget:self action:@selector(segmentChanged)
        forControlEvents:UIControlEventValueChanged];
    self.navigationItem.titleView = seg;
    [seg release];
}
@end
```

► *Rezept 9.10: Texteingaben von Benutzern filtern*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.16 Rezept: Seitenindikatoren hinzufügen

Die Klasse `UIPageControl` stellt eine Reihe von Punkten dar, die anzeigen, welches Element einer mehrseitigen Ansicht zurzeit zu sehen ist. Ein Beispiel für diese Art von Steuerelement bilden die Punkte am unteren Rand der SpringBoard-Startseite. Leider ist die Funktionsweise der Klasse `UIPageControl` enttäuschend. Sie lässt sich schwer handhaben, schwer antippen und wird sich für die meisten Benutzer als ärgerlich erweisen. Wenn Sie sie verwenden, müssen Sie alternative Navigationsmöglichkeiten beisteuern, damit der Seitenindikator eher als eine Anzeige und weniger als ein Steuerelement funktioniert.

Abbildung 9.12 zeigt ein Seitensteuerelement mit drei Seiten. Das Antippen links oder rechts vom grellen Seitenindikator löst `UIControlEventValueChanged`-Ereignisse aus, wobei eine beliebige Methode angestoßen wird, die Sie als Aktion der Steuerung festgelegt haben. Sie können den neuen Wert der Steuerung abfragen, indem Sie `currentPage` aufrufen, und die verfügbare Seitenzählung festlegen, indem Sie die Eigenschaft `numberOfPages` korrigieren. In SpringBoard ist die Anzahl der Punkte für die Seiten auf neun begrenzt, aber in Ihrer eigenen Anwendung können Sie auch mehr Punkte verwenden, vor allem im Querformat.

In *Rezept 9.11* werden drei Seiten mit Bildern mithilfe einer `UIScrollView`-Instanz angezeigt. Die Benutzer können sich mit Durchstreichgesten durch die Bilder bewegen, wobei der Seitenindikator aktualisiert wird und jeweils anzeigt, welche Seite gerade dargestellt wird. Ebenso können die

Benutzer auf das Seitensteuerelement tippen, woraufhin die Rollansicht die ausgewählte Seite ins Blickfeld schiebt. Diese Zwei-Wege-Bedienung wird durch einen Target-Action-Callback im Seitensteuerelement und einen Delegierungs-Callback in der Rollansicht ermöglicht. Jeder dieser Callbacks aktualisiert jeweils das andere Objekt, um eine enge Kopplung zwischen ihnen herzustellen.

▸ *Abbildung 9.12: Die Klasse* `UIPageControl` *bietet einen interaktiven Indikator für mehrseitige Darstellungen. Durch Antippen links oder rechts vom aktiven Punkt können Benutzer neue Seiten auswählen. Zumindest tun sie das in der Theorie. Die Seitensteuerung ist nur schwer anzutippen, erfordert vom Benutzer eine extreme Genauigkeit und weist eine mangelhafte Reaktionsfähigkeit auf.*

```
@implementation TestBedViewController
- (void) pageTurn: (UIPageControl *) aPageControl
{
    // Aktualisiert die Rollansicht entsprechend der Auswhal einer
    // neuen Seite
    int whichPage = aPageControl.currentPage;

    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.3f];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];

    sv.contentOffset = CGPointMake(320.0f * whichPage, 0.0f);

    [UIView commitAnimations];
}
- (void) scrollViewDidScroll: (UIScrollView *) aScrollView
{
    // Aktualisiert den Seitenindikator entsprechend der Rollansicht
    CGPoint offset = aScrollView.contentOffset;
    pageControl.currentPage = offset.x / 320.0f;
```

```
}

- (void) viewDidLoad
{
    // Erstellt die Rollansicht und legt ihre Inhaltsgröße und die
    // Delegierung
    // fest
    sv = [[UIScrollView alloc] initWithFrame:
        CGRectMake(0.0f, 0.0f, 320.0f, BASEHEIGHT)];
    sv.contentSize = CGSizeMake(NPAGES * 320.0f, sv.frame.size.height);
    sv.pagingEnabled = YES;
    sv.delegate = self;
    [sv release];

    // Lädt alle Seiten
    for (int i = 0; i < NPAGES; i++)
    {
        NSString *filename = [NSString stringWithFormat:
            @"image%d.png", i+1];
        UIImageView *iv = [[UIImageView alloc] initWithImage:[UIImage
            imageNamed:filename]];
        iv.frame = CGRectMake(i * 320.0f, 0.0f, 320.0f, BASEHEIGHT);
        [sv addSubview:iv];
        [iv release];
    }

    [self.view addSubview:sv];

    // Initialisiert den in IB erstellten Seitenindikator
    pageControl.numberOfPages = 3;
    pageControl.currentPage = 0;
    [pageControl addTarget:self action:@selector(pageTurn)
        forControlEvents:UIControlEventValueChanged];
}
@end
```

▸ *Rezept 9.11: Den* `UIPageControl`*-Indikator verwenden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.17 Rezept: Eine anpassbare mehrseitige Rollansicht erstellen

In *Rezept 9.11* wurde eine einfache mehrseitige Rollansicht vorgestellt, in der aber keinerlei dynamische Interaktionen möglich waren. Das Beispiel umfasst zu Anfang und zu Ende drei Seiten. In der Praxis ist es sinnvoller, wenn Sie in Seitensteuerelementen im laufenden Betrieb Seiten hinzufügen und löschen können. Genau das leistet *Rezept 9.12*. Es fügt dem Steuerelement Schaltflächen hinzu, mit denen einzelne Ansichten in der `UIScrollView` erstellt und aus ihr entfernt werden können.

Hierzu werden nicht zwei, sondern vier verschiedene Steuerelemente eingesetzt, um die Hinzufügen/Löschen-Schnittstelle aus *Abbildung 9.13* zu gestalten. Es handelt sich dabei um:

- eine Schaltfläche zum Hinzufügen im Stil der Standardschaltfläche zum Hinzufügen von Kontakten
- eine Löschschaltfläche im gleichen Stil
- eine Bestätigungsschaltfläche mit einem X, die über der Löschschaltfläche eingeblendet wird
- eine bildschirmfüllende, vollständig leere Schaltfläche zum Abbrechen

▸ *Abbildung 9.13: Mit den Schaltflächen + und - können die Benutzer einzelne Seitenansichten zur Rollansicht hinzufügen und aus ihr löschen. Zum Löschen ist ein zusätzlicher Schritt erforderlich, da eine Bestätigungsschaltfläche eingeblendet wird.*

Diese Schaltflächen funktionieren wie folgt. Solange es weniger als acht Seiten gibt, kann der Benutzer auf **Add** tippen, um eine neue Ansicht in der `UIScrollView` zu erstellen. Dabei wird die Anzahl der Seiten im Seitenindikator aktualisiert und die neue Ansicht ins Blickfeld geschoben. Außerdem wird die Anzahl der Seiten überprüft. Wenn sie das Maximum erreicht, deaktiviert der Code die Schaltfläche **Add**. Der Grenzwert von acht Seiten ist willkürlich. Sie können den Code auf eine größere oder kleinere Zahl anpassen.

Beim Antippen von **Delete** wird mit einer Animation eine Bestätigungsschaltfläche eingeblendet und eine unsichtbare Abbruch-Schaltfläche aktiviert, die den Rest des Bildschirms bedeckt. Wenn der Benutzer auf **Confirm** (Bestätigen) tippt, wird die Seite gelöscht. Tippt er auf irgendeine andere Stelle, wird die Aktion abgebrochen und die Bestätigungsschaltfläche ausgeblendet, ohne die Seite zu entfernen.

Das Löschen mit Bestätigung entspricht der Richtlinie von Apple, Löschvorgänge nur mit Vorsichtsmaßnahmen durchzuführen, die auch beim Bearbeiten von Tabellen und in anderen Benutzeroberflächen zum Tragen kommen. Zum Löschen einer Seite sind zwei Tippvorgänge erforderlich, sodass der Benutzer den Vorgang ohne Verluste abbrechen kann. Dies verhindert das versehentliche Löschen einer Seite und gibt dem Benutzer eine sichere Möglichkeit zum Abbrechen, falls er den Vorgang nicht fortsetzen möchte.

```
@implementation TestBedViewController
- (void) pageTurn: (UIPageControl *) aPageControl
{
   // Aktualisiert den Seitenindikator und blendet die neue Seite ein
   int whichPage = aPageControl.currentPage;
   [UIView beginAnimations:nil context:NULL];
   [UIView setAnimationDuration:0.3f];
   [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
   sv.contentOffset = CGPointMake(320.0f * whichPage, 0.0f);
   [UIView commitAnimations];
}

- (void) scrollViewDidScroll: (UIScrollView *) aScrollView
{
   // Spiegelt die Rollvorgänge durch den Benutzer im Seitenindikator
   // wider
   CGPoint offset = aScrollView.contentOffset;
   pageControl.currentPage = offset.x / 320.0f;
}

- (UIColor *)randomColor
{
   // Gibt eine Zufallsfarbe zurück
   float red = (64 + (random() % 191)) / 256.0f;
   float green = (64 + (random() % 191)) / 256.0f;
   float blue = (64 + (random() % 191)) / 256.0f;
   return [UIColor colorWithRed:red green:green blue:blue alpha:1.0f];
}

  (void) addPage
{
   // Neue Seiten werden stets am Ende der Rollansicht eingefügt
```

```
    pageControl.numberOfPages = pageControl.numberOfPages + 1;
    pageControl.currentPage = pageControl.numberOfPages - 1;

    // Erhöht die Größe der Rollansicht und fügt eine neue Seite hinzu
    sv.contentSize = CGSizeMake(pageControl.numberOfPages * 320.0f,
       BASEHEIGHT);
    UIView *aView = [[UIView alloc] initWithFrame:
       CGRectMake(pageControl.currentPage * 320.0f, 0.0f, 320.0f,
       BASEHEIGHT)];
    aView.backgroundColor = [self randomColor];
    [sv addSubview:aView];
    [aView release];
}

- (void) requestAdd: (UIButton *) button
{
    // Fügt die Seite ein und aktualisiert die Schaltflächen
    [self addPage];
    addButton.enabled = (pageControl.numberOfPages < 8) ? YES : NO;
    deleteButton.enabled = YES;
    [self pageTurn:pageControl];
}

- (void) deletePage
{
    // Gelöscht wird stets die zurzeit angezeigte Seite
    int whichPage = pageControl.currentPage;
    pageControl.numberOfPages = pageControl.numberOfPages - 1;

    // Entfernt die betroffene Seite
    NSMutableArray *properViews = [NSMutableArray array];
    for (UIView *view in sv.subviews)
       if ([[[view class] description] isEqualToString:@"UIView"] &&
          (view.frame.size.width == 320.0f))
          [properViews addObject:view];

    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.3f];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];

    UIView *whichView = [properViews objectAtIndex:whichPage];

    // Schiebt andere Seiten ins Blickfeld
    for (int i = whichPage + 1; i < [properViews count]; i++)
    {
```

```
        UIView *aView = [properViews objectAtIndex:i];
        CGRect frame = aView.frame;
        frame.origin.x = frame.origin.x - 320.0f;
        aView.frame = frame;
    }

    [UIView commitAnimations];

    // Entfernt die Seite nach Abschluss der Animation
    [whichView performSelector:@selector(removeFromSuperview)
        withObject:nil afterDelay:0.3f];

    sv.contentSize = CGSizeMake(sv.contentSize.width - 320.0f,
        BASEHEIGHT);
}

// Entfernt die Bestätigungsschaltfläche und verbirgt die
// Abbruch-Schaltfläche
- (void) hideConfirmAndCancel
{
    cancelButton.enabled = NO;

    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.3f];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    confirmButton.center = CGPointMake(deleteButton.center.x + 100.0f,
        deleteButton.center.y);
    [UIView commitAnimations];
}

// Führt die Löschung nach Bestätigung durch und aktualisiert
// die Schaltflächen
- (void) confirmDelete: (UIButton *) button
{
    [self deletePage];
    addButton.enabled = YES;
    deleteButton.enabled = (pageControl.numberOfPages > 1) ? YES : NO;
    [self pageTurn:pageControl];
    [self hideConfirmAndCancel];
}

// Beim Abbrechen werden einfach die Bestätigugns- und die
// Abbruchschaltfläche verborgen
- (void) cancelDelete: (UIButton *) button
{
```

```
    [self hideConfirmAndCancel];
}

// Zeigt als Reaktion auf eine Löschanforderung die Bestätigungsschalt-
// fläche
- (void) requestDelete: (UIButton *) button
{
    // Zeigt die Abbruch- und Bestätigungsschaltfläche an
    [cancelButton.superview bringSubviewToFront:cancelButton];
    [confirmButton.superview bringSubviewToFront:confirmButton];
    cancelButton.enabled = YES;

    // Blendet die Bestätigungsschaltfläche mit einer Animation ein
    confirmButton.center = CGPointMake(deleteButton.center.x + 100.0f,
        deleteButton.center.y);
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.3f];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    confirmButton.center = deleteButton.center;
    [UIView commitAnimations];
}

- (void) viewDidLoad
{
    // Erstellt die Rollansicht und legt ihre Inhaltsgröße und Delegie-
    // rung fest
    sv = [[UIScrollView alloc] initWithFrame:CGRectMake(0.0f, 0.0f,
        320.0f, BASEHEIGHT)];
    sv.contentSize = CGSizeZero;
    sv.pagingEnabled = YES;
    sv.delegate = self;
    [self.view addSubview:sv];
    [sv release];

    pageControl.numberOfPages = 0;
    [pageControl addTarget:self action:@selector(pageTurn)
        forControlEvents:UIControlEventValueChanged];

    // Lädt alle Seiten
    for (int i = 0; i < INITPAGES; i++) [self addPage];
    pageControl.currentPage = 0;

    // Schiebt die Bestätigungsschaltfläche vom Bildschirm
    confirmButton.center = CGPointMake(deleteButton.center.x + 100.0f,
        deleteButton.center.y);
```

```
    // Legt die Target-Action-Paare für alle Schaltflächen fest
    [addButton addTarget:self action:@selector(requestAdd)
       forControlEvents:UIControlEventTouchUpInside];
    [cancelButton addTarget:self action:@selector(cancelDelete)
       forControlEvents:UIControlEventTouchUpInside];
    [deleteButton addTarget:self action:@selector(requestDelete)
       forControlEvents:UIControlEventTouchUpInside];
    [confirmButton addTarget:self action:@selector(confirmDelete)
       forControlEvents:UIControlEventTouchUpInside];
}
@end
```

► *Rezept 9.12: Seiten im laufenden Betrieb hinzufügen und entfernen*

Den Rezeptcode finden

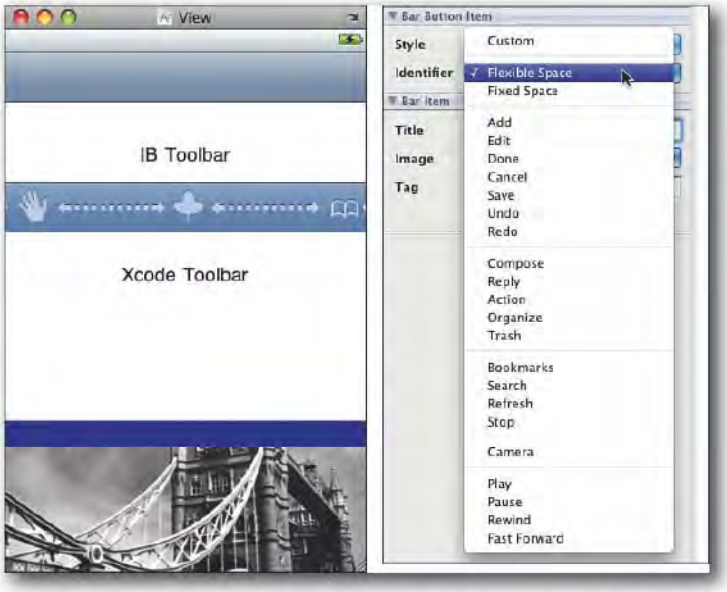
Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.18 Symbolleisten erstellen

Symbolleisten können Sie sowohl in Interface Builder (IB) als auch in Xcode erstellen, aber wenn es ans Eingemachte geht, ist es in Xcode sehr viel einfacher. Das liegt daran, dass die Oberfläche von IB zum Hinzufügen und Bearbeiten der Schaltflächen einer Symbolleiste ziemlich umständlich ist. Sie müssen ständig zwischen verschiedenen Paletten und Informationsfeldern wechseln, was sehr schnell ziemlich unübersichtlich wird.

Nachdem Sie eine Symbolleiste in eine IB-Ansicht gezogen haben, müssen Sie die einzelnen Elemente für die Schaltflächen der Leiste hinzufügen und anpassen. Ziehen Sie für jedes Element, das Sie hinzufügen möchten, einen Schaltflächenelement auf die Leiste. Zu den möglichen Elementen gehören sowohl Ansichtselemente wie Schaltflächen als auch Abstandhalter zwischen ihnen (siehe *Abbildung 9.14*, links).

Nachdem Sie die Elemente für die Leistenschaltflächen hinzugefügt haben, können Sie im zugehörigen Attribut-Informationsfeld ⌘ + 1 aus *Abbildung 9.14* (rechts) auswählen, für was für eine Art Element die einzelnen Schaltflächen stehen sollen. Mit dem Stil `Custom` können Sie benutzerdefinierte Text- und Bildelemente erstellen. Ansonsten treffen Sie Ihre Auswahl aus der Liste der vom System definierten Symbole. Dort finden Sie Symbole für das Abspielen von Medien, den Zugriff auf die Kamera, die Bearbeitung von Listen usw.



▸ *Abbildung 9.14: Das Hinzufügen von Elementen für Leistenschaltflächen kann in Interface Builder ein aufwendiger Vorgang sein.*

Wenn Sie Systemelemente verwenden, müssen Sie darauf achten, dass Sie sie so einsetzen, wie es in den Apple-Richtlinien für Benutzeroberflächen vorgesehen ist. Die Prüfer im App Store halten nicht viel von einer »kreativen« Interpretation dieser Symbole.

Vermeiden Sie es auch, eigene Schaltflächen zu erstellen, die Produkten oder Marken von Apple ähneln. Es wurden schon Apps abgelehnt, weil die Symbole darin wie das iPhone- oder das Apple-Logo aussahen.

9.18.1 Symbolleisten in Xcode erstellen

Symbolleisten können Sie in Xcode auf einfache Weise erstellen, vorausgesetzt, Sie haben einige praktische Makros definiert. Die folgenden Makros geben die richtigen Leistenschaltflächen-Elemente für die vier verfügbaren Arten von Elementen zurück:

```
#define BARBUTTON(TITLE, SELECTOR) [[[UIBarButtonItem alloc]
   initWithTitle:TITLE style:UIBarButtonItemStylePlain target:self
   action:SELECTOR] autorelease]
#define IMGBARBUTTON(IMAGE, SELECTOR) [[[UIBarButtonItem alloc]
   initWithImage:IMAGE style:UIBarButtonItemStylePlain target:self
   action:SELECTOR] autorelease]
#define SYSBARBUTTON(ITEM, SELECTOR) [[[UIBarButtonItem alloc]
   initWithBarButtonSystemItem:ITEM target:self action:SELECTOR] autore-
lease]
#define CUSTOMBARBUTTON(VIEW) [[[UIBarButtonItem alloc]
   initWithCustomView:VIEW] autorelease]
```

Mögliche Elementarten sind Text-, Bild- und Systemelemente sowie benutzerdefinierte Ansichten. Jedes dieser Makros erstellt ein automatisch freigegebenes `UIBarButtonItem`-Element, das Sie in einer `UIToolbar` platzieren können. *Rezept 9.13* zeigt diese Makros in Aktion. Sie können daran ablesen, wie Sie die einzelnen Elementarten einschließlich Abstandhaltern hinzufügen. Es ist auch möglich, in Symbolleisten benutzerdefinierte Ansichten einzubauen, wie es in *Rezept 9.13* geschieht. Dort wird eine `UISwitch`-Instanz als Leistenschaltflächen-Element verwendet, wie Sie in *Abbildung 9.15* sehen.

Das Leistenschaltflächen-Element für einen festen Abstand ist der einzige Fall, bei dem Sie mehr tun müssen, als diese Makros anzuwenden. Hier müssen Sie die Eigenschaft `width` des Elements angeben, um festzulegen, wie viel Platz es einnehmen soll.

```
@implementation TestBedViewController
- (void) action
{
   // Hier findet keine Aktion statt
}
- (void) viewDidLoad
{
   UIToolbar *tb = [[UIToolbar alloc] initWithFrame:
      CGRectMake(0.0f, 0.0f, 320.0f, 44.0f)];
   tb.center = CGPointMake(160.0f, 200.0f);
   NSMutableArray *tbitems = [NSMutableArray array];

   // Richtet die Elemente in der Symbolleiste ein
   [tbitems addObject:BARBUTTON(@"Title", @selector(action))];
   [tbitems addObject:SYSBARBUTTON(UIBarButtonSystemItemAdd,
      @selector(action))];
   [tbitems addObject:IMGBARBUTTON([UIImage
      imageNamed:@"TBUmbrella.png"], @selector(action))];
   [tbitems addObject:CUSTOMBARBUTTON([[[UISwitch alloc] init]
      autorelease])];
   [tbitems addObject:SYSBARBUTTON(UIBarButtonSystemItemFlexibleSpace,
      nil)];
   [tbitems addObject:IMGBARBUTTON([UIImage
      imageNamed:@"TBPuzzle.png"], @selector(action))];

   // Fügt einen festen Abstand von 20 Pixel Breite ein
   UIBarButtonItem *bbi = [[[UIBarButtonItem alloc]
      initWithBarButtonSystemItem:UIBarButtonSystemItemFixedSpace
      target:nil action:nil] autorelease];
   bbi.width = 20.0f;
   [tbitems addObject:bbi];
```

```
    tb.items = tbitems;
    [self.view addSubview:tb];
    [tb release];
}
@end
```

▸ *Rezept 9.13: Symbolleisten in Xcode erstellen*

▸ *Abbildung 9.15: Benutzerdefinierte Symbolleistenelemente können auch Ansichten wie diesen Schalter umfassen.*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 9 und öffnen das Projekt zu diesem Rezept.

9.18.2 Tipps und Tricks zu Symbolleisten

Bei der Arbeit mit Symbolleisten können die folgenden Tipps nützlich sein:

- **Feste Abstände können eine definierte Breite haben.** `UIBarButtonSystemItemFixedSpace`-Elemente sind die einzigen `UIBarButtonItem`-Elemente, denen eine Breite zugewiesen werden kann. Erstellen Sie daher das Abstandhalterelement, legen Sie seine Breite fest, und fügen Sie es erst dann dem Elemente-Array hinzu.
- **Verwenden Sie einen einfachen flexiblen Abstand für eine links- oder rechtsbündige Ausrichtung.** Das Hinzufügen eines einfachen `UIBarButtonSystemItemFlexibleSpace` zu Beginn einer Liste von Elementen bewirkt eine rechtsbündige Ausrichtung aller übrigen

Elemente, und das Hinzufügen am Ende sorgt für eine linksbündige Ausrichtung. Verwenden Sie zwei Abstände – einen am Beginn und einen am Ende –, um Elemente zu zentrieren.

> **Berücksichtigen Sie fehlende Elemente.** Wenn Sie Schaltflächenelemente in Leisten situationsbedingt ausblenden wollen, verwenden Sie keine flexiblen Abstände, um sie loszuwerden, sondern ersetzen Sie das betreffende Element durch einen Abstand fester Breite, der der ursprünglichen Größe des Elements entspricht. Das erhält das Layout und belässt alle anderen Elemente an der gleichen Position, sowohl vor als auch nach dem Ausblenden des Elements.

9.19 Ein letzter Punkt: Intelligente Beschriftungen

Leider ist die eingebaute Klasse `UILabel` nicht sehr intelligent, wenn es darum geht, antippbare Elemente wie Telefonnummern und Webadressen bereitzustellen. An dieser Stelle kommt die Klasse `UITextView` ins Spiel. Textansichten weisen in der Version 3.0 die neue Eigenschaft `dataDetectorTypes` auf, die angibt, welche Datentypen in antippbare URLs umgewandelt werden. Die verfügbaren Typen sind Telefonnummern (`UIDataDetectorTypePhoneNumber`) und Links (`UIDataDetectorTypeLink`). Um alle Typen zu aktivieren, verwenden Sie wie im folgenden Beispiel `UIDataDetectorTypeAll`:

```
- (void) viewDidLoad
{
    UITextView *tv = (UITextView *)[self.view viewWithTag:101];
    tv.dataDetectorTypes = UIDataDetectorTypeAll;
}
```

In Interface Builder finden Sie im Attribut-Informationsfeld für Textansichten auch einzelne Markierungsfelder für Links und Telefonnummern.

▸ *Abbildung 9.16: Diese »Beschriftung« ist in Wirklichkeit eine Textansicht mit aktivierter Datenerkennung.*

Wenn Sie `UILabel`- durch `UITextView`-Instanzen ersetzen, müssen Sie die Rollbarkeit der Ansicht deaktivieren. Setzen Sie dazu im Code oder in Interface Builder die Eigenschaft `editable` der Ansicht auf `NO`. Verwenden Sie Wagenrücklaufkonstanten (`\n`), um Zeilenumbrüche vorzusehen, und bedenken Sie die Textausrichtung genau. *Abbildung 9.16* zeigt eine `UITextView`-Ansicht, die als Beschriftung dient und automatisch einen URL erstellt.

Bei der Arbeit mit eingebetteten URLs müssen Sie darauf achten, dass Links die Benutzer zu den angeforderten Ressourcen führen, ohne eine weitere Bestätigung abzuwarten. Bei Telefonnummern dagegen wird der Benutzer vor dem Wählen um Bestätigung gebeten.

Hinweis

Wollen Sie schon immer einmal andere Schriftarten verwenden als diejenigen, die Cocoa Touch bietet? Das iPhone-Open-Source-Projekt FontLabel (*http://github.com/zynga/FontLabel*) von iPhone-Guru Kevin Ballard verwendet die Core Graphics-Klasse `CGFont`, um die Einschränkungen der iPhone-Klasse `UIFont` zu umgehen. Während der Abfassung dieses Buches befand sich FontLabel noch aktiv in der Entwicklung, wobei es regelmäßig Aktualisierungen gab.

9.20 Zusammenfassung

Dieses Kapitel behandelte viele Möglichkeiten, um mit den Steuerelementen in Ihren Anwendungen umzugehen und den größten Nutzen aus ihnen zu ziehen. Bevor Sie zum nächsten Kapitel übergehen, überdenken Sie die folgenden Aspekte:

- Dass ein Element zur Klasse `UIControl` gehört, heißt noch lange nicht, dass Sie es wie nicht wie eine `UIView`-Ansicht behandeln können. Weisen Sie ihm Unteransichten zu, passen Sie es in der Größe an, animieren Sie es, verschieben Sie es auf dem Bildschirm, oder markieren Sie es für spätere Verwendung.
- Core Graphics und Quartz 2D ermöglichen es Ihnen, grafische Elemente nach Bedarf zu erstellen. Kombinieren Sie den Komfort der SDK-Klassen mit den Vorteilen der Echtzeitbearbeitung, um Ihrer Darstellung Würze hinzuzufügen.
- Wenn das Steuerelement, das Sie brauchen, im iPhone SDK nicht vorhanden ist, passen Sie ein vorhandenes Steuerelement an, oder erstellen Sie ein völlig neues.
- Apple gibt Spitzenbeispiele für hervorragende Benutzeroberflächen. Ahmen Sie diese Vorbilder nach, wenn Sie neue Arten der Interaktion erstellen. Ein Beispiel ist die Bestätigungsschaltfläche, die in diesem Kapitel gegen versehentliches Löschen eingesetzt wurde.
- Interface Builder (IB) bietet nicht immer die beste Möglichkeit zum Gestalten von Oberflächen. Wenn es um Symbolleisten geht, können Sie durch die Verwendung von Xcode Zeit sparen, anstatt jedes einzelne Element manuell in IB anzupassen.

10 Benutzer benachrichtigen

Manchmal müssen Sie die Aufmerksamkeit des Benutzers erregen. Eventuell kommen neue Nachrichten herein oder der Systemstatus ändert sich. Vielleicht möchten Sie Ihrem Benutzer auch mitteilen, dass eine Wartezeit besteht, bevor irgendetwas Weiteres passiert – oder dass eben diese Wartezeit vorüber ist und er wieder aufmerksam sein muss. Das iPhone verfügt über viele Möglichkeiten, um die Aufmerksamkeit des Benutzers zu erlangen: von Benachrichtigungen über Fortschrittsanzeigen bis zu akustischem Klingeln. In diesem Kapitel erfahren Sie, wie Sie diese Hinweise in Ihre Programme einbauen können, und Sie werden Ihr Spektrum an Benutzerbenachrichtigungen erweitern. Außerdem sehen Sie Praxisbeispiele, die diese Klassen veranschaulichen, und Sie lernen, wie Sie die Aufmerksamkeit Ihres Benutzers zur richtigen Zeit erhalten.

10.1 Benutzer durch Benachrichtigungen direkt ansprechen

Benachrichtigungen »sprechen« mit Ihren Benutzern. Member der Klassen `UIAlertView` und `UIActionSheet` werden über andere Ansichten dargestellt, um ihre Nachricht mitzuteilen. Diese schlanken Klassen fügen Ihren Programmen eine Dialogfähigkeit hinzu, die in zwei Richtungen funktioniert. Benachrichtigungen kommunizieren optisch mit Benutzern und können diese zu einer Antwort auffordern. Dazu präsentieren Sie die Benachrichtigung auf dem Bildschirm, bekommen vom Benutzer eine Bestätigung und blenden die Benachrichtigung anschließend aus, damit der Benutzer mit anderen Aufgaben fortfahren kann.

Falls Sie meinen, dass Benachrichtigungen nichts weiter sind als Nachrichten mit einer angehängten **OK**-Schaltfläche (wie in *Abbildung 10.1*), denken Sie noch einmal darüber nach. `UIAlertSheet`-Objekte stellen eine unglaubliche Flexibilität bereit. Mit Benachrichtigungs-Sheets können Sie ganze

Menüs, Texteingaben, Abfragen und mehr erstellen. In den Rezepten dieses Kapitels erfahren Sie, wie Sie ein breites Spektrum nützlicher Benachrichtigungen in Ihren eigenen Programmen verwenden können.

10.1.1 Einfache Benachrichtigungen erstellen

Stellen Sie ein `UIAlertView`-Objekt zum Anlegen von Benachrichtigungs-Sheets bereit, und initialisieren Sie es mit einem Titel und einem Schaltflächen-Array. Der Titel ist ein `NSString`, und das Schaltflächen-Array besteht aus `NSStrings`, die jeweils für eine einzelne Schaltfläche stehen.

Die im Folgenden gezeigte Methode erstellt die einfachste mögliche Art von Benachrichtigung und stellt sie dar, nämlich eine Nachricht mit einer **OK**-Schaltfläche. Die Benachrichtigung wird automatisch freigegeben, wodurch die Notwendigkeit für Delegierungen und Callbacks vermieden wird. Wenn Sie Benachrichtigungen nicht automatisch freigeben, müssen Sie dafür sorgen, dass eine Delegierung die Verantwortung für die Freigabe übernimmt, nachdem der Benutzer auf die Schaltfläche getippt hat.

```
- (void) showAlert: (NSString *) theMessage
{
   UIAlertView *av = [[[UIAlertView alloc] initWithTitle:@"Title"
      message:theMessage delegate:nil
      cancelButtonTitle:@"OK" otherButtonTitles:nil] autorelease];
   [av show];
}
```

Um weitere Schaltflächen hinzuzufügen, müssen Sie sie als Parameter von `otherButtonTitles:` angeben, wobei Sie die Liste mit `nil` abschließen müssen. Dieses Argument nimmt eine beliebige Anzahl von Parametern entgegen, wobei `nil` der Methode mitteilt, dass die Liste abgeschlossen ist.

Der folgende Code erstellt eine Benachrichtigung mit drei Schaltflächen (**Cancel**, **Option** und **OK**). Da keine Delegierung deklariert wird, gibt es keine Möglichkeit, die Benachrichtigung zu erfassen und festzustellen, auf welche der drei Schaltflächen der Benutzer getippt hat. Die Benachrichtigung wird angezeigt, bis der Benutzer darauf tippt, und wird dann ohne weitere Auswirkung entfernt.

```
- (void) showAlert: (NSString *) theMessage
{
   UIAlertView *av = [[[UIAlertView alloc] initWithTitle:@"Title"
      message:theMessage delegate:nil cancelButtonTitle:@"Cancel"
      otherButtonTitles: @"Option", @"OK", nil] autorelease];
   [av show];
}
```

Bei der Arbeit mit Benachrichtigungen ist der Platz häufig von entscheidender Bedeutung. Wenn Sie mehr als zwei Schaltflächen verwenden, wird die Benachrichtigung in mehreren Zeilen angezeigt. *Abbildung 10.1* zeigt zwei verschiedene Benachrichtigungen: einmal mit zwei Schaltflächen (die nebeneinander angezeigt werden), einmal mit drei Schaltflächen (Zeile für Zeile angeordnet). Die Anzahl der Schaltflächen in einer Benachrichtigung sollte nie mehr als drei oder vier betragen.

Weniger Schaltflächen sind besser, wobei eine oder zwei ideal sind. Wenn Sie mehr Schaltflächen benötigen, sollten Sie statt Benachrichtigungsansichten lieber Action-Sheet-Objekte verwenden, die weiter hinten in diesem Kapitel besprochen werden.

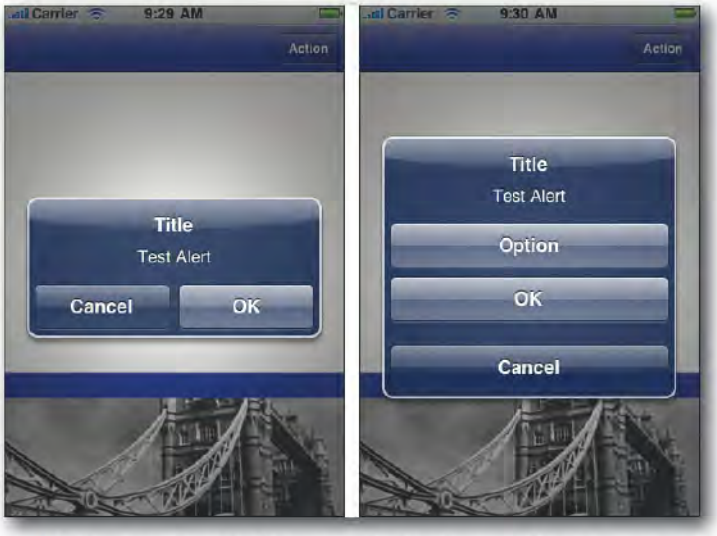
Bei `UIAlertview`-Objekten wird keine »Standard«-Schaltfläche hervorgehoben. Die einzige Hervorhebung erhält die Abbruch-Schaltfläche (**Cancel**), wie Sie in *Abbildung 10.1* sehen. Abbruch-Schaltflächen stehen in Benachrichtigungen links oder unten.

10.1.2 Delegierungen für Benachrichtigungen

Zum Abrufen der Benutzerauswahl werden in Benachrichtigungen Delegierungen eingesetzt. Setzen Sie die Delegierung auf Ihr primäres (aktives) `UIViewController`-Objekt, es sei denn, Sie haben zwingende Gründe, dies nicht zu tun. Die Delegierung implementiert das Protokoll `UIAlertViewDelegate`. `UIAlertView`-Instanzen erfordern diese Delegierungsunterstützung, um mindestens das Antippen von Schaltflächen zu verarbeiten.

Delegierungsmethoden ermöglichen die Anpassung der Reaktion darauf, dass verschiedene Schaltflächen betätigt werden. Sie können diese Delegierungsunterstützung wie schon gezeigt weglassen, wenn Sie lediglich eine Nachricht mit einer **OK**-Schaltfläche ausgeben möchten.

Nachdem der Benutzer die Benachrichtigung gesehen und darauf reagiert hat, rufen Sie den Delegierungsmethodenaufruf `alertView: clickedButtonAtIndex:` auf. Aus seinem zweiten Argument können Sie ablesen, welche Schaltfläche angetippt wurde. Die Nummerierung der Schaltflächen beginnt mit null, wobei eine Abbruch-Schaltfläche, sofern sie definiert ist, immer die Nummer null trägt. Auch wenn sie in einigen Ansichten links und in anderen unten erscheint, bleibt ihre Nummerierung stets gleich. Dies gilt jedoch nicht für Action-Sheet-Objekte, die weiter hinten in diesem Kapitel behandelt werden.



► *Abbildung 10.1: Benachrichtigungen funktionieren am besten, wenn sie nur eine oder zwei Schaltflächen enthalten (links). Gibt es mehr als zwei Schaltflächen, werden diese in Listenform angeordnet, was zu einer weniger eleganten Darstellung führt (rechts).*

Das folgende Beispiel einer Benachrichtigungsdarstellung mit Callback gibt die Nummer der ausgewählten Schaltfläche an die Debugging-Konsole aus:

```
@interface TestBedViewController : UIViewController <UIAlertViewDelegate>
@end

@implementation TestBedViewController
- (void) alertView:(UIAlertView *) alertView
   clickedButtonAtIndex: (int) index
{
   printf("User selected button %d\n", index);
   [alertView release];
}

- (void) showAlert: (NSString *) message
{
   UIAlertView *av = [[UIAlertView alloc] initWithTitle:@"Title"
      message:message delegate:self cancelButtonTitle:@"Cancel"
      otherButtonTitles:@"One", @"Two", @"Three", nil];
   av.tag = MAIN_ALERT;
   [av show];
}
@end
```

Wenn Sie mit mehreren Benachrichtigungen auf einmal arbeiten, sollten Sie die Objekte kennzeichnen. Dadurch können Sie festlegen, welche Benachrichtigung einen gegebenen Callback hervorgerufen hat. Anders als Steuerelemente mit Target-Action-Paaren lösen alle Benachrichtigungen dieselbe Methode aus. Durch eine `switch`-Anweisung, die sich auf die Benachrichtigungskennzeichen stützt, können Sie die Reaktionen auf die einzelnen Benachrichtigungen unterscheiden.

Hinweis

Beachten Sie, dass die Benachrichtigung in diesem Code nicht automatisch freigegeben wird. Das Objekt wird im Callback freigegeben.

10.1.3 Die Benachrichtigung anzeigen

Wie Sie gesehen haben, wird die Methode `show` verwendet, um die Benachrichtigung anzuweisen, auf dem Bildschirm zu erscheinen. Die Darstellung der Benachrichtigung erfolgt modal, d. h., der Bildschirm dahinter wird abgedunkelt und die Benutzerinteraktion mit der Anwendung hinter dem modalen Fenster wird blockiert. Diese modale Interaktion dauert an, bis der Benutzer die Benachrichtigung bestätigt, indem er auf eine Schaltfläche tippt, gewöhnlich auf **OK** oder **Cancel**.

Nachdem Sie das Sheet für die Benachrichtigung erstellt haben, müssen Sie sie anpassen, indem Sie ihre Eigenschaft `message` anpassen. Dabei handelt es sich um den optionalen Text, der zwischen

dem Titel der Benachrichtigung und ihren Schaltflächen erscheint. Wie Sie in den weiteren Rezepten in diesem Kapitel sehen, können Sie auch den Rahmen der Benachrichtigung ändern und Unteransichten hinzufügen.

10.1.4 Benachrichtigungsklassen

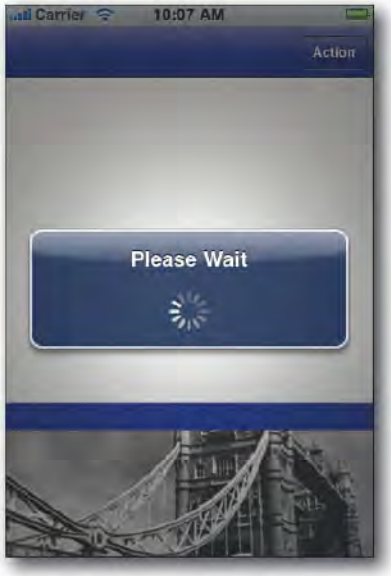
In den ersten Versionen der iPhone-Firmware wurden `UIActionSheet` und `UIAlertView` von derselben Klasse implementiert, nämlich `UIAlertView`. Diese eine Klasse war sowohl für eingeblendete Benachrichtigungen als auch für Menüfunktionen zuständig. Später ersetzte Apple Benachrichtigungs-Sheets durch `UIModalView` und erstellte diese neuen Objekte als Unterklassen dieser Basisklasse.

Dann wiederum entfernte Apple `UIModalView`, und in den neuen Versionen des SDK sind `UIActionSheet` und `UIAlertView` nicht mehr von dieser Klasse abgeleitet (sondern von `UIView`). Wie die früheren Versionen sind sie sich in ihrem Verhalten ähnlich und verwenden auch eine ähnliche zugrunde liegende Technologie für die Darstellung auf dem Bildschirm.

Aus dieser geschichtlichen Entwicklung lässt sich eine wichtige Lehre ziehen. Auch wenn Apple hinter den APIs und den veröffentlichten Methoden steht, können Sie sich nicht darauf verlassen, dass die zugrunde liegenden Klassen unverändert bleiben. Das iPhone ist eine Plattform, die sich rasch weiterentwickelt.

10.2 Rezept: Benachrichtigungen ohne Schaltflächen

Wenn Sie eine asynchrone Nachricht anzeigen möchten, ohne dass der Benutzer darauf reagieren muss, können Sie eine `UIAlertView`-Instanz ohne Schaltflächen erstellen. Eine solche Benachrichtigung können Sie wie die normale Version mit Schaltflächen anlegen und anzeigen. Damit haben Sie eine hervorragende Möglichkeit, um eine »Bitte warten«-Nachricht anzuzeigen, wie Sie in *Abbildung 10.2* sehen.



► *Abbildung 10.2: Wenn Sie einer Benachrichtigung keine Schaltflächen hinzufügen, können Sie auf fortlaufende Vorgänge hinweisen.*

Benachrichtigungen ohne Schaltflächen stellen eine besondere Herausforderung dar, da sie keinen korrekten Callback zur Delegierungsmethode durchführen. Sie verschwinden nicht automatisch, auch dann nicht, wenn sie angetippt werden. Stattdessen müssen Sie sie manuell entfernen, wenn sie nicht mehr angezeigt werden sollen. Dazu rufen Sie `dismissWithClickedButtonIndex:animated:` auf.

In *Rezept 10.1* wird unter dem Titel der Benachrichtigung eine Instanz von `UIActivityIndicator` eingefügt, um die rotierende Aktivitätsanzeige darzustellen, die Sie unten in der Benachrichtigung aus *Abbildung 10.2* sehen. Dadurch erhält der Benutzer eine optische Rückmeldung darüber, dass irgendein Vorgang läuft, der zurzeit eine Benutzerinteraktion unterbindet. Dieser »Vorgang« ist in *Rezept 10.1* einfach eine Wartezeit von drei Sekunden. In der Praxis setzen Sie diese Art von Benachrichtigung sinnvoller ein.

Sobald Sie eine Benachrichtigung erstellt haben, funktioniert sie wie eine andere Ansicht, sodass Sie ihr Unteransichten hinzufügen und ihr Erscheinungsbild ändern können. Leider sind in der Bibliothek von Interface Builder keine Benachrichtigungsansichten zu finden, sodass die gesamte Anpassung im Code erfolgen muss, wie Sie hier sehen. In *Rezept 10.1* wird die Unteransicht erstellt und hinzugefügt, nachdem die Benachrichtigung mit `show` angezeigt wurde. Durch die Darstellung der Benachrichtigung erhalten Sie eine echte Bildschirmansicht, die Sie ändern und anpassen können.

Achten Sie darauf, dass Benachrichtigungen in eigenen Fenstern angezeigt werden. Diese Ansicht gehört nicht zur Hierarchie des Hauptfensters. Außerdem müssen Sie daran denken, dass das Entfernen von Schaltflächen die Gesamtgeometrie der Darstellung durcheinander bringen kann. Der Platz, den die Schaltflächen normalerweise einnahmen, verschwindet nicht. In *Rezept 10.1* wird dieser Platz für die Aktivitätsanzeige genutzt. Wenn Sie nur Text verwenden, fügen Sie einen Wagenrücklauf (@"\n") am Anfang der Nachricht ein, um die leere Fläche am unteren Rand, in der sich normalerweise die Schaltflächen befinden, durch Leerraum am oberen Rand auszugleichen.

```
- (void) performDismiss
{
    [baseAlert dismissWithClickedButtonIndex:0 animated:NO];
}

- (void) action: (UIBarButtonItem *) item
{
    baseAlert = [[[UIAlertView alloc] initWithTitle:@"Please Wait"
        message:nil delegate:self cancelButtonTitle:nil
        otherButtonTitles: nil] autorelease];
    [baseAlert show];

    // Erstellt die Aktivitätsanzeige und fügt sie hinzu
    UIActivityIndicatorView *aiv = [[UIActivityIndicatorView alloc]
        initWithActivityIndicatorStyle:
        UIActivityIndicatorViewStyleWhiteLarge];
```

```
    aiv.center = CGPointMake(baseAlert.bounds.size.width / 2.0f,
        baseAlert.bounds.size.height - 40.0f);
    [aiv startAnimating];
    [baseAlert addSubview:aiv];
    [aiv release];

    // Entfernt die Schaltfläche automatisch nach drei Sekunden
    [self performSelector:@selector(performDismiss) withObject:nil
        afterDelay:3.0f];
}
```

► *Rezept 10.1: Benachrichtigungen ohne Schaltflächen anzeigen und entfernen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.3 Rezept: Modale Benachrichtigungen mit Ausführungsschleifen erstellen

Die indirekte Natur von Benachrichtigungen, vor allem die Verwendung von Delegierungs-Callbacks, kann zu unnötig kompliziertem Code führen. Es ist jedoch relativ einfach, eine eigene Klasse zu erstellen, die unmittelbar den Wert der ausgewählten Schaltfläche zurückgibt. Betrachten Sie dazu den folgenden Code, der eine Antwort von der Benachrichtigung aus *Abbildung 10.3* (links) anfordert und die Antwort verwendet, die die Klassenmethode zurückgibt.

```
- (void) confirm: (id) sender
{
    NSUInteger answer = [ModalAlert confirm:@"Are you sure?"];
    [self showAlert:[NSString stringWithFormat:@"You %@ confirm",
        answer ? @"did" : @"did not"]];
}
```

Eine Benachrichtigung zu erstellen, die unmittelbar ein Ergebnis zurückgibt, erfordert Einfallsreichtum. Die Klasse `ModalAlert` in *Rezept 10.2* weist eine zweite Ausführungsschleife auf. In diesem Code wird die Benachrichtigung so erstellt, wie Sie es normalerweise erwarten, aber sobald sie dargestellt ist, wird `CFRunLoopRun()` aufgerufen. Dadurch hält die Methode an und wartet darauf, dass der Benutzer die Interaktion mit der Benachrichtigung abschließt. Die Methode schreitet nicht voran, solange die Ausführungsschleife läuft.

► *Abbildung 10.3: Diese modalen Benachrichtigungen geben unmittelbare Antworten zurück, da sie ihre eigenen Ausführungsschleifen enthalten.*

Es ist die Aufgabe der Delegierungsklasse dieser modalen Benachrichtigung (`ModalAlertDelegate`), die Ausführungsschleife beim Antippen einer Schaltfläche zu beenden und den Wert des gewählten Elements zurückzugeben. Wenn der Benutzer die Interaktion abschließt, kann die aufrufende Methode endlich über die Ausführungsschleife hinaus fortgesetzt werden.

Die Klasse `ModalAlert` enthält zwei Klassenmethoden, die die Schaltflächen **CANCEL/OK** bzw. **YES/NO** aus *Abbildung 10.3* darstellen. Sie geben 0 und 1 bzw. 1 und 0 zurück (die Auswahl von **CANCEL** und von **No** führt jeweils zum Wert 0).

Dieses Rezept können Sie leicht für eine andere Anzahl von Schaltflächen und für andere Titel anpassen. Wenn Sie nicht sicher sind, mit wie vielen Schaltflächen Sie es zu tun bekommen, können Sie an die benutzerdefinierten Klassen ein Array übergeben. Mit der `UIAlertView`-Methode `addButtonWithTitle:` können Sie die Deklaration mit variabler Argumentzahl umgehen (der Initialisierungsaufruf enthält also eine Reihe von durch Kommata getrennten Argumenten mit schließendem `nil`), um Schaltflächen aus einem Array hinzuzufügen, beispielsweise wie folgt:

```
ModalAlertDelegate *madelegate = [[ModalAlertDelegate alloc]
   initWithRunLoop:currentLoop];
UIAlertView *alertView = [[UIAlertView alloc] initWithTitle:question
   message:nil delegate:madelegate cancelButtonTitle:cancelTitle
   otherButtonTitles:nil];
for (int i = 1; i < buttons.count; i++) [alertView
   addButtonWithTitle:[buttons objectAtIndex:i]];
[alertView show];
```

Mit dieser Methode können Sie zwar eine Benachrichtigung nach der anderen ausführen, doch kann es dabei vorkommen, dass sich die Aufrufe gegenseitig im Wege stehen. Räumen Sie genug Zeit ein, damit die vorhergehende Benachrichtigung verschwinden kann, bevor Sie die nächste darstellen. Sollte eine Benachrichtigung nicht auf dem Bildschirm erscheinen, kann das möglicherweise an einer solchen Überschneidung liegen. Verwenden Sie in einem solchen Fall einen verzögerten Selektor, um die nächste Benachrichtigungsanforderung aufzurufen. Eine Zehntelsekunde bietet ausreichend Zeit, um die Einblendung der neuen Benachrichtigung zu ermöglichen.

```
@interface ModalAlertDelegate : NSObject <UIAlertViewDelegate>
{
   CFRunLoopRef currentLoop;
   NSUInteger index;
}
@property (readonly) NSUInteger index;
@end

@implementation ModalAlertDelegate
@synthesize index;

// Initialisierung mit der bereitgestellten Ausführungsschleife
-(id) initWithRunLoop: (CFRunLoopRef)runLoop
{
   if (self = [super init]) currentLoop = runLoop;
   return self;
}

// Ruft die Ergebnisse ab, nachdem der Benutzer auf eine Schaltfläche
// getippt hat
-(void) alertView: (UIAlertView*)aView clickedButtonAtIndex:
      (NSInteger)anIndex
{
   index = anIndex;
   CFRunLoopStop(currentLoop);
}
@end

@implementation ModalAlert
+(NSUInteger) queryWith: (NSString *)question
      button1: (NSString *)button1 button2: (NSString *)button2
{
   CFRunLoopRef currentLoop = CFRunLoopGetCurrent();

   // Erstellt die Benachrichtigung
   ModalAlertDelegate *madelegate = [[ModalAlertDelegate alloc]
      initWithRunLoop:currentLoop];
```

```
    UIAlertView *alertView = [[UIAlertView alloc]
        initWithTitle:question message:nil delegate:madelegate
        cancelButtonTitle:button1 otherButtonTitles:button2, nil];
    [alertView show];

    // Wartet auf Antwort
    CFRunLoopRun();

    // Ruft die Antwort ab
    NSUInteger answer = madelegate.index;
    [alertView release];
    [madelegate release];
    return answer;
}

// Stellt eine Ja/Nein-Frage
+ (BOOL) ask: (NSString *) question
{
    return ([ModalAlert queryWith:question
        button1: @"Yes" button2: @"No"] == 0);
}

// Stellt eine Abbrechen/OK-Frage

+ (BOOL) confirm: (NSString *) statement
{
    return [ModalAlert queryWith:statement
        button1: @"Cancel" button2: @"OK"];
}
@end
```

▶ *Rezept 10.2: Benachrichtigungen erstellen, die unmittelbar Ergebnisse zurückgeben*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.4 Rezept: Texteingaben vom Benutzer anfordern

Benachrichtigungsansichten bilden eine besonders einfache Möglichkeit, um Texteingaben von Benutzern anzufordern. Die Instanzen dieser Ansichten übernehmen den Bildschirm und zwingen den Benutzer dazu, eine Antwort zu geben, bevor mit den Ergebnissen weitergearbeitet werden kann. Wie

in *Rezept 10.2* werden die Antworten direkt abgerufen, ohne dass Delegierungs-Callbacks vonnöten sind. Der folgende Code fragt z. B. den Namen des Benutzers ab und verwendet dann unmittelbar den betreffenden String:

```
-(void) action: (UIBarButtonItem *) item
{
   NSString *answer = [ModalAlert ask:@"What is your name?"
      withTextPrompt:@"Name"];
   [self showAlert:[NSString stringWithFormat:
      @"Nice to meet you, %@.", answer]];
}
```

Dazu können Sie den Ansatz mit Ausführungsschleifen und die Klassen `ModalAlert` und `ModalAlertDelegate` aus *Rezept 10.2* verwenden. Es sind nur einige leichte Änderungen erforderlich.

Rezept 10.3 erstellt eine Benachrichtigung, fügt ihr ein Textfeld hinzu und zeigt sie an. Leider verhindert die normale Position von Benachrichtigungen auf dem Bildschirm die Verwendung der Tastatur für das Textfeld, da die Tastatur die Benachrichtigung teilweise verdeckt. Dieses Problem können Sie lösen, indem Sie die Benachrichtigung so positionieren, dass die Tastatur unter ihr erscheint.

Die folgende Methode verschiebt das Textfeld in einen Bereich oberhalb des Raums, den normalerweise die Tastatur einnimmt, sodass sich die beiden Elemente nicht verdecken. Hier wird ein hartkodierter Wert für den Mittelpunkt der Benachrichtigung verwendet. Eine bessere Möglichkeit besteht darin, die Grenzen der Tastatur abzufragen und daraus zu berechnen, wie weit das Textfeld verschoben werden muss.

```
// Verschiebt die Benachrichtigung so, dass Platz für die Tastatur
// bleibt
- (void) moveAlert: (UIAlertView *) alertView
{
   CGContextRef context = UIGraphicsGetCurrentContext();
   [UIView beginAnimations:nil context:context];
   [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
   [UIView setAnimationDuration:0.25f];
   if (![self isLandscape])
      alertView.center = CGPointMake(160.0f, 180.0f);
   else
      alertView.center = CGPointMake(240.0f, 90.0f);
   [UIView commitAnimations];

   [[alertView viewWithTag:TEXT_FIELD_TAG] becomeFirstResponder];
}
```

Der vorstehende Code, der in *Rezept 10.3* aufgerufen wird, schiebt die Benachrichtigung aus dem Weg und legt das Textfeld als First Responder fest. Dadurch können die Benachrichtigung und die Tastatur gleichzeitig angezeigt werden, wie Sie in *Abbildung 10.4* sehen.

Hinweis

Die Konsolenwarnung `wait_fences: failed to receive reply` deutet gewöhnlich darauf hin, das ein Kindelement vor seinem Elternelement gerendert wird. Diese Warnung können Sie verhindern, indem Sie in der Methode `alertView:clickedButtonAtIndex:` benutzerdefinierte Ansichten aus einer Benachrichtigung entfernen.

```
+(NSString *) textQueryWith: (NSString *)question
   prompt: (NSString *)prompt button1: (NSString *)button1
   button2:(NSString *) button2
{
   // Erstellt die Benachrichtigung
   CFRunLoopRef currentLoop = CFRunLoopGetCurrent();
   ModalAlertDelegate *madelegate = [[ModalAlertDelegate alloc]
      initWithRunLoop:currentLoop];
   UIAlertView *alertView = [[UIAlertView alloc]
      initWithTitle:question message:@"\n" delegate:madelegate
      cancelButtonTitle:button1 otherButtonTitles:button2, nil];

   // Erstellt das Textfeld
   UITextField *tf = [[UITextField alloc]
      initWithFrame:CGRectMake(0.0f, 0.0f, 260.0f, 30.0f)];
   tf.borderStyle = UITextBorderStyleRoundedRect;
   tf.tag = TEXT_FIELD_TAG;
   tf.placeholder = prompt;
   tf.clearButtonMode = UITextFieldViewModeWhileEditing;
   tf.keyboardType = UIKeyboardTypeAlphabet;
   tf.keyboardAppearance = UIKeyboardAppearanceAlert;
   tf.autocapitalizationType = UITextAutocapitalizationTypeWords;
   tf.autocorrectionType = UITextAutocorrectionTypeNo;
   tf.contentVerticalAlignment =
      UIControlContentVerticalAlignmentCenter;

   // Zeigt die Benachrichtigung an und wartet auf das Ende der
   // Darstellung
   [alertView show];
   while (CGRectEqualToRect(alertView.bounds, CGRectZero));

   // Bestimmt den Mittelpunkt des Textfeldes und fügt dieses hinzu
   CGRect bounds = alertView.bounds;
   tf.center = CGPointMake(bounds.size.width / 2.0f,
      bounds.size.height / 2.0f - 10.0f);
```

```
    [alertView addSubview:tf];
    [tf release];

    // Legt das Feld als First Responder fest und bewegt es an die
    // richtige Position
    [madelegate performSelector:@selector(moveAlert)
       withObject:alertView afterDelay: 0.7f];

    // Startet die Ausführungsschleife
    CFRunLoopRun();

    // Ruft die Auswahl des Benutzers ab
    NSUInteger index = madelegate.index;
    NSString *answer = [[madelegate.text copy] autorelease];
    if (index == 0) answer = nil; // cancel ist in Position 0

    [alertView release];
    [madelegate release];
    return answer;
}
```

▸ *Rezept 10.3: Eine modale Benachrichtigung für die Textabfrage erstellen*

▸ *Abbildung 10.4: Wenn Sie sorgfältig mit dem Platz umgehen und Titel und erklärenden Text weglassen, können Sie in einer `UIAlertView`-Ansicht mehrere Textfelder auf einmal darstellen. Allerdings sollten Sie sich lieber auf ein oder zwei Felder beschränken.*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.5 Rezept: Benachrichtigungsansichten mit variabler Anzahl von Argumenten

Es gibt Methoden, die eine variable Anzahl von Argumenten annehmen und mit Auslassungspunkten hinter dem letzten Parameter deklariert werden. Sowohl `NSLog` als auch `printf` fallen in diese Kategorie. Diesen Methoden können Sie einen Formatierungsstring sowie eine beliebige Anzahl von Argumenten übergeben.

Da die meisten Benachrichtigungen hauptsächlich Text enthalten, ist es praktisch, Methoden für Benachrichtigungen mit Formatierungsstrings zu erstellen. *Rezept 10.4* legt die Methode `say:` an, die die an sie übergebenen Argumente sammelt und daraus einen String zusammensetzt. Dieser String wird dann an eine automatisch freigegebene Benachrichtigungsansicht übergeben, was eine praktische unmittelbare Darstellung ermöglicht.

Die Methode `say:` analysiert ihre Parameter in keiner Weise, sondern fasst einfach das erste Argument als Formatierungsstring auf und übergibt die restlichen Elemente an die `NSString`-Methode `initWithFormat:arguments:`. Dadurch entsteht ein String, der anschließend als Titel zu einer Benachrichtigungsansicht mit einer Schaltfläche übergeben wird.

Wenn Sie eigene Hilfsmethoden mit variabler Argumentzahl definieren, können Sie mehrere Schritte auslassen, bei denen Sie sonst einen String mit einem bestimmten Format zusammensetzen und dann eine Methode aufrufen müssten. Mit `say:` können Sie so etwas wie folgt in einem einzigen Aufruf erledigen:

```
[self say:@"I am so happy to meet you, %@", yourName];
```

Hinweis

Um die in *Rezept 10.4* gezeigten Aufrufe mit variabler Argumentzahl nutzen zu können, müssen Sie `<stdarg.h>` importieren.

```
- (void) say: (id)formatstring,...
{
   va_list arglist;
   va_start(arglist, formatstring);
   id statement = [[NSString alloc] initWithFormat:formatstring
      arguments:arglist];
   va_end(arglist);
```

```
    UIAlertView *av = [[[UIAlertView alloc] initWithTitle:statement
        message:nil delegate:self cancelButtonTitle:@"Okay"
        otherButtonTitles:nil] autorelease];
    [av show];
    [statement release];
}

-(void) action: (UIBarButtonItem *) item
{
    NSDateFormatter *formatter = [[[NSDateFormatter alloc] init]
        autorelease];
    formatter.dateFormat = @"MM/dd/YY HH:mm:ss";
    NSString *timestamp = [formatter stringFromDate:[NSDate date]];

    [self say:@"At the chime, the time will be %@", timestamp];
}
```

► *Rezept 10.4: Eine* `UIAlertView`*-Ansicht mithilfe einer Methode mit variabler Argumentzahl erstellen*

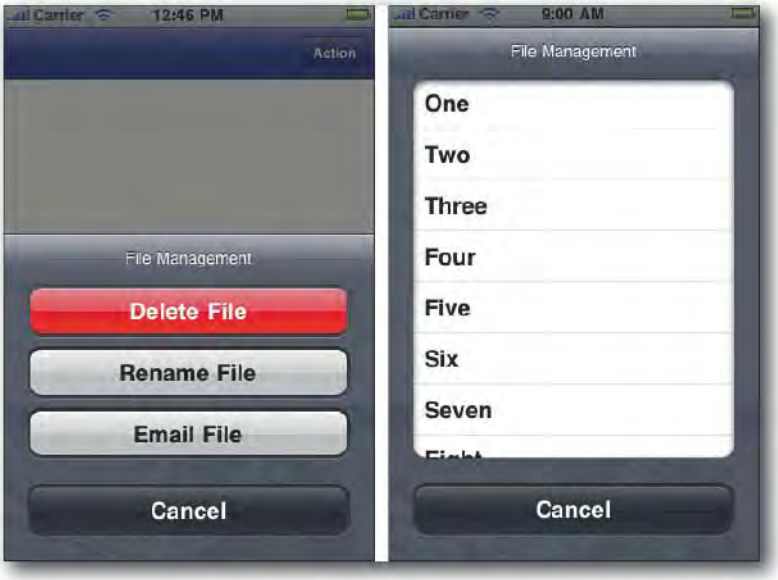
Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.6 Rezept: Einfache Menüs darstellen

Wenn es um Menüs geht, bilden `UIActionSheet`-Instanzen die Möglichkeit für das iPhone. Sie verschieben die Wahlmöglichkeiten (im Grunde eine Liste von Schaltflächen, die mögliche Aktionen auflisten) auf den Bildschirm und warten darauf, dass der Benutzer reagiert. Action-Sheets unterscheiden sich von Pop-Ups, die unabhängig von der Oberfläche sind und sich eher eignen, um die Aufmerksamkeit zu erregen. Menüs gleiten ins Sichtfeld und gliedern sich besser in den laufenden Vorgang im Programm ein. Cocoa Touch unterstützt zwei Wege zur Darstellung von Menüs:

- `showInView` Die Darstellung Ihres Sheets in einer Ansicht ist der ideale Weg zur Verwendung von Menüs und wird hier verwendet. Diese Methode schiebt das Menü von unten ins Blickfeld (siehe *Abbildung 10.5*).
- `showFromToolBar:` und `showFromTabBar` Symbolleisten, Symbolbildschirme und andere Arten von Anzeigeleisten, die horizontal gruppierte Schaltflächen enthalten (wie sie in vielen Programmen am unteren Rand zu sehen sind), stimmen das Menü durch diese Methoden mit der Titelleiste ab und schieben sie genau dorthin, wo sie sein soll.



▸ *Abbildung 10.5: Verwenden Sie* `showInView:`*, um einfache Menüdarstellungen zu gestalten. Das Menü gleitet von unten in die Ansicht. Obwohl die Schaltfläche* **Delete File** *hier grau erscheint (links), ist sie auf dem iPhone rot und weist auf unwiderrufliche Aktionen mit möglichen negativen Auswirkungen für den Benutzer hin. Wenn Sie die Anzahl der Menüelemente erhöhen, wird die rollbare Liste aus der rechten Abbildung angezeigt.*

Rezept 10.4 zeigt, wie eine einfache `UIActionSheet`-Instanz initialisiert und dargestellt werden kann. In der Initialisierungsmethode finden Sie etwas, das bei `UIAlertView` fehlt: eine unwiderrufliche Schaltfläche. Diese Schaltfläche löst eine Aktion aus, von der es kein Zurück gibt, beispielsweise das Löschen einer Datei (siehe *Abbildung 10.5*). Die kräftige rote Farbe warnt den Benutzer vor dieser Wahl. Setzen Sie diese Gestaltungsmöglichkeit sparsam ein.

Action-Sheet-Werte werden in der Reihenfolge der Schaltfläche zurückgegeben. Im Beispiel von *Abbildung 10.5* trägt die Schaltfläche **Delete** die Nummer 0 und **Cancel** die Nummer 3. Dies widerspricht dem Verhalten von Benachrichtigungsansichten, bei denen **Cancel** immer den Wert 0 aufweist.

10.6.1 Rollbare Menüs

Als grobe Faustregel gilt, dass Sie bis zu sieben Schaltflächen (einschließlich der Abbruch-Schaltfläche) im Hochformat und vier im Querformat unterbringen können. Wenn Sie über diese Anzahl hinausgehen, wird bei iPhone OS 3.0 und höher die Rolldarstellung aus *Abbildung 10.5* (rechts) ausgelöst. Beachten Sie, dass die Abbruch-Schaltfläche unter dieser Liste erscheint. Ihre Nummerierung entspricht jedoch der in der Darstellung des kleineren Menüs, da sie immer nach allen anderen Schaltflächen nummeriert wird. Wie *Abbildung 10.5* zeigt, ist diese Darstellung nicht sehr elegant und sollte daher nach Möglichkeit vermieden werden.

Hinweis

Sie können hier den Ansatz mit einer zweiten Ausführungsschleife wie in *Rezept 10.2* verwenden, um die Ergebnisse von Action-Sheets wie die von Benachrichtigungsansichten abzurufen.

```
- (void)actionSheet:(UIActionSheet *)actionSheet
clickedButtonAtIndex:(NSInteger)buttonIndex
{
   [actionSheet release];
   [self say:@"User Pressed Button %d\n", buttonIndex + 1];
}

-(void) action: (UIBarButtonItem *) item
{
   UIActionSheet *menu = [[UIActionSheet alloc] initWithTitle:
      @"File Management" delegate:self cancelButtonTitle:@"Cancel"
      destructiveButtonTitle:@"Delete File"
      otherButtonTitles:@"Rename File", @"Email File", nil];
   [menu showInView:self.view];
}
```

► *Rezept 10.5: Einfache Menüs anzeigen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.7 Text in Action-Sheets anzeigen

Action-Sheets bieten viele der Textdarstellungsmöglichkeiten, die es auch in Benachrichtigungsansichten gibt, weisen dabei aber eine weit größere Anzeigefläche auf. *Rezept 10.6* veranschaulicht, wie Sie eine Textnachricht mit einem `UIActionSheet`-Objekt anzeigen. Dieser Code baut auf *Rezept 10.4* auf, passt die Methode aber an eine Action-Sheet-Darstellung an.

```
- (void) show: (id)formatstring,...
{
   va_list arglist;
   va_start(arglist, formatstring);
   id statement = [[NSString alloc] initWithFormat:formatstring
      arguments:arglist];
   va_end(arglist);
```

```
    UIActionSheet *actionSheet = [[[UIActionSheet alloc]
        initWithTitle:statement delegate:nil cancelButtonTitle:nil
        destructiveButtonTitle:nil otherButtonTitles:@"OK", nil]
        autorelease];
    [actionSheet showInView:self.view];
    [statement release];
}
```

► *Rezept 10.6: Text in Action-Sheets darstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.8 »Bitte warten«: Fortschrittsanzeige für Benutzer

Das Warten ist eine wesentliche Erfahrung bei der Bedienung von Computern, und dies wird in naher Zukunft auch so bleiben. Es ist unsere Aufgabe als Entwickler, diese Tatsache unseren Benutzern mitzuteilen. Cocoa Touch bietet mehrere Klassen an, die Ihre Benutzer bitten, darauf zu warten, dass der Vorgang abgeschlossen wird. Diese Fortschrittsanzeigen gibt es in zwei Formen: als drehendes Rad, das für die Dauer seiner Darstellung besteht, und als Leiste, die sich von links nach rechts füllt, während der Vorgang fortschreitet. Folgende Klassen stellen diese Anzeigen bereit:

- `UIActivityIndicatorView` Eine Fortschrittsanzeige, die dem Benutzer ein sich drehendes Rad anzeigt und ihm somit ohne genauere Angabe über den Fortschritt bedeutet, zu warten. Die iPhone-Aktivitätsanzeige ist zwar klein, doch zieht die lebendige Animation den Blick des Benutzers auf sich und eignet sich daher gut für kurze Unterbrechungen in normalen Programmen. In *Rezept 10.1* haben Sie eine einfache Benachrichtigung mit einer solchen Aktivitätsanzeige gesehen.
- `UIProgressView` Diese Ansicht stellt einen Fortschrittsbalken dar. Er gibt eine genaue Rückmeldung darüber, wie viel Arbeit bereits erledigt ist und wie viel noch aussteht. Dabei nimmt er nur relativ wenig Platz in Anspruch. Er stellt sich als schmales Recheck dar, das von links nach rechts aufgefüllt wird, während der Vorgang fortschreitet. Dieses klassische Element einer Benutzeroberfläche ist am besten für längere Verzögerungen geeignet, bei denen der Benutzer wissen möchte, wie viel Prozent der Arbeit bereit erledigt sind.

Hüten Sie sich vor Blockierungen. Beide Klassen müssen im Hauptthread verwendet werden, wie es bei GUI-Elementen allgemein der Fall ist. Rechenintensiver Code kann verhindern, dass Ansichten in Echtzeit angezeigt werden. Wenn Sie asynchrone Rückmeldungen einblenden müssen, verwen-

den Sie mehrere Threads. Die in Rezept 7.11 von Kapitel 7, *Mit Bildern arbeiten*, vorgestellte Kantenerkennung bildet ein gutes Beispiel. Dabei wird eine `UIActivityIndicatorView` im Hauptthread verwendet, während die Berechnung in einem zweiten Thread stattfindet.

10.8.1 UIActivityIndicatorView verwenden

Instanzen von `UIActivityIndicatorView` bilden kleine Ansichten, die das standardmäßige rotierende Fortschrittsrädchen anzeigen. Bei der Arbeit mit diesen Ansichten sollten Sie vor allem ein Wort berücksichtigen: *klein*. Alle Aktivitätsanzeigen sind winzig und sehen merkwürdig aus, wenn sie über ihren Normalzustand hinaus vergrößert werden.

Das iPhone bietet verschiedene Spielarten der Klasse `UIActivityIndicatorView`. `UIActivityIndicatorViewStyleWhite` und `UIActivityIndicatorViewStyleGray` sind jeweils 20 × 20 Pixel groß. Die weiße Version wirkt am besten vor schwarzem Hintergrund, die graue vor einem weißen. Die Gestaltung ist schlank und scharf.

Überlegen Sie sorgfältig, ob Sie die weiße oder die graue Ansicht verwenden. Eine komplett weiße Darstellung ist auf einem weißen Hintergrund nicht sichtbar. Leider gibt es den größten, deutlichsten Indikator mit einer Größe von 37 × 10 nur als `UIActivityIndicatorViewStyleWhiteLarge` für die Verwendung auf dunklen Hintergründen.

```
UIActivityIndicatorView *aiv = [[UIActivityIndicatorView alloc]
    initWithActivityIndicatorStyle:
    UIActivityIndicatorViewStyleWhiteLarge];
```

Sie müssen die Anzeige nicht mittig auf dem Bildschirm platzieren, sondern können sie nach Belieben ausrichten. Der Indikator fügt sich in jegliche Hintergrundansicht ein, da er selbst einen transparenten Hintergrund aufweist. Die vorherrschende Farbe der Hintergrundansicht hilft Ihnen bei der Auswahl der Fortschrittsanzeige.

Für den allgemeinen Gebrauch fügen Sie die Aktivitätsanzeige als Unteransicht zu dem Fenster, der Ansicht, der Symbol- oder der Navigationsleiste hinzu, die Sie überlagern möchten (siehe *Rezept 10.1*). Weisen Sie die Anzeige zu, und initialisieren Sie sie mit einem Rahmen, bevorzugt in der jeweiligen Elternansicht zentriert.

Starten Sie die Anzeige, indem Sie `startAnimating` senden. Cocoa Touch übernimmt den Rest und versteckt die Anzeige, wenn sie nicht benötigt wird.

10.9 Rezept: Eine Ansicht mit UIProgressView erstellen

Fortschrittsanzeigen ermöglichen es den Benutzern, den Aufgabenfortschritt zu verfolgen, anstatt einfach die Meldung »Bitte warten« auszugeben. Sie zeigen Balken an, die sich von links nach rechts füllen. Diese Balken verdeutlichen den prozentualen Anteil der Aufgabe, der bereits abgeschlossen ist. Fortschrittsbalken sind am besten für längere Wartezeiten geeignet, da sie dem Benutzer eine Rückmeldung geben und ihm ein Gefühl von Kontrolle vermitteln.

Um eine Fortschrittsanzeige zu erstellen, weisen Sie sie zu und richten ihren Frame ein. Zu ihrer Verwendung führen Sie `setProgress:` aus. Dazu wird ein Argument benötigt, nämlich eine Fließkommazahl zwischen 0,0 (kein Fortschritt) und 1,0 (abgeschlossen). Fortschrittsbalken gibt es in zwei Varianten: weiß und hellgrau. Mit der Methode `setStyle:` wählen Sie die passende aus.

Anders als bei anderen Fortschrittsanzeigen liegt es ganz bei Ihnen, den Fortschrittbalken anzuzeigen oder auszublenden. Es gibt keine `setVisible:`-Methode. Fortschrittsbalken zu Action-Sheets hinzuzufügen, vereinfacht sowohl das Ein- als auch das Ausblenden der Anzeige. Ein anderer Vorteil besteht darin, dass der Rest des Bildschirms bei der Anzeige von Benachrichtigungs-Sheets abgedunkelt wird, was eine modale Darstellung des Aufgabenfortschritts erzwingt. Benutzer können so lange nicht weiter mit der Benutzeroberfläche interagieren, bis Sie die Benachrichtigung wieder ausblenden. *Rezept 10.7* zeigt ein Beispiel, in dem `UIActionSheet/UIProgressView` verwendet wird, um die Anzeige aus *Abbildung 10.6* zu erstellen. Mehrere Zeilenwechsel im Titel des Action-Sheets sorgen dafür, dass der Fortschrittsbalken nicht den Titeltext verdeckt.

▸ *Abbildung 10.6: Mit* `UIProgressView`*-Instanzen zeigen Sie den Fortschritt bei einer längeren Verzögerung an. Wenn Sie sie zu einem* `UIActionSheet` *hinzufügen, vereinfachen Sie ihre Darstellung und das Ausblenden.*

```
@interface TestBedViewController : UIViewController <UIActionSheetDelegate>
{
    float amountDone;
    UIProgressView *progressView;
    UIActionSheet *actionSheet;
}
@property (retain) UIActionSheet *actionSheet;
@end

@implementation TestBedViewController
```

```
@synthesize actionSheet;

// Dieser Callback simuliert einen Fortschritt via setProgress:
- (void) incrementBar: (id) timer
{
   amountDone += 1.0f;
   [progressView setProgress: (amountDone / 20.0)];
   if (amountDone > 20.0)
   {
      [self.actionSheet dismissWithClickedButtonIndex:0
         animated:YES];
      self.actionSheet = nil;
      [timer invalidate];
   }
}

// Lädt den Fortschrittsbalken in ein Action-Sheet
-(void) action: (UIBarButtonItem *) item
{
   amountDone = 0.0f;
   self.actionSheet = [[[UIActionSheet alloc]
      initWithTitle:@"Downloading data. Please Wait\n\n\n"
      delegate:nil cancelButtonTitle:nil destructiveButtonTitle: nil
      otherButtonTitles: nil] autorelease];
   progressView = [[UIProgressView alloc]
      initWithFrame:CGRectMake(0.0f, 40.0f, 220.0f, 90.0f)];
   [progressView setProgressViewStyle: UIProgressViewStyleDefault];
   [actionSheet addSubview:progressView];
   [progressView release];

   // Erledigt die Beispielaktualisierungen
   [progressView setProgress:(amountDone = 0.0f)];
   [NSTimer scheduledTimerWithTimeInterval: 0.5 target: self
      selector:@selector(incrementBar) userInfo: nil repeats: YES];
   [actionSheet showInView:self.view];
   progressView.center = CGPointMake(actionSheet.center.x,
      progressView.center.y);
}
@end
```

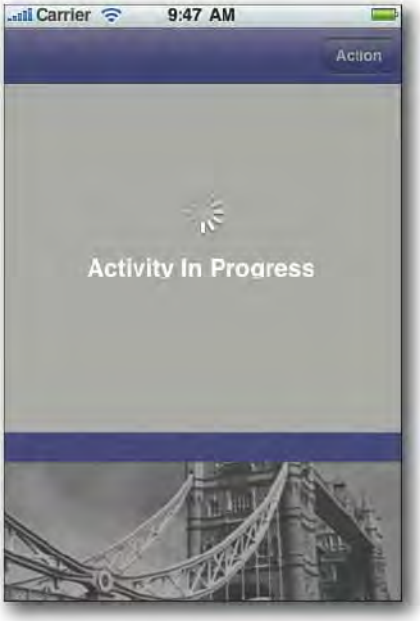
▸ *Rezept 10.7: Fortschrittsanzeige in einem Action-Sheet*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.10 Rezept: Benutzerdefinierte Überlagerungen erstellen

Mit `UIAlertView` und `UIActionSheet` können Sie zwar hervorragende modale Fortschrittsanzeigen erstellen, doch können Sie auch von Grund auf eine eigene anlegen. *Rezept 10.8* zeigt eine einfache, getönte `UIView`-Überlagerung mit einer `UIActivityIndicatorView`-Ansicht, um die modale Fortschrittsanzeige aus *Abbildung 10.7* darzustellen.



▸ *Abbildung 10.7: Mit benutzerdefinierten Ansichten sind praktische modale Benachrichtigungen ohne Verwendung der vorgefertigten Apple-Klassen möglich.*

Diese Ansicht wurde in Interface Builder gestaltet und mit der benutzerdefinierten Klasseneigenschaft `overlay` verbunden. Sie nimmt den gesamten Bildschirm ein, sodass keine simulierten Bildschirmelemente aktiviert werden müssen. Bei der bildschirmfüllenden Anzeige überdeckt die Überlagerung auch die Navigationsleiste, weil die Ansicht zum Anwendungsfenster hinzugefügt werden muss und nicht, wie man denken könnte, zur Ansicht des Hauptansichtscontrollers. Letztere Ansicht nimmt nur den Platz unter der Navigationsleiste ein, sodass der Zugang zu den Schaltflächen und anderen Steuerelementen in der Leiste frei bleibt.

Um Benutzeraktionen zu verhindern, setzt die Überlagerung die Eigenschaft `userInteractionEnabled` auf `YES`, wodurch alle Berührungsereignisse abgefangen werden, sodass Sie die normale

Oberfläche unterhalb der Benachrichtigung nicht erreichen. Dadurch kommt eine modale Darstellung zustande, in der eine Interaktion erst möglich ist, wenn die Benachrichtigung wieder ausgeblendet wurde.

In diesem Beispiel wird nur eine Hochformatansicht verwendet. Da die Ansicht nicht zu einem Ansichtscontroller gehört, kann und wird sie bei einer Drehung des iPhones nicht aktualisiert. Wenn Sie zwischen Hoch- und Querformat unterscheiden müssen, können Sie den Orientierungswert abrufen, bevor Sie die Überlagerung anzeigen. Diese Vorgehensweise zeigt *Rezept 10.10*.

```
- (void) finish
{
   [(UIActivityIndicatorView *)[self.overlay viewWithTag:202]
      stopAnimating];
   [self.overlay removeFromSuperview];
}

- (void) action: (id) sender
{
   // Fügt die Unteransicht hinzu
   [self.view.window addSubview:self.overlay];

   // Startet die Aktivitätsanzeige
   [(UIActivityIndicatorView *)[self.overlay viewWithTag:202]
      startAnimating];

   // Ruft die Beendigungsmethode mit Verzögerung auf
   [self performSelector:@selector(finish) withObject:nil
      afterDelay:3.0f];
}
```

► *Rezept 10.8: Eine benutzerdefinierte Überlagerung mit einer Benachrichtigung anzeigen und entfernen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.11 Antippbare Überlagerungen

Mit benutzerdefinierten Überlagerungen können Sie sowohl Informationen anzeigen als auch modale Sequenzen herstellen. *Rezept 10.9* erstellt die Klasse `TappableOverlay`. Wenn ein Benutzer auf eine solche Ansicht tippt, wird sie vom Bildschirm entfernt. Durch dieses Verhalten ist sie für die Anzeige von Informationen geeignet, für die normalerweise die Klasse `UIAlertView` verwendet wird.

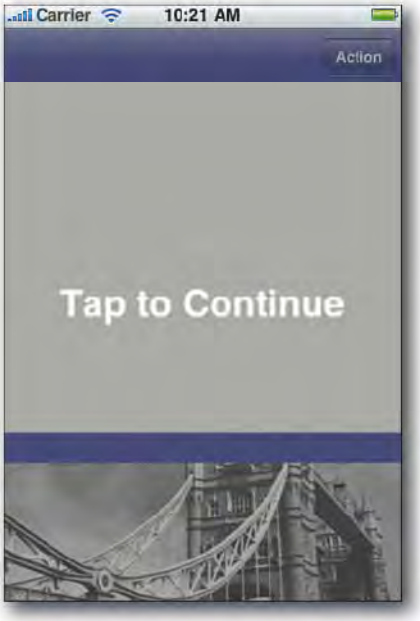
Um diese Klasse zu nutzen, erstellen Sie in Interface Builder eine Ansichtsinstanz und fügen ihr so viele Unteransichten und Gestaltungselemente hinzu, wie Sie brauchen. Importieren Sie über **File ▸ Read Class File** die Headerdatei `TappableOverlay.h`, ändern Sie im Identitäts-Informationsfeld ⌘ + 4 die Ansichtsklasse von `UIView` auf `TappableOverlay`, und speichern Sie das Projekt.

Um die Ansicht darzustellen, fügen Sie sie wie in *Rezept 10.8* zum Fenster hinzu:

```
- (void) action: (id) sender
{
   // Add the overlay
[   self.view.window addSubview:self.overlay];
}
```

Es ist keine weitere Programmierung erforderlich. Die Ansicht wartet, bis ein Benutzer auf sie tippt, und wenn dies geschieht, entfernt sie sich selbst vom Bildschirm.

Abbildung 10.8 zeigt ein einfaches Beispiel für diese Art von Überlagerung. Darin ist einfach nur der Hinweis »Tap to Continue« (Antippen, um fortzufahren) zu lesen. Es ist nicht schwer, sich auszumalen, wie sich dieses Prinzip erweitern lässt, um alle möglichen Arten von passenden Informationen anzuzeigen und so eine benutzerdefinierte Alternative zur Klasse `UIAlertView` zu erhalten. Wie *Rezept 10.8* passt sich auch dieses Beispiel nicht an eine Änderung der Geräteorientierung an.



▸ *Abbildung 10.8: Diese einfache Überlagerung entfernt sich selbst vom Bildschirm, wenn der Benutzer auf sie tippt.*

```
@interface TappableOverlay : UIView
@end
@implementation TappableOverlay
- (void)touchesEnded:(NSSet *)touches withEvent:(UIEvent *)event
{
```

```
    // Remove this view when it is touched
    [self removeFromSuperview];
}
@end
```

► *Rezept 10.9: Eine benutzerdefinierte, entfernbare Benachrichtigungsansicht erstellen, die auf Benutzerberührungen reagiert*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.12 Von oben eingeblendete Benachrichtigungen mit Orientierungswechsel

Die in *Rezept 10.8* eingeführten Möglichkeiten für modale Benachrichtigungen können Sie erweitern, um eine nicht interaktive Überlagerung zu erstellen, die als Hintergrund für eine von oben ins Blickfeld geschobene Benachrichtigung fungiert. In *Rezept 10.10* enthält eine solche Überlagerung eine Ansicht mit einer eingebetteten Schaltfläche (siehe *Abbildung 10.9*). Die Ansicht wird über ein Paar einfacher `UIView`-Animationsblöcke dargestellt und entfernt, wobei die Schaltfläche **Okay** die Methode `dismiss:` auslöst, die die Ansicht vom Bildschirm schiebt.

► *Abbildung 10.9: Diese modal dargestellte Nachricht wird von oben in die Ansicht geschoben und wird entfernt, wenn der Benutzer auf* **Okay** *tippt.*

Die Nachrichtenansicht wurde in Interface Builder als standardmäßige `UIView` erstellt und in der Methode `viewDidLoad` als Unteransicht zur Überlagerung hinzugefügt. Anstatt wie in *Rezept 10.8* die Überlagerung zum Hauptfenster hinzuzufügen und von dort zu entfernen, wird hier die Eigenschaft `alpha` der Überlagerung zum Ein- und Ausblenden verwendet.

Anders als in den beiden vorhergehenden Rezepten wird hier auf die Bildschirmausrichtung geachtet. Größe und Darstellung werden an die aktuelle Orientierung des iPhones angepasst. Dies geschieht auf zwei Weisen. Erstens wird eine affine Transformation auf die Überlagerung angewendet, wenn sich die Orientierung ändert. Zweitens werden die Rahmen der Überlagerung und der Nachrichtenansicht vor der Darstellung angepasst, sodass ihre Form der des aktuellen Fensters entspricht.

Bei diesem Beispiel wird die Überlagerung von oben ins Blickfeld geschoben, aber die Berechnungen lassen sich ganz einfach anpassen, um die Einblendung von den Seiten (verwenden Sie den x-Wert des Ursprungs statt des y-Werts) oder von unten (addieren Sie 320 oder 480 zur Höhe der Ansicht) erfolgen zu lassen. Alternativ können Sie die Ansicht auch zentrieren und ihre Größe animieren, sodass sie nicht ins Blickfeld gleitet, sondern sich aufbläht.

```
- (void) dismiss: (id) sender
{
   // Schiebt die Nachrichtenansicht mit einer Animation weg
   [UIView beginAnimations:nil context:NULL];
   [UIView setAnimationDuration:0.3f];
   [UIView setAnimationCurve:UIViewAnimationCurveLinear];
   mvframe.origin = CGPointMake(0.0f, -300.0f);
   self.messageView.frame = mvframe;
   [UIView commitAnimations];

   // Verbirgt die Überlagerung
   [self.overlay performSelector:@selector(setAlpha) withObject:nil
      afterDelay:0.3f];
}

- (void) action: (id) sender
{
   // Passt die Größe der Überlagerung an die Bildschirmorientierung an
   self.overlay.frame = self.view.window.frame;
   mvframe.size.width = UIDeviceOrientationIsPortrait([[UIDevice
      currentDevice] orientation]) ? 320.0f : 480.0f;
   mvframe.origin = CGPointMake(0.0f, -mvframe.size.height);
   self.messageView.frame = mvframe;

   // Zeigt die Überlagerung an
   if (!self.overlay.superview)
      [self.view.window addSubview:self.overlay];
   self.overlay.alpha = 1.0f;
```

```
    // Schiebt die Nachrichtenansicht mit einer Animation ins Bild
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:0.3f];
    [UIView setAnimationCurve:UIViewAnimationCurveLinear];
    mvframe.origin = CGPointMake(0.0f, 20.0f);
    self.messageView.frame = mvframe;
    [UIView commitAnimations];
}

- (void) viewDidLoad
{
    self.navigationItem.rightBarButtonItem = BARBUTTON(@"Action",
        @selector(action));

    // Initialisiert die Überlagerung und die Nachrichtenansicht
    self.overlay.alpha = 0.0f;
    [self.overlay addSubview:self.messageView];
    mvframe = messageView.frame;
    mvframe.origin = CGPointMake(0.0f, -300.0f);
    self.messageView.frame = mvframe;
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
        (UIInterfaceOrientation)interfaceOrientation
{
    // Wendet entsprechend der Orientierung Transformationen auf die
    // Überlagerung an
    if (interfaceOrientation ==
        UIInterfaceOrientationPortraitUpsideDown)
        self.overlay.transform = CGAffineTransformMakeRotation(M_PI);
    else if (interfaceOrientation ==
        UIInterfaceOrientationLandscapeLeft)
        self.overlay.transform = CGAffineTransformMakeRotation(-M_PI /
            2.0f);
    else if (interfaceOrientation ==
        UIInterfaceOrientationLandscapeRight)
        self.overlay.transform = CGAffineTransformMakeRotation(M_PI /
            2.0f);
    else
        self.overlay.transform = CGAffineTransformIdentity;
    return YES;
}
```

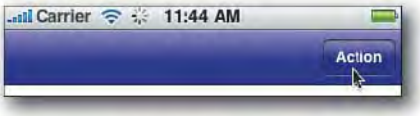
▶ *Rezept 10.10: Eine von oben eingeschobene Überlagerung mit Orientierungswechsel erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.13 Rezept: Die Netzwerkanzeige verwenden

Wenn Ihre Anwendung hinter den Kulissen auf das Internet zugreift, ist es höflich, die Benutzer dies wissen zu lassen. Statt einer bildschirmfüllenden Benachrichtigung gibt es dafür in Cocoa Touch eine einfache Anwendungseigenschaft für eine sich drehende Netzwerkanzeige in der Statusleiste. *Abbildung 10.10* zeigt diese Anzeige in Aktion. Sie finden sie zwischen der WiFi-Anzeige und der Uhrzeit.



▸ *Abbildung 10.10: Die Netzwerkanzeige wird von einer `UIApplication`-Eigenschaft gesteuert.*

Rezept 10.11 zeigt, wie Sie auf diese Eigenschaft zugreifen. Dabei gibt es nicht mehr zu tun, als die Anzeige ein- oder auszuschalten. In der Praxis führen Sie Netzwerkaktivitäten wahrscheinlich in einem zweiten Thread aus. Daher müssen Sie diese Eigenschaft im Hauptthread ändern, damit die Benutzeroberfläche korrekt aktualisiert werden kann.

```
- (void) action: (id) sender
{
    // Schaltet die Netzwerkanzeige ein und aus
    UIApplication *app = [UIApplication sharedApplication];
    app.networkActivityIndicatorVisible =
        !app.networkActivityIndicatorVisible;
}
```

▸ *Rezept 10.11: Die Netzwerkanzeige in der Statusleiste einblenden*

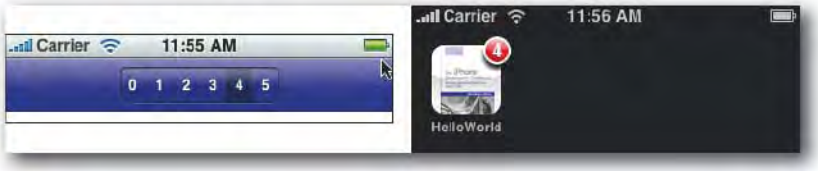
Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.14 Anwendungen mit Badges versehen

Wenn Sie bereits das iPhone oder den iPod touch verwendet haben, kennen Sie wahrscheinlich auch die kleinen roten Badges, die über Programmen auf dem Startbildschirm erscheinen. Sie können zum Beispiel die Anzahl verpasster Telefonanrufe oder E-Mails anzeigen, die seit dem letzten Öffnen dieser Programme hereingekommen sind.

Um aus dem Programm selbst heraus ein solches Badge zu setzen, legen Sie für die Eigenschaft `applicationIconBadgeNumber` einen Integerwert fest. Um das Badge auszublenden, geben Sie als Wert null an. *Rezept 10.12* zeigt, wie Sie Anwendungs-Badges lesen und festlegen können. In diesem Rezept wird der Wert eines unterteilten Steuerelements mit der zuletzt verwendeten Badge-Nummer verglichen. Wenn Benutzer die Einstellung des unterteilten Steuerelements ändern, aktualisiert der Code dementsprechend das Badge. *Abbildung 10.11* zeigt diesen Vorgang. Dort sehen Sie das Steuerelement in der Anwendung und die dadurch hervorgerufene Badge-Nummer.



▶ *Abbildung 10.11: Das unterteilte Steuerelement aus Rezept 10.12 aktualisiert die Badge-Nummer der Anwendung.*

```
@implementation TestBedViewController
- (void) updateBadge: (UISegmentedControl *) seg
{
   // Setzt die Badge-Nummer auf den Index des ausgewählten Segments
   [UIApplication sharedApplication].applicationIconBadgeNumber =
      seg.selectedSegmentIndex;
}

- (void) viewDidLoad
{
   // Erstellt das segmentierte Steuerelement für die Badge-Nummer
   UISegmentedControl *seg = [[UISegmentedControl alloc]
      initWithItems:[@"0 1 2 3 4 5" componentsSeparatedByString:
      @" "]];
   seg.segmentedControlStyle = UISegmentedControlStyleBar;
   seg.selectedSegmentIndex = MIN([UIApplication
      sharedApplication].applicationIconBadgeNumber, 5);
   [seg addTarget:self action:@selector(updateBadge)
      forControlEvents:UIControlEventValueChanged];
   self.navigationItem.titleView = seg;
   [seg release];
}
@end
```

▶ *Rezept 10.12: Anwendungs-Badges lesen und aktualisieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.15 Rezept: Einfache Audiobenachrichtigungen

Audiobenachrichtigungen »reden« direkt mit dem Benutzer. Sie geben eine sofortige Rückmeldung – vorausgesetzt, der Benutzer ist nicht hörgeschädigt. Glücklicherweise hat Apple eine grundlegende Tonwiedergabe über Systemaudiodienste in das Cocoa Touch SDK eingebaut, was ähnlich wie auf einem Macintosh funktioniert.

Die Alternative sind Audio Queue-Aufrufe und `AVAudioPlayer`. Die Audio Queue-Wiedergabe ist aufwendig zu programmieren und sehr viel komplizierter, als es für einfache Benachrichtigungstöne erforderlich wäre. Im Gegensatz dazu können Sie Systemtöne mit wenigen Codezeilen laden und abspielen. Auch `AVAudioPlayer` hat seine Nachteile, denn hier kann es zu Konflikten mit der iPod-Audiowiedergabe kommen. Im Gegensatz dazu können die Systemaudiodienste einen Ton abspielen, ohne die gerade gespielte Musik zu unterbrechen, was jedoch zugegebenermaßen nicht unbedingt das ist, was Sie erreichen wollen, da dabei die Gefahr besteht, dass die Benachrichtigung in der Musik untergeht.

Benachrichtigungstöne funktionieren laut Apple am besten, wenn sie kurz gehalten sind, vorzugsweise 30 Sekunden oder weniger. Das Systemaudio spielt nur PCM- und IMA-Audio, was das Format möglicher Töne auf AIFF, WAV und CAF beschränkt.

10.15.1 Systemtöne

Um einen Systemton zu erstellen, rufen Sie `AudioServicesCreateSystemSoundID` mit einem Datei-URL auf, der auf die Klangdatei zeigt. Dieser Aufruf gibt ein initialisiertes Systemsoundobjekt zurück, das Sie dann nach Belieben abspielen können. Rufen Sie einfach `AudioServicesPlaySystemSound` mit dem Soundobjekt auf. Mit diesem einen Aufruf ist schon die ganze Arbeit erledigt.

```
AudioServicesPlaySystemSound(mySound);
```

In ihrer Standardimplementierung werden Systemtöne durch die Voreinstellung **Töne** unter **Einstellungen** gesteuert. Sind Töne deaktiviert, wird der Klang nicht abgespielt. Um diese Voreinstellung zu überschreiben und den Ton immer wiederzugeben, können Sie wie folgt ein entsprechendes Eigenschaftsflag setzen:

```
// Kennzeichnung als Nicht-UI-Ton
AudioServicesCreateSystemSoundID(baseURL, &mysound);
AudioServicesPropertyID flag = 0; // 0 bedeutet, dass der Ton stets
                                  // wiedergegeben wird
```

```
AudioServicesSetProperty(kAudioServicesPropertyIsUISound,
   sizeof(SystemSoundID), &mysound,
   sizeof(AudioServicesPropertyID), &flag);
```

Bei der Wiedergabe von iPod-Audio wird der Systemton im Allgemeinen mit derselben Lautstärke abgespielt, sodass die Benutzer die Benachrichtigung verpassen könnten. Ziehen Sie daher den Einsatz von Vibration neben oder anstelle von Musik in Betracht. Den aktuellen Wiedergabestatus können Sie wie folgt prüfen: Schließen Sie `<MediaPlayer/MediaPlayer.h>` ein, und stellen Sie eine Verbindung zum MediaPlayer-Framework her.

```
if ([MPMusicPlayerController iPodMusicPlayer].playbackState ==
   MPMusicPlaybackStatePlaying)
```

Um Ihrem Programm mitzuteilen, dass die Wiedergabe eines Tons abgeschlossen ist, können Sie einen optionalen Callback hinzufügen, indem Sie `AudioServicesAddSystemSoundCompletion` aufrufen. Sofern Sie nicht hintereinandergereihte kurze Töne verwenden, können Sie auf diesen Schritt jedoch gewöhnlich verzichten.

Räumen Sie auf, indem Sie `AudioServicesDisposeSystemSoundID` mit dem betroffenen Ton aufrufen. Dadurch werden das Soundobjekt und alle zugehörigen Ressourcen freigegeben.

Hinweis

Um diese Systemtondienste zu verwenden, müssen Sie `AudioToolbox/AudioServices.h` in den Code aufnehmen und eine Verbindung zum Audio Toolbox-Framework herstellen.

10.15.2 Vibration

Wie Audiobenachrichtigungen erlangen Vibrationen die sofortige Aufmerksamkeit des Benutzers. Darüber hinaus funktionieren Vibrationen bei nahezu allen Benutzern, auch bei Hör- oder Sehgeschädigten. Mit denselben Systemaudiodiensten können Sie sowohl Vibrationen auslösen als auch Sounds abspielen. Alles, was Sie benötigen, ist folgender einzeiliger Aufruf, wie er auch in *Rezept 10.13* verwendet wurde:

```
AudioServicesPlaySystemSound (kSystemSoundID_Vibrate);
```

Sie können die Vibrationsparameter nicht ändern. Jeder Aufruf ruft ein ein- oder zweisekündiges Brummen hervor. Bei Plattformen ohne Vibrationsunterstützung (wie dem iPod Touch) geschieht nichts, es wird aber auch kein Fehler ausgelöst.

10.15.3 Alarm

Die Audiodienste bieten auch als sogenannten Alarm eine Kombination aus Vibration und Ton an, die wie folgt aufgerufen wird:

```
AudioServicesPlayAlertSound(mySound);
```

Dieser Aufruf, der auch in *Rezept 10.13* veranschaulicht wird, spielt den angeforderten Klang ab und lässt das iPhone vibrieren oder spielt einen zweiten Alarm ab. Wenn der Benutzer auf dem iPhone **Einstellungen ▸ Töne ▸ Klingeln ▸ Vibrieren** eingeschaltet hat, vibriert das Gerät. iPod touch-Geräte der zweiten Generation und moderner spielen den Ton ohne Vibration (da es diese Funktion dort nicht gibt) über den eingebauten Lautsprecher ab. Auf einem iPod touch der ersten Generation wird auf dem Gerätelautsprecher anstelle des gewünschten Tons eine Benachrichtigungsmelodie abgespielt, während der angeforderte Ton über die Kopfhörer ausgegeben wird.

10.15.4 Verzögerungen

Wenn Sie einen Systemsound auf dem iPhone zum ersten Mal wiedergeben, kann es zu einer Verzögerung kommen. Um dies zu vermeiden, können Sie den Ton bei der Initialisierung der Anwendung stumm abspielen.

Hinweis

Beim Test auf iPhone-Geräten müssen Sie darauf achten, dass Sie das Klingeln nicht mit dem Schalter auf der linken Seite ausgestellt haben. Darauf sind schon viele iPhone-Entwickler hereingefallen. Wenn die Benachrichtigungstöne stets wiedergegeben werden müssen, sollten Sie die Klasse `AVAudioPlayer` verwenden, die in Kapitel 15, *Audio, Video und Media Kit*, besprochen wird.

```
@implementation TestBedViewController
- (void) playSound
{
   if ([MPMusicPlayerController iPodMusicPlayer].playbackState ==
      MPMusicPlaybackStatePlaying)
      AudioServicesPlayAlertSound(mysound);
   else
      AudioServicesPlaySystemSound(mysound);
}

- (void) vibrate
{
   AudioServicesPlaySystemSound (kSystemSoundID_Vibrate);
}

- (void) viewDidLoad
{
   // Erstellt den Klang
   NSString *sndpath = [[NSBundle mainBundle]
      pathForResource:@"basicsound" ofType:@"wav"];
   CFURLRef baseURL = (CFURLRef)[NSURL fileURLWithPath:sndpath];
```

```
    // Bezeichnet den Klang als Nicht-UI-Sound
    AudioServicesCreateSystemSoundID(baseURL, &mysound);
    AudioServicesPropertyID flag = 0; // 0 bedeutet, dass der Ton stets
                                      // wiedergegeben wird
    AudioServicesSetProperty(kAudioServicesPropertyIsUISound,
       sizeof(SystemSoundID), &mysound,
       sizeof(AudioServicesPropertyID), &flag);

    self.navigationItem.rightBarButtonItem = BARBUTTON(@"Sound",
       @selector(playSound));
    self.navigationItem.leftBarButtonItem = BARBUTTON(@"Vibrate",
       @selector(vibrate));
}

-(void) dealloc
{
    // Aufräumen
    if (mysound) AudioServicesDisposeSystemSoundID(mysound);
    [super dealloc];
}
@end
```

▸ *Rezept 10.13: Töne, Alarme und Vibrationen mithilfe der Audiodienste abspielen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 10 und öffnen das Projekt zu diesem Rezept.

10.16 Ein letzter Punkt: Lautstärkebenachrichtigung anzeigen

Auf dem iPhone gibt es eine eingebaute Benachrichtigung, mit der die Benutzer die Systemlautstärke anpassen können. Sie besteht aus einem Schieberegler und einer **Done**-Schaltfläche (siehe *Abbildung 10.12*). Diese Benachrichtigung können Sie mithilfe der folgenden Media Player-Funktion aufrufen:

```
- (void) action
{
    // Zeigt die Media Player-Benachrichtigung zur Lautstärkeregelung
    MPVolumeSettingsAlertShow();
}
```

Die Sichtbarkeit dieser Benachrichtigung prüfen Sie mit `MPVolumeSettingsAlertIsVisible()`. Dadurch wird ein boolescher Wert zurückgegeben, der angibt, ob sich die Benachrichtigung auf dem Bildschirm befindet. Mit `MPVolumeSttingsAlertHide()` verbergen Sie die Benachrichtigung unabhängig davon, ob der Benutzer auf **Done** tippt oder nicht. Damit diese Funktionen verfügbar sind, müssen Sie eine Verbindung mit dem MediaPlayer-Framework herstellen und die entsprechenden Headerdateien importieren.

▸ *Abbildung 10.12: Die Benachrichtigung mit der Lautstärkeregelung aus den Media Player-Klassen*

10.17 Zusammenfassung

Dieses Kapitel hat Möglichkeiten vorgestellt, um direkt mit den Benutzern zu kommunizieren. Sie haben gesehen, wie Sie Benachrichtigungen erstellen – optische, akustische und fühlbare –, die die Aufmerksamkeit des Benutzers auf sich ziehen und eine sofortige Rückmeldung verlangen können. Nutzen Sie diese Beispiele, um die Interaktivität Ihrer Programme zu verbessern und einige einzigartige Funktionen auszulösen, die nur auf dem iPhone möglich sind. Die folgenden Gedanken sollten Sie aus der Lektüre dieses Kapitels mitnehmen:

- Immer wenn eine Aufgabe eine merkbare Zeitspanne benötigt, sollten Sie an die Benutzer denken und ihnen eine Art von Fortschrittsrückmeldung anzeigen. Das iPhone bietet viele Möglichkeiten dazu, von Schwebefenstern zu Fortschrittsbalken und darüber hinaus. Um eine Blockierung zu vermeiden, müssen Sie die Elemente der Aufgabe, die nicht zur Anzeige der Benutzeroberfläche dienen, unter Umständen in einen anderen Thread auslagern.

10.17 Zusammenfassung

- Benachrichtigungen können die Aufmerksamkeit des Benutzers auf sich ziehen. Sie dienen dazu, dem Benutzer Antworten zu entlocken, während Informationen weitergegeben werden. Wie Sie in diesem Kapitel gesehen haben, lassen sie sich nahezu unbegrenzt anpassen. Es ist möglich, die gesamte Anwendung um ein einfaches `UIAlertSheet` herum zu erstellen.
- Haben Sie keine Angst vor Ausführungsschleifen. Eine modale Antwort von einer Benachrichtigung oder einem Action-Sheet erlaubt es Ihnen, die Auswahl der Benutzer unmittelbar zu erfassen, ohne auf asynchrone Callbacks zurückgreifen zu müssen.
- Wenn sich die blauen, systemeigenen Anzeigen nicht für das Design Ihrer Anwendung eignen, verzichten Sie darauf. Sie können mit Instanzen von `UIView` und Animationen ganz einfach Ihre eigenen Benachrichtigungen und Menüs erstellen.
- Rückmeldungen durch Töne wie Klicken oder Piepsen können Ihre Programme verbessern und die Interaktion interessanter gestalten. Wenn Sie Systemsoundaufrufe verwenden, geraten die Töne nicht mit den iPod-Funktionen in Konflikt und stören nicht die laufende Musikwiedergabe. Seien Sie aber auch nicht zu aufdringlich, sondern setzen Sie Benachrichtigungstöne sparsam und sinnvoll ein, um die Benutzer nicht zu stören.

11 Tabellenansichten erstellen und verwalten

Tabellen ermöglichen eine Interaktion auf der Grundlage von scrollbaren Listen, die besonders gut auf einem kleinen, beengten Gerät funktionieren. Viele, wenn nicht sogar die meisten Programme, die mit dem iPhone und dem iPod touch ausgeliefert werden, basieren auf Tabellen, darunter *Kontakte, Einstellungen, iPod, YouTube, Aktien* und *Wetter*. Aufgrund der eingeschränkten Bildschirmgröße des iPhones sind Tabellen mit ihrer Scrollmöglichkeit und der Auswahl einzelner Elemente ideal, um Informationen und Inhalte in einfach zu verändernder Form bereitzustellen. In diesem Kapitel erfahren Sie, wie Tabellen für das iPhone funktionieren und welche Arten von Tabellen für Sie als Entwickler verfügbar sind. Außerdem lernen Sie, wie Sie Tabellenfunktionen in Ihren eigenen Programmen verwenden können.

11.1 Einführung in UITableView und UITableViewController

Die Standardtabelle des iPhones besteht aus einer einfachen verschiebbaren Liste einzelner Zellen, die einen interaktiven Datenindex bereitstellen. Benutzer können nach oben oder unten scrollen oder blättern, bis sie das Element finden, das sie verwenden möchten. Anschließend können sie unabhängig von anderen Zeilen mit diesem Element arbeiten. Beim iPhone sind Tabellen allgegenwärtig, denn nahezu jedes Standardsoftwarepaket verwendet sie, und außerdem bilden sie den Kern vieler Programme von Drittherstellern. In diesem Abschnitt erfahren Sie, wie Tabellen funktionieren und welche Elemente Sie kombinieren müssen, um Ihre eigene Tabelle zu erstellen.

Das iPhone-SDK unterstützt mehrere Arten von Tabellen, von denen viele als Abkömmlinge der Klasse `UITableView` implementiert sind. Neben der standardmäßigen scrollbaren Zellenliste, die die allgemeinste Tabellenimplementierung darstellt, können Sie mehrere spezialisierte Tabellen erstellen. Dazu gehört die Art von Tabellen, die Sie im Programm *Einstellungen* sehen können, Tabellen mit einem grauen Hintergrund und abgerundeten Ecken, in Abschnitte eingeteilte Tabellen mit einem Index wie die im Programm *Kontakte* und verwandte Klassen mit Drehtabellen (Picker) wie denen, die zur Eingabe eines Datum in der Kalenderapplikation verwendet werden. Alle Tabellen funktionieren unabhängig von ihrem Typ grundsätzlich auf die gleiche Weise. Sie enthalten Zellen, die von Datenquellen bereitgestellt werden, und reagieren durch den Aufruf von klar definierten Methoden auf Benutzeraktionen.

`UITableViewController` ist eine Unterklasse von `UIViewController`. Wie ihre Elternklasse ermöglicht sie die Erstellung von Bildschirmansichten bei minimalem Arbeitsaufwand und größtmöglicher Dienlichkeit. Die Klasse `UITableViewController` vereinfacht die Erstellung einer `UITableView`-Ansicht, wobei die wiederholten Schritte, die zur direkten Arbeit an Tabelleninstanzen notwendig sind, verringert werden oder gar nicht mehr nötig sind. `UITableViewController` übernehmen die lästigen Einzelheiten des Ansichtslayouts und vereinfachen den Einsatz von Tabellen, indem sie eine lokale `tableView`-Instanzvariable hinzufügen und eine automatische Tabellenprotokollunterstützung für Delegierungen der Datenquellen bieten.

11.1.1 Tabellen erstellen

Um Tabellen zu implementieren, müssen Sie drei Schlüsselelemente definieren: das Tabellenlayout, die Elemente in der Tabelle und die Reaktion auf Benutzeraktionen. Dazu müssen Sie Beschreibungen und Methoden zu Ihrem Programm hinzufügen. Das grafische Layout erstellen Sie beim Aufbau der Ansichten. Außerdem definieren Sie die Datenquellen, die die Tabellenzellen bei Bedarf füllen, und implementieren die Delegierungsmethoden, die auf Benutzereingaben wie die Änderung der Zeilenauswahl reagieren

Das Layout der Ansicht festlegen

`UITableView`-Instanzen sind, wie der Name bereits vermuten lässt, Ansichten. Sie stellen interaktive Tabellen auf dem Bildschirm des iPhones dar. Die Klasse `UITableView` erbt von der Klasse `UIScrollView`, woher auch die Scrollmöglichkeiten für die Tabelle stammen. Wie andere Ansichten auch, definieren `UITableView`-Instanzen ihre Grenzen über Rahmen (Frames) und können Kinder oder Eltern anderer Elemente sein. Um eine Tabellenansicht anzulegen, weisen Sie sie zu, initialisieren sie wie andere Ansichten mit einem Rahmen und fügen anschließend alle Verwaltungsdetails hinzu, indem Sie Datenquellen und Delegierungsobjekte zuweisen.

`UITableViewController` übernehmen die Gestaltungsarbeit für Sie. Die Klasse `UITableView Controller` erstellt einen standardmäßigen `UIViewController` und füllt diesen mit einer einzelnen `UITableView`. wobei sein Rahmen so eingerichtet wird. dass die Navigations- und Symbolleisten erscheinen. Auf diese Tabellenansicht können Sie über die Instanzvariable `tableView` zugreifen.

Eine wichtige Anmerkung: Stellen Sie beim Anlegen einer Unterklasse von `UITableViewController` sicher, dass Sie bei der Definition einer `loadView`-Methode deren Oberklassenimplementierung aufrufen:

```
- (void) loadView
{
   [super loadView];
   ...der Rest der Methode...
}
```

Dadurch wird die Tabellenansicht richtig eingerichtet, während Sie gleichzeitig eigene Funktionen wie Schaltflächen für Navigationselemente in der Unterklasse hinzufügen können. Wenn Sie einen `UITableViewController` in Interface Builder erstellen, brauchen Sie keine besonderen Aufrufe von `loadView`.

Eine Datenquelle zuweisen

`UITableView`-Instanzen stützen sich auf eine externe Quelle, die neue oder bestehende Zellen bei Bedarf füllt. Diese externe Quelle wird Datenquelle (*data source*) genannt und ist das Objekt, das bei einer Tabellenanfrage für die Rückgabe einer Zelle verantwortlich ist.

Datenquellen stellen Tabellenzellen aufgrund eines Indexpfades bereit. Indexpfade sind Member der Klasse `NSIndexPath` und beschreiben den Pfad durch einen Datenbaum zu einem bestimmten Knoten, also zu einem bestimmten Abschnitt und einer bestimmten Zeile Bei einfachen Tabellen wird häufig nur ein einziger Abschnitt verwendet, doch können Tabellendaten durch mehrere Abschnitte auch logisch gruppiert werden. Instanzen von `UITableView` geben über einen Indexpfad den gewünschten Abschnitt und die gewünschte Zeile in diesem Abschnitt an.

Die Aufgabe der Datenquelle besteht darin, diesen Pfad mit einer konkreten `UITableViewCell`-Instanz herzustellen und diese Zelle bei Bedarf zurückzugeben. Einen Indexpfad erstellen Sie, indem Sie Abschnitt und Zeile angeben:

```
myIndexPath = [NSIndexPath indexPathForRow:5 inSection:0];
```

Zum Abruf der Werte verwenden Sie die Eigenschaften `row` und `section` des Indexpfadobjekts.

Das iPhone SDK enthält einen eingebauten Mechanismus zur Wiederverwendung von Tabellenzellen. Wenn Zellen beim Scrollen aus dem sichtbaren Bereich der Tabelle geraten, kann die Tabelle sie in einer Warteschlange für die Wiederverwendung zwischenspeichern. Wenn Sie Zellen für die Wiederverwendung kennzeichnen, können Sie sie bei Bedarf aus dieser Warteschlange herausnehmen. Dadurch wird Arbeitsspeicher eingespart. Außerdem haben Sie so eine schnelle und effiziente Möglichkeit, um Zellen abzurufen, wenn die Benutzer schnell durch eine lange Liste auf dem Bildschirm blättern. In *Rezept 11.8* wird die Wiederverwendung von Zellen ausführlicher vorgestellt.

In Tabellen sind Sie nicht auf einen einzigen Zellentyp beschränkt. Das folgende Codefragment wählt eine von zwei möglichen Zellenarten aus, die von einer Wiederverwendungswarteschlange abgerufen werden. Standardzellen haben eine einzige Beschriftung, Zellen für Untertitel noch eine zweite. Der Bezeichner ist willkürlich vom Entwickler gewählt.

```
UITableViewCell *cell;
UITableViewCellStyle style;
NSString *identifier;

if (item.notes)
{
   style = UITableViewCellStyleSubtitle;
   identifier = @"notescell";
}
else
{
   style = UITableViewCellStyleDefault;
   identifier = @"basecell";
}

cell = [aTableView dequeueReusableCellWithIdentifier:identifier];
if (!cell)
   cell = [[[UITableViewCell alloc] initWithStyle:style
      reuseIdentifier:identifier] autorelease];
```

Mit der Eigenschaft `dataSource` der Tabelle weisen Sie ein Objekt als Datenquelle zu, wobei das Objekt das Protokoll `UITableViewDataSource` implementieren muss. Normalerweise dient der `UITableViewController`, der die Tabellenansicht besitzt, als deren Datenquelle. Wenn Sie mit Unterklassen von `UITableViewController` arbeiten, müssen Sie das Protokoll nicht deklarieren, da die Elternklasse es implizit unterstützt und automatisch den Controller als Datenquelle zuweist.

Nachdem Sie eine Datenquelle zugewiesen haben, können Sie die Tabelle mit Zellen füllen, indem Sie die Methode `tableView:cellForRowAtIndexPath:` implementieren. Beim Aufruf ihrer Methode `reloadData` beginnt die Tabelle mit der Abfrage ihrer Datenquelle, um die auf dem Bildschirm erkennbaren Zellen zu laden. Es ist jederzeit möglich, `reloadData` aufzurufen, um den Inhalt einer Tabelle zwangsweise neu zu laden.

11.1.2 Eine Delegierung zuweisen

Wie viele andere Interaktionsobjekte von Cocoa Touch verwenden `UITableView`-Instanzen Delegierungen, um sinnvoll auf Aktionen von Benutzern zu reagieren. Die Delegierung einer Tabelle kann auf Ereignisse wie Scrollvorgänge oder die Auswahl von Zeilen reagieren. Sie weist die Tabelle an, die Verantwortung für die Reaktion an das festgelegte Objekt abzugeben, das normalerweise der `UITableViewController` ist, der die Tabellenansicht besitzt.

Wenn Sie direkt mit einer `UITableView`-Ansicht arbeiten, können Sie die Standardmethode `setDelegate:` verwenden, um die Delegierung der Tabelle einzurichten. Diese Delegierung muss das Protokoll `UITableViewDelegate` implementieren. Wenn Klassen ein Delegierungsprotokoll

implementieren sollen, fügen Sie eine Deklaration in der Headerdatei der Klasse hinzu. Die Deklaration von Protokollen wird in Kapitel 3, *Objective-C-Trainingslager*, erläutert.

Wenn Sie mit `UITableViewController` arbeiten, lassen Sie die Methode `setDelegate:` und die Protokollzuweisung weg, da sich diese Klasse automatisch darum kümmert. Der vollständige Satz der Delegierungsmethoden ist in der SDK-Dokumentation von Apple aufgeführt. Die grundlegendsten Delegierungsmethoden werden in diesem Kapitel behandelt.

Hinweis

`UITableView`-Instanzen bieten neben Aufrufen von Delegierungsmethoden auch Benachrichtigungen, wodurch die verschiedenen Threads Ihres Programms miteinander kommunizieren können, indem sie Aktualisierungen über das standardmäßige `NSNotificationCenter` senden. Sie können Ihr Programm diese Benachrichtigungen mithilfe von standardmäßigen `NSNotificationCenter`-Beobachtern abonnieren lassen, damit es herausfinden kann, wann sich der Status der Tabelle ändert. Die einzige offizielle Tabellenbenachrichtigung im SDK 3.0 ist `UITableViewSelectionDidChangeNotification`.

11.2 Rezept: Eine sehr einfache Tabelle erstellen

Die Klasse `UITableViewController` bettet ein `UITableView`-Objekt in ein `UIViewController`-Objekt ein, das die Tabellenansicht verwaltet. Der Zugriff auf die Ansicht erfolgt über die Eigenschaft `tableView`. Die Controller legen automatisch sich selbst als Datenquelle und Delegierungsmethoden für die Tabellenansicht fest. Das ist schon fast ein »Plug&Play«-Vorgang. Bei einer sehr einfachen Tabelle müssen Sie nur einige Daten beibringen und wenige Datenquellenfunktionen hinzufügen, um Zellen einzufügen und die Anzahl der Zeilen und Abschnitte zu melden.

11.2.1 Die Tabelle füllen

Fast jedes Array aus Strings kann zum Einrichten und Füllen einer Tabelle verwendet werden. *Rezept 11.1* nutzt die Fähigkeit der Klasse `UIFont`, die verfügbaren Schriftarten als Liste von Strings aufzuführen. Ein Aufruf von `[UIFont familyNames]` gibt ein Array zurück, das die Namen dieser Schriftarten enthält. Dieses Rezept erstellt eine grundlegende Tabelle mit den Namen der Schriftarten.

Abbildung 11.1 zeigt die von diesem Code hervorgerufene Oberfläche, wie sie auf dem iPhone-Simulator erscheint. Die Ausführung auf dem Simulator führt zu einem ungewöhnlich umfangreichen Satz von Schriftarten, der auf den verfügbaren Schriften des Macintoshs und nicht auf denen des iPhones beruht.

▸ *Abbildung 11.1: Es ist einfach, eine* `UITableView` *aus einem Array mit Strings heraus mit Zellen zu füllen. Hier wird die Schriftartenliste der Klasse* `UIFont` *aufgeführt. Wird eine Schriftart angetippt, aktualisiert sich die Schrift der Titelleiste mit der ausgewählten Schrift.*

11.2.2 Datenquellenfunktionen

Zur Anzeige einer Tabelle muss jede Tabellendatenquelle drei Kernmethoden implementieren. Diese Methoden definieren die Struktur der Tabelle und stellen die Inhalte bereit:

- `numberOfSectionsInTableView` Tabellen können Daten in Abschnitten oder als eine einzige Liste darstellen. Bei einfachen Tabellen sollte immer 1 für einen einzelnen Abschnitt zurückgegeben werden, sodass die Tabelle als einzige Liste erscheint. Für Listen mit mehreren Abschnitten verwenden Sie den Wert 2 oder höher.
- `tableView: numberOfRowsInSection` Diese Methode gibt die Anzahl der Zeilen pro Abschnitt zurück, was bei einfachen Listen die Anzahl der Zeilen der gesamten Tabelle ist. Für umfangreichere Listen sollten Sie einen Weg zur Anzeige der Abschnitte anbieten. Die Nummerierung der Abschnitte beginnt bei 0.
- `tableView: cellForRowAtIndexPath` Diese Methode gibt eine Zelle an die aufrufende Methode zurück. Verwenden Sie die Eigenschaften `row` und `section` des Indexpfads, um festzustellen, welche Zelle bereitgestellt werden soll, und um die Vorteile der Wiederverwendung von Zellen zur Schonung des Arbeitsspeichers wo immer möglich zu nutzen.

11.2.3 Zellen wiederverwenden

Eine Möglichkeit, mit der das iPhone Speicher einspart, ist die Wiederverwendung von Zellen. Sie können jeder Zelle einen ID-String zuweisen, der festlegt, um welche Art von Zelle es sich handelt und wann eine außerhalb des Bildschirms gescrollte Zelle wiederverwendet werden soll. Verwen-

den Sie für die verschiedenen Typen von Zellen unterschiedliche Arten von IDs. Bei einfachen Tabellen reicht ein einzelner Bezeichner aus, beispielsweise `@"BaseCell"` in *Rezept 11.1*. Die Strings sind frei wählbar, weshalb Sie sie so bezeichnen können, wie Sie möchten. Allerdings sollten die Namen bei Verwendung von mehreren Zelltypen aussagekräftig sein. In *Rezept 11.8* wird die Wiederverwendung von Zellen genauer behandelt.

Bevor Sie eine neue Zelle zuweisen, sollten Sie immer überprüfen, ob eine wiederverwendbare Zelle verfügbar ist. Gibt Ihre Tabelle bei einer Anfrage an `dequeueReusableCellWithIdentifier:` den Wert `nil` zurück, müssen Sie eine neue Zelle erstellen.

Wird dagegen eine Zelle zurückgegeben, aktualisieren Sie sie mit den passenden Informationen für den momentanen Zeilen und Abschnittsindex. Sie müssen keine Zellen zur Wiederverwendungswarteschlange hinzufügen, da Cocoa Touch dies für Sie übernimmt.

11.2.4 Beispiel einer Schriftentabelle

Rezept 11.1 verdeutlicht, wie eine einfache Listentabelle erstellt werden kann. Es legt eine Tabelle an und füllt diese mit verfügbaren Schriftfamilien. Wird eine Schriftart angetippt, weist der Ansichtscontroller sie der Beschriftung in der blauen Navigationsleiste im oberen Bildschirmbereich zu. Dieses Verhalten ist in der Delegierungsmethode `tableView: didSelectRowAtIndexPath:` definiert, die beim Antippen einer Zeile aufgerufen wird.

Die Verwendung von `UITableViewController` als Delegierung ist hier eine gute Wahl, da die Aktion des Benutzers mit der Tabelle Einfluss auf deren Ansicht hat. Wenn Sie lieber eine andere Delegierung verwenden, können Sie stattdessen `setDelegate:` zusammen mit diesem Objekt aufrufen, um die Standardeinstellungen von `UITableViewController` zu überschreiben.

Zwischen Version 2.x und 3.x des SDK hat Apple schwer wiegende Änderungen bei Tabellenansichten vorgenommen. Vor Version 3.0 konnten Sie die Eigenschaften `text` und `image` einer Zelle direkt festlegen. Seit 3.0 gibt es die Eigenschaften `textLabel`, `detailLabel` und `imageView`, die jetzt auf ein Objekt der Benutzeroberfläche zeigen (zwei `UILabels` und eine `UIImageView`) und direkten Zugriff auf diese Objekte bieten.

Hinweis

In Tabellen können Sie die Farbe für die ausgewählte Zelle festlegen, indem Sie zwischen einer blauen und einer grauen Überlagerung auswählen. Setzen Sie die Eigenschaft `selectionStyle` entweder auf `UITableViewCellSelectionStyleBlue` oder auf `UITableViewCellSelectionStyleGray`. Sollten Sie keine Hervorhebung wünschen, verwenden Sie `UITableViewCellSelectionStyleNone`. Dadurch kann die Zelle immer noch ausgewählt werden, die Überlagerung wird aber nicht angezeigt.

```
#define MAINLABEL ((UILabel *)self.navigationItem.titleView)

@interface TableListViewController : UITableViewController
@end

@implementation TableListViewController

- (NSInteger)numberOfSectionsInTableView:(UITableView *)aTableView
{
   // Diese einfache Tabelle hat nur einen Abschnitt
   return 1;
}

- (NSInteger)tableView:(UITableView *)aTableView
   numberOfRowsInSection:(NSInteger)section
{
   // Anzahl der verwendeten Zeilen
   return [UIFont familyNames].count;
}

- (UITableViewCell *)tableView:(UITableView *)tView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Versucht, eine Zelle von der Warteschlange zu nehmen. Ist dies
   // nicht möglich, wird eine neue erstellt.
   UITableViewCellStyle style = UITableViewCellStyleDefault;
   UITableViewCell *cell = [tView
      dequeueReusableCellWithIdentifier:@"BaseCell"];
   if (!cell)
      cell = [[[UITableViewCell alloc] initWithStyle:style
         reuseIdentifier:@"BaseCell"] autorelease];

   // Legt den Zellentext fest
   cell.textLabel.text = [[UIFont familyNames]
      objectAtIndex:indexPath.row];
   return cell;
}

- (void)tableView:(UITableView *)tableView
   didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Aktualisiert bei Auswahl einer Zelle den Text in der Titelansicht
   NSString *font = [[UIFont familyNames]
      objectAtIndex:indexPath.row];
```

```
    [MAINLABEL setText:font];
    [MAINLABEL setFont:[UIFont fontWithName:font size:18.0f]];
}

- (void) loadView
{
    // Fügt zur Titelansicht der Navigationsleiste eine benutzerdefinierte
    // Beschriftung hinzu
    [super loadView];
    self.navigationItem.titleView = [[[UILabel alloc]
        initWithFrame:CGRectMake(0.0f, 0.0f, 200.0f, 30.0f)]
        autorelease];
    [MAINLABEL setBackgroundColor:[UIColor clearColor]];
    [MAINLABEL setTextColor:[UIColor whiteColor]];
    [MAINLABEL setTextAlignment:UITextAlignmentCenter];
}
@end
```

► *Rezept 11.1: Eine einfache Tabelle erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.2.5 Rezept: Die Hintergrundfarbe einer Tabelle ändern

Um für den Hintergrund einer Tabelle eine andere Farbe zu verwenden als Weiß, richten Sie wie in *Rezept 11.2* die Eigenschaft `backgroundColor` der Tabellenansicht ein. Die einzelnen Zellen erben diese Farbe, sodass sämtliche Bestandteile der Tabelle diese Farbe zeigen. Achten Sie darauf, dass die Farbe des Textes in den Zellen einen Kontrast zur Hintergrundfarbe bildet. Bei einem dunkelvioletten Hintergrund wie in diesem Rezept eignet sich sehr gut ein helles Weiß.

Leider können Sie den Hintergrund einzelner Zellen nicht direkt festlegen. Zwar können Sie die Eigenschaft `backgroundColor` einer Zelle ändern, aber die Farbänderung findet fast vollständig hinter den Beschriftungsansichten statt. Diese Beschriftungen stehen vor dem Hintergrund der Zellen und verdecken ihn. Sichtbare Änderungen an der Zelle gibt es nur wenige, wenn überhaupt. Um die größtmögliche Sichtbarkeit des Hintergrunds zu erreichen (die auch nicht sehr groß ist), legen Sie als Tabellenformat `UITableViewStyleGrouped` fest.

```
- (void)applicationDidFinishLaunching:(UIApplication *)application
{
    // Erstellt einen Tabellenansichtscontroller und legt dessen
    // Hintergrundfarbe fest
    TableListViewController *tlvc = [[TableListViewController alloc]
        init];
    tlvc.tableView.backgroundColor = COOKBOOK_PURPLE_COLOR;

    // Initialisiert den Navigationscontroller
    UINavigationController *nav = [[UINavigationController alloc]
        initWithRootViewController:tlvc];
    nav.navigationBar.tintColor = COOKBOOK_PURPLE_COLOR;

    // Erstellt das Hauptfenster
    UIWindow *window = [[UIWindow alloc] initWithFrame:[[UIScreen
        mainScreen] bounds]];
    [window addSubview:nav.view];
    [window makeKeyAndVisible];
}
```

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.2.6 Die Hintergrundfarbe nach der Position in der Tabelle festlegen

Da `UITableViews` Unterklassen von `UIScrollView` sind, können Sie die Hintergrundfarbe daran anpassen, wie weit der Benutzer durch die Tabelle geblättert hat, und dabei z. B. die Farbe aufhellen oder dunkler machen. Verwenden Sie den Prozentwert der durch das Blättern zurückgelegten Strecke als Multiplikationsfaktor für die einzelnen Komponenten der Hintergrundfarbe.

Im Standardzustand sind alle Instanzen von `UITableViewController` als `UIScrollView`-Delegierungen eingerichtet. Es ist keine zusätzliche Arbeit erforderlich, um die folgende `UIScrollViewDelegate`-Methode zu Ihrer Implementierung von `UITableViewController` hinzuzufügen. Der folgende Code berechnet die Sättigung der Hintergrundfarbe aus dem aktuellen Tabellen-Offset:

```
- (void) scrollViewDidScroll: (UIScrollView *) sv
{
    float percent = sv.contentOffset.y / sv.contentSize.height;
    percent = 0.25 + 3 * (MAX(MIN(1.0f, percent), 0.0f) / 4.0f);
    self.tableView.backgroundColor = [UIColor
```

```
        colorWithRed:percent * 0.20392 green:percent * 0.19607
        blue:percent * 0.61176 alpha: 1.0f];
}
```

Bei der Aktualisierung der Hintergrundfarbe sollten Sie die folgenden Punkte beachten. Erstens müssen Sie den Divisor um die Höhe der Tabelle verringern, wenn Sie das »Nachfedern« nicht aktivieren (wenn die Tabelle beim Einblenden also nicht über die Begrenzungen des Inhalts hinausgeht und dann zurückspringt). Zweitens müssen Sie beim Einrichten der Tabelle die ursprünglichen Farben festlegen, da die Farbe sonst »springt«, wenn der Benutzer die Tabelle zum ersten Mal berührt. Drittens ist dieser Ansatz zwar nicht sehr rechenintensiv, erfordert aber eine ständige Aktualisierung des Bildschirms, weshalb Sie bei Anwendungen, die den Prozessor stark belasten, darauf verzichten sollten.

11.3 Eine Tabelle mit einem Hintergrundbild erstellen

Rezept 11.3 geht über die Änderung der Hintergrundfarbe in *Rezept 11.2* hinaus und erstellt eine Tabelle mit einem Hintergrundbild. Anstatt den Hintergrund undurchsichtig zu färben, wird er mit einem Alpha-Wert von 0 transparent gemacht. Wenn Sie dann dem Anwendungsfenster vor dem Hinzufügen der Tabellenansicht ein Hintergrundbild hinzufügen, scheint dieses durch die Tabelle durch, wie Sie in *Abbildung 11.2* sehen.

▶ *Abbildung 11.2: Wenn Sie einen durchsichtigen Tabellenhintergrund mit einem Hintergrundbild kombinieren, erscheint die Tabelle über dem Bild.*

Die Tabelle rollt über das Bild hinweg, das statisch im Hintergrund bleibt. Verwenden Sie ein Bild, das zum Zusammenhang passt (z. B. das Firmenlogo), und hellen Sie es z. B. durch Herabsetzen der Sättigung so weit auf, dass der Text der Tabelle lesbar bleibt. Stimmen Sie die Textfarbe auf das Hintergrundbild ab.

```
- (void)applicationDidFinishLaunching:(UIApplication *)application
{
    // Erstellt einen Tabellenansichtscontroller mit durchsichtigem
    // Hintergrund
    TableListViewController *tlvc = [[TableListViewController alloc]
        init];
    tlvc.tableView.backgroundColor = [UIColor clearColor];

    // Initialisiert den Navigationscontroller
    UINavigationController *nav = [[UINavigationController alloc]
        initWithRootViewController:tlvc];
    nav.navigationBar.tintColor = COOKBOOK_PURPLE_COLOR;

    // Lädt das Hintergrundbild in eine Ansicht
    UIImageView *iv = [[[UIImageView alloc] initWithImage:[UIImage
        imageNamed:@"Backsplash.png"]] autorelease];

    // Erstellt das Hauptfenster
    UIWindow *window = [[UIWindow alloc] initWithFrame:[[UIScreen
        mainScreen] bounds]];
    [window addSubview:iv];
    [window addSubview:nav.view];
    [window makeKeyAndVisible];
}
```

► *Rezept 11.3: Eine Tabelle über einem statischen Bild anzeigen*

Den Rezeptcode finden

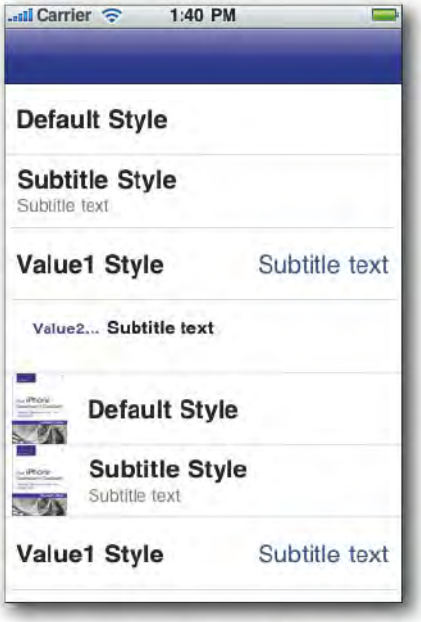
Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.4 Rezept: Unterschiedliche Zellentypen verwenden

Um auf dem iPhone nützliche Zellen verwenden zu können, bietet das SDK 3.0 vier verschiedene Arten von Tabellenansichtszellen an (siehe *Abbildung 11.3*). Der Standardtyp wurde zwar schon in Version 2.0 verwendet, hat inzwischen aber einen neuen Namen erhalten. Diese Zellentypen bilden eine neue Möglichkeit, um Tabellenzellen zu erstellen und mit ihnen zu arbeiten.

Vor Version 3.0 mussten Sie den Text einer Zelle direkt zuweisen. Jetzt weisen Zellen die Eigenschaften `textLabel` und `detailTextLabel` auf, die Zugriff auf die Beschriftungen selbst bieten. Damit können Sie die Textaspekte der Beschriftungen nach Belieben festlegen. Die folgende Übersicht stellt die vier neuen Typen vor:

- `UITableViewCellStyleDefault` Zellen dieser Art weisen ein einzelnes linksbündiges Textlabel und optional ein Bild auf. Wenn Sie Bilder verwenden, wird die Beschriftung nach rechts geschoben, wobei der Platz für den Text schrumpft. Zwar können Sie auch auf das Label `detailTextLabel` zugreifen und es ändern, doch wird es auf dem Bildschirm nicht angezeigt.
- `UITableViewCellStyleSubtitle` In diesen Zellen, die auch in der Anwendung *iPod* verwendet werden, wird das standardmäßige Textlabel ein wenig nach oben geschoben, um Platz für das kleinere Detaillabel darunter zu machen, das in Grau dargestellt wird. Wie in der Standardzelle ist auch in der Untertitelzelle optional ein Bild möglich.
- `UITableViewCellStyleValue1` Dieses Zellenformat, das in der Anwendung *Einstellungen* zum Einsatz kommt, zeigt ein großes, schwarzes Hauptlabel auf der linken Seite und ein etwas kleineres, blaues Untertitellabel rechts daneben. Bilder sind hier nicht möglich.
- `UITableViewCellStyleValue2` Diese Art von Zelle wird in den Anwendungen *Telefon/Kontakte* verwendet und besteht aus einem kleinen blauen Hauptlabel links und einem kleinen schwarzen Untertitellabel rechts. Wegen der geringen Breite des Hauptlabels wird ein Großteil des Textes abgeschnitten und durch Auslassungspunkte ersetzt. Auch in Zellen dieses Typs können Sie keine Bilder aufnehmen.



▸ *Abbildung 11.3: Cocoa Touch bietet vier Standardzellentypen, in denen teilweise auch Bilder angezeigt werden können.*

Rezept 11.4 zeigt den Code, mit dem die Zellen aus *Abbildung 11.3* erstellt wurden. Jede Zelle wird mit ihrem Typ beschriftet, und derselbe Text dient auch als ID für die Wiederverwendung. Außer in den ersten vier Zellen werden überall Bilder hinzugefügt, sodass Sie erkennen können, dass dies nur in der Standard- und der Untertiteldarstellung möglich ist.

```
- (UITableViewCell *)tableView:(UITableView *)tView
cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   UITableViewCellStyle style;
   NSString *cellType;

   // Wählt das Zellenformat und eine ID aus
   switch (indexPath.row % 4)
   {
      case 0:
         style = UITableViewCellStyleDefault;
         cellType = @"Default Style";
         break;
      case 1:
         style = UITableViewCellStyleSubtitle;
         cellType = @"Subtitle Style";
         break;
      case 2:
         style = UITableViewCellStyleValue1;
         cellType = @"Value1 Style";
         break;
      case 3:
         style = UITableViewCellStyleValue2:
         cellType = @"Value2 Style";
         break;
   }

   // Nimmt wenn möglich eine Zelle aus der Warteschlange und erstellt
   // anderenfalls eine neue
   UITableViewCell *cell = [tView
      dequeueReusableCellWithIdentifier:cellType];
   if (!cell)
      cell = [[[UITableViewCell alloc] initWithStyle:style
      reuseIdentifier:cellType] autorelease];

   // Fügt nach den ersten vier Zellen Bilder zu allen Zellen hinzu
   if (indexPath.row > 3)
      cell.imageView.image = [UIImage imageNamed:@"icon.png"];

   // Legt den Text der Zellen fest
   cell.textLabel.text = cellType;
   cell.detailTextLabel.text = @"Subtitle text";
   return cell;
}
```

► *Rezept 11.4: Tabellenzellen mit verschiedenen Formaten erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.5 Benutzerdefinierte Zellen in Interface Builder erstellen

In Interface Builder (IB) können Sie auf einfache Weise benutzerdefinierte `UITableViewCell`-Instanzen erstellen, ohne Unterklassen bilden zu müssen. Dabei können Sie die Zellen direkt in IB anlegen und dann in den Code laden, wie es in *Rezept 11.5* geschieht. Das große Problem bei der Verwendung von Interface Builder besteht darin, dass Ihre selbst gestalteten Elemente von jeglichen Zellinhalten überdeckt werden, wie *Abbildung 11.4* zeigt. Wenn Sie das Layout aus dem linken Bild gestalten wollen, erhalten Sie als Ergebnis gewöhnlich das Layout aus dem rechten Bild.

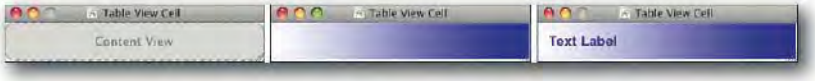
▸ *Abbildung 11.4: Bei der Arbeit mit Zellen, die in Interface Builder gestaltet wurden, wird die eigene Grafik, die Sie links sehen, von den eingebauten Zellinhalten überdeckt, wie das rechte Bild zeigt.*

Das liegt daran, dass beim Zuweisen der Labeleigenschaften einer Zelle die Labelansicht (samt allen erforderlichen unterstützenden Ansichten für das Label) erstellt wird, nachdem die Interface Builder-Zelle geladen wurde. Die zusätzlichen Ansichten werden oberhalb der Zelle platziert, sodass sie eigene Grafiken und andere IB-Elemente verdecken, mit denen Sie die Zelle versehen haben.

Sie könnten die einzelnen Unteransichten der Zelle bearbeiten und deren Hintergrundfarbe jeweils auf transparent setzen, doch Apple sieht es nicht gern, wenn Sie auf diese Weise in den Ansichten herumgeistern. Es gibt aber eine SDK-freundliche Lösung dafür. *Abbildung 11.5* (links) zeigt eine grundlegende `UITableViewCell` in Interface Builder, während in *Abbildung 11.5* (Mitte) dieselbe Zelle, aber mit einer Überlagerung in Form der selbst erstellten Grafik aus diesem Rezept zu sehen ist. Dies ist der Inhalt, den die Zelle darstellen muss, ohne dass er wie in *Abbildung 11.4* überdeckt wird.

Der Trick zur Beibehaltung der Zellgrafik besteht darin, nicht die eingebauten Eigenschaften `label` und `detailLabel` der Zelle zu verwenden, sondern stattdessen ein eigenes Label hinzuzufügen (siehe *Abbildung 11.5*, rechts). Die Labelansicht wird gekennzeichnet (hier mit `101`) und über diese ID aus der Zelle abgerufen. Die ID können Sie im Attribut-Informationsfeld von Interface Builder festlegen. In dem folgenden Makro wird die ID zum Abrufen des benutzerdefinierten Labels verwendet:

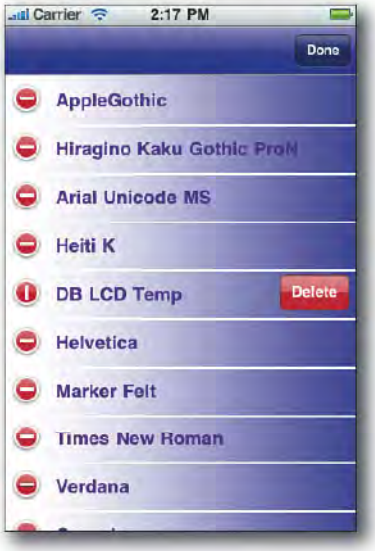
```
#define TEXTLABEL ((UILabel *)[cell viewWithTag:101])
```



▸ *Abbildung 11.5: Das linke Bild zeigt eine in Interface Builder erstellte Standardzelle ohne Inhalt. In der Mitte sehen Sie dieselbe Zelle, nachdem ihr eine benutzerdefinierte Grafik hinzugefügt wurde. Mit einem benutzerdefinierten Textlabel (rechts) kann eine Zelle wie in Abbildung 11.4 (links) korrekt beschriftet werden.*

Wenn Sie mehr als ein Label brauchen, z. B. für einen Untertitel, fügen Sie in Interface Builder einfach ein weiteres hinzu, und weisen Sie diesem wieder eine eindeutige ID zu.

Tabellen, die auf diese Weise erstellt wurden, sind mit allen Tabellenfunktionen vollständig kompatibel. Wie *Abbildung 11.6* zeigt, können Sie in benutzerdefinierten Zellen eine standardmäßige Bearbeitung vornehmen. Aktualisierung, Einrückung und alle anderen Verhaltensweisen laufen genauso ab wie in anderen Tabellenzellen. Wenn die vorgefertigten Labels und Bilder nicht für Ihre Zwecke geeignet sind, erstellen Sie eigene Ansichten, und fügen Sie sie als Unteransichten der Zelle hinzu.



▸ *Abbildung 11.6: Zellen, die in Interface Builder mit benutzerdefinierten Labeln erstellt wurden, sind bei sämtlichen standardmäßigen Bearbeitungsaufgaben voll funktionsfähig. Dies gilt auch für die Einrückung zur Anzeige von Lösch- und Bestätigungsschaltflächen.*

11.5.1 Tipps zum Erstellen von benutzerdefinierten Zellen

Beachten Sie beim Erstellen von benutzerdefinierten Tabellenansichtszellen in Interface Builder folgende Tipps:

- Erstellen Sie die neue `xib`-Datei, indem Sie in Xcode **File ▸ New File ▸ User Interface ▸ Empty XIB** wählen. Geben Sie der Datei einen aussagekräftigen Namen, z. B. `BaseCell.xib`, und speichern Sie sie.
- Öffnen Sie die leere `xib`-Datei in Interface Builder, und ziehen Sie eine `UITableViewCell` in das Projektfenster.
- Passen Sie den Zellinhalt an, indem Sie Grafiken und andere Schnittstellenelemente hinzufügen. Achten Sie darauf, dass Klassen zur Textbearbeitung wie `UITextField` und `UITextView` in Tabellenansichtszellen nicht gut funktionieren, sofern Sie keine besondere Sorgfalt walten lassen.
- Versuchen Sie beim Hinzufügen von eigenen Elementen auf der rechten Seite genügend Platz zu lassen (etwa 40 Pixel), damit die Zelle im Bearbeitungsmodus nach rechts verschoben werden kann. Anderenfalls werden die Elemente abgeschnitten.
- Legen Sie die Wiederverwendungs-ID (z. B. `"BaseCell"`) im Attribut-Informationsfeld der Zelle ⌘ + 1 fest. Das ID-Feld befindet sich oben in dem Informationsfeld.
- Das Bild der Zelle für den normalen und für den ausgewählten Zustand können Sie zwar im Informationsfeld festlegen, aber in der Praxis werden die Bilder gewöhnlich auf der Grundlage der aktuellen Daten generiert. Um eventuelle Bildeinstellungen sollten Sie sich im Code kümmern (mit den Eigenschaften `image` und `selectedImage`). Achten Sie darauf, dass die verwendeten Bilder die richtige Größe aufweisen. Hinweise dazu finden Sie in den Rezepten zum Erstellen von Vorschaubildern in *Kapitel 7, Mit Bildern arbeiten*.
- Sie können in Interface Builder kein Zellenformat auswählen, und nachdem die `nib`-Datei geladen wurde, können Sie das Format auch nicht mehr ändern. Wenn Sie ein anderes als das Standardformat benötigen, müssen Sie die Zelle im Code erstellen.
- Legen Sie für die Zellen die Höhe fest, die Sie brauchen, und passen Sie dann die Eigenschaft `rowHeight` der Tabelle entsprechend an.
- In Interface Builder gibt es zwar im Attribut-Informationsfeld der Zelle eine Option für den Separator, aber stattdessen sollten Sie die Eigenschaften `separatorStyle` und `separatorColor` der Tabellenansicht verwenden.

```
- (UITableViewCell *)tableView:(UITableView *)tView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Versucht, eine bereits erstellte Zelle wiederherzustellen.

   // Wenn es keine gibt, wird eine neue initialisiert.
   UITableViewCell *cell = [tView
      dequeueReusableCellWithIdentifier:@"BaseCell"];
```

```
    if (!cell)
        cell = [[[NSBundle mainBundle] loadNibNamed:@"BaseCell"
            owner:self options:nil] lastObject];

    // Setzt den Zellentext
    [TEXTLABEL setText:
        [[UIFont familyNames] objectAtIndex:indexPath.row]];
    return cell;
}
```

► *Rezept 11.5: In Interface Builder erstellte benutzerdefinierte Zellen verwenden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.5.2 Eigene Auswahlverhalten hinzufügen

Cocoa Touch bietet verschiedene Möglichkeiten, um die Auswahl einer Zelle durch den Benutzer hervorzuheben. Das Auswahlverhalten können Sie anpassen, indem Sie die drei möglichen Aspekte verändern, nämlich das angezeigte Bild, die Schriftfarbe und den Zellenhintergrund. Diese Einstellungen nehmen Sie in den Eigenschaften `selectedImage`, `selectedTextColor` und `selectedBackgroundView` vor.

Das Bild für den ausgewählten Zustand ersetzt das Bild, das Sie der Zelle (über die Eigenschaft `image` – siehe *Rezept 11.4*) hinzugefügt haben, wenn der Benutzer die Zelle auswählt. Dieses Bild muss dieselbe Größe aufweisen wie das ursprüngliche, damit das Zellenlayout stabil bleibt. Beispielsweise können Sie ein »leeres« Bild, also einen Platzhalter, durch einen Pfeil, einen Winkel oder einen Finger ersetzen, der auf die ausgewählte Zelle zeigt.

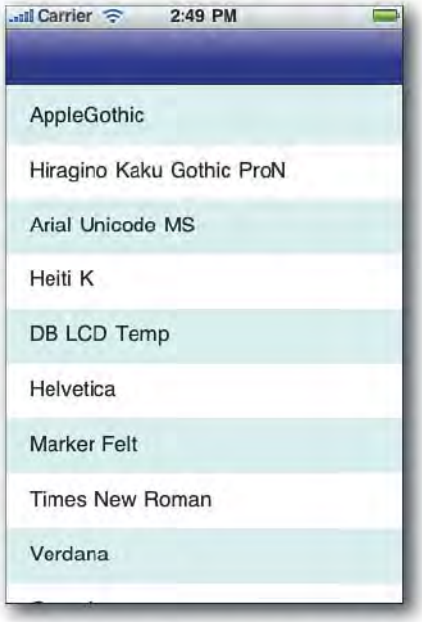
Die Eigenschaft `selectedTextColor` gilt offiziell als unerwünscht, wird aber in Interface Builder immer noch verwendet. Die Dokumentation von Apple schlägt vor, die neue Eigenschaft `textLabel` zu verwenden, es wird aber kein einfacher Weg aufgezeigt, um diese Eigenschaft für die Aktualisierung beim Auswählen oder Aufheben der Auswahl mit dem Objekt zu verbinden. Bis Apple sich um dieses Problem kümmert, können Sie die im folgenden Code vorgestellte Notlösung verwenden, durch die Warnungen über die erwünschte Eigenschaft zur Kompilierungszeit vermieden werden:

```
cell.selectedBackgroundView = [[[UIImageView alloc]
    initWithImage:[UIImage imageNamed:@"cellart.png"]] autorelease];

// Gilt als unerwünscht, wird aber immer noch in IB verwendet
[cell performSelector:@selector(setSelectedTextColor:)
    withObject:COOKBOOK_PURPLE_COLOR];
```

Die Eigenschaft `selectedBackgroundView` funktioniert genau so, wie die reguläre Eigenschaft `backgroundView` funktionieren sollte, was sie aber nicht tut (vergleiche *Abbildung 11.4*). Wird eine Zelle ausgewählt, erscheint der Hintergrund für diesen Zustand hinter dem Text, wobei Text und Grafik perfekt miteinander verschmelzen.

11.6 Rezept: Abwechselnde Zellenfarben

Obwohl der Wechsel zwischen weißen und blauen Zellen eine häufig verwendete und sehr empfohlene Tabellenfunktion ist, hat Apple diesen Stil nicht ins iPhone SDK aufgenommen. Mit der Technik zur Gestaltung von benutzerdefinierten Zellen aus *Rezept 11.5* können Sie eine in Interface Builder erstellte Zelle importieren. *Rezept 11.6* verwendet nicht eine, sondern zwei `xib`-Dateien für benutzerdefinierte Zellen und baut daraus die Struktur aus abwechselnd weißen und blauen Zellen auf, die Sie in *Abbildung 11.7* sehen.



▸ *Abbildung 11.7: Mit benutzerdefinierten Zellen können Sie abwechselnd weiße und blaue Zellen erstellen.*

Eine einfache Überprüfung auf eine gerade oder ungerade Zeilennummer (`row % 2`) legt fest, ob der Hintergrund mit Blau oder Weiß aufgefüllt wird. Da diese Tabelle nur aus einem Abschnitt besteht, ist die Mathematik einfach. Abwechselnd blaue und weiße Zellen eignen sich am besten für nicht gruppierte, nicht in Abschnitte eingeteilte Tabellen, sowohl was das Aussehen als auch die Programmierung angeht.

Beachten Sie, wie in diesem Rezept Zellen-IDs verwendet werden, um bereits geladene Zellen nach Bedarf wiederzuverwenden. Für die `xib`-Dateien wird der Dateiname als ID verwendet, was den Code erheblich vereinfacht. Sofern vorhandene blaue oder weiße Zellen aus der Wiederverwendungswarteschlange bezogen werden können, werden keine neuen erstellt.

Dieses Zellenformat eignet sich zwar für Bearbeitungsvorgänge, sowohl für das Löschen als auch die Neuanordnung, doch sollten Sie die Tabelle nach jeder Änderung durch den Benutzer neu laden, um die Blau/Weiß-Reihenfolge zu erhalten. Wenn der Benutzer ein Element verschiebt, behält es seine ursprüngliche Farbe bei, was zu einer optischen Uneinheitlichkeit führen kann, bis die Bearbeitung abgeschlossen ist. Geben Sie den Befehl zum Neuladen der Tabelle mit einem verzögerten Selektor von mindestens einer Viertel- bis einer halben Sekunde aus.

```
- (UITableViewCell *)tableView:(UITableView *)tView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Wählt den Zelltyp aus
   NSString *identifier = (indexPath.row % 2) ?
      @"WhiteCell" : @"BlueCell";

   // Versucht, eine Zelle aus der Warteschlange zu nehmen, und lädt
   // eine neue, wenn dies nicht möglich ist.
   UITableViewCell *cell = [tView dequeueReusableCellWithIdentifier:
      identifier];
   if (!cell)
      cell = [[[NSBundle mainBundle] loadNibNamed:identifier
         owner:self options:nil] lastObject];

   // Legt den Text der Zelle fest
   [(UILabel *)[cell viewWithTag:101] setText:
      [[UIFont familyNames] objectAtIndex:indexPath.row]];
   return cell;
}
```

▸ *Rezept 11.6: Tabellen mit abwechselnder Zellenfarbe erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

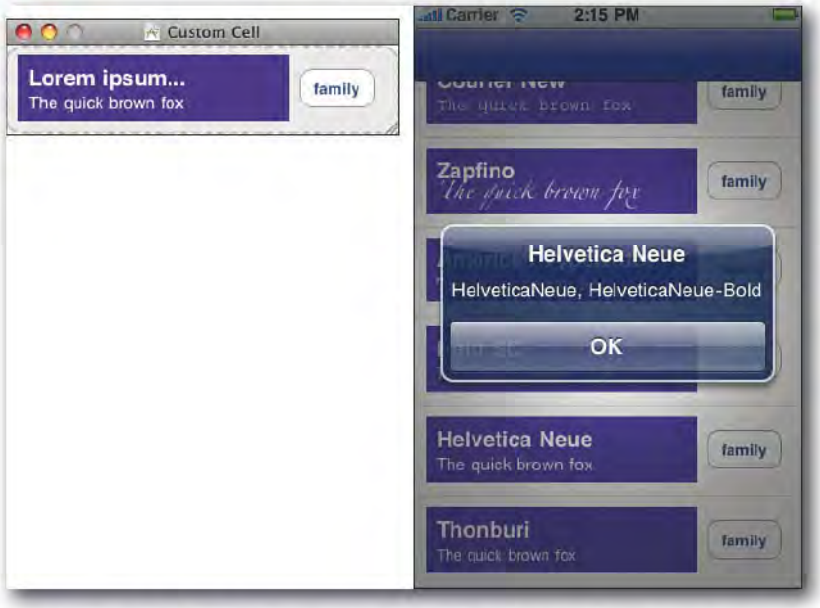
11.7 Rezept: Benutzerdefinierte Zellen mit eingebauten Steuerelementen anlegen

Wenn Sie in Interface Builder eigene Zellen gestalten, sind Sie nicht darauf beschränkt, Hintergrundgrafiken und Labels hinzuzufügen. Es ist nicht schwer, Schaltflächen oder andere Steuerelemente zu ergänzen, die auf zellspezifische Ereignisse reagieren. *Rezept 11.7* erstellt die in *Abbildung 11.8* gezeig-

ten Zellen. Sie enthalten ein Haupttextlabel zur Anzeige eines Schriftnamens, ein Untertitellabel, das einen Standardsatz in dieser Schriftart anzeigt, und die Schaltfläche **FAMILY**, die alle Mitglieder der betreffenden Schriftfamilie in einer Benachrichtigung anzeigt.

Anstatt wie in *Rezept 11.6* eine benutzerdefinierte `UITableViewCell` zu verwenden, wird in *Rezept 11.7* eine Unterklasse erstellt. `CustomCell` weist drei Outlets auf, nämlich die Schaltfläche, die beiden Labels und die Aktion `buttonPress:`, die bei `TouchUpInside`-Ereignissen aufgerufen wird. Die Verbindungen zwischen den `IBOutlets`, der `IBAction` und ihren Zeilen werden unmittelbar in Interface Builder hergestellt.

Damit dieses Rezept funktionieren kann, müssen Sie eine Instanz von `UITableViewCell` erstellen, den Klassenheader für `CustomCell` importieren (**FILE ▸ READ CLASS FILE**) und die Klasse der Instanz im Identitätsinformationsfeld (**TOOLS ▸ IDENTITY INSPECTOR**, ⌘ + 4) von `UITableViewCell` in `CustomCell` ändern. Anschließend können Sie in Interface Builder die Callback-Verbindungen für die Outlets und die Schaltfläche durch Ziehen der entsprechenden Elemente herstellen.



▸ *Abbildung 11.8: Die in Interface Builder zu dieser benutzerdefinierten Zelle hinzugefügte Schaltfläche (links) startet eine Benachrichtigung mit einer Liste der Mitglieder der Schriftfamilie (rechts). Jede Schaltfläche ist mit ihrer eigenen Zelle verbunden und funktioniert unabhängig davon, ob die Zelle ausgewählt ist.*

```
@interface CustomCell : UITableViewCell {
    IBOutlet UIButton *button;
    IBOutlet UILabel *primaryLabel;
    IBOutlet UILabel *secondaryLabel;
}

@property (assign) UIButton *button;
```

```
@property (assign) UILabel *primaryLabel;
@property (assign) UILabel *secondaryLabel;

- (IBAction) buttonPress: (UIButton *) aButton;
@end

@implementation CustomCell
@synthesize button;
@synthesize primaryLabel;
@synthesize secondaryLabel;

- (IBAction) buttonPress: (UIButton *) aButton
{
   NSString *fontName = self.primaryLabel.text;
   NSArray *fonts = [UIFont fontNamesForFamilyName:fontName];
   UIAlertView *av = [[[UIAlertView alloc] initWithTitle:fontName
      message:[fonts componentsJoinedByString:@", "] delegate:nil
      cancelButtonTitle:@"OK" otherButtonTitles:nil] autorelease];
   [av show];
}
@end
```

► *Rezept 11.7: Callbacks für Schaltflächen in Zellen erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.8 Rezept: Den Status von Steuerelementen in benutzerdefinierten Zellen speichern

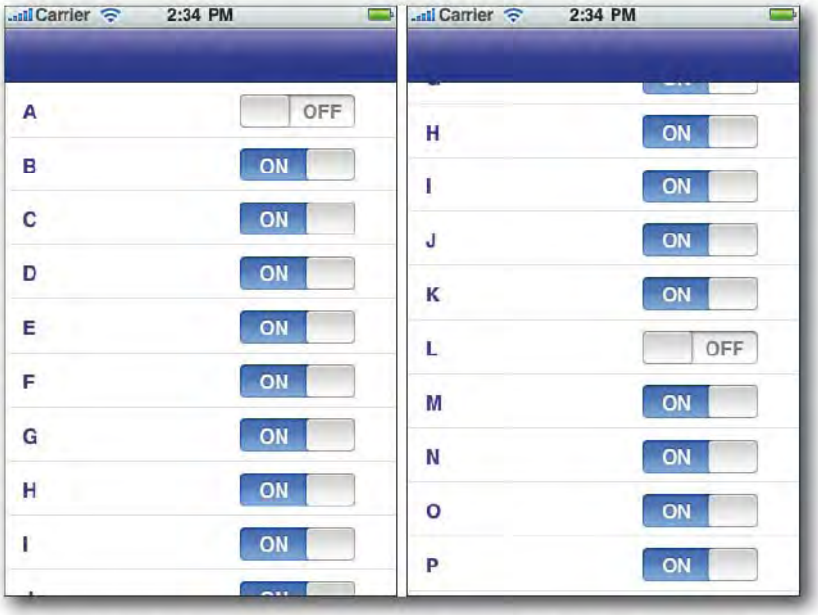
Zellen haben kein nennenswertes »Gedächtnis«. Sie wissen nicht, wie sie beim letzten Mal von der Anwendung benutzt wurden, denn sie sind einfach nur Ansichten und sonst nichts. Das bedeutet, dass es zu unerwarteten und sinnlosen Ergebnissen kommen kann, wenn Sie Zellen wiederverwenden, ohne sie mit irgendeiner Form von Datenmodell zu verbinden. Das ist eine natürliche Folge des MVC-Entwurfsmodells.

Stellen Sie sich folgende Situation vor. Sie haben eine Reihe von Zellen erstellt, die jeweils über einen Umschalter verfügen. Die Benutzer können auf diese Schalter zugreifen und ihren Wert ändern. Eine Zelle, die außerhalb des Bildschirms gerät, landet in der Warteschlange zur Wiederverwendung. Daher kann ein Tabellenelement einen bereits umgeschalteten Zustand aufweisen, auch wenn der Benutzer es noch gar nicht berührt hat.

Abbildung 11.9 veranschaulicht dieses Problem. Die Zelle für Element A wurde für Element L wiederverwendet und wird daher mit der Einstellung **OFF** angezeigt, obwohl der Benutzer Element L noch gar nicht angefasst hat. Es ist die Zelle, die die Einstellung beibehält, nicht das logische Element. Stützen Sie sich daher nicht auf Zellen, um den Status aufzubewahren.

Um das Problem zu lösen, müssen Sie den Status der Zellen an einem gespeicherten Modell überprüfen. So bleibt die Ansicht im Einklang mit der Semantik der Anwendung. *Rezept 11.8* verwendet ein benutzerdefiniertes Dictionary, um den Zellstatus mit dem Zellinhalt zu verknüpfen. Es gibt auch andere Möglichkeiten, um dieses Problem zu lösen, aber dieses einfache Beispiel gibt Ihnen schon einen Vorgeschmack auf den erforderlichen Abgleich zwischen Modell und Ansicht einer Datenquelle, deren Ansichten Statusinformationen darstellen.

Da der Status im Tabellenansichtscontroller gespeichert wird, muss jeder Aufruf in der Lage sein, gewissermaßen »nach Hause zu telefonieren«, wenn der Schalterstatus aktualisiert wird. Die benutzerdefinierte Eigenschaft `tableViewController`, die hier festgelegt wird, ist für die Rückwärtsverknüpfung da, während die Eigenschaft `customSwitch` auf den aktuellen, vom Benutzer festgelegten Zustand zugreift.



▶ *Abbildung 11.9: Die Zelle zur Darstellung von Element A (links) wird für Element L wiederverwendet (rechts), behält aber die vorherige Schalterstellung bei.*

```
- (UITableViewCell *)tableView:(UITableView *)tView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Nimmt wenn möglich eine Zelle aus der Warteschlange
   CustomCell *cell = (CustomCell *)[tView
      dequeueReusableCellWithIdentifier:@"BaseCell"];
```

```
    if (!cell)
        cell = [[[NSBundle mainBundle] loadNibNamed:@"BaseCell"
            owner:self options:nil] lastObject];

    // Bestimmt Schlüssel und Status aufgrund der Zeile
    NSString *key = [ALPHA objectAtIndex:indexPath.row];
    cell.customLabel.text = key;
    cell.tableViewController = self;
    if (self.switchStates)
    {
        NSNumber *state;
        if (state = [self.switchStates objectForKey:key])
            cell.customSwitch.on = [state boolValue];
        else
        {
            cell.customSwitch.on = YES;
            [self.switchStates setObject:[NSNumber numberWithBool:YES]
                forKey:key];
        }
    }

    return (UITableViewCell *)cell;
}

- (void) updateSwitch:(UISwitch *) aSwitch forItem: (NSString *) anItem
{
    // Der Schalter sendet bei einer Wertänderung einen Callback
    if (self.switchStates)
        [self.switchStates setObject:[NSNumber
            numberWithBool:aSwitch.on] forKey: anItem];
}
```

▸ *Rezept 11.8: Wiederverwendete Tabellenzellen mit dem gespeicherten Status aktualisieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.8.1 Die Wiederverwendung von Zellen sichtbar machen

Rezept 11.8 hilft bei der Lösung von Problemen mit Abweichungen zwischen Zelle und Modell. Der folgende Code macht sichtbar, wie die Zellen wiederverwendet werden. Hierbei wird jede neue Zelle gekennzeichnet, sobald sie erstellt wird, sodass sich nachverfolgen lässt, wie die einzelnen Zellen im Lebenszyklus einer sehr umfangreichen Tabelle verwendet und wiederverwendet werden. In diesem Fall besteht die Tabelle aus einer Million Einträgen. Probieren Sie dieses Fragment aus (eine vollständige Version finden Sie im Beispielcode zum Buch), und blättern Sie kräftig in beiden Richtungen durch die Liste. Wenn Sie dabei nur ausreichend sprunghaft vorgehen, können Sie die Anordnung der Zellen ziemlich durcheinanderbringen. Sie werden aber auch feststellen, dass Sie selbst für eine Million Einträge insgesamt nur maximal elf Tabellenzellen benötigen.

```
@implementation TableListViewController
- (NSInteger)numberOfSectionsInTableView:(UITableView *)aTableView
{
   return 1;
}

- (NSInteger)tableView:(UITableView *)aTableView
   numberOfRowsInSection:(NSInteger)section
{
   return 999999; // Sehr viel!
}

- (UITableViewCell *)tableView:(UITableView *)tView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   UITableViewCellStyle style = UITableViewCellStyleDefault;
   UITableViewCell *cell = [tView
      dequeueReusableCellWithIdentifier:@"BaseCell"];

   // Erstellt eine neue Zelle mit eindeutiger Nummer, wenn keine Zeile
   // aus der Warteschlange entnommen werden kann
   if (!cell)
   {
      cell = [[[UITableViewCell alloc] initWithStyle:style
         reuseIdentifier:@"BaseCell"] autorelease];
      cell.textLabel.text = [NSString stringWithFormat:
         @"Cell %d", ++count];
   }
   return cell;
}
@end
```

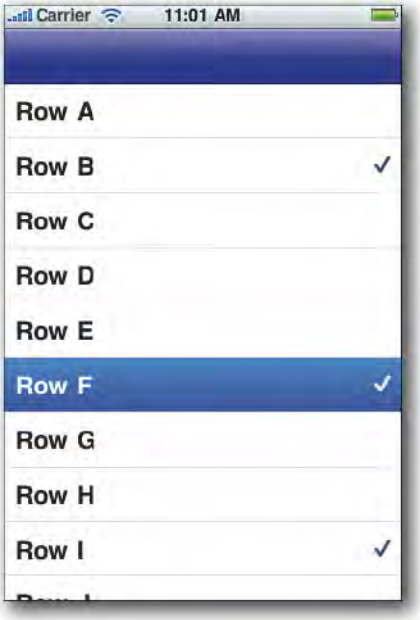
Jede Zelle implementiert die Methode `prepareForReuse`, die aufgerufen wird, bevor die Anforderung der Tabellenansicht zur Entnahme aus der Warteschlange eine Zelle zurückgibt. Um den Inhalt einer Zelle vor deren Wiederverwendung zurückzusetzen, können Sie eine Unterklasse von `UITableViewCell` erstellen und darin diese Methode überschreiben.

11.9 Rezept: Tabellenzellen abhaken

Hilfsansichten erweitern den normalen Funktionsumfang von `UITableViewCell`. Die gebräuchlichsten Hilfselemente sind die Schaltfläche **Löschen** und die Ziehleisten für die Neuanordnung, doch können Sie auch Häkchen für eine interaktive *1-aus-n-* oder *n-aus-n*-Auswahl hinzufügen. Bei dieser Art von Auswahl können Sie den Benutzer beispielsweise auffordern, das auszuwählen, was er zum Abendbrot essen möchte oder welche Elemente aktualisiert werden sollen. Diese Art von Optionsschalter oder Markierungsfeld bereichert die Interaktion mit Tabellen. *Rezept 11.9* veranschaulicht, wie diese Art von Tabelle erstellt werden kann.

Abbildung 11.10 zeigt solche Häkchen in einer Benutzeroberfläche, nämlich einer standardmäßigen `UITableView` mit Hilfszellen. Markierungshäkchen erscheinen neben den ausgewählten Elementen. Werden die Elemente angetippt, erscheinen bzw. verschwinden die Häkchen. Wie in *Rezept 11.8* wird hier ein gemeinsames Dictionary verwendet, um nachzuverfolgen, welche logischen Elemente abgehakt sind, sodass durch die Wiederverwendung von Zellen keine Unstimmigkeiten auftreten können.

Für abgehakte Elemente wird der Hilfsansichtstyp `UITableViewCellAccessoryCheckmark` verwendet, für nicht abgehakte Elemente `UITableViewCellAccessoryNone`. Diese Typen legen Sie über die Eigenschaft `accessoryType` der Zelle fest.



▸ *Abbildung 11.10: Hilfselemente mit Markierungshäkchen stellen eine bequeme Möglichkeit dar, um 1-aus-n oder n-aus-n Elemente aus einer Liste auszuwählen.*

Beachten Sie, dass hier die Zelle ausgewählt wird und nicht das mit Ihr verbundene logische Element (wobei dieses Element jedoch in dem gemeinsamen `stateDictionary` aktualisiert wird). Die Zellen bleiben bei der nächsten Wiederverwendung im jeweiligen abgehakten oder nicht abgehakten Zustand, sodass Sie bei der Entnahme einer Zelle aus der Warteschlange die Hilfsansicht stets so festlegen müssen, dass sie dem Status im Dictionary entspricht. Die Beibehaltung des Zellenstatus wird in *Rezept 11.8* behandelt.

```
- (UITableViewCell *)tableView:(UITableView *)tView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Ruft eine Zelle ab oder erstellt sie
   UITableViewCellStyle style = UITableViewCellStyleDefault;
   UITableViewCell *cell = [tView
      dequeueReusableCellWithIdentifier:@"BaseCell"];
   if (!cell) cell = [[[UITableViewCell alloc] initWithStyle:style
      reuseIdentifier:@"BaseCell"] autorelease];

   // Legt das Zellenlabel fest
   NSString *key = [@"Row " stringByAppendingString:[ALPHA
      objectAtIndex:indexPath.row]];
   cell.textLabel.text = key;

   // Setzt das Markierungshäkchen der Zelle
   NSNumber *checked = [self.stateDictionary objectForKey:key];
   if (!checked) [self.stateDictionary setObject:(checked = [NSNumber
      numberWithBool:NO]) forKey:key];
   cell.accessoryType = checked.boolValue ?
      UITableViewCellAccessoryCheckmark :
      UITableViewCellAccessoryNone;
   return cell;
}

- (void)tableView:(UITableView *)tableView
   didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Ruft Zelle und Schlüssel ab
   UITableViewCell *cell = [self.tableView
      cellForRowAtIndexPath:indexPath];
   NSString *key = cell.textLabel.text;

   // Kehrt den Wert um und speichert ihn
   BOOL isChecked = !([[self.stateDictionary objectForKey:key]
      boolValue]);
   NSNumber *checked = [NSNumber numberWithBool:isChecked];
   [self.stateDictionary setObject:checked forKey:key];
```

```
    // Aktualisiert das Hilfselement der Zelle mit dem Häkchen
    cell.accessoryType = isChecked ? UITableViewCellAccessoryCheckmark
        : UITableViewCellAccessoryNone;
}
```

► *Rezept 11.9: Hilfselemente mit Markierungshäkchen in Zellen verwenden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.10 Die Hervorhebung der Auswahl von Zellen entfernen

Bei der Arbeit mit Tabellen müssen Sie manchmal verhindern, dass der Zellenstatus erhalten bleibt. Das ist der Fall, wenn die Benutzer zwar mit der Tabelle umgehen und Zellen berühren sollen, die Auswahl der Zelle aber nach dem Ende der Benutzeraktion nicht fortbestehen soll. Cocoa Touch bietet zwei Möglichkeiten, um die dauerhafte Zellenauswahl in Tabellen zu verhindern.

Erstens können Sie die Eigenschaft `selectionStyle` einer Zelle auf `UITableViewCellSelectionStyleNone` setzen. Dadurch wird die graue oder blaue Überlagerung deaktiviert, die – wie in Zeile F von *Abbildung 11.10* – auf einer ausgewählten Zelle angezeigt wird. Die Zelle ist nach wie vor ausgewählt, wird aber nicht hervorgehoben dargestellt. Wenn die Auswahl der Zelle jedoch noch andere Auswirkungen hat als die bloße Anzeige von Informationen, ist dies nicht der geeignete Ansatz. Stattdessen müssen Sie das folgende Vorgehen anwenden.

Bei dem zweiten Ansatz wird die Zelle zwar hervorgehoben, diese Hervorhebung aber nach Abschluss der Benutzeraktion entfernt, indem Sie der Tabelle mitteilen, dass die Auswahl der betreffenden Zeile aufgehoben wird. In *Rezept 11.10* löst die Auswahl durch einen Benutzer eine verzögerte Aufhebung der Auswahl nach einer halben Sekunde auf (über die in diesem Rezept definierte benutzerdefinierte Methode `deselect:`). Diese Methode ruft die Methode `deselectRowAtIndexPath:` der Tabellenansicht auf, die die aktuelle Auswahl entfernt. Mit diesem Ansatz können Sie sowohl die Benutzeraktion durch eine Hervorhebung bestätigen als auch eine statusfreie Darstellung ermöglichen, in der der Benutzer nicht sieht, was zurzeit ausgewählt ist.

```
// Hebt die Auswahl auf
- (void) deselect: (id) sender
{
    [self.tableView deselectRowAtIndexPath:[self.tableView
        indexPathForSelectedRow] animated:YES];
}

// Reagiert auf die Benutzerauswahl
```

```
- (void)tableView:(UITableView *)tableView
   didSelectRowAtIndexPath:(NSIndexPath *)newIndexPath
{
   printf("User selected row %d\n", [newIndexPath row] + 1);
   [self performSelector:@selector(deselect:) withObject:nil
      afterDelay:0.5f];
}
```

► *Rezept 11.10: Die Auswahl einer Tabellenzeile aufheben*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.11 Hilfselemente für Einblenddreiecke verwenden

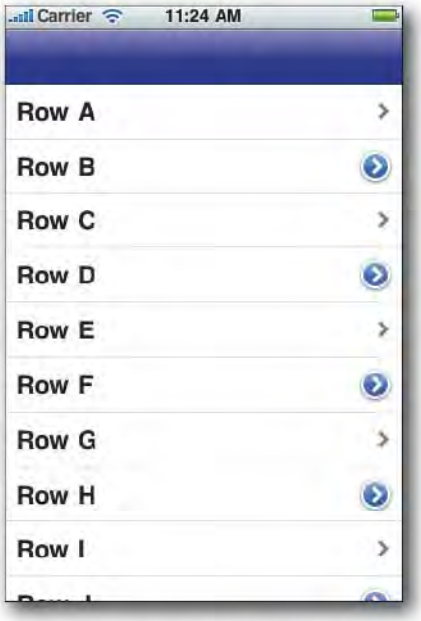
Mit Einblenddreiecken sind die kleinen grauen oder blauen, nach rechts weisenden Winkel gemeint, die sich auf der rechten Seite von Tabellenzellen befinden. Sie ermöglichen die Verknüpfung einer Zelle mit einer zusätzlichen Ansicht. Zum Beispiel verweisen diese Dreiecke in den Programmen *Kontakte* und *Kalender* auf Ansichten, die Ihnen die Anpassung von Kontaktinformationen und Terminen ermöglichen. *Abbildung 11.11* zeigt eine Tabellenansicht, in der jede Zelle über ein Einblend-Steuerelement verfügt, wobei beide verfügbaren Typen verwendet werden.

Das blaue und das graue Einblenddreieck haben unterschiedliche Rollen. Die blaue Version, `UITableViewCellAccessoryDetailDisclosureButton`, ist tatsächlich eine Schaltfläche. Sie reagiert auf Antippen und weist darauf hin, dass sich hinter ihr eine vollständig interaktive Detailansicht verbirgt. Die graue Version `UITableViewCellAccessoryDisclosureIndicator` verfolgt keine Berührungen nach und leitet Benutzer zu einer Ansicht weiter, die genauere Optionen zur aktuellen Auswahl enthält.

Diese beiden Hilfselemente sehen Sie im Programm *Einstellungen* in Aktion. Auf dem Einstellungsbildschirm für Wi-Fi-Netzwerke führen die Einblenddreiecke zu den Angaben über das jeweilige Wi-Fi-Netzwerk: IP-Adresse, Subnetzmaske, Router, DNS usw. Die Einblendanzeige für **Anderes** erlaubt es Ihnen, ein neues Netzwerk hinzuzufügen, indem Sie einen Bildschirm für die Eingabe von Netzwerkinformationen öffnen. Anschließend erscheint ein neues Netzwerk mit einem eigenen Einblenddreieck.

Solche Erweiterungselemente werden auch immer dann angezeigt, wenn ein Bildschirm zu einem Untermenü führt. Bei der Arbeit mit Untermenüs ist nur die Verwendung des grauen Einblenddreiecks möglich. Die allgemeine Regel hierbei lautet: Für Untermenüs verwenden Sie graue Dreiecke, für Objektanpassungen blaue. Bei grauen Winkeln müssen Sie auf die Zellauswahl reagieren, bei den blauen auf Berührungen der Hilfsschaltflächen.

Rezept 11.11 veranschaulicht die Verwendung von Einblenddreiecken in Ihren Programmen. Dieser Code setzt den `accessoryType` für jede Zelle auf `UITableViewCellAccessoryDetailDisclosure➥Button` und, was besonders wichtig ist, `editingAccessoryType` auf `UITableViewCell➥AccessoryNone`. Wenn die Steuerelemente zum Löschen oder Neuanordnen erscheinen, wird das Einblenddreieck ausgeblendet, sodass die Benutzer die vollständige Kontrolle bei der Bearbeitung haben und nicht versehentlich eine neue Ansicht öffnen.



▸ *Abbildung 11.11: Mit den nach rechts zeigenden Einblenddreiecken können Sie einzelne Tabellenelemente mit einer anderen Ansicht verknüpfen.*

Um Berührungen des Einblenddreiecks zu verarbeiten, ermöglicht Ihnen die Methode `tableView:➥accessoryButtonTappedForRowWithIndexPath:`, die angetippte Zeile abzurufen und eine entsprechende Antwort zu implementieren. Dieses Beispiel verschiebt lediglich einen neuen `UIViewController`, der ein gespeichertes Bild anzeigt. In der Praxis würden Sie zu einer Ansicht wechseln, die mehr über das ausgewählte Objekt erklärt oder die es ermöglicht, aus weiteren Optionen auszuwählen.

Die grauen Einblenddreiecke erfordern einen anderen Ansatz. Da es sich bei diesen Hilfselementen nicht um Schaltflächen handelt, reagieren sie auf die Auswahl einer Zelle statt auf die Berührung. Fügen Sie Ihrer Programmlogik`tableView:didSelectRowAtIndexPath:` hinzu, um die Einblendansicht auf dem Navigationsstack zu platzieren oder einen modalen Ansichtscontroller darzustellen.

```
- (UITableViewCell *)tableView:(UITableView *)tView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Ruft eine Zelle ab oder erstellt sie
   UITableViewCellStyle style = UITableViewCellStyleDefault;
   UITableViewCell *cell = [tView
      dequeueReusableCellWithIdentifier:@"BaseCell"];
```

```
    if (!cell) cell = [[[UITableViewCell alloc] initWithStyle:style
        reuseIdentifier:@"BaseCell"] autorelease];

    // Legt das Zellenlabel fest
    NSString *key = [@"Row " stringByAppendingString:[ALPHA
        objectAtIndex:indexPath.row]];
    cell.textLabel.text = key;

    cell.accessoryType =
        UITableViewCellAccessoryDetailDisclosureButton;
    cell.editingAccessoryType = UITableViewCellAccessoryNone;

    return cell;
}

// Reagiert auf Berührung der Hilfsschaltfläche
-(void)tableView:(UITableView *)tableView
    accessoryButtonTappedForRowWithIndexPath:(NSIndexPath *)indexPath
{
    [[self navigationController] pushViewController:[ImageController
        newController] animated:YES];
}
```

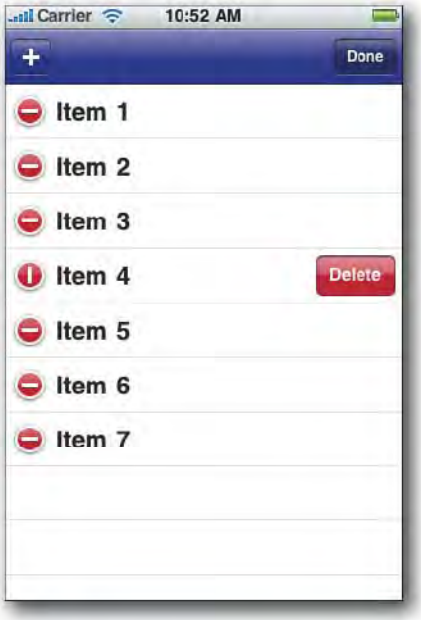
► *Rezept 11.11: Einblenddreiecke zur Anzeige von neuen »Detail«-Ansichten verwenden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.12 Rezept: Zellen löschen

Bei der täglichen Arbeit wird jeder iPhone-Benutzer schnell mit den kleinen roten Kreisen vertraut, die das Löschen von Zellen aus Tabellen ermöglichen, wobei viele Benutzer auch die grundlegende Funktion nutzen, etwas zum Löschen wegzustreichen. Das interaktive Löschen ist eine der am besten gestalteten Funktionen des iPhones. *Rezept 11.12* zeigt, wie Sie eine Tabelle erstellen können, die sinnvoll auf das Löschen von Zellen reagiert. In diesem Beispiel können Benutzer auch neue Zellen durch Antippen der entsprechenden Schaltfläche hinzufügen oder sie entweder durch Wegstreichen oder durch die roten Steuerelemente im Bearbeitungsmodus löschen (siehe *Abbildung 11.12*).



▸ *Abbildung 11.12: Die roten Steuerelemente zum Entfernen ermöglichen Ihren Benutzern das interaktive Löschen von Elementen einer interaktiven Tabelle.*

11.12.1 Steuerelemente zum Entfernen anzeigen

Seit SDK 3.0 ist es besonders einfach, Steuerelemente zum Entfernen in Ihre Programme einzufügen. Wenn Zellen bearbeitet werden sollen, rufen Sie `[self.tableView setEditing:YES animated:YES]` auf. Dadurch wird die Eigenschaft `editing` der Tabelle aktualisiert, sodass die Steuerelemente aus *Abbildung 11.12* angezeigt werden.

Verwenden Sie in Benutzeroberflächen auf dem iPhone nach Möglichkeit Animationen, um einen Status in den nächsten zu überführen, sodass die Benutzer auf die Änderung des Modus auf dem Bildschirm vorbereitet werden. In dem seltenen Fall, dass Sie die Änderung aus einem wichtigen Grund nicht animieren möchten, übergehen Sie `NO` anstelle von `YES`.

Rezept 11.12 verwendet eine einzige Methode namens `enterEditMode`. Wenn ein Benutzer auf die Schaltfläche **Edit** in der Navigationsleiste tippt, ruft die Anwendung diese Methode auf. Sie entfernt die aktuelle Elementauswahl, führt die Methode `setBarButtonItems` aus, die die Beschriftung von **Edit** in **Done** ändert, und aktiviert die Eigenschaft `editing` der Tabelle.

11.12.2 Steuerelemente zum Entfernen ausblenden

Wenn die Benutzer die Bearbeitung abgeschlossen haben und wieder zur normalen Tabellenansicht zurückkehren möchten, gehen Sie umgekehrt vor. Entfernen Sie die Steuerelemente (`[self.tableView setEditing:NO animated:YES]`), und aktualisieren Sie die Schaltfläche in der Navigationsleiste, sodass sie wieder den alten Zustand annimmt. In *Rezept 11.12* wird geprüft, ob noch Elemente übrig geblieben sind, und es wird die Schaltfläche **Edit** ausgeblendet, wenn dies nicht der Fall ist.

11.12.3 Löschanforderungen verarbeiten

Beim Löschen von Zeilen kommuniziert die Tabelle über den Callback `tableView: commitEditing Style: forRowAtIndexPath:` mit Ihrem Programm. Das Element wird aus der sichtbaren Tabelle entfernt, die zugrunde liegenden Daten werden aber nicht geändert. Sie müssen das Element auch aus der Datenquelle entfernen, da es sonst bei der nächsten Aktualisierung der Tabelle wieder erscheint. Diese Methode bietet die Möglichkeit, die Datenquelle zu aktualisieren und damit auf den Löschvorgang zu reagieren, den der Benutzer ausgeführt hat.

Hier findet die eigentliche Löschung des Elements aus der Datenstruktur statt, die die Datenquellenmethoden versorgt (in diesem Rezept über ein `NSMutableArray` mit dem Titeln), und hier werden in der Praxis alle Aktionen wie das Löschen von Dateien als Folge der Benutzeraktion durchgeführt. In diesem Beispielcode verschwindet die Zelle, aber es ergeben sich keine praktischen Konsequenzen daraus. Das Beispiel basiert nicht auf einem Modell der Praxis. Stattdessen verliert die Tabelle einfach nur den betroffenen nummerierten Zellentitel.

Sowohl das Hinzufügen als auch das Löschen von Elementen werden von derselben Methode gehandhabt, nämlich von `updateItemAtIndexPath:withString:`. Das mag eine merkwürdige Art zum Umgang mit solchen Anforderungen sein, da eine zusätzliche Methode und zusätzliche Schritte erforderlich sind. Dieser Ansatz aber bildet die Grundlage für die Widerrufsmöglichkeiten, die in *Rezept 11.14* besprochen werden. Die Verwendung von `NSUndoManager` mit einer einzigen Aktualisierungsmethode ermöglicht einheitliche Widerrufsmöglichkeiten für diese beiden Operationen.

11.12.4 Zellen durchstreichen

Das Durchstreichen stellt eine elegante Möglichkeit zum Löschen von Elementen in `UITableView`-Instanzen dar. Um das Durchstreichen zu aktivieren, müssen Sie nichts tun. Die Tabelle kümmert sich um alles, sofern Sie die Methode `commitEditingStyle` bereitgestellt haben.

Zum Löschen fahren die Benutzer mit dem Finger leicht von links nach rechts über die Zelle. Die rechteckige Bestätigungsschaltfläche erscheint rechts von der Zelle, aber auf der linken Seite erscheinen *keine* runden Steuerelemente zum Entfernen.

Nachdem Benutzer eine Zelle weggestrichen und den Vorgang bestätigt haben, verarbeitet die Methode `tableView: commitEditingStyle: forRowAtIndexPath:` die Datenaktualisierungen so wie den Bearbeitungsmodus.

11.12.5 Zellen hinzufügen

Rezept 11.12 hat auch eine Schaltfläche zum Hinzufügen, für die das Systemschaltflächenelement mit dem Pluszeichen verwendet wird (oben links in *Abbildung 11.12*). Mit dieser Schaltfläche können die Benutzer neue Tabellenzellen anlegen. Dazu hängt die Methode `addItem:` einen neuen Zellentitel am Ende des Arrays `items` an und weist die Tabelle an, die Datenquelle mit `reloadData` zu aktualisieren. Dadurch kann der normale Tabellenmechanismus die Daten überprüfen und die Tabellenansicht aufgrund der aktualisierten Datenquellen wiederherstellen.

```
@implementation TableListViewController
@synthesize count;
@synthesize items;

- (NSInteger)numberOfSectionsInTableView:(UITableView *)aTableView
{
   return 1;
}

- (NSInteger)tableView:(UITableView *)aTableView
   numberOfRowsInSection:(NSInteger)section
{
   return self.items.count;
}

- (void) setBarButtonItems
{
   // Zeigt stets die Schaltfläche zum Hinzufügen (+) an
   self.navigationItem.leftBarButtonItem =
      SYSBARBUTTON(UIBarButtonSystemItemAdd, @selector(addItem:));

   // Zeigt beim Bearbeiten die Schaltfläche Done an.
   // Wenn keine Bearbeitung stattfindet, wird Edit nur angezeigt, wenn
   // Elemente vorhanden sind.
   if (self.tableView.isEditing)
      self.navigationItem.rightBarButtonItem =
         SYSBARBUTTON(UIBarButtonSystemItemDone,
         @selector(leaveEditMode));
   else
      self.navigationItem.rightBarButtonItem = self.items.count ?
         SYSBARBUTTON(UIBarButtonSystemItemEdit,
         @selector(enterEditMode)) : nil;
}

- (UITableViewCell *)tableView:(UITableView *)tView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Gibt eine Zelle aus der Warteschlange zurück oder erstellt
   // eine neue
   UITableViewCellStyle style = UITableViewCellStyleDefault;
   UITableViewCell *cell = [tView
      dequeueReusableCellWithIdentifier:@"BaseCell"];
   if (!cell)
      cell = [[[UITableViewCell alloc] initWithStyle:style
         reuseIdentifier:@"BaseCell"] autorelease];
```

```
    cell.textLabel.text = [items objectAtIndex:indexPath.row];
    return cell;
}

- (void) updateItemAtIndexPath: (NSIndexPath *) indexPath
    withString: (NSString *) string
{
    // Es ist nicht möglich, ein nil-Element einzufügen. Die Übergabe von
    // nil entspricht einer Löschanforderung.
    if (!string)
        [self.items removeObjectAtIndex:indexPath.row];
    else
        [self.items insertObject:string atIndex:indexPath.row];

    [self.tableView reloadData];
    [self setBarButtonItems];
}

- (void) addItem: (id) sender
{
    // Fügt ein neues Element hinzu
    NSIndexPath *newPath = [NSIndexPath
        indexPathForRow:self.items.count inSection:0];
    NSString *newTitle = [NSString stringWithFormat:@"Item %d",
        count++];
    [self updateItemAtIndexPath:newPath withString:newTitle];
}

- (void)tableView:(UITableView *)aTableView
    commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
    forRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Löscht ein Element
    [self updateItemAtIndexPath:indexPath withString:nil];
}

-(void)enterEditMode
{
    [self.tableView deselectRowAtIndexPath:[self.tableView
        indexPathForSelectedRow] animated:YES];
    [self.tableView setEditing:YES animated:YES];
    [self setBarButtonItems];
}
```

```
-(void)leaveEditMode
{
    [self.tableView setEditing:NO animated:YES];
    [self setBarButtonItems];
}

- (void) loadView
{
    [super loadView];
    count = 1;
    self.items = [NSMutableArray array];
    [self setBarButtonItems];
}
@end
```

▶ *Rezept 11.12: Zellen im laufenden Betrieb löschen*

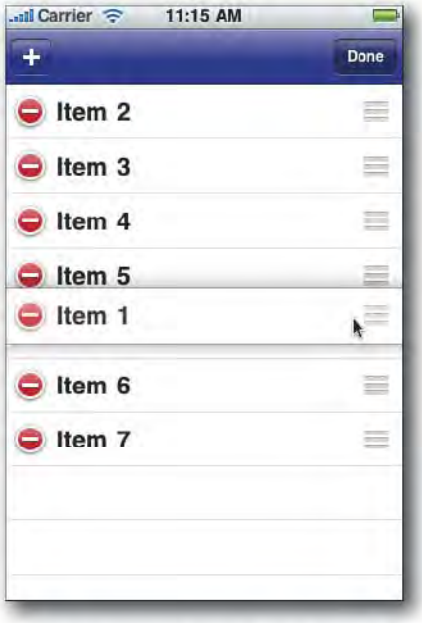
Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.13 Rezept: Zellen umordnen

Sie geben Ihren Benutzern deutlich mehr Möglichkeiten an die Hand, wenn Sie ihnen erlauben, die Zellen einer Tabelle direkt neu anzuordnen. *Abbildung 11.13* zeigt eine Tabelle mit dem Steuerelement zum Umordnen, das wie ein Stapel grauer Balken aussieht. Die Benutzer können diese Funktion anwenden, um anstehende Aufgaben nach Priorität zu sortieren oder um auszuwählen, welche Lieder zuerst in einer Wiedergabeliste stehen sollen. Das iPhone wird mit integrierter Unterstützung für die Umsortierung ausgeliefert, die Sie Ihren Programmen sehr einfach hinzufügen können. *Rezept 11.13* zeigt, wie das geht. Dazu müssen Sie nur eine einzige Tabellen-Delegierungsmethode hinzufügen.

Das interne Datenmodell muss unbedingt die Änderungen widerspiegeln, die der Benutzer in der Ansicht vornimmt. Implementieren Sie die Methode `tableView:moveRowAtIndexPath:toIndexPath`, um die Datenquelle mit den Änderungen auf dem Bildschirm zu synchronisieren. Dies geschieht auf die gleiche Weise wie die Übernahme der Bearbeitungsvorgänge beim Löschen von Zellen. In diesem Beispiel wird das Objekt anhand des Zellentitels im Array `items` an die neue Position verschoben.



▸ *Abbildung 11.13: Steuerelemente zur Neuanordnung erscheinen im Bearbeitungsmodus auf der rechten Seite jeder Zelle und werden als drei aufeinandergestapelte graue Linien dargestellt. In diesem Screenshot ist zu sehen, wie Element 1 unter Element 5 gezogen wird.*

Um die Neusortierung von Zellen zu ermöglichen, müssen Sie diese Methode in irgendeiner Form bereitstellen. Gibt es diese Methode nicht, zeigt die Tabelle im Bearbeitungsmodus keine Steuerelemente für die Neuanordnung an.

```
-(void) tableView: (UITableView *) tableView
   moveRowAtIndexPath: (NSIndexPath *) oldPath
   toIndexPath:(NSIndexPath *) newPath
{
   // Ändert die Datenreihenfolge als Reaktion auf eine Benutzeraktion
   NSString *title = [[self.items objectAtIndex:oldPath.row] retain];
   [self.items removeObjectAtIndex:oldPath.row];
   [self.items insertObject:title atIndex:newPath.row];
   [title release];

   [self setBarButtonItems];
}
```

▸ *Rezept 11.13: Tabellenzellen umsortieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.14 Rezept: Widerrufsmöglichkeiten zu Tabellen hinzufügen

Die in Kapitel 8, *Gesten und Berührungen*, eingeführte Klasse `NSUndoManager` bietet Widerrufs- und Wiederherstellungsmöglichkeiten für Cocoa Touch-Objekte. Bei Tabellen können Sie den gleichen grundlegenden Ansatz verwenden. Als Erstes erstellen und initialisieren Sie einen Undo-Manager für den Tabellenansichts-Controller.

```
self.undoManager = [[[NSUndoManager alloc] init] autorelease];
[self.undoManager setLevelsOfUndo:999];
```

Lassen Sie Ihren gesunden Menschenverstand walten, wenn Sie festlegen, wie viele Aktionen rückgängig gemacht werden können. Denken Sie daran, dass bei jedem Widerruf die als Argument übergebenen Objekte beibehalten werden. Wenn Sie es wie in diesem Beispiel nur mit kurzen Strings zu tun haben, können Sie eine sehr hohe Anzahl widerrufbarer Aktionen festlegen, da die Speicherbelastung nur minimal ist. Bei größeren Objekten dagegen sollten Sie die Anzahl stärker einschränken. Drei bis zehn widerrufbare Aktionen sollten für die meisten Bedürfnisse ausreichen.

11.14.1 Schüttelgesteuerte Bearbeitung ermöglichen

Wenn Sie schüttelgesteuerte Widerrufsmöglichkeiten anbieten möchten, deklarieren Sie dies in der Anwendungsdelegierung oder im Ansichtscontroller:

```
[[UIApplication sharedApplication]
   setApplicationSupportsShakeToEdit:YES];
```

Außerdem müssen Sie dem Controller die Möglichkeit einräumen, den Status als First Responder anzunehmen, sobald er auf dem Bildschirm erscheint. Die folgende Methode macht dies möglich:

```
- (BOOL)canBecomeFirstResponder {
   return YES;
}

// Wird zum First Responder, wenn die Ansicht erscheint
- (void)viewDidAppear:(BOOL)animated {
   [super viewDidAppear:animated];
   [self becomeFirstResponder];
}

// Gibt den First-Responder-Status auf, wenn die Ansicht verschwindet
- (void)viewWillDisappear:(BOOL)animated {
   [super viewWillDisappear:animated];
   [self resignFirstResponder];
}
```

Sollten Sie die schüttelgesteuerte Bearbeitung für Tabellen anbieten? Die Benutzer müssen daran denken, dass es diese Funktion gibt, an die die meisten nicht gewöhnt sind. Anschließend müssen Sie das Gerät schütteln, darauf warten, dass eine Benachrichtigung erscheint, und ein Menüelement auswählen. Das bedeutet viel Aufwand, und vor allem dann, wenn mehrere Aktionen widerrufen oder wiederhergestellt werden müssen, kann dies dazu führen, dass die Benutzer Ihre Anwendung nicht mehr nutzen wollen. Durch die Anzeige von Schaltflächen zum Widerrufen und Wiederstellen wie in *Rezept 11.14* bieten Sie eine offensichtliche Möglichkeit, um sich vorwärts und rückwärts durch den Bearbeitungsverlauf zu bewegen, was eher zur Zufriedenheit der Benutzer beiträgt.

11.14.2 Schaltflächen zum Widerrufen und Wiederherstellen hinzufügen

Cocoa Touch bietet zwei Systemschaltflächenelemente zum Widerrufen und Wiederherstellen. Der folgende Code passt die Methode `setBarButtonItems` aus *Rezept 11.12* an und fügt der Navigationsleiste eine eigene Symbolleiste hinzu. Die Schaltflächen **Undo** und **Redo** erscheinen nur, wenn der Undo-Manager die betreffenden Aktionen ausführen kann. Die Leiste ist an beiden Enden mit flexiblen Abstandshaltern ausgepolstert. Wenn Widerruf bzw. Wiederherstellung nicht möglich sind, nimmt ein Element mit fester Breite den Platz der betreffenden Schaltfläche ein.

```
- (void) setBarButtonItems
{
   // Fügt eine Hinzufügen-Schaltfläche hinzu
   self.navigationItem.leftBarButtonItem =
      SYSBARBUTTON(UIBarButtonSystemItemAdd, @selector(addItem:));

   // Zeigt entweder "Edit" oder "Done" an
   if (self.tableView.isEditing)
      self.navigationItem.rightBarButtonItem =
         SYSBARBUTTON(UIBarButtonSystemItemDone,
         @selector(leaveEditMode));
   else
      self.navigationItem.rightBarButtonItem = self.items.count ?
         SYSBARBUTTON(UIBarButtonSystemItemEdit,
         @selector(enterEditMode)) : nil;

   // Erstellt ein neues Array für Leistenelemente
   NSMutableArray *barItems = [NSMutableArray array];
   UIBarButtonItem *spacer =
      SYSBARBUTTON(UIBarButtonSystemItemFixedSpace, nil);
   spacer.width = 64;

   // Fügt einen Abstandshalter hinzu
   [barItems
      addObject:SYSBARBUTTON(UIBarButtonSystemItemFlexibleSpace,
      nil)];
```

```
    // Fügt eine Undo-Schaltfläche hinzu, wenn der Undo-Manager etwas
    // widerrufen kann
    if ([self.undoManager canUndo])
       [barItems addObject:SYSBARBUTTON(UIBarButtonSystemItemUndo,
          @selector(undo:))];
    else
       [barItems addObject:spacer];

    // Fügt einen Abstandshalter hinzu
    [barItems
       addObject:SYSBARBUTTON(UIBarButtonSystemItemFlexibleSpace,
       nil)];

    // Fügt eine Redo-Schaltfläche hinzu, wenn der Undo-Manager etwas
    // wiederherstellen kann
    if ([self.undoManager canRedo])
       [barItems addObject:SYSBARBUTTON(UIBarButtonSystemItemRedo,
       @selector(redo:))];
    else
       [barItems addObject:spacer];

    // Fügt einen Abstandshalter hinzu
    [barItems
       addObject:SYSBARBUTTON(UIBarButtonSystemItemFlexibleSpace,
       nil)];

    // Erstellt die Symbolleiste
    UIToolbar *tb = [[[UIToolbar alloc] initWithFrame:
       CGRectMake(0.0f, 0.0f, 200.0f, 48.0f)] autorelease];
    tb.barStyle = UIBarStyleBlack;
    tb.tintColor = COOKBOOK_PURPLE_COLOR;
    [tb setItems:barItems animated:YES];
    self.navigationItem.titleView = tb;
}
```

11.14.3 Aktionen widerrufen und wiederherstellen

Die eigentlichen Befehle zum Widerrufen und Wiederherstellen sind trivial. Die Hauptarbeit zur Bereitstellung von Widerrufsmöglichkeiten in Tabellen besteht darin, den Undo-Manager vorzubereiten, und nicht in der Ausführung der Befehle. In Kapitel 19, *Core Data*, finden Sie einen weiteren, einfacheren Ansatz zur Undo/Redo-Verwaltung.

```
- (void) undo: (id) sender
{
   // Macht das erste Element auf dem Undo-Stack rückgängig
   [self.undoManager undo];
   [self setBarButtonItems];
}

- (void) redo: (id) sender
{
   // Stellt das erste Element auf dem Redo-Stack wieder her
   [self.undoManager redo];
   [self setBarButtonItems];
}
```

11.14.4 Den Widerruf von Tabellenoperationen vorbereiten

Rezept 11.14 zeigt die Methoden der vier wichtigsten Tabellenoperationen mit Möglichkeiten für den Widerruf. Hierbei handelt es sich um die Methoden zum Hinzufügen und Löschen, die gemeinsame Aktualisierungsmethode, die sie aufrufen, sowie um die Methode, die sich um Umsortierungsvorgänge kümmert.

Wie ich in diesem Kapitel bereits erwähnt habe, sorgt die Hilfskonstruktion einer zweiten Methode zum Hinzufügen und Löschen dafür, dass für diese Operationen ein Widerruf möglich ist. Die Aktualisierungsmethode bereitet hier den Undo-Aufruf vor, um ein Element hinzuzufügen, das gelöscht werden sollte, bzw. umgekehrt. Durch diese Kombination kann der Undo-Manager an einer zentralen Stelle ansetzen.

Für die Sortiermethode, die Zeilen in neue Pfade verschiebt, gibt es sogar noch eine einfachere Lösung. Sie tauscht lediglich den neuen und den alten Indexpfad aus und verwendet diesen Aufruf für den Undo-Manager. Es gibt jedoch zwei mögliche Fallstricke bei der Verschiebemethode, auf die Sie achten müssen.

Erstens müssen Sie prüfen, ob bei einer Umsortierung überhaupt etwas verschoben wurde. Der Test `oldPath.row == newPath.row` sorgt dafür, dass solche »Verschiebungen, die gar keine sind«, nicht auf den Undo-Stack gelangen. Bei der normalen Umsortierung (ohne Widerrufsmöglichkeit) können Sie diesen Schritt auslassen, da solche »Nicht-Zeilentauschvorgänge« keine Auswirkungen auf das Verhalten der Benutzeroberfläche haben. Allerdings führen sie zu Problemen, wenn es einen Undo-Stack gibt. Die Benutzer können sich nicht vorstellen, warum der gerade angeforderte Widerruf keinerlei Wirkung zeigt, während die Anwendung tatsächlich etwas »widerrufen«, nämlich ein Element mit sich selbst ausgetauscht hat. Vermeiden Sie es, die Benutzer auf diese Weise zu verwirren, sondern halten Sie solche Elemente vom Undo-Stack fern.

Zweitens können Tabellenansichten erst dann neu geladen werden, wenn eine Verschiebeoperation abgeschlossen ist. Da der Benutzer etwas mit den Zellen tut und sie an eine neue Position zieht, müssen Sie die Tabelle normalerweise nicht von der Verschiebemethode aus neu laden. Tatsächlich könnte dies sogar zu einer Endlosschleife und dem Absturz der Anwendung führen.

Bei der Arbeit mit einem Undo-Manager muss sich die Tabelle jedoch auf irgendeine Weise aktualisieren können, um wieder dem aktualisierten Modell zu entsprechen. Sie müssen die Tabelle neu laden, sodass die Zellen die Daten nach dem Widerruf widerspiegeln. Wenn Sie wie in diesem Rezept einen verzögerten Selektor hinzufügen, kann die Verschiebemethode abgeschlossen werden, bevor die Tabelle neu geladen wird. Dadurch kann die Umsortierung ohne Absturz komplett erfolgen, während es gleichzeitig die Möglichkeit gibt, die Tabelle nach Undo- und Redo-Aufrufen zu aktualisieren.

```
- (void) updateItemAtIndexPath: (NSIndexPath *) indexPath
   withString: (NSString *) string
{
   // Tauscht den String für den Widerruf gegen nil aus oder umgekehrt
   NSString *undoString = string ? nil : [self.items
      objectAtIndex:indexPath.row];
   [[self.undoManager prepareWithInvocationTarget:self]
      updateItemAtIndexPath:indexPath withString:undoString];

   // Ein nil-Element kann nicht eingefügt werden. Die Übergabe von nil
   // entspricht einer Löschanforderung.
   if (!string)
      [self.items removeObjectAtIndex:indexPath.row];
   else
      [self.items insertObject:string atIndex:indexPath.row];

   [self.tableView reloadData];
   [self setBarButtonItems];
}

- (void) addItem: (id) sender
{
   // Fügt ein neues Element auf der Grundlage der aktuellen Anzahl hinzu
   NSIndexPath *newPath = [NSIndexPath
      indexPathForRow:self.items.count inSection:0];
   NSString *newTitle = [NSString stringWithFormat:@"Item %d",
      count++];
   [self updateItemAtIndexPath:newPath withString:newTitle];
}

- (void)tableView:(UITableView *)aTableView
   commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
   forRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Löscht ein Element
   [self updateItemAtIndexPath:indexPath withString:nil];
}
```

```
-(void) tableView: (UITableView *) tableView
   moveRowAtIndexPath: (NSIndexPath *) oldPath
   toIndexPath:(NSIndexPath *) newPath
{
   // Erfasst und ignoriert alle Nicht-Verschiebungen
   if (oldPath.row == newPath.row) return;

   // Bereitet den Widerruf eines Zeilentauschs vor
   [[self.undoManager prepareWithInvocationTarget:self]
      tableView:self.tableView moveRowAtIndexPath:newPath
      toIndexPath:oldPath];

   // Führt den Austausch durch
   NSString *item = [[self.items objectAtIndex:oldPath.row] retain];
   [self.items removeObjectAtIndex:oldPath.row];
   [self.items insertObject:item atIndex:newPath.row];
   [item release];

   // Aktualisiert die Leistenschaltflächenelemente und lädt die Daten neu
   [self setBarButtonItems];
   [self.tableView performSelector:@selector(reloadData)
      withObject:nil afterDelay:0.25f];
}
```

▸ *Rezept 11.14: Undo-Elemente für Tabellenoperationen vorbereiten*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.15 Tabellen sortieren

Eine Tabelle ist eine Datenquelle. Wenn Sie die Informationen in einer Tabelle sortieren und die Daten neu laden, erhalten Sie eine sortierte Tabelle. *Rezept 11.15* zeigt eine Methode für einen Tabellenansichtscontroller, die auf Anforderung eine Sortierung durchführt.

In diesem Rezept werden drei Sortierreihenfolgen abgedeckt: alphabetisch aufsteigend, alphabetisch absteigend und nach Stringlänge. Für die beiden letzteren Möglichkeiten ist eine Erweiterung der Klasse `NSString` erforderlich. Diese einfache Klassenkategorie verfügt zusätzlich über einen umgekehrten Vergleich und einen Stringlängenvergleich.

Rezept 11.15 zeigt nur einen Teil der Implementierung. Es veranschaulicht, wie das Datenmodell auf die verschiedenen Sortierungsarten reagiert. Wenn der Benutzer auf eine Segmentschaltfläche tippt, wird das Array `items` durch eine Version mit der ausgewählten Sortierung ersetzt. In *Kapitel 19, Core Data*, finden Sie einen Ansatz unter Zuhilfenahme von Core Data. Dabei erfolgt eine Sortierung, während die Ergebnisse aus einem dauerhaften Datenspeicher abgerufen werden.

```
@implementation NSString (sortingExtension)
- (NSComparisonResult) reverseCompare: (NSString *) aString
{
   // Kehrt den normalen Vergleich ohne Berücksichtigung von Groß- und
   // Kleinschreibung um
   return -1 * [self caseInsensitiveCompare:aString];
}

- (NSComparisonResult) lengthCompare: (NSString *) aString
{
   // Gibt eine Sortierung aufgrund der Stringlänge zurück
   if (self.length == aString.length) return NSOrderedSame;
   if (self.length > aString.length) return NSOrderedDescending;
   return NSOrderedAscending;
}
@end

@implementation TableListViewController
- (void) updateSort: (UISegmentedControl *) seg
{
   // Wendet die ausgewählte Sortierung auf die Tabellenelemente an
   if (seg.selectedSegmentIndex == 0)
      self.items = [self.items sortedArrayUsingSelector:
            @selector(caseInsensitiveCompare:)];
   else if (seg.selectedSegmentIndex == 1)
      self.items = [self.items sortedArrayUsingSelector:
         @selector(reverseCompare:)];
   else if (seg.selectedSegmentIndex == 2)
      self.items = [self.items sortedArrayUsingSelector:
         @selector(lengthCompare:)];

   [self.tableView reloadData];
}
@end
```

▸ *Rezept 11.15: Eine UITableView sortieren*

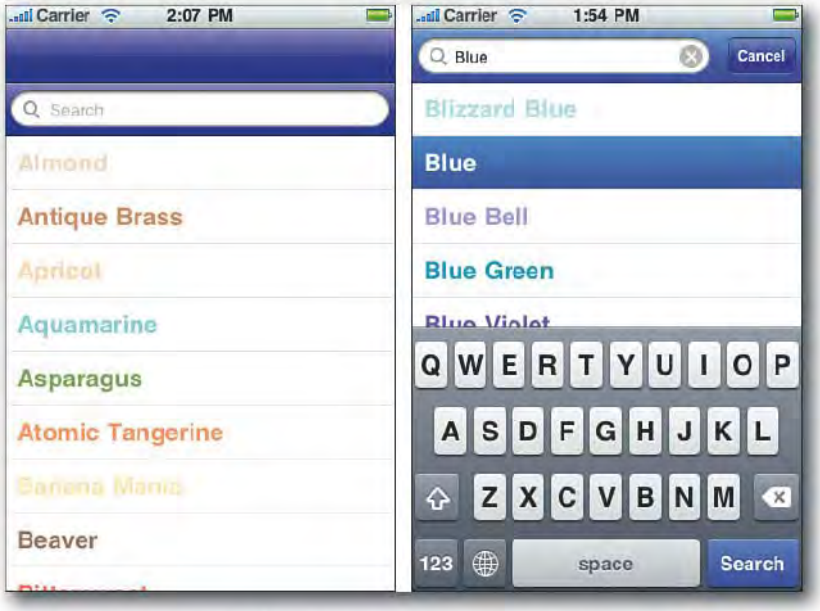
Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.16 Rezept: Tabellen durchsuchen

Mit der neuen eingebauten Suchfunktion ab SDK 3.0 können die Benutzer die Inhalte einer Tabelle in Echtzeit filtern. Für diese Suche werden zwei wichtige Klassen verwendet, die bereits früher verfügbare `UISearchBar` und der neue `UISearchDisplayController`. Zusammen ahmen sie das Verhalten der Anwendung *Kontakte* nach, in der sich am Kopf der Tabelle eine Suchleiste befindet.

Um die Suchleiste zu finden, müssen Sie in der Tabelle bis ganz nach oben blättern, wie Sie in *Abbildung 11.14* (links) sehen. Die Suchleiste befindet sich zu Anfang nicht in der Navigationsleiste. Sobald der Benutzer auf das Suchfeld tippt, ändert sich die Ansicht, und die Suchleiste wird in den Navigationsbereich verschoben, wie *Abbildung 11.14* (rechts) zeigt. Dort bleibt sie, bis der Benutzer auf **Cancel** tippt, woraufhin wieder die ungefilterte Tabellenansicht erscheint.



▸ *Abbildung 11.14: Um eine Suche auszulösen, muss der Benutzer zunächst bis ganz nach oben blättern. Die Suchleiste erscheint als erstes Element in der Headeransicht der Tabelle (links). Sobald der Benutzer auf die Suchleiste tippt und sie damit aktiviert, springt sie in die Navigationsleiste, um eine anhand der Suchkriterien gefilterte Liste der Elemente darzustellen (rechts).*

11.16.1 Den Suchanzeigecontroller erstellen

Suchanzeigecontroller verwalten die Anzeige von Daten eines anderen Controllers, in diesem Fall des standardmäßigen `UITableViewControllers`. Der Suchanzeigecontroller zeigt eine Untermenge der Daten an, indem er die Datenquelle gewöhnlich durch ein Prädikat filtert.

Zur Initialisierung versorgen Sie den Suchanzeigecontroller mit einer Suchleiste und einem Inhaltscontroller. Der im Folgenden gezeigte Suchanzeigecontroller verwendet eine standardmäßige Suchleiste, die mit dem folgenden Code erstellt wurde. Als Inhaltscontroller übergeben Sie die Hauptinstanz von `UITableViewController`, in der Sie die Elemente definieren.

Legen Sie die Textfeldaspekte der Suchleiste wie üblich fest, aber richten Sie keine Delegierung ein. Die Suchleiste arbeitet mit dem Suchanzeigecontroller ohne ausdrückliche Delegierung von Ihrer Seite zusammen.

Beim Einrichten des Suchergebniscontrollers müssen Sie wie hier gezeigt sowohl die Datenquelle für die Suchergebnisse als auch die Delegierung festlegen. Beide zeigen auf die Unterklasse des primären Tabellenansichtscontrollers zurück, in der Sie die normalen Datenquellen- und Delegierungsmethoden an die Erfordernisse der durchsuchbaren Tabelle anpassen.

```
// Erstellt eine Suchleiste
self.searchBar = [[[UISearchBar alloc] initWithFrame:
   CGRectMake(0.0f, 0.0f, 320.0f, 44.0f)] autorelease];
self.searchBar.tintColor = COOKBOOK_PURPLE_COLOR;
self.searchBar.autocorrectionType = UITextAutocorrectionTypeNo;
self.searchBar.autocapitalizationType =
   UITextAutocapitalizationTypeNone;
self.searchBar.keyboardType = UIKeyboardTypeAlphabet;
self.tableView.tableHeaderView = self.searchBar;

// Erstellt den Suchanzeigecontroller
self.searchDC = [[[UISearchDisplayController alloc]
   initWithSearchBar:self.searchBar contentsController:self]
   autorelease];
self.searchDC.searchResultsDataSource = self;
self.searchDC.searchResultsDelegate = self;
```

11.16.2 Methoden für durchsuchbare Datenquellen erstellen

Die Anzahl der in der Tabelle angezeigten Elemente wechselt mit dem Suchvorgang, weshalb Sie jeweils die korrekte Zeilenanzahl melden müssen. Um herauszufinden, ob zurzeit der Tabellenansichts- oder der Suchanzeigecontroller verantwortlich ist, vergleichen Sie die Tabellenansichtsparameter mit der integrierten Eigenschaft `tableView`. Sind sie identisch, haben Sie es mit der normalen Tabellenansicht zu tun, andernfalls ist der Suchanzeigecontroller in Aktion und verwendet seine eigene Tabellenansicht. Passen Sie die Zeilenzahl entsprechend an.

```
- (NSInteger)tableView:(UITableView *)aTableView
   numberOfRowsInSection:(NSInteger)section
{
   // Gibt die Zeilenanzahl der normalen Tabelle zurück
   if (aTableView == self.tableView)
      return self.crayonColors.allKeys.count;

   // Gibt die Zeilenanzahl der Suchtabelle zurück
   NSPredicate *predicate = [NSPredicate predicateWithFormat:
      @"SELF contains[cd] %@", self.searchBar.text];
   self.filteredArray = [self.crayonColors.allKeys
      filteredArrayUsingPredicate:predicate];
   return self.filteredArray.count;
}
```

Um die Anzahl der Elemente zu melden, die mit dem Text im Suchfeld übereinstimmen, verwenden Sie ein Prädikat. Prädikate bilden eine äußerst einfache Möglichkeit, um ein Array zu filtern und nur die Elemente zurückzugeben, die mit einem Suchstring übereinstimmen. Jeder String, der den Text im Suchfeld enthält, bildet eine Übereinstimmung, sodass er im gefilterten Array verbleibt. Alternativ können Sie auch `beginswith` verwenden, damit nur Elemente gefunden werden, die mit dem gesuchten Text anfangen.

Mit Prädikaten ist weit mehr möglich als der hier gezeigte einfache Stringvergleich. Sie können sie für alle möglichen Arten von komplexen Objekten einsetzen, um anspruchsvolle Filterfunktionen für die Anzeige von Tabellen bereitzustellen. Dies gilt z. B. auch für Core Data-Objekte.

Zum Filtern ist mehr erforderlich, als nur die Zeilenanzahl zu melden. Sie müssen die Datenquelle filtern, um Zellinstanzen zu füllen und zurückzugeben. Die folgende Methode untersucht die aktuelle Tabellenansicht und gibt die Zellen zurück, die entweder dem Standardschlüssel oder der gefilterten Menge entsprechen.

```
- (UITableViewCell *)tableView:(UITableView *)aTableView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Entnimmt eine Zelle aus der Warteschlange oder erstellt eine neue
   UITableViewCellStyle style = UITableViewCellStyleDefault;
   UITableViewCell *cell = [aTableView
      dequeueReusableCellWithIdentifier:@"BaseCell"];
   if (!cell) cell = [[[UITableViewCell alloc] initWithStyle:style
      reuseIdentifier:@"BaseCell"] autorelease];

   // Ruft den Buntstift und dessen Farbe ab
   NSArray *keyCollection = (aTableView == self.tableView) ?
      DEFAULTKEYS : FILTEREDKEYS;
   NSString *crayon = [keyCollection objectAtIndex:indexPath.row];
   cell.textLabel.text = crayon;
```

```
    if (![crayon hasPrefix:@"White"])
        cell.textLabel.textColor = [self.crayonColors
            objectForKey:crayon];
    else
        cell.textLabel.textColor = [UIColor blackColor];
    return cell;
}
```

11.16.3 Delegierungsmethoden

Nicht nur die Datenquellen müssen für die Suche eingerichtet sein. Wie *Rezept 11.16* zeigt, ist es entscheidend, den Kontext einer Benutzerberührung zu bestimmen, um in den Delegierungsmethoden für die richtige Reaktion zu sorgen. Wie die zuvor gezeigten Datenquellenmethoden vergleicht auch diese Delegierungsmethode die mit dem Callback gesendeten Tabellenansichtsparameter mit den integrierten Parametern. Nach dem Ergebnis dieses Vergleichs entscheidet die Methode, wie sie reagieren soll. In diesem Fall geht es darum, sowohl die Such- als auch die Navigationsleiste mit der zurzeit ausgewählten Farbe zu versehen.

```
// Reagiert durch Farbänderung auf die Benutzerauswahl
- (void)tableView:(UITableView *)aTableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    NSArray *keyCollection = (aTableView == self.tableView) ?
        DEFAULTKEYS : FILTEREDKEYS;
    NSString *crayon = [keyCollection objectAtIndex:indexPath.row];
    self.navigationController.navigationBar.tintColor =
        [self.crayonColors objectForKey:crayon];
    self.searchBar.tintColor = [self.crayonColors objectForKey:crayon];
}
```

► *Rezept 11.16: Tabellenansichten vergleichen, um korrekt auf Benutzereingaben zu reagieren*

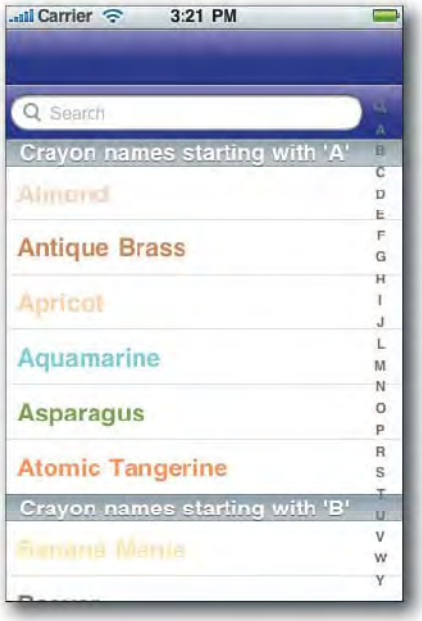
Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.17 Rezept: Tabellen in Abschnitte unterteilen

In vielen iPhone-Anwendungen werden nicht nur Zeilen, sondern auch Abschnitte verwendet, um Listen noch mehr Struktur zu geben und Elemente logisch zu gruppieren. Die am häufigsten verwendete Abschnittseinteilung ist das Alphabet, allerdings können Sie Daten natürlich auch auf andere Weise gliedern. Es steht Ihnen frei, jedes Gliederungsschema zu verwenden, das für Ihre Anwendung sinnvoll ist.

Abbildung 11.15 zeigt eine Tabelle, in der Abschnitte verwendet werden, um gruppierte Namen anzuzeigen. Jeder Abschnitt hat eine eigene Überschrift (z. B. »Crayon names starting with...«, also »Buntstiftnamen mit dem Anfangsbuchstaben...«), und ein Index auf der rechten Seite ermöglicht einen schnellen Zugriff auf die einzelnen Abschnitte. Wie Sie sehen, werden in diesem Index keine Abschnitte für K, Q, X und Z aufgeführt. In dem Rezept werden leere Abschnitte aus dem Index entfernt.



► *Abbildung 11.15: In Tabellen mit Abschnitten können Sie sowohl Überschriften als auch einen Index anzeigen, damit die Benutzer die gewünschten Informationen so schnell wie möglich finden können.*

11.17.1 Eine Datenstruktur mit Abschnitten erstellen

Wenn Sie mit Gruppen und Abschnitten arbeiten, müssen Sie zweidimensional denken. Mit einem Abschnittsarray können Sie die Einträge der einzelnen Abschnitte speichern und auf sie zugreifen. Erstellen Sie dazu ein Array aus Arrays. Das Abschnittsarray speichert für jeden Abschnitt ein Array, das die Titel der einzelnen Zelle enthält. Der folgende Code erstellt die Abschnittsarrays und füllt sie, indem er jeweils den ersten Buchstabens eines Elementnamens heranzieht und dessen Position im Alphabet bestimmt.

```
// Erstellt das Abschnittsarray
self.sectionArray = [NSMutableArray array];
for (int i = 0; i < 26; i++)

{

   [self.sectionArray  addObject:[NSMutableArray array]];

}
```

```
// Füllt die Arrays nach dem Anfangsbuchstaben
for (NSString *string in rawCrayons)
{

   [self.crayonColors setObject:CRAYON_COLOR(string)

      forKey:CRAYON_NAME(string)];
   NSUInteger firstLetter = [ALPHA rangeOfString:[string
      substringToIndex:1]].location;
   if (firstLetter != NSNotFound) [[self.sectionArray
      objectAtIndex:firstLetter] addObject:CRAYON_NAME(string)];
}
```

Diese Implementierung stützt sich auf zwei Dinge: Zunächst müssen die Wörter bereits sortiert sein, denn in jedem Unterabschnitt werden die Wörter in der Reihenfolge hinzugefügt, in der sie im Array vorliegen. Zweitens müssen die Wörter mit den Abschnitten übereinstimmen. Einträge, die mit Punkten oder Nummern anfangen, führen hierbei zu Fehlern. Sie können einfach den Abschnitt »Andere« hinzufügen, in dem solche Namen untergebracht werden, was in diesem einfachem Beispiel allerdings nicht vorkommt.

Alphabetisch sortierte Abschnitte sind zwar hilfreich und stellen wie bereits erwähnt wahrscheinlich die häufigste Art der Gruppierung dar, doch können Sie eine beliebige Gruppierungsstruktur verwenden. Ordnen Sie beispielsweise Personen nach Abteilungen, Edelsteine nach Güteklassen oder Termine nach Datum. Unabhängig von der Art der Gruppierung ist ein Array aus Arrays die Datenquelle für Tabellenansichten, die sich am besten für unterteilte Tabellen eignet.

11.17.2 Abschnitte und Zeilen zählen

Für unterteilte Tabellen müssen zwei wichtige Datenquellenmethoden angepasst werden:

- `numberOfSectionsInTableView` Diese Methode legt fest, wie viele Abschnitte in der Tabelle erscheinen, und bestimmt die Anzahl der angezeigten Gruppen. Wenn Sie – wie hier empfohlen – ein Abschnittsarray verwenden, geben Sie die Anzahl der Elemente darin zurück (also `[mySectionArray count]`). Sofern Sie die Anzahl der Elemente bereits im Voraus kennen (in diesem Beispiel 26), können Sie den Betrag hartkodieren.
- `tableView:numberOfRowsInSection` Diese Methode wird mit einer Abschnittsnummer aufgerufen. Legen Sie fest, wie viele Zeilen in diesem Abschnitt erscheinen. Geben Sie bei der empfohlenen Datenstruktur nur die Anzahl der Elemente des *n*-ten Unterarrays zurück: `[[mySectionArray objectAtIndex: sectionNumber] count]`.

Diese Methoden erweitern die in *Rezept 11.16* vorgestellte durchsuchbare Tabelle. Wie *Abbildung 11.14* zeigt, sind unterteile Tabellen und ihre Indizes mit Suchfunktionieren kompatibel. Das kleine Suchsymbol oben im Index führt die Benutzer zur Suchleiste oben in der Tabelle. In diesem Beispiel sind die Suchergebnisse nicht in Abschnitte unterteilt, weshalb als Anzahl der Abschnitte 1 statt 26 zurückgegeben wird.

```
- (NSInteger)numberOfSectionsInTableView:(UITableView *)aTableView
{
   // Abschnittsanzahl der normalen Tabelle
   if (aTableView == self.tableView) return 26;

   // Abschnittsanzahl der Suchtabelle
   return 1;
}

- (NSInteger)tableView:(UITableView *)aTableView
   numberOfRowsInSection:(NSInteger)section
{
   // Zellenanzahl der normalen Tabelle
   if (aTableView == self.tableView) return [[self.sectionArray
      objectAtIndex:section] count];

   // Zellenanzahl der Suchtabelle
   NSPredicate *predicate = [NSPredicate predicateWithFormat:

      @"SELF contains[cd] %@", self.searchBar.text];
   self.filteredArray = [self.crayonColors.allKeys
      filteredArrayUsingPredicate:predicate];
   return self.filteredArray.count;
}
```

11.17.3 Zellen zurückgeben

In unterteilten Tabellen werden sowohl die Angaben über die Zeilen als auch die Angaben über die Abschnitte herangezogen, um Zellendaten zu finden. In den Rezepten weiter vorn in diesem Kapitel wurde ein lineares Array mit einem Zeilennummernindex verwendet. Bei Tabellen mit Abschnitten ist jedoch der gesamte Indexpfad erforderlich, um in einer Zelle sowohl den Abschnitts- als auch den Zeilenindex zu finden.

```
- (UITableViewCell *)tableView:(UITableView *)aTableView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Entnimmt eine Zelle der Warteschlange oder erstellt eine neue
   UITableViewCellStyle style = UITableViewCellStyleDefault;
   UITableViewCell *cell = [aTableView
      dequeueReusableCellWithIdentifier:@"BaseCell"];
   if (!cell) cell = [[[UITableViewCell alloc] initWithStyle:style
      reuseIdentifier:@"BaseCell"] autorelease];

   NSString *crayon;
```

```
    // Ruft den Buntstift und seine Farbe ab
    if (aTableView == self.tableView)
        crayon = [[self.sectionArray objectAtIndex:indexPath.section]
            objectAtIndex:indexPath.row];
    else
        crayon = [FILTEREDKEYS objectAtIndex:indexPath.row];

    // Legt den Zellentext fest und färbt ihn
    cell.textLabel.text = crayon;
    if (![crayon hasPrefix:@"White"])
        cell.textLabel.textColor = [self.crayonColors
            objectForKey:crayon];
    else
        cell.textLabel.textColor = [UIColor blackColor];
    return cell;
}
```

11.17.4 Titel für Überschriften erstellen

Es ist nur sehr wenig Arbeit notwendig, um Abschnittsüberschriften zu einer gruppierten Tabelle hinzuzufügen. Die optionale Methode `tableView:titleForHeaderInSection:` versieht jeden Abschnitt mit einer Überschrift. Ihr wird ein Integer übergeben, und als Rückgabewert geben Sie einen Titel an. Befinden sich in einem gegebenen Abschnitt der Tabelle keine Elemente oder gibt es nur einen Abschnitt (z. B. in der Suchtabelle), wird `nil` zurückgegeben.

```
- (NSString *)tableView:(UITableView *)aTableView
    titleForHeaderInSection:(NSInteger)section
{
    // Normale Tabelle
    if (aTableView == self.tableView)
    {
        // Leere Abschnitte
        if ([[self.sectionArray objectAtIndex:section] count] == 0)
            return nil;

        // Gefüllte Abschnitte
        return [NSString stringWithFormat:
            @"Crayon names starting with '%@'",
            [[ALPHA substringFromIndex:section] substringToIndex:1]];
    }
    else
        // Suchtabelle
        return nil;
}
```

11.17.5 Einen Abschnittsindex erstellen

Tabellen, die `sectionIndexTitelsForTabelView:` implementieren, zeigen die Art von Indexansicht an, die Sie rechts in *Abbildung 11.14* sehen. Diese Methode wird beim Erstellen der Tabellenansicht aufgerufen, und das zurückgegebene Array bestimmt, welche Elemente auf dem Bildschirm dargestellt werden. Geben Sie `nil` zurück, damit kein Index angezeigt wird, wie es hier für die Suchtabelle geschieht. Apple empfiehlt, Abschnittsindizes nur zu einfachen Tabellenansichten hinzuzufügen, also zu solchen, die mit dem standardmäßigen einfachen Format `UITableViewStylePlain` erstellt wurden. Ein (etwas verunglücktes) Beispiel für eine gruppierte Tabelle mit Abschnittsindex sehen Sie in *Abbildung 11.16*.

```
- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)aTableView
{
   if (aTableView == self.tableView) // Normale Tabelle
   {
      NSMutableArray *indices = [NSMutableArray
         arrayWithObject:UITableViewIndexSearch];
      for (int i = 0; i < 26; i++)
         if ([[self.sectionArray objectAtIndex:i] count])
            [indices addObject:[[ALPHA substringFromIndex:i]
               substringToIndex:1]];
      return indices;
   }
   else return nil; // Suchtabelle
}
```

Das erste Element, das diesem Index hinzugefügt wird, ist die Konstante `UITableViewIndexSearch`. Dadurch wird das kleine Vergrößerungsglas angezeigt, das darauf hinweist, dass die Tabelle durchsucht werden kann, und das den Benutzer zum Kopf der Liste führt.

In diesem Beispiel werden zwar nur Titel aus einem einzigen Buchstaben verwendet, doch ist dies keine allgemeine Einschränkung: Sie können auch ganze Wörter benutzen. Wenn Sie sich die Mühe machen, die Unicode-Darstellung herauszufinden, können Sie auch Bilder wie Emoji-Elemente (für iPhone-Benutzer in Japan) verwenden, die zur iPhone-Zeichenbibliothek gehören.

```
[indices addObject:@"\ue057"];
```

11.17.6 Abweichungen im Index beheben

Indizes führen die Benutzer an eine andere Stelle der Tabelle. Grundlage dafür ist ein durch Antippen gewählter Versatz. Wie ich bereits erwähnt habe, werden in dieser Tabelle keine Abschnitte für K, Q, X und Z angezeigt. Deren Fehlen kann zu einer Abweichung der angezeigten Ergebnisse von der Benutzerauswahl führen.

Um dieses Problem zu lösen, implementieren Sie die optionale Methode `tableView:sectionForSectionIndexTitel:`. Deren Aufgabe besteht darin, den Indextitel eines Abschnitts (den die Methode `sectionIndexTitlesForTableView:` zurückgibt) mit einer Abschnittsnummer zu verknüpfen. Dadurch werden jegliche mögliche Abweichungen überschrieben, sodass die Indexauswahl des Benutzers genau mit dem angezeigten Abschnitt übereinstimmt.

```
- (NSInteger)tableView:(UITableView *)tableView
   sectionForSectionIndexTitle:(NSString *)title
   atIndex:(NSInteger)index
{
   if (title == UITableViewIndexSearch)
   {
      // Sorgt für den Sprung zur Suchleiste
      [self.tableView scrollRectToVisible:self.searchBar.frame
         animated:NO];
      return -1;
   }

   // Gibt den Abschnitt zu einem Buchstaben zurück
   return [ALPHA rangeOfString:title].location;
}
```

Der hier verwendete Aufruf von `scrollRectToVisible:animated:` verschiebt die Suchleiste manuell an Ort und Stelle, wenn der Benutzer auf das Vergrößerungsglas tippt. Anderenfalls müsste der Benutzer von Abschnitt O aus zurückblättern, also dem Abschnitt für den Buchstaben A.

11.17.7 Delegierungen in unterteilten Tabellen

Wie bei Datenquellenmethoden besteht der Trick zur Implementierung von Delegierungsmethoden in unterteilten Tabellen darin, die Eigenschaften `section` und `row` des Indexpfades zu verwenden. Diese Eigenschaften ermöglichen den doppelten Zugriff, der erforderlich ist, um erst das richtige Abschnittsarray und dann darin das gewünschte Element zu finden. *Rezept 11.17* zeigt, wie Sie die Such- und die Navigationsleiste aktualisieren, wenn Sie die durch Berührung einer unterteilten Tabelle ausgewählte Farbe abrufen.

```
// Reagiert durch Aktualisierung der Farbe auf die Benutzerauswahl
- (void)tableView:(UITableView *)aTableView
   didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Bestimmt, welcher Buntstift ausgewählt wurde
   NSString *crayon;
   if (aTableView == self.tableView)
      crayon = [[self.sectionArray objectAtIndex:indexPath.section]
         objectAtIndex:indexPath.row];
   else
```

```
        crayon = [FILTEREDKEYS objectAtIndex:indexPath.row];

    // Aktualisiert die Farbe der Navigations- und der Suchleiste
    self.navigationController.navigationBar.tintColor =
        [self.crayonColors objectForKey:crayon];
    self.searchBar.tintColor = [self.crayonColors objectForKey:crayon];
}
```

► *Rezept 11.17: In einer unterteilten Tabelle auf Benutzerberührungen reagieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.18 Rezept: Gruppierte Tabellen erstellen

Auf dem iPhone gibt es Tabellen in zwei Formaten: gruppierte Tabellen und einfache Tabellenlisten. Letztere haben Sie bereits kennengelernt, da sich die Rezepte in diesem Kapitel darauf konzentrierten, wie sie erstellt werden. Gruppierte Listen finden Sie im Programm *Einstellungen* des iPhones. Diese Listen werden auf einem blaugrauen Hintergrund dargestellt, wobei jeder Unterabschnitt in einem leicht abgerundeten Rechteck erscheint. *Abbildung 11.16* zeigt die von *Rezept 11.18* erstellte gruppierte Liste.

Um das Format zu ändern, müssen Sie nicht mehr tun, als den Tabellenansichtscontroller mit einem anderen Stil zu initialisieren. Das können Sie ausdrücklich beim Erstellen einer neuen Instanz erledigen:

```
myTableViewController = [[UITableViewController alloc]
    initWithStyle:UITableViewStyleGrouped];
```

Sie können aber auch den Ansatz aus *Rezept 11.18* verwenden. Durch Überschreiben der Methode `init` stellen Sie sicher, dass die neuen Instanzen dieser Unterklasse eine Tabelle im gruppierten Format hervorrufen.

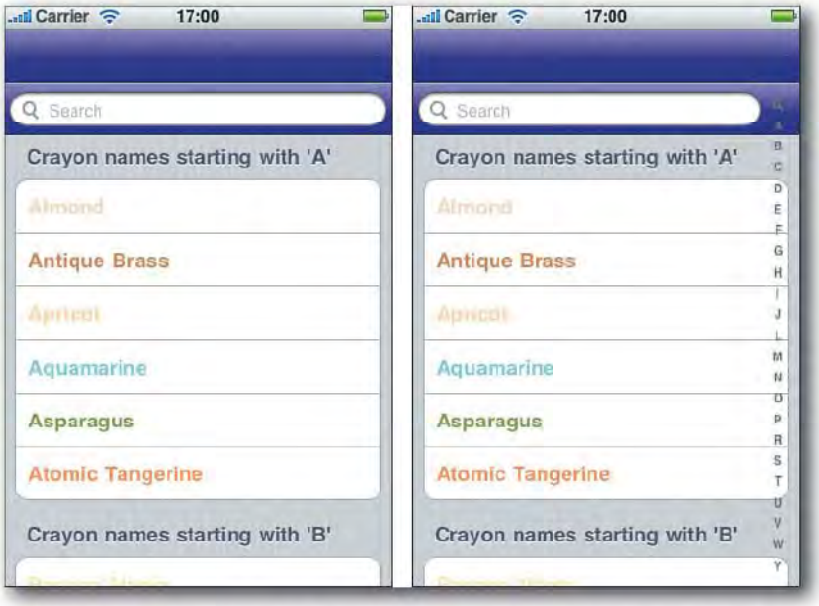
```
- (TableListViewController *) init
{
    // Legt das gruppierte Format fest
    self = [super initWithStyle:UITableViewStyleGrouped];
    return self;
}
```

► *Rezept 11.18: Das gruppierte Format durch Überschreiben der Methode* `init` *des Tabellenansichtscontrollers festlegen*

Apple rät davon ab, hier einen Abschnittsindex wie den in *Abbildung 11.16* (rechts) zu verwenden, da er über dem rechten Rand der gruppierten Zellen verläuft, was zu einer unschönen Darstellung führt.

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.



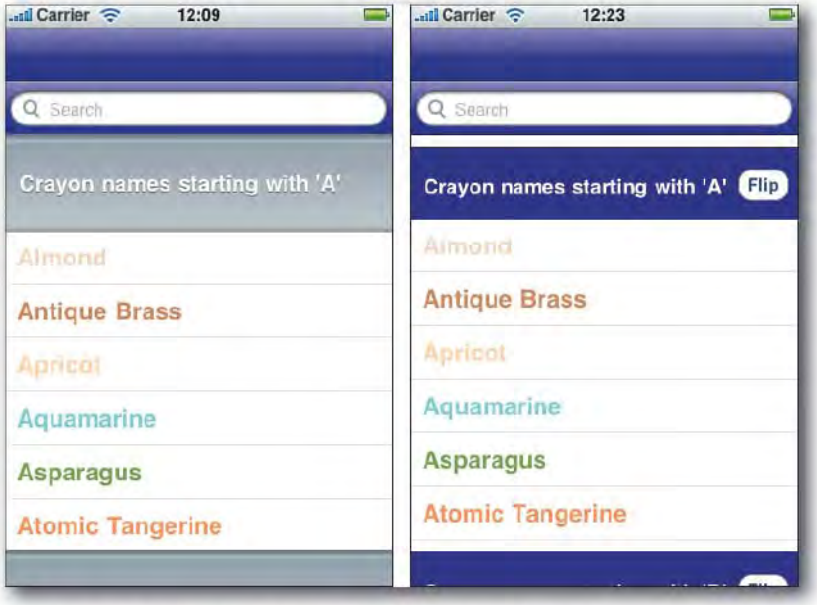
▸ *Abbildung 11.16: Gruppierte Tabellen zeigen eine andere Darstellung als Standardtabellen (links). Apple rät davon ab, in gruppierten Tabellen einen Abschnittsindex zu verwenden. Wie Sie sehen, überschreitet der Index die Zellenbegrenzungen (rechts).*

11.19 Rezept: Header und Footer anpassen

In Abschnitte unterteilte Tabellen lassen sich auf vielfältige Weise anpassen. Sie haben bereits gelesen, wie Sie ein `UISearchBar`-Suchfeld mithilfe der Eigenschaft `tableHeaderView` anpassen können. Diese und die zugehörige Eigenschaft `tableFooterView` können Sie allen Arten von Ansichten zuweisen, um jeweils eigene Unteransichten dafür zu bilden. Damit können Sie Labels, Textfelder, Schaltflächen und andere Steuerelemente hinzufügen, um den Funktionsumfang einer Tabelle zu erweitern.

Header und Footer sind nicht nur für die Tabelle als Ganzes möglich. Auch die einzelnen Abschnitte weisen anpassbare Header- und Footeransichten auf. Um benutzerdefinierte Ansichten zu erstellen, können Sie die Höhe ändern oder Elemente austauschen. In *Abbildung 11.17* wird links eine größere

Höhe verwendet als üblich, was durch die Implementierung der optionalen Methode `tableView: heightForHeaderInSection:` erreicht wurde. Das Bild auf der rechten Seite zeigt benutzerdefinierte Ansichten. Die einfarbige Headeransicht mit ihren Unteransichten für Label und Schaltflächen wird aus einer `xib`-Datei geladen und über die optionale Methode `tableView:viewForHeaderInSection:` zurückgegeben. Entsprechende Methoden gibt es auch für Footer.



▸ *Abbildung 11.17: Delegierungsmethoden für Tabellenansichten ermöglichen es, die Höhe und die Ansicht von Abschnittsheadern und -footern festzulegen.*

Rezept 11.19 zeigt die Verwendung dieser Methoden. Der benutzerdefinierte Header wird mit einer Höhe von 70 Pixel versehen und aus einer `xib`-Datei gesetzt. Der Label wird festgelegt, und die Schaltfläche wird mit einer einfachen Umkehrungsanimation verbunden. Dies ist nur eine triviale Demonstration einer Funktion, die weit mehr kann, als dieses einfache Beispiel andeutet.

```
// Meldet die Höhe der einzelnen Abschnittsheader
- (CGFloat)tableView:(UITableView *)tableView
   heightForHeaderInSection:(NSInteger)section
{
   return 70.0f;
}

// Die Schaltfläche dient nur zur Demonstration und ruft eine Umkehr-
// Animation auf
- (void) flip: (UIButton *) button
{
```

```
    [UIView beginAnimations:nil context:nil];
    [UIView setAnimationCurve:UIViewAnimationCurveEaseInOut];
    [UIView setAnimationDuration:1.0];
    [UIView setAnimationTransition:
       UIViewAnimationTransitionFlipFromRight forView:self.view
       cache:YES];
    [UIView commitAnimations];
}

// Gibt den Titel der einzelnen Abschnitte zurück
- (NSString *)tableView:(UITableView *)aTableView
    titleForHeaderInSection:(NSInteger)section
{
    if (aTableView == self.tableView)
    {
       if ([[self.sectionArray objectAtIndex:section] count] == 0)
          return nil;
       return [NSString stringWithFormat:

          @"Crayon names starting with '%@'",

          [[ALPHA substringFromIndex:section]
          substringToIndex:1]];
    }
    else return nil;
}

- (UIView *)tableView:(UITableView *)tableView
    viewForHeaderInSection:(NSInteger)section
{
    // Die nib-Datei enthält als einziges Objekt die Headeransicht
    UIView *hView = [[[NSBundle mainBundle] loadNibNamed:@"HeaderView"
       owner:self options:nil] lastObject];

    UILabel *label = (UILabel *)[hView viewWithTag:101];
    label.text = [self tableView:self.tableView
       titleForHeaderInSection:section];

    UIButton *button = (UIButton *)[hView viewWithTag:102];
    [button addTarget:self action:@selector(flip:)
       forControlEvents:UIControlEventTouchUpInside];

    return hView;
}
```

► *Rezept 11.19: Benutzerdefinierte Ansichten für Abschnittsheader erstellen*

Den Rezeptcode finden

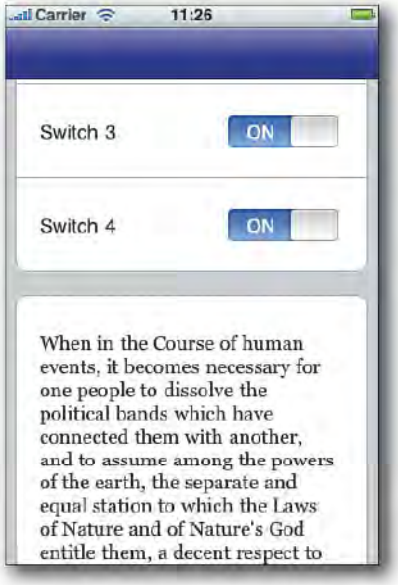
Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.20 Gruppierte Tabellen mit verschiedenen Zellentypen und -höhen erstellen

Wenn alphabetisch sortierte und in Abschnitte aufgeteilte Listentabellen die iPhone-Entsprechungen zur Kunst von M. C. Escher sind, da jeder Abschnittsblock genau in die Negativräume anderer Abschnitte der Liste passt, dann stellen die Freiformgruppen die Bilder von Marc Chagall dar. Jedes Einzelteil ist ein freihändig gezeichnetes Kunstwerk.

Es ist relativ einfach, alle bisher in diesem Kapitel vorgestellten Tabellen anzulegen, wenn Sie erst einmal den Trick kennen. Die Perfektionierung von gruppierten Tabellen (von iPhone-Anhängern normalerweise als »Voreinstellungstabellee« bezeichnet, da diese Art im Programm *Einstellungen* verwendet wird) ist und bleibt eine Illusion. Die Erstellung solcher Tabellen ist eine Collage, bei der Sie Stück für Stück am Erscheinungsbild feilen.

Werkzeuge wie Interface Builder ermöglichen es Ihnen, eine beliebige Anzahl von benutzerdefinierten Tabellenzellen mit jeweils eigenen Inhalten und Höhen anzulegen. Es ist Ihre Aufgabe, all dieses Material im Programmcode zusammenzutragen und daraus eine Tabelle zu erstellen. Sie sind dafür verantwortlich, dass die richtige Art von Zelle bereitgestellt und die individuelle Höhe der einzelnen Zellentypen korrekt gemeldet wird. Bei Anwendungen in der Praxis müssen Sie auch dafür sorgen, dass die Zelle auf sinnvolle Weise auf Benutzeraktionen reagiert.



▸ *Abbildung 11.18: Voreinstellungstabellen müssen manuell gestaltet werden. Dazu legen Sie die einzelnen Zeilen und Gruppen durch die entsprechenden Datenquellenmethoden fest.*

Wenn Sie die Grundlagen beherrschen, können Sie Voreinstellungstabellen nach eigenem Gusto gestalten und formen. *Abbildung 11.18* zeigt eine einfache Voreinstellungstabelle, die aus zwei Gruppen besteht: einer Reihe von Schaltern und einem Block mit Text (sowie einer Untertitelzelle, die zurzeit nicht auf dem Bildschirm zu sehen ist). *Rezept 11.20* verdeutlicht, wie viel Arbeit selbst für diese kleine Tabelle erforderlich ist.

Leider erfordert das Hinzufügen neuer Objekte oder das Aktualisieren älterer sehr viel Feinarbeit, die nirgendwo zentralisiert ist. Sie müssen jede der Datenquellenmethoden einsehen und mit Ihren neuen oder überarbeiteten Objekten aktualisieren.

11.20.1 Gruppierte Voreinstellungstabellen erstellen

Beim Erstellen eines neuen `UITableViewControllers` für eine Voreinstellungstabelle gibt es keine Besonderheiten. Sie weisen ihn zu, initialisieren ihn mit dem Stil der gruppierten Tabelle, und schon sind Sie fertig. Die eigentliche Herausforderung stellen die Datenquellen- und Delegierungsmethoden dar. Folgende Methoden müssen Sie definieren:

- `numberOfSectionsInTableView:` Alle Voreinstellungstabellen enthalten Objektgruppen. Jede Gruppe ist in einem abgerundeten Rechteck untergebracht. Geben Sie die Anzahl der Gruppen, die Sie festlegen wollen, als Integer zurück.
- `tableView: titleForHeaderInSection:` Fügen Sie dieser optionalen Methode die Titel für jeden Abschnitt hinzu. Anschließend geben Sie einen `NSString` mit dem angeforderten Abschnittsnamen zurück. In *Rezept 11.20* werden keine Titel verwendet.
- `tableView: numberOfRowsInSection:` Jeder Abschnitt kann eine beliebige Anzahl von Zellen enthalten. Diese Methode gibt einen Integer zurück, der die Anzahl der Zeilen (also der Zellen) für die betreffende Gruppe anzeigt.
- `tableView: heightForRowAtIndexPath:` Tabellen mit flexiblen Zeilenhöhen sind berechnungsintensiver. Falls Sie variable Höhen benötigen (wie in *Rezept 11.20*), implementieren Sie diese optionale Methode, um die Höhen festzulegen. Der Wert wird nach Abschnitt und Zeile zurückgegeben.
- `tableView: cellForRowAtIndexPath:` Diese standardmäßige Zelle-für-Zeile-Methode haben Sie bereits in diesem Kapitel gesehen. Was sich hier unterscheidet, ist die Implementierung. Anstatt eine Art von Zelle zu verwenden, erstellt *Rezept 11.20* verschiedenartige wiederverwendbare Zellen (mit unterschiedlichen Wiederverwendungs-Tags) für die einzelnen Typen. Wie dieses Rezept zeigt, wird es sehr viel komplizierter, wenn mehrere Zellenarten verwendet werden. Stellen Sie sicher, dass Sie Ihre Wiederverwendungswarteschlange umsichtig verwalten. Die Menge der Zellen ist in diesem Beispiel trivial, kann in der Praxis aber sehr viel komplizierter werden.
- `tableView: didSelectRowAtIndexPath:` Je nach der ausgewählten Zellenart stellen Sie in dieser Delegierungsmethode fallweise unterschiedliche Reaktionen für die Zellenauswahl bereit.

Hinweis

Das Open-Source-Projekt *llamasettings* unter Google Code (*http://llamasettings.googlecode.com*) erstellt automatisch gruppierte Tabellen aus Eigenschaftenlisten für iPhone-Einstellungsbündel. Damit können Sie in Ihre Anwendung *Einstellungen* aufnehmen, ohne die Benutzer dazu zu zwingen, das Programm zu verlassen. Das Projekt kann ohne Lizenzierungsgebühren in kommerzielle iPhone SDK-Anwendungen aufgenommen werden.

```
- (NSInteger)numberOfSectionsInTableView:
   (UITableView *)aTableView
{
   // Die Beispieltabelle umfasst zwei Abschnitte
   return 2;
}

- (NSInteger)tableView:(UITableView *)aTableView
   numberOfRowsInSection:(NSInteger)section
{
   // Der erste Abschnitt hat vier Zeilen, der zweite zwei
   if (section == 0) return 4;
   else if (section == 1) return 2;
   return 0;
}

- (UITableViewCell *)tableView:(UITableView *)tView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   UITableViewCell *cell;

   if (indexPath.section == 0)
   {
      // Lädt eine einfache Zelle mit einem Schalter und einem Label
      cell = [tView dequeueReusableCellWithIdentifier:@"SwitchCell"];
      if (!cell)
         cell = [[[NSBundle mainBundle] loadNibNamed:@"switchcell"
            owner:self options:nil] lastObject];
      [(UILabel *)[cell viewWithTag:101] setText:[NSString
         stringWithFormat:@"Switch %d\n", indexPath.row + 1]];
   }
   else if (indexPath.section == 1)
   {
      // Das erste Element ist ein großer Textblock
      if (indexPath.row == 0)
      {
```

```
            cell = [tView dequeueReusableCellWithIdentifier:
               @"LibertyCell"];
            if (!cell)
               cell = [[[NSBundle mainBundle] loadNibNamed:
               @"libertycell" owner:self options:nil] lastObject];
         }
         // Das zweite Element ist eine standardmäßige Untertitelzelle
         else if (indexPath.row == 1)
         {
            cell = [[[UITableViewCell alloc]
               initWithStyle:UITableViewCellStyleSubtitle
               reuseIdentifier:@"SubtitleCell"] autorelease];
            cell.textLabel.text = @"Hello World";
            cell.detailTextLabel.text = @"Subtitle World";
         }
      }

      return cell;
}

// Die Höhe jeder einzelnen Zelle zu melden, kann sehr rechenintensiv
// sein und zu Leistungseinbußen führen.
- (CGFloat)tableView:(UITableView *)tableView
   heightForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Bestimmt die Zellenhöhe aufgrund von Abschnitt und Zeile
   if (indexPath.section == 0) return 80.0f;
   if (indexPath.section == 1)
   {
      if (indexPath.row == 0) return 340.0f;
      if (indexPath.row == 1) return 40.0f;
   }

   return 0.0f;

}
```

► *Rezept 11.20: Komplexe gruppierte Tabellen mit unterschiedlichen Zellenhöhen erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.21 Mehrfache Drehtabellen erstellen

Manchmal bevorzugen Benutzer es, aus mehreren Listen gleichzeitig auszuwählen. Das ist der Zeitpunkt, an dem `UIPickerView`-Instanzen glänzen können, denn sie erstellen Tabellen mit individuellen rotierenden Rädern (siehe *Abbildung 11.19*). Die Benutzer können dabei mit jedem einzelnen Rad arbeiten, um ihre Auswahl zu treffen.

Diese Tabellen sind den standardmäßigen `UITableViews` zwar sehr ähnlich, nutzen jedoch eigenständige Daten- und Delegierungsprotokolle.

- **Die Klasse `UIPickerViewController` gibt es nicht.** `UIPickerView`-Instanzen dienen als Unteransichten anderer Ansichten. Sie sind nicht als Hauptmerkmal einer Programmansicht gedacht. Instanzen dieser Art können Sie nur in einer anderen Ansicht wie dem Action-Sheet in *Abbildung 11.19* erstellen.
- **Picker-Ansichten benutzen Zahlen, keine Objekte.** Die Komponenten, also die Räder, werden mit Zahlen und nicht mit `NSIndexPath`-Instanzen indiziert. Die Klasse ist informeller als `UITableView`.

▸ *Abbildung 11.19: `UIPickerView`-Instanzen ermöglichen es Benutzern, von unabhängig voneinander verschiebbaren Rädern auszuwählen.*

- **Die Höhe von Picker-Ansichten ist festgelegt.** Es ist nicht möglich, die Picker-Ansichten durch einfache Veränderung ihres Rahmens so in der Größe anzupassen, wie Sie es bei `UITableViews` tun können. Picker-Ansichten haben im Hochformat eine Größe von 320 × 216 Pixel, im Querformat von 480 × 162 Pixel. Alle anderen Rahmengrößen sehen verzerrt oder abgeschnitten aus. Dies sind die gleichen Abmessungen wie für die Standardtastatur des iPhones.

  Aus einer Datenquelle können Sie entweder Titel oder Ansichten hinzufügen. Bei Picker-Ansichten ist beides möglich.

11.21.1 UIPickerView-Ansichten erstellen

Sie können jede Rahmengröße für Ihre `UIPickerView` verwenden, solange sie 216 Pixel hoch und 320 Pixel breit ist (Hochformat) bzw. 162 Pixel hoch und 480 Pixel breit (Querformat). Allerdings können Sie die Tabelle an jede Stelle des Bildschirms verschieben, an der Sie sie benötigen.

Beim Erstellen von Picker-Ansichten müssen Sie an zwei grundlegende Dinge denken. Zunächst sollten Sie die Auswahlkennzeichnung aktivieren, also die blaue Leiste, die über ausgewählten Objekten liegt. Dazu setzen Sie `showsSelectionIndicator` auf `YES`. Wenn Sie den Picker in Interface Builder hinzufügen, ist dies bereits die Standardeinstellung.

Zweitens dürfen Sie nicht vergessen, die Delegierung und die Datenquelle zuzuweisen, da Sie sonst keine Daten zu der Ansicht hinzufügen, keine Funktionen definieren und nicht auf Änderungen der Auswahl reagieren können. Der Hauptansichtscontroller sollte die Protokolle `UIPickerViewDelegate` und `UIPickerViewDataSource` implementieren.

Implementieren Sie mindestens die drei folgenden entscheidenden Datenquellenmethoden, damit Ihre `UIPickerView` richtig funktioniert:

- `numberOfComponentsInPickerView` Gibt einen Integer für die Anzahl der Spalten zurück.
- `pickerView: numberOfRowsInComponent` Gibt einen Integer für die maximale Anzahl der Zeilen pro Rad zurück. Diese Zahlen müssen nicht identisch sein, da ein Rad beispielsweise viele Zeilen und ein anderes nur sehr wenige haben kann.
- `pickerView: titleForRow: forComponent` Legt den Text fest, mit dem eine Zeile einer gegebenen Komponente beschriftet ist, und gibt einen `NSString` zurück. (Die Rückgabe einer Ansicht anstelle eines Strings wird im nächsten Abschnitt behandelt.)

Neben diesen Datenquellenmethoden sollten Sie eine weitere Delegierungsmethode bereitstellen, die auf die Auswahl reagiert, die die Benutzer an den Rädern treffen:

- `pickerView: didSelectRow: inComponent` Fügen Sie das gesamte programmspezifische Verhalten zu dieser Methode hinzu. Falls notwendig, können Sie `pickerView` abfragen, um `selectedRowInComponent:` für alle Auswahlräder in der Ansicht zurückzugeben.

Rezept 11.21 erstellt das einfache Auswahlrad aus *Abbildung 11.19*, das ein Zahlenschloss simuliert, über das die Benutzer eine Kombination eingeben können. Wenn Sie den Picker in eine `UIAlertSheet`-Instanz einbetten, kann er in die Ansicht hinein- und aus ihr herausgleiten.

```
@interface TestBedViewController : UIViewController <UIPickerViewDelegate,
    UIPickerViewDataSource, UIActionSheetDelegate>
@end

@implementation TestBedViewController
- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)pickerView
{
    // Der Picker hat drei Räder
    return 3;
}
```

```
- (NSInteger)pickerView:(UIPickerView *)pickerView
   numberOfRowsInComponent:(NSInteger)component
{

   // Jedes Rad hat 20 Zeilen
   return 20;
}

- (NSString *)pickerView:(UIPickerView *)pickerView
   titleForRow:(NSInteger)row forComponent:(NSInteger)component
{
   // Jede Zelle enthält eine Zahl und entweder R (für rechts) oder
   // L (für links)
   return [NSString stringWithFormat:@"%@-%d",
      component == 1 ? @"R" : @"L", row];
}

- (void)actionSheet:(UIActionSheet *)actionSheet
   clickedButtonAtIndex:(NSInteger)buttonIndex
{
   // Zeigt die aktuelle Kombination
   UIPickerView *pickerView = (UIPickerView *)[actionSheet
      viewWithTag:101];
   self.title = [NSString stringWithFormat:@"L%d-R%d-L%d",
      [pickerView selectedRowInComponent:0],
      [pickerView selectedRowInComponent:1],
      [pickerView selectedRowInComponent:2]];
   [actionSheet release];
}

- (void) action: (id) sender
{
   // Schafft ausreichend Platz für den Picker
   NSString *title = UIDeviceOrientationIsLandscape([UIDevice
      currentDevice].orientation) ?
      @"\n\n\n\n\n\n\n\n\n" :
      @"\n\n\n\n\n\n\n\n\n\n\n\n" ;

   // Erstellt das Action-Sheet und zeigt es an
   UIActionSheet *actionSheet = [[UIActionSheet alloc]
      initWithTitle:title delegate:self
      cancelButtonTitle:nil
      destructiveButtonTitle:nil
```

```
        otherButtonTitles:@"Set Combo", nil];
    [actionSheet showInView:self.view];

    // Fügt die Unteransicht mit dem Picker hinzu, sobald das Sheet
    // angezeigt wird
    UIPickerView *pickerView = [[[UIPickerView alloc] init]
        autorelease];
    pickerView.tag = 101;
    pickerView.delegate = self;
    pickerView.dataSource = self;
    pickerView.showsSelectionIndicator = YES;

    [actionSheet addSubview:pickerView];
}

- (BOOL)shouldAutorotateToInterfaceOrientation:
    (UIInterfaceOrientation)interfaceOrientation
{
    // Erlaubt die Darstellung sowohl im Hoch- als auch im Querformat
    return YES;
}

- (void) viewDidLoad
{

    self.navigationController.navigationBar.tintColor =

        COOKBOOK_PURPLE_COLOR;
    self.navigationItem.rightBarButtonItem = BARBUTTON(@"Action",
        @selector(action:));
}
@end
```

▶ *Rezept 11.21: Eine* `UIPickerView` *für eine mehrspaltige Auswahl verwenden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.22 Rezept: Einen Picker mit Ansichten verwenden

Picker-Ansichten funktionieren genauso gut mit Ansichten wie mit Titeln. *Abbildung 11.20* zeigt eine Picker-Ansicht, die Spielkartenfarben darstellt. Diese Bilder gibt die Datenquellenmethode `pickerView: viewForRow: forComponent: reusingView:` zurück. Dabei können Sie jede Form von Ansicht verwenden, z. B. Labels, Schieberegler, Schaltflächen usw. In dem Beispiel von *Rezept 11.22* wird eine einfache `UIImageView` für die Darstellung der der vier Spielkartensymbole verwendet.

▶ *Abbildung 11.20: Diese* `UIPickerView` *zeigt eine Folge von Bildern mit Spielkartensymbolen. Die Benutzer können eine Kombination aus drei Elementen auswählen.*

Picker-Ansichten nutzen ein einfaches Verfahren zur Wiederverwendung, bei dem die zur möglichen Wiederverwendung bereitgestellten Ansichten zwischengespeichert werden. Wenn der letzte Parameter der Callback-Methode nicht `nil` ist, kann die Ansicht durch Aktualisierung ihrer Einstellungen oder Inhalte wiederverwendet werden. Der Code in *Rezept 11.22* sucht nach einer solchen Ansicht und weist nur dann eine neue Bildansicht zu, wenn er keine findet.

Die Höhe muss nicht der Höhe der tatsächlichen Ansicht entsprechen. Wenn Sie `pickerView: rowHeightForComponent:` implementieren, können Sie damit die Zeilenhöhe der einzelnen Komponenten festlegen. In *Rezept 11.22* wird eine Zeilenhöhe von 120 Pixeln verwendet, was genügend Platz für die einzelnen Bilder lässt und die Illusion hervorruft, dass der Picker nicht wie in *Rezept 11.21* einen Anfang und ein Ende hat, sondern endlos durchlaufen kann.

Beachten Sie die große Anzahl der Zeilen, nämlich eine Million. Der Grund für diese hohe Zahl besteht darin, richtige Zylinder zu simulieren. Normalerweise haben Picker-Ansichten ein erstes und ein letztes Element, bei denen der Drehvorgang endet. In diesem Rezept wird ein anderer Ansatz verfolgt, um so zu tun, als wären die Komponenten echte Zylinder, bei denen das letzte Element wieder an das erste angrenzt.

Dazu wird in diesem Picker eine weit höhere Anzahl von Zeilen verwendet, als irgendein Benutzer überhaupt jemals nutzen könnte. Der Picker wird durch den Aufruf von `selectRow:inComponent:Animated:` in der Mitte dieser Zeilenmenge initialisiert. Jede Komponenten-»Zeile« wird durch den Modulo aus der tatsächlich gemeldeten Zeile und der Anzahl der anzuzeigenden einzelnen Elemente bestimmt, in diesem Fall durch `% 4`. Der Code weiß zwar, dass der Picker eine Million Zeilen pro Rad aufweist, aber der Benutzer sieht ein zylindrisches Rad mit nur vier Zeilen.

```
- (NSInteger)pickerView:(UIPickerView *)pickerView
   numberOfRowsInComponent:(NSInteger)component
{
   // Gibt eine absurd hohe Anzahl von Zeilen pro Rad zurück
   return 1000000;
}

- (CGFloat)pickerView:(UIPickerView *)pickerView
   rowHeightForComponent:(NSInteger)component
{
   // Stellt die Zeilenhöhe passend zur Grafik ein
   return 120.0f;
}

- (UIView *)pickerView:(UIPickerView *)pickerView
   viewForRow:(NSInteger)row forComponent:(NSInteger)component
   reusingView:(UIView *)view
{
   // Erstellt bei Bedarf eine neue Ansicht und fügt die Grafik hinzu
   UIImageView *imageView;
   imageView = view ? (UIImageView *) view : [[UIImageView alloc]
      initWithFrame:CGRectMake(0.0f, 0.0f, 60.0f, 60.0f)];
   NSArray *names = [NSArray arrayWithObjects:@"club.png",
      @"diamond.png", @"heart.png", @"spade.png", nil];
   imageView.image = [UIImage imageNamed:
      [names objectAtIndex:(row % 4)]];
   return imageView;
}

- (void)actionSheet:(UIActionSheet *)actionSheet
   clickedButtonAtIndex:(NSInteger)buttonIndex
{
   // Legt die aktuelle Radauswahl als Titel fest
   UIPickerView *pickerView = (UIPickerView *)[actionSheet
      viewWithTag:101];
   NSArray *names = [NSArray arrayWithObjects:
      @"C", @"D", @"H", @"S", nil];
   self.title = [NSString stringWithFormat:@"%@•%@•%@";
```

```
        [names objectAtIndex:([pickerView
        selectedRowInComponent:0] % 4)],
        [names objectAtIndex:([pickerView
        selectedRowInComponent:1] % 4)],
        [names objectAtIndex:([pickerView
        selectedRowInComponent:2] % 4)]];
    [actionSheet release];
}

- (void) action: (id) sender
{
    // Schafft genügend Platz für den Picker
    NSString *title = UIDeviceOrientationIsLandscape([UIDevice
        currentDevice].orientation) ?
        @"\n\n\n\n\n\n\n\n\n" :
        @"\n\n\n\n\n\n\n\n\n\n\n\n" ;

    // Erstellt das Action-Sheet und zeigt es an
    UIActionSheet *actionSheet = [[UIActionSheet alloc]
        initWithTitle:title delegate:self
        cancelButtonTitle:nil destructiveButtonTitle:nil
        otherButtonTitles:@"Set Combo", nil];
    [actionSheet showInView:self.view];

    // Fügt die Picker-Ansicht hinzu, sobald das Sheet angezeigt wird
    UIPickerView *pickerView = [[[UIPickerView alloc] init]
        autorelease];
    pickerView.tag = 101;
    pickerView.delegate = self;
    pickerView.dataSource = self;
    pickerView.showsSelectionIndicator = YES;

    [actionSheet addSubview:pickerView];

    // Wählt ein zufälliges Element in der Mitte der Tabelle aus
    [pickerView selectRow:50000 + (random() % 4) inComponent:0
        animated:YES];
    [pickerView selectRow:50000 + (random() % 4) inComponent:1
        animated:YES];
    [pickerView selectRow:50000 + (random() % 4) inComponent:2
        animated:YES];
}
```

▶ *Rezept 11.22: Die Drehung eines geschlossenen Zylinders simulieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.23 Rezept: UIDatePicker verwenden

Für den Fall, dass Ihre Benutzer Kalenderdaten eingeben sollen, stellt Apple eine ordentliche Unterklasse von `UIPickerView` bereit, die verschiedene Arten von Zeitangaben verarbeitet. *Abbildung 11.21* zeigt die vier eingebauten Formate von `UIDatePicker`, aus denen Sie auswählen können. Zur Verfügung stehen die Auswahl der Uhrzeit, des Datums, eine Kombination von beiden sowie ein Countdown-Timer. *Rezept 11.23* veranschaulicht alle vier Formate.

11.23.1 Die Datumsauswahl erstellen

Eine Picker-Ansicht zur Datumsauswahl gestalten Sie genauso wie eine `UIPickerView`. Die Geometrie ist identisch. Anschließend werden die Dinge sehr viel einfacher, da Sie keine Delegierung einrichten und auch keine Datenquellenmethoden definieren müssen. Sie müssen auch kein Protokoll deklarieren, sondern nur einen Modus für die Datumsauswahl zuweisen. Mögliche Modi sind `UIDatePickerModeTime`, `UIDatePickerModeDate`, `UIDatePickerModeDateAndTime` und `UIDatePickerModeCountDownTimer`.

Optional können Sie ein Ziel für den Fall hinzufügen, dass sich die Auswahl ändert (`UIControlEvent ValueChanged`), und die Callback-Methode für das Target-Action-Paar erstellen.

Folgende Eigenschaften können Sie in der Klasse `UIDatePicker` zu Ihrem Vorteil nutzen:

- `date` Richtet die Eigenschaft `date` zur Initialisierung des Pickers oder zum Abrufen der Daten ein, die der Benutzer mithilfe der Rädern festgelegt hat.
- `maximumDate` und `minimumDate` Diese Eigenschaften setzen die Grenzen der Datums- und Zeitauswahl. Weisen Sie jeder ein standardmäßiges `NSDate` zu. Damit können Sie den Benutzer z. B. auf die Auswahl eines Datums des nächsten Jahres beschränken, anstatt ihn ein Datum eingeben zu lassen und anschließend zu prüfen, ob es sich in einem akzeptablen Zeitrahmen befindet.
- `minuteInterval` Manchmal müssen Sie die Auswahl auf Intervalle von 5, 10, 15 oder 30 Minuten beschränken, z. B. in Anwendungen, mit denen Termine festgelegt werden. Verwenden Sie die Eigenschaft `minuteInterval`, um diesen Wert festzulegen. Allerdings muss 60 ohne Rest durch die eingegebene Zahl teilbar sein.
- `countDownDuration` Verwenden Sie diese Eigenschaft, um den maximal verfügbaren Wert eines Countdown-Timers einzurichten. Sie können höchstens 23 Stunden und 59 Minuten (86.399 Sekunden) eingeben.

▶ *Abbildung 11.21: Das iPhone bietet vier Modelle für den Datums-Picker an. Mit der Eigenschaft* `datePickerMode` *wählen Sie eines davon für Ihr Programm aus.*

```
@implementation TestBedViewController
- (void)actionSheet:(UIActionSheet *)actionSheet
    clickedButtonAtIndex:(NSInteger)buttonIndex
{
    // Ruft den Picker ab
    UIDatePicker *datePicker = (UIDatePicker *)[actionSheet
        viewWithTag:101];

    // Legt das Datumsformat anhand des ausgewählten Segments fest
    NSDateFormatter *formatter = [[[NSDateFormatter alloc] init]
        autorelease];
    switch ([(UISegmentedControl *)self.navigationItem.titleView
        selectedSegmentIndex])
    {
```

```
        case 0: // Uhrzeitauswahl
            formatter.dateFormat = @"h:mm a";
            break;
        case 1: // Datumsauswahl
            formatter.dateFormat = @"dd MMMM yyyy";
            break;
        case 2: // Auswahl von Datum und Uhrzeit
            formatter.dateFormat = @"MM/dd/YY h:mm a";
            break;
        case 3: // Countdown-Picker
            formatter.dateFormat = @"HH:mm";
            break;
        default:
            break;
    }

    // Erstellt einen Zeitstempel und zeigt ihn an
    NSString *timestamp = [formatter stringFromDate:datePicker.date];
    [(UILabel *)[self.view viewWithTag:103] setText:timestamp];
    [actionSheet release];
}

- (void) action: (id) sender
{
    // Räumt Platz für die Darstellung im Hoch- oder Querformat ein
    NSString *title = UIDeviceOrientationIsLandscape(
        [UIDevice currentDevice].orientation) ?
        @"\n\n\n\n\n\n\n\n\n" : @"\n\n\n\n\n\n\n\n\n\n\n\n" ;

    // Erstellt das Action-Sheet und zeigt es an
    UIActionSheet *actionSheet = [[UIActionSheet alloc]
        initWithTitle:title delegate:self
        cancelButtonTitle:nil destructiveButtonTitle:nil
        otherButtonTitles:@"Set", nil];
    [actionSheet showInView:self.view];

    // Erstellt den Datums-Picker und fügt ihn  hinzu
    UIDatePicker *datePicker = [[[UIDatePicker alloc] init]
        autorelease];
    datePicker.tag = 101;
    datePicker.datePickerMode =
    [(UISegmentedControl *)self.navigationItem.titleView
        selectedSegmentIndex];
    [actionSheet addSubview:datePicker];
}
```

```
- (void) viewDidLoad
{

   self.navigationController.navigationBar.tintColor =

      COOKBOOK_PURPLE_COLOR;
   self.navigationItem.rightBarButtonItem = BARBUTTON(@"Action",
      @selector(action:));

   // Erstellt ein segmentiertes Schaltflächenelement zur Auswahl des
   // Datumsformats
   UISegmentedControl *seg = [[[UISegmentedControl alloc]
      initWithItems:[@"Time Date DT Count"
      componentsSeparatedByString:@" "]] autorelease];
   seg.segmentedControlStyle = UISegmentedControlStyleBar;
   seg.selectedSegmentIndex = 0;
   self.navigationItem.titleView = seg;
}
@end
```

▸ *Rezept 11.23:* `UIDatePicker` *zur Auswahl von Datum und Uhrzeit verwenden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 11 und öffnen das Projekt zu diesem Rezept.

11.24 Ein letzter Punkt: Datumsangaben formatieren

Die Klasse `NSDateFormatter` hat sich zwar seit ihren Anfängen sehr stark entwickelt und enthält jetzt Elemente, die sich sehr gut bearbeiten und an verschiedene Kalender und Kulturen anpassen lassen, doch ist es immer noch hilfreich, eine Übersicht der gebräuchlichsten Datums- und Zeitformate zur Hand zu haben. In *Tabelle 11.1* finden Sie eine solche Übersicht, in der die am häufigsten verwendeten Standardcodes aufgeführt sind. Hierbei handelt es sich um Codes wie diejenigen, die in *Rezept 11.23* in der Eigenschaft `date` des Datums-Pickers zur Formatierung der Ergebnisse im Label in der Bildschirmmitte verwendet wurden. In *Listing 11.1* werden die Formate aus *Tabelle 11.1* genutzt, um ein `NSString`-Hilfsprogramm zu erstellen, das einen String mit dem Format Monat-Tag-Jahr in ein Datum umwandelt, also z. B. `@"05-22-1934"`.

Typ	Code	Erklärung
Tag im Monat	d	1 bis 31, keine führenden Nullen
	dd	01 bis 31, führende Nullen
Monat	M L	1 bis 12, numerischer Monatswert ohne führende Nullen
	MM LL	01 bis 12, numerischer Monatswert mit führenden Nullen
	MMM LLL	Jan bis Dec, Monatsabkürzung mit drei Buchstaben
	MMMM LLLL	January bis December, ausgeschriebener Monatsname
Jahr	y u	Vierstellige Jahreszahl, z. B. 2009
	yy	Zweistellige Jahreszahl, z. B. 09
Stunde	h K	1 bis 12, ohne führende Nullen
	hh KK	01 bis 12, mit führenden Nullen
	H k	0 bis 23, ohne führende Nullen
	HH kk	00 bis 23, mit führenden Nullen
Minuten	m	0 bis 59, ohne führende Nullen
	mm	00 bis 59, mit führenden Nullen
Sekunden	s	0 bis 59, ohne führende Nullen
	ss	00 bis 59, mit führenden Nullen
AM/PM	a	

Wochentag	`ccc` `EEE`	Sun bis Sat, Abkürzung in drei Buchstaben
	`cccc` `EEEE`	Sunday bis Saturday, vollständige Namen
	`c` `e`	0 bis 7, Nummer des Wochentags
	`cc` `ee`	00 bis 07, Nummer des Wochentags mit führenden Nullen
Woche im Monat	`F`	1 bis 5, ohne führende Nullen
	`FF`	01 bis 05, mit führenden Nullen
Tag im Jahr	`D`	1 bis 366, ohne führende Nullen
	`DD`	01 bis 366, mit führender Null
	`DDD`	001 bis 366, mit zwei führenden Nullen
Kalenderwoche	`w`	1 bis 52, ohne führende Nullen
	`ww`	01 bis 52, mit führenden Nullen
Millisekunde des Tages	`A`	0 bis 86399999, ohne führende Nullen
Julianisches Datum	`g`	Anzahl der Tage seit dem 1. Januar 4713 v. Chr.
Ära	`G`	BC, AD (v. Chr., n. Chr.)
	`GGGG`	Before Christ, Anno Domini (ausgeschrieben)
Quartal	`q` `Q`	1 bis 4, Quartalsnummer
	`qq` `QQ`	01 bis 04, mit führenden Nullen
	`qqq` `QQQ`	Q1 bis Q4
	`qqqq` `QQQQ`	1st quarter, 2nd quarter, 3rd quarter, 4th quarter

Zeitzone	v	Abkürzung der Zeitzone mit zwei Buchstaben, z. B. MT
	V	Abkürzung der Zeitzone mit drei Buchstaben, z. B. MDT
	z	
	vv	Zeitzonenoffset von westeuropäischer Zeit (Greenwich Mean Time, GMT) nach RFC 822, z. B. GMT-06:00.
	VV	
	ZZZ	
	vvvv	Zeitzonenname, z. B. Mountain Time
	VVVV	Position der Zeitzone, z. B. United States (Denver)
	zzzz	Vollständiger Name der Zeitzone, z. B. Mountain Daylight Time
	z	GMT-Offset z. B. -06:00
Andere Zeichen	:	Doppelpunkt, Bindestrich, Schrägstrich
	-	
	/	

▸ *Tabelle 11.1: Standardformatcodes für die Klasse* `NSDateFormatter`

```
@interface NSString (DateUtility)
- (NSDate *) dateFromString;
@end

@implementation NSString (DateUtility)
- (NSDate *) dateFromString
{
    // Gibt ein Datum aus einem String zurück
    NSDateFormatter *formatter =
    [[[NSDateFormatter alloc] init] autorelease];
    formatter.dateFormat = @"MM-dd-yyyy";
    NSDate *date = [formatter dateFromString:self];
    return date;
}
@end
```

▸ *Listing 11.1: Einen String mithilfe von Datumsformaten in ein Datum konvertieren*

11.25 Zusammenfassung

In diesem Kapitel wurden iPhone-Tabellen von den einfachsten bis zu komplizierten Varianten vorgestellt. Hier haben Sie alle grundlegenden iPhone-Tabellenfunktionen gesehen, von der Darstellung einfacher Tabellen über die Bearbeitung und Umsortierung bis zum Widerrufen von Aktionen. Außerdem haben Sie verschiedene erweiterte Elemente wie benutzerdefinierte Zellen aus `xib`-Dateien, indizierte alphabetische Auflistungen und Picker-Ansichten kennengelernt. Mit den Fertigkeiten, die Sie in diesem Kapitel erworben haben, können Sie eine breite Palette von tabellengestützten Anwendungen für das iPhone, den iPod touch oder das iPad erstellen. Nehmen Sie aus diesem Kapitel die folgenden Punkte mit:

- Seien Sie sich des Unterschieds zwischen Datenquellen und Delegierungsmethoden bewusst. Datenquellen füllen Tabellen mit sinnvollen Zellen, Delegierungsmethoden reagieren auf Benutzeraktionen.
- `UITableViewController` vereinfachen Anwendungen, die auf einer zentralen `UITableView` aufbauen. Zögern Sie jedoch nicht, `UITableView`-Instanzen auch direkt zu verwenden, wenn dies in Ihrer Anwendung erforderlich ist. Stellen Sie dabei aber sicher, dass Sie die Protokolle `UITableViewDelegate` und `UITableViewDataSource` ausdrücklich unterstützen.
- Indexsteuerelemente bilden eine hervorragende Möglichkeit, um sich schnell durch lange, geordnete Listen zu bewegen. Nutzen Sie diese leistungsfähigen Funktionen, wenn Sie es mit Tabellen zu tun haben, in denen sich die Benutzer ansonsten kaum orientieren könnten. In gruppierten Tabellen dagegen sollten Sie Indexsteuerelemente aus ästhetischen Gründen vermeiden.
- Datums-Picker sind hochgradig spezialisierte Auswahlelemente, die ihre Aufgabe sehr gut erfüllen, nämlich Angaben von Datum und Uhrzeit von den Benutzern einzufordern. Picker-Ansichten dagegen sind weniger spezialisiert, erfordern aber mehr Arbeit von Ihnen, um sie ans Laufen zu bringen.
- In diesem Kapitel wurde die Klasse `NSPredicate` eingeführt, die weit flexibler und leistungsfähiger ist, als die hier vorgestellten Tabellen anzeigen. Die Möglichkeiten dieser Klasse werden in *Kapitel 18, Verbindung mit dem Adressbuch*, und *19, Core Data*, ausführlicher behandelt.
- Voreinstellungstabellen sind zwar kompliziert zu programmieren, sind aber ein beliebtes Merkmal in iPhone-Anwendungen. Damit können Sie viele verschiedene Möglichkeiten zur interaktiven Eingabe auf einer ansprechenden und leicht zu durchblätternden Seite kombinieren. Wenn Sie all die kniffligen Aspekte gemeistert haben, erweisen sich solche Tabellen als leistungsfähige Werkzeuge in Ihrem Programmierrepertoire.

12 Verbindungen mit GameKit und Bonjour

In diesem Kapitel wird GameKit vorgestellt, die einfachste Möglichkeit, um zwei iPhone-Geräte miteinander zu verbinden. GameKit ist die neue Ad-hoc-Netzwerklösung von Apple für Peer-to-Peer-Verbindungen. Grundlage ist eine Technologie namens Bonjour, die eine einfache, konfigurationsfreie Kommunikation zwischen den Geräten ermöglicht. Mit GameKit und Bonjour können Sie Spiele und Hilfsprogramme erstellen, die Informationen zwischen mehreren iPhones oder zwischen einem iPhone und einem Desktop-System austauschen. In diesem Kapitel lernen Sie, wie Sie mithilfe von GameKit verbundene Anwendungen schreiben. Sie erweitern die GameKit-Kommunikation, um ein System zum gemeinschaftlichen Zeichnen zu erstellen, das sowohl Benutzerberührungen als auch die ausgewählte Farbe überträgt. Außerdem erfahren Sie, wie Sie GameKit Voice zu Ihren Anwendungen hinzufügen, um Voice-Chats im Stil der Walkie-Talkie-Kommunikation zu ermöglichen. Darüber hinaus lernen Sie auch noch etwas grundlegende Bonjour-Programmierung, um die Einschränkungen von GameKit zu überwinden und die Kommunikation mit dem iPhone auch auf den Desktop auszudehnen.

12.1 Rezept: Einfache GameKit-Dienste bereitstellen

GameKit ermöglicht eine Peer-to-Peer-Anbindung zwischen iPhone- und iPod touch-Geräten. Dieses Framework ist neu seit dem SDK 3.0 und hilft dabei, vernetzte Anwendungen zu schreiben, die Daten in Echtzeit austauschen.

In der Standardimplementierung erstellt und verwaltet GameKit ein Ad-hoc-Bluetooth-Netzwerk, in dem die Geräte sich gegenseitig finden, eine Verbindung erstellen und darüber Daten übertragen können. Seit der iPhone-Firmware 3.1 kann GameKit auch Geräte im selben WiFi-Netzwerk finden, Verbindung dazu aufnehmen und Daten übertragen. Die Verwendung von Bluetooth ist eine schnel-

le und zuverlässige Möglichkeit zur Kommunikation zwischen den Geräten, doch leider wird die Nutzung von Bluetooth über GameKit in den iPhones und iPod touches der ersten Generation nicht unterstützt. GameKit bietet neben dem Bluetooth- auch einen Online-Modus an (über WiFi oder das Internet), aber während ich diese Zeilen schreibe, handelt es sich dabei im Grunde genommen um eine Form von »Bringen Sie Ihre eigene Technologie mit«.

Sie können mit GameKit nicht nur Spiele erstellen, wie der Name andeutet, sondern weit mehr. Beispielsweise sind Anwendungen für gemeinschaftliches Layout, den Austausch von Bildern, Chats usw. möglich. Solange die Anwendung auf beiden Geräten vorhanden ist, können Sie die GameKit-Verbindung mit (Firmware 3.1 und höher) und ohne ein gemeinsames WiFi-Netzwerk herstellen.

12.1.1 GameKit-Einschränkungen für Bluetooth

Nähe ist alles, was Sie brauchen. Bluetooth-gestützte GameKit-Anwendungen sind auf einen Abstand von etwa zehn Metern beschränkt. Wenn Sie sich Ihr Zielpublikum vorstellen, denken Sie also an Personen, die zusammen im Zug oder im Auto reisen, die sich in einem Besprechungsraum versammeln oder im selben Büro arbeiten. Innerhalb dieser Reichweite kann Ihre Anwendung dann auf einfache Weise eine Peer-to-Peer-Verbindung herstellen.

Bei der Übertragung kurzer Informationen zeigt GameKit eine hervorragende Leistung. Apple empfiehlt, die Menge der zu übertragenden Daten auf maximal 1000 Byte zu beschränken. Zwar kann GameKit auch größere Datenpakete von bis zu 95 Kilobyte auf einmal handhaben, aber das Framework ist nicht für die allgemeine Datenübertragung von Gerät zu Gerät gedacht. Wenn Sie versuchen, zu viele Daten auf einmal zu senden, treten Übermittlungsfehler auf.

Wenn Sie umfangreiche Dateien übertragen müssen, müssen Sie diese in Teilstücke zerlegen, mit denen GameKit zurechtkommt. Verwenden Sie die üblichen Handshake- und Prüfsummentechniken, um die Zuverlässigkeit der Daten sicherzustellen.

12.1.2 Geräte-Einschränkungen

Bluetooth-Netzwerke via GameKit sind nicht für alle iPhone- und iPod touch-Geräte geeignet, sondern funktionieren nur beim iPhone-Modell 3G und neuer und beim iPod touch der zweiten Generation und neuer. Es ist nicht möglich, eine GameKit-Bluetooth-Anwendung auf iPhone- und iPod touch-Geräten der ersten Generation bereitzustellen. Außerdem wird Bluetooth via GameKit auch im Simulator nicht vollständig unterstützt. Sie können zwar Geräte in der Nähe erkennen, aber keine Verbindung mit ihnen aufnehmen.

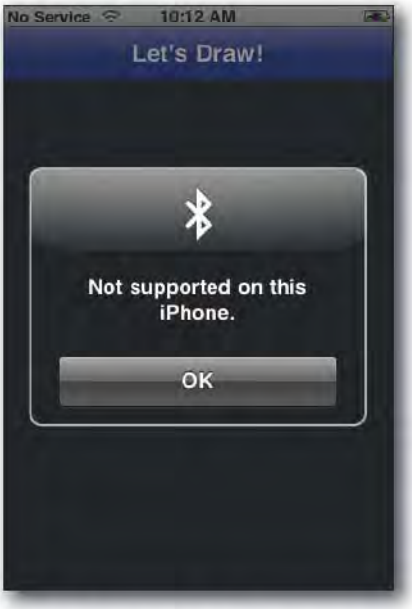
Ab der SDK-Version 3.1 hat Apple Unterstützung für den Schlüssel `peer-peer` im Eintrag `UIRequiredDeviceCapabilities` von `Info.plist` vorgesehen. Dieser Schlüssel gibt an, dass die Anwendung eine Peer-to-Peer-Anbindung über Bluetooth erfordert. In Firmware vor Version 3.1 können Sie erforderliche Gerätefunktionen wie `telephony` und `microphone` angeben, es gibt jedoch keinen Schlüssel für ein Bluetooth-Netzwerk als Voraussetzung.

Machen Sie in Ihrem Marketingmaterial deutlich, für welche Geräte Ihre Anwendung geeignet ist und für welche nicht, vor allem, wenn die Verbindung mit iPhones in der Nähe ein Kernstück des Programms ist. Auf nicht kompatiblen Geräten können die Benutzer GameKit-Bluetooth-Funktionen nicht nutzen. Wenn sie es dennoch versuchen, erhalten sie eine Meldung wie die aus *Abbildung 12.1*, die ein Bluetooth-Logo zeigt und angibt, dass diese Funktion auf dem betreffenden iPhone (oder iPod) nicht unterstützt wird.

Aus diesem Grund sollten Sie eine Reservelösung im Online-Modus vorsehen. Über dieselbe grafische Schnittstelle, die zum Bluetooth-gestützten »Nähere Umgebung«-Modus führt, können die Benutzer auch auf andere Netzwerkoptionen zugreifen. *Rezept 12.8* stellt diese zweite Verbindungsschicht vor.

12.1.3 Sitzungen

Die Peer-to-Peer-Verbindungen von GameKit werden mithilfe von Bluetooth-Netzwerkfunktionen eingerichtet. Bei Bonjour handelt es sich um den Markennamen von Apple für konfigurationsfreie Netzwerke. Dieses Protokoll ermöglicht es Geräten, Netzwerkdienste anzukündigen und zu entdecken. Es ist seit Version 10.2 in Mac OS X eingebaut und bietet diese Funktionen, ohne auf sich selbst aufmerksam zu machen. Beispielsweise steckt Bonjour hinter den Funktionen, mit denen Benutzer freigegebene Musik für iTunes aufspüren oder ohne Konfiguration Verbindung mit drahtlos angeschlossenen Druckern aufnehmen können. Diese Dienste erscheinen automatisch, wenn sie verfügbar sind, und verschwinden anderenfalls wieder. Das ist ein leistungsfähiges Merkmal des Betriebsystems!



▸ *Abbildung 12.1: GameKit-Funktionen sind nicht auf allen iPhone- und iPod touch-Geräten verfügbar. Die Modelle der ersten Generation unterstützen keine GameKit-Bluetooth-Verbindungen. Vermeiden Sie diesen Fehler, indem Sie die Geräteanforderung* `peer-peer` *in der Datei* `Info.plist` *Ihrer Anwendung festlegen.*

GameKit bietet die Möglichkeiten von Bonjour, ohne dass Sie die oftmals komplizierten Bonjour-Callbacks für die Registrierung und Erkennung von Diensten schreiben müssen. Mit GameKit fordern Sie eine Verbindung über einen »Peer-Picker«-Controller an und verwalten dann eine »Sitzung«, sobald die Verbindung eingerichtet ist.

Die Sitzungsobjekte von GameKit bilden einen zentralen Punkt für die Verwaltung der Datenübertragung. Jede Sitzung trägt einen von Ihnen gewählten, eindeutigen Namen, um sich selbst anzukündigen. Wenn eine Anwendung nach einem anderen Gerät sucht, mit dem sie sich verbinden kann, zieht sie diesen Namen heran, um kompatible Dienste zu erkennen.

Wenn Sie mit einem Bonjour-Browserdienst nach diesem Namen suchen, werden Sie keinen Erfolg haben, denn Apple kodiert die Dienstnamen. So wird ein Dienst namens `MacBTClientSample` z. B. zum Bonjour-Dienst `_11d7n7p5tob54j._udp.`. GameKit wandelt die von Ihnen angegebenen Namen automatisch um, sodass es passende Dienste finden kann.

Leider gibt es keine Mac OS X- oder Windows-API, um auf einem Desktop-System Dienste zu erstellen, mit denen Sie sich in GameKit einklinken können. Die Namensverschlüsselung von Apple sorgt mehr oder weniger dafür, dass die standardmäßige Bonjour-Kommunikation nicht bei GameKit-Anwendungen auf dem iPhone funktioniert.

Wie Sie diese Einschränkung umgehen können, lesen Sie weiter hinten in diesem Kapitel, wo es um die direkte Arbeit mit Bonjour geht.

12.1.4 Server, Clients und Peers

In GameKit sind drei Sitzungsmodi möglich, sodass Anwendungen als Server, als Clients und als Peers fungieren können. Server bieten einen Dienst an und initialisieren eine Sitzung, sodass Clients danach suchen und Verbindungen aufnehmen können. Dies ist die Art von Verhalten, die ein intelligenter Drucker aufweist, damit Clients seine Fähigkeiten erkennen und nutzen können. Für Geräte mit einem festen Funktionsumfang ist das praktisch, aber für die meisten iPhone-Anwendungen ist dies nicht die beste Wahl, vor allem nicht für Spiele.

Peers fungieren gleichzeitig als Server und als Clients, indem sie gleichzeitig Dienste ankündigen und nach ihnen suchen. Wenn ein Peer einen Dienst auswählt, wird seine Client/Server-Rolle sowohl vor dem Benutzer als auch vor dem Entwickler verborgen. Dadurch lässt sich der Peer-Ansatz für iPhones sehr einfach realisieren, denn Sie müssen keine getrennten Server- und Clientanwendungen erstellen. Eine einzige Peer-Anwendung erledigt sämtliche Aufgaben.

12.1.5 Die Verbindungsaufnahme mit dem Peer

Die Auswahl des Peers erledigt die Klasse `GKPeerPickerController`. Sie stellt eine Reihe vorgefertigter interaktiver Benachrichtigungsdialogfelder bereit, die die Ankündigung der Geräteverfügbarkeit und die Auswahl eines Peers automatisieren. Es ist jedoch nicht zwingend erforderlich, diese Klasse zu verwenden. Sie können sie auch umgehen und eine eigene Klasse erstellen, um nach Peers zu suchen und sich mit ihnen zu verbinden. Für einfache Verbindungen jedoch bietet `GKPeerPickerController` eine fertige Schnittstelle, die die Notwendigkeit umgeht, Peers zu finden und die Verbindung mit ihnen auszuhandeln.

Um den Peer-Picker zu verwenden, weisen Sie ihn zu, legen eine Delegierung fest (die das Protokoll `GKPeerPickerControllerDelegate` implementieren muss) und zeigen ihn an. Für Ziele älter als SDK 3.1 müssen Sie beim Picker auf die automatische Freigabe verzichten. Warten Sie stattdessen auf den Delegierungs-Callback, und geben Sie ihn dann frei. Dadurch stellen Sie sicher, dass

die Anwendung nicht abstürzt, wenn die Animation zum Entfernen des Pickers fehlschlägt. Dieses Problem wurde in der Firmware-Version 3.1 behoben. Ab Version 3.1 können Sie auswählen, ob Sie die Freigabe im Callback vornehmen (kompatibel mit 3.0) oder die automatische Freigabe nutzen (kompatibel mit 3.1 und höher).

Wie bereits erwähnt wurde, ermöglicht GameKit zwei Arten von Verbindungen: in der näheren Umgebung (über Bluetooth) und online (über das Internet und WiFi). In dem Code der ersten Rezepte dieses Kapitels werden ausschließlich Verbindungen in der Umgebung verwendet, doch gegen Ende des Kapitels finden Sie auch ein Rezept mit Online-Verbindungen. Die Kommunikation über das Internet ist komplizierter und zurzeit nur minimal dokumentiert. Dagegen sind die Möglichkeiten für Bluetooth-Verbindungen in der Umgebung bequem, leicht zu verwenden und stehen zur Aufnahme in Ihre Anwendungen bereit.

Den Peer-Picker anzeigen

Der folgende Code weist einen neuen Peer-Picker-Controller zu und zeigt ihn an, wobei sein Verbindungsformat auf `Nearby` (also die nähere Umgebung) gesetzt wird. Dabei wird der (weiter hinten in diesem Kapitel behandelte) optionale Interaktionsschritt übergangen, in dem der Benutzer zwischen dem Online- und dem Umgebungsmodus wählt. Wird der Picker dargestellt, zeigt er die Oberfläche aus *Abbildung 12.2*. Um GameKit-Sitzungen einzurichten, müssen Sie keinen Peer-Picker verwenden. Das iPhone SDK erlaubt jetzt auch, eigene Schnittstellen für die Arbeit mit den zugrunde liegenden GameKit-Verbindungen zu erstellen. Ein Beispiel dafür befindet sich im Beispielcode zu diesem Kapitel.

```
// Erstellt einen neuen Peer-Picker und zeigt ihn an
GKPeerPickerController *picker = [[GKPeerPickerController alloc]
   init];
picker.delegate = self;
picker.connectionTypesMask = GKPeerPickerConnectionTypeNearby;
[picker show];
```

► *Abbildung 12.2: Dies ist der erste Bildschirm, den die Benutzer bei Peer-to-Peer-Verbindungen über Bluetooth sehen.*

Wenn die Maske auch den Online-Typ umfasst (`GKPeerPickerConnectionTypeOnline`), fragt der Picker den Benutzer zunächst, welche Art von Verbindung er aufbauen möchte, bevor er zu der Oberfläche für Umgebungsverbindungen aus *Abbildung 12.2* oder zu einer Online-Schnittstelle übergeht, die Sie selbst erstellen müssen.

Den Verbindungsaufbau abbrechen

Benutzer können die Peer-Picker-Benachrichtigung abbrechen. In diesem Fall empfängt die Delegierung den Callback `peerPickerControllerDidCancel:`. Wenn Sie in Ihrer Anwendung eine Schaltfläche für die Verbindung anbieten, müssen Sie dafür sorgen, dass sie an diesem Punkt wiederhergestellt wird, damit der Benutzer es erneut versuchen kann.

Das Sitzungsobjekt erstellen

Die Picker-Delegierung muss auf Anforderung ein Sitzungsobjekt liefern. Sitzungen stellen eine abstrakte Klasse bereit, die ein Daten-Socket zwischen den Geräten erstellt und verwaltet. Sie gehören zur Klasse `GKSession` und müssen mit einer Sitzungs-ID initialisiert werden. Dabei handelt es sich um einen eindeutigen String, der verwendet wird, um den Bonjour-Dienst zu erstellen und zwei iPhone-Geräte (Peers), die den gleichen Service ankündigen um sich miteinander zu verbinden. Wenn Sie den Anzeigenamen auf `nil` setzen, wird in der Sitzung der eingebaute Gerätename verwendet.

```
- (GKSession *)peerPickerController:(GKPeerPickerController *)picker
    sessionForConnectionType:(GKPeerPickerConnectionType)type
{
    // Erstellt eine neue Sitzung, falls keine vorhanden ist
    if (!self.session) {
        self.session = [[GKSession alloc] initWithSessionID:
            (self.sessionID ? self.sessionID : @"Sample Session")
            displayName:nil sessionMode:GKSessionModePeer];

        self.session.delegate = self;
    }
    return self.session;
}
```

Diese Methode ist zwar optional, doch sollten Sie sie implementieren, damit Sie die Sitzungs-ID und den Sitzungsmodus festlegen können. Wenn der Peer-Picker ein anderes iPhone- oder iPod-Gerät erkennt, das einen Dienst mit derselben ID ankündigt, zeigt er diesen Peer wie in *Abbildung 12.3* an.

Es kann nur Sekunden, aber auch mehrere Minuten dauern, bis die Peer-Picker-Liste erscheint. Bei der Entwicklung sollten Sie dem Bonjour-Netzwerk-Stack die Gelegenheit geben, alle vorherigen Sitzungen zu entfernen, während Sie durch den Code iterieren. Das ist gewöhnlich der Grund für die langen Verzögerungen. Apple empfiehlt, das Debugging stets nach einem sauberen Neustart durchzuführen. Wenn die Debug-Verzögerungen ein erträgliches Maß übersteigen, sollten Sie neu starten.

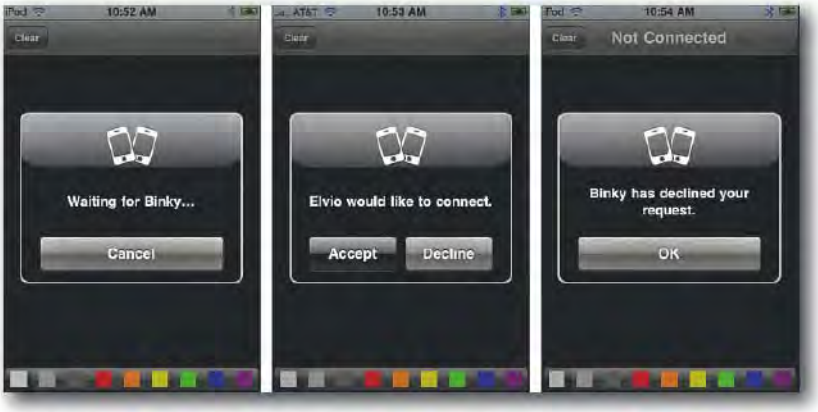
Im normalen Betrieb betragen die Verzögerungen bei der Verbindung gewöhnlich höchstens 45 Sekunden. Weisen Sie die Benutzer darauf hin, dass sie sich gedulden sollen. In *Abbildung 12.3* trägt das zweite iPhone mit der Anwendung den Namen »Binky«. Wenn der Benutzer auf diesen Namen tippt, geht sein iPhone automatisch in den Clientmodus, Binky dagegen geht in den Servermodus.

Client- und Servermodus

Wenn ein Gerät in den Clientmodus wechselt, kündigt es seinen Dienst nicht mehr an. Auf der Servereinheit ändert sich das Dialogfeld zur Auswahl eines iPhone- oder iPod touch-Geräts, das Sie in *Abbildung 12.3* gesehen haben. Der Pee-Nname des Clients wird abgedunkelt, und darunter erscheinen die Worte »is not available« (»ist nicht verfügbar«). Wenige Sekunden später (was allerdings auch bis zu einer Minute dauern kann, weshalb Sie die Benutzer auf diese Verzögerung hinweisen sollten) aktualisieren beide Geräte ihre Peer-Picker-Anzeige.

Abbildung 12.4 zeigt den Peer-Picker auf dem Server und auf dem Client während dieses Vorgangs. Der Client wartet, während der Server die Verbindungsanforderung empfängt (links). Auf dem Server muss der Benutzer die Verbindung akzeptieren oder ablehnen (Mitte). Lehnt er ab, wird der Clientbenutzer mit einem aktualisierten Peer-Picker darüber informiert (rechts). Wird die Verbindung dagegen angenommen, empfangen beide Delegierungen einen neuen Callback.

▸ *Abbildung 12.3: Der Peer-Picker führt alle Geräte auf, die als Peers fungieren können.*



▶ *Abbildung 12.4: Bei der Auswahl eines Partners geht der Client in den Wartemodus über (links), während der Benutzer des Servers entscheidet, ob er die Verbindung annehmen oder ablehnen soll (Mitte). Bei einer Ablehnung erhält der Client eine entsprechende Benachrichtigung (rechts).*

Der Delegierungs-Callback sorgt dafür, dass die Peers die Peer-Picker entfernen und ihre Datenhandler einrichten. Achten Sie darauf, den Picker zu diesem Zeitpunkt freizugeben.

```
- (void)peerPickerController:(GKPeerPickerController *)picker
    didConnectPeer:(NSString *)peerID
    toSession: (GKSession *) session
{
    // Entfernt den Picker und gibt ihn frei, legt den Datenhandler fest
    [picker dismiss];
    [picker release];
    [self.session setDataReceiveHandler:self withContext:nil];
    isConnected = YES;
}
```

12.1.6 Daten senden und empfangen

Der Datenhandler (in diesem Fall `self`) muss die Methode `receiveData:fromPeer:inSession:context:` implementieren. Für die an diese Methode gesendeten Daten wird ein `NSData`-Objekt verwendet. Es gibt keine Hooks oder Handles für den Empfang und die Verarbeitung von Teildaten. Da die Daten in einem Stück ankommen, sollten Sie ihre Menge klein halten (unter 1000 Bytes), damit zwischen den Anwendungen eine schnelle Interaktion möglich bleibt.

```
- (void) receiveData:(NSData *)data fromPeer:(NSString *)peer
    inSession: (GKSession *)session context:(void *)context
{
    // Hier werden die Daten verarbeitet
}
```

Daten senden Sie über das Sitzungsobjekt, wobei eine zuverlässige, aber auch eine unzuverlässige Übertragung möglich ist. Bei der zuverlässigen Variante erfolgt eine Fehlerüberprüfung. Die Übertragung wird dann so lange versucht, bis die Daten korrekt gesendet wurden. Durch die TCP-Übermittlung wird sichergestellt, dass alle Elemente in der Sendereihenfolge ankommen. Bei der unzuverlässigen Übertragung werden die Daten nur ein einziges Mal ohne Wiederholung per UDP gesendet, wobei sie auch in einer anderen Reihenfolge ankommen können. Wenn Sie eine korrekte Zustellung der Daten brauchen, verwenden Sie die zuverlässige Übertragung (`GKSendDataReliable`). Bei kleinen Datenmengen, die fast unmittelbar ankommen müssen, greifen Sie dagegen auf die unzuverlässige Variante zurück.

```
- (void) sendDataToPeers: (NSData *) data
{
   // Sendet die Daten und prüft, ob der Vorgang erfolgreich war
   NSError *error;
   BOOL didSend = [self.session sendDataToAllPeers:data
      withDataMode:GKSendDataReliable error:&error];
   if (!didSend)
      NSLog(@"Error sending data to peers: %@",
         [error localizedDescription]);
}
```

Der einzige Fehler, dem Sie hierbei begegnen können, resultiert gewöhnlich daraus, dass bei der zuverlässigen Übertragung zu viele Daten in die Warteschlange gestellt wurden. Dies führt zu dem Fehler »Puffer voll«.

12.1.7 Statusänderungen

Der folgende Delegierungs-Callback für Sitzungen meldet, wenn sich der Status eines Peers ändert. Die beiden möglichen Zustände, um die es hier geht, sind der verbundene Status (die Verbindung ist endgültig hergestellt, nachdem der Peer-Picker entfernt wurde) und der getrennte Status (der andere Benutzer beendet die Anwendung, löst die Verbindung manuell oder bewegt sich außerhalb der Reichweite).

```
- (void)session:(GKSession *)session peer:(NSString *)peerID
   didChangeState:(GKPeerConnectionState)state
{
   /* STATES:
      GKPeerStateAvailable, = 0,
      GKPeerStateUnavailable, = 1,
      GKPeerStateConnected, = 2,
      GKPeerStateDisconnected, = 3,
      GKPeerStateConnecting = 4 */

   if (state == GKPeerStateConnected)
   {
      // Verarbeitet den verbundenen Status
   }
```

```
    if (state == GKPeerStateDisconnected)
    {
        // Verarbeitet die Trennung der Verbindung
    }
}
```

Um eine Sitzung zwangsweise zu beenden, verwenden Sie die Methode `disconnectFromAllPeers`:

```
- (void) disconnect
{
    // Trennt die Verbindung und setzt die Eigenschaft session zurück
    [self.session disconnectFromAllPeers];
    self.session = nil;
}
```

12.1.8 Eine GameKit-Hilfsklasse erstellen

Rezept 12.1 bündelt den gesamten Vorgang der Peer-Verbindung in eine vereinfachte Hilfsklasse, die Ihnen die meisten Einzelheiten der GameKit-Verbindungen und der Datenübertragung abnimmt. Gleichzeitig veranschaulicht sie auch, wie Sie diese Funktionen nutzen können, und – was noch wichtiger ist – zerlegt den Vorgang in seine beiden Hauptbestandteile, nämlich die Verbindungsaufnahme und die Datenübertragung.

Verbindung aufnehmen

Jeder GameKit-Client, den Sie schreiben, muss passend auf den aktuellen Verbindungsstatus reagieren. Sie müssen in der Lage sein, eine solche Verbindung aufzubauen und zu reagieren, wenn sie aktiv wird oder ausfällt. Die hier vorgestellte Klasse bietet sowohl `connect`- als auch `disconnect`-Anforderungen. Zur Überwachung von Verbindungen gehört vor allem, einen booleschen Statuswert umzuschalten (`isConnected`) und jegliche Schaltflächen zu aktualisieren, mit denen sich der Zustand ändern lässt.

Um diese Aktualisierungen zu vereinfachen, können Sie in dieser Klasse einen Ansichtscontroller zuweisen (über die Eigenschaft `viewController`), wobei die Navigationsschaltfläche auf der rechten Seite automatisch aktualisiert wird. Zu Anfang handelt es sich um eine Schaltfläche zum Herstellen der Verbindung (**Connect**), doch wenn der Benutzer darauf tippt, verschwindet sie, bis der Benutzer den Vorgang abbricht oder die Verbindung vollständig aufgebaut ist. Besteht eine Verbindung, wird die Schaltfläche zu **Disconnect** aktualisiert und stellt jetzt einen Callback zur Methode `disconnect` der Hilfsklasse bereit.

Daten übertragen

Die Klasse `GameKitHelper` kümmert sich um die Einzelheiten des Verbindungsstatus, sodass Sie sie einsetzen können, um einfache GameKit-Anwendungen zu erstellen. Die Handhabung der Daten jedoch bleibt Ihnen überlassen. Betrachten Sie das folgende Codefragment. Es zeigt die gesamte Implementierung des Ansichtscontrollers für eine Chat-Anwendung und veranschaulicht die Datenübertragungsmethoden dafür.

```
@implementation TestBedViewController
- (void)textViewDidChange:(UITextView *)textView
{
   // Nimmt Aktualisierungen nur vor, wenn eine Verbindung besteht
   if (![GameKitHelper sharedInstance].isConnected) return;

   NSString *text = sendView.text;

   // Prüft, ob tatsächlich Text vorhanden ist. Wenn nein, wird eine
   // besondere Löschanforderung gesendet.
   if (!text || (text.length == 0)) text = @"xyzzyclear";
   NSData *textData = [text dataUsingEncoding:NSUTF8StringEncoding];
   [GameKitHelper sendData:textData];
}

-(void) receivedData: (NSData *) data
{
   NSString *text = [[[NSString alloc] initWithData:data
      encoding:NSUTF8StringEncoding] autorelease];

   // Prüft beim Aktualisieren des Textes, ob eine Löschanforderung
   // vorliegt
   receiveView.text = [text isEqualToString:@"xyzzyclear"] ?
      @"" : text;
}

- (void) clear
{
   // Verarbeitet die Löschanforderung
   sendView.text = @"";
}

- (void) viewDidLoad
{

   self.navigationController.navigationBar.tintColor =

      COOKBOOK_PURPLE_COLOR;
   self.navigationItem.leftBarButtonItem = BARBUTTON(@"Clear",
      @selector(clear));

   // Initialisiert die Hilfsklasse
   [GameKitHelper sharedInstance].sessionID = @"Typing Together";
```

```
    [GameKitHelper sharedInstance].dataDelegate = self;
    [GameKitHelper assignViewController:self];

    // Zeigt die Tastatur an
    [sendView becomeFirstResponder];
}
@end
```

Wie Sie sehen, überwacht diese Anwendung eine Ansicht für zu sendenden Text, und wenn sich diese ändert (da der Benutzer Eingaben macht), sendet sie den Inhalt der Ansicht via GameKit an einen Peer. Gleichzeitig wartet sie auch auf Daten, und wenn Sie sie empfängt, aktualisiert sie die Ansicht für empfangenen Text, um zu zeigen, was der Benutzer des Peer-Geräts geschrieben hat. Die Schaltfläche **Clear** löscht den Text in der Sendeansicht.

Diese Anwendung veranschaulicht den zweiten Teil des GameKit-Vorgangs, nämlich die Handhabung der Daten. Die Hilfsklasse aus *Rezept 12.1* erstellt ein Datendelegierungsprotokoll, das der hier gezeigte Ansichtscontroller für den Text-Chat abonniert. Die Daten werden über die benutzerdefinierte Delegierungsmethode `receivedData:` übergeben, sodass die Ansicht für empfangenen Text mit dem Text aktualisiert werden kann, der auf dem Peer-Gerät eingegeben wurde.

Auf ähnliche Weise übergibt die Delegierungsmethode `textViewDidChange:` der Textansicht die Verantwortung für die Übertragung des Textes an die Klasse `GameKitHelper`: Sie ruft die Methode `sendData:` auf, um die Daten an die angeschlossenen Peers zu übermitteln.

Hinweis

Das Problem, dass Pakete in falscher Reihenfolge empfangen werden, wird in *Rezept 12.1* nicht angesprochen. Ein Beispiel für die Handhabung von Netzwerkpaketen finden Sie im `GKTank`-Beispielcode von Apple. Dieser Code untersucht den Zeitstempel und die IDs der Pakete, um dafür zu sorgen, dass sie in der richtigen Reihenfolge verarbeitet werden.

Die Hilfsklasse

Rezept 12.1 enthält die Implementierung der Klasse `GameKitHelper`. Der zugehörige Beispielcode zu diesem Rezept zeigt die Klasse in Aktion und erstellt die zuvor besprochene Text-Chat-Anwendung. Die Klasse wurde für eine Wiederverwendung entworfen und lässt sich sehr leicht von der Text-Chat-Anwendung entkoppeln und für andere Zwecke nutzen, wie Sie im nächsten Rezept sehen.

```
@implementation GameKitHelper
@synthesize dataDelegate;
@synthesize viewController;
@synthesize sessionID;
@synthesize session;
@synthesize isConnected;
```

```
// Das Makro hilft dabei, die Selektoren für die Datendelegierungs-
// Callbacks zu prüfen und dann zu senden
#define DO_DATA_CALLBACK(X, Y) if (self.dataDelegate && \
   [self.dataDelegate respondsToSelector:@selector(X)]) \
   [self.dataDelegate performSelector:@selector(X) withObject:Y];

#pragma mark Shared Instance

static GameKitHelper *sharedInstance = nil;
+ (GameKitHelper *) sharedInstance
{
   if(!sharedInstance) sharedInstance = [[self alloc] init];
      return sharedInstance;
}

#pragma mark Data Sharing

// Sendet Daten an alle angeschlossenen Peers
- (void) sendDataToPeers: (NSData *) data
{
   NSError *error;
   BOOL didSend = [self.session sendDataToAllPeers: data
      withDataMode:GKSendDataReliable error:&error];
   if (!didSend)

      NSLog(@"Error sending data to peers: %@",

         [error localizedDescription]);
   DO_DATA_CALLBACK(sentData:,
         (didSend ? nil : [error localizedDescription]));
}

// Leitet den Datenempfang an die Datendelegierung um
- (void) receiveData:(NSData *)data fromPeer:(NSString *)peer
   inSession: (GKSession *)session context:(void *)context
{
   DO_DATA_CALLBACK(receivedData:, data);
}

#pragma mark Connections

// Beginnt durch Anzeige eines Peer-Pickers eine neue Verbindung
- (void) startConnection
{
```

```
    if (!self.isConnected)
    {
        GKPeerPickerController *picker = [[GKPeerPickerController
            alloc] init];
        picker.delegate = self;
        picker.connectionTypesMask = GKPeerPickerConnectionTypeNearby;
        [picker show];
        if (self.viewController)
            self.viewController.navigationItem.rightBarButtonItem =
                nil;
    }
}

// Entfernt den Peer-Picker beim Abbrechen des Vorgangs
- (void) peerPickerControllerDidCancel:
    (GKPeerPickerController *)picker
{
    [picker release];
    if (self.viewController)
        self.viewController.navigationItem.rightBarButtonItem =
            BARBUTTON(@"Connect", @selector(startConnection));
}

// Richtet bei erfolgreichem Verbindungsaufbau den Datenhandler ein
- (void)peerPickerController:(GKPeerPickerController *)picker
    didConnectPeer:(NSString *)peerID
    toSession: (GKSession *) session
{
    [picker dismiss];
    [picker release];
    [self.session setDataReceiveHandler:self withContext:nil];

    isConnected = YES;
    DO_DATA_CALLBACK(connectionEstablished, nil);
}

// Stellt Sitzungsinformationen wie ID und Modus bereit
- (GKSession *)peerPickerController:(GKPeerPickerController *)picker
    sessionForConnectionType:(GKPeerPickerConnectionType)type
{
    if (!self.session) {
        self.session = [[GKSession alloc] initWithSessionID:
            (self.sessionID ? self.sessionID : @"Sample Session")
            displayName:nil sessionMode:GKSessionModePeer];
        self.session.delegate = self;
    }
```

```
    return self.session;
}

#pragma mark Session Handling

// Trennt die aktuelle Sitzung
- (void) disconnect
{
    [self.session disconnectFromAllPeers];
    self.session = nil;
}

// Erkennt Statusänderungen des anderen Peers
- (void)session:(GKSession *)session peer:(NSString *)peerID
    didChangeState:(GKPeerConnectionState)state
{

    NSArray *states = [NSArray arrayWithObjects:@"Available",

        @"Unavailable", @"Connected", @"Disconnected", @"Connecting", nil];

    NSLog(@"Peer state is now %@", [states objectAtIndex:state]);


    if (state == GKPeerStateConnected)
    {
        if (self.viewController)
            self.viewController.navigationItem.rightBarButtonItem =
                BARBUTTON(@"Disconnect", @selector(disconnect));
    }

    if (state == GKPeerStateDisconnected)
    {
        self.isConnected = NO;
        showAlert(@"Lost connection with peer. You are no longer \
            connected to another device.");
        [self disconnect];
        if (self.viewController)
            self.viewController.navigationItem.rightBarButtonItem =
                BARBUTTON(@"Connect", @selector(startConnection));

        DO_DATA_CALLBACK(connectionLost, nil);
    }
}
```

```
// Hilfsmethode zur Einrichtung des Ansichtscontrollers
- (void) assignViewController: (UIViewController *) aViewController
{
   self.viewController = aViewController;
   self.viewController.navigationItem.rightBarButtonItem =
      BARBUTTON(@"Connect", @selector(startConnection));
}

#pragma mark Class utility methods

// Diese Klassenmethoden leiten zu Instanzmethoden um.
// Sie dienen hier nur der Bequemlichkeit.
+ (void) connect
{
   [[self sharedInstance] startConnection];
}

+ (void) disconnect
{
   [[self sharedInstance] disconnect];
}

+ (void) sendData: (NSData *) data
{
   [[self sharedInstance] sendDataToPeers:data];
}

+ (void) assignViewController: (UIViewController *) aViewController
{
   [[self sharedInstance] assignViewController:aViewController];
}
@end
```

▶ *Rezept 12.1: Die Klasse* `GameKitHelper`

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 12 und öffnen das Projekt zu diesem Rezept.

12.2 Rezept: Hinter die Kulissen schauen

Zurzeit protokolliert GameKit während der Ausführung seine Statusinformationen, hauptsächlich durch `NSLog`-Aufrufe, die die Ingenieure von Apple vorgesehen haben. Diese Informationen können Sie an der Debug-Konsole einsehen, aber mit folgendem Trick auch in eine Datei umleiten (sie werden dann nicht an der Konsole ausgegeben) und dann in einer Textansicht Ihrer Anwendung anzeigen. In *Rezept 12.2* wird der Standard-C-Aufruf `freopen()` verwendet, um `stderr`-Daten (wie sie von `NSLog()` hervorgerufen werden) in eine Datei umzuleiten. Anschließend wird eine `NSTimer`-Instanz eingerichtet, um diese Datei zu überwachen. Wenn sich der Inhalt der Datei ändert, wird die Textansicht mit der betreffenden Ausgabe aktualisiert. Eine solche Umleitung können Sie bei GameKit und bei jeder anderen Anwendung einsetzen, die irgendeine Form von Konsolenausgabe hervorruft.

Beachten Sie, wie in diesem Rezept der Inhaltsoffset für die Textansicht aktualisiert wird. Hierbei wird sichergestellt, dass der Text am unteren Rand der Ansicht nach einer Aktualisierung stets angezeigt wird. Dazu wird der Offset auf die vollständige Größe des Inhalts abzüglich der Seitenhöhe gesetzt.

```
@implementation TestBedViewController

- (void) listenForStderr: (NSTimer *) timer;
{
   // Überprüft die stderr-Ausgabe auf neue Informationen
   NSString *contents = [NSString
      stringWithContentsOfFile:STDERR_OUT encoding:NSUTF8StringEncoding

      error:NULL];
   contents = [contents stringByReplacingOccurrencesOfString:@"\n"
      withString:@"\n\n"];
   if ([contents isEqualToString:textView.text]) return;
   [textView setText:contents];
   stextView.contentOffset = CGPointMake(0.0f,
      MAX(textView.contentSize.height -
      textView.frame.size.height, 0.0f));
}

- (void) viewDidLoad
{

   self.navigationController.navigationBar.tintColor =

      COOKBOOK_PURPLE_COLOR;
   // Richtet die GameKit-Sitzung ein
   [GameKitHelper sharedInstance].sessionID = @"Peeking at GameKit";
   [GameKitHelper assignViewController:self];
```

```
    // Leitet die stderr-Ausgabe in eine Datei um
    freopen([STDERR_OUT fileSystemRepresentation], "w", stderr);
    [NSTimer scheduledTimerWithTimeInterval:1.0f target:self
        selector:@selector(listenForStderr:) userInfo:nil repeats:YES];
}
@end
```

▶ *Rezept 12.2: GameKit überwachen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 12 und öffnen das Projekt zu diesem Rezept.

12.3 Rezept: Komplexe Daten mit GameKit übertragen

Einfache Strings über GameKit zu übertragen wie in *Rezept 12.1* ist nur der Anfang. Schon bald werden Sie zu komplexeren Objekten und Daten übergehen müssen. Eigenschaftenlisten bieten eine gute Möglichkeit, um benutzerdefinierte Objekte zu übertragen, da sie sich leicht in `NSData`-Objekte serialisieren und wieder zurückverwandeln lassen.

Eigenschaftenlisten bilden einen praktischen abstrakten Datentyp. Ein Eigenschaftenlistenobjekt kann auf Daten (`NSData`), Strings (`NSString`), Arrays (`NSArray`), Dictionarys (`NSDictionary`), Datumsangaben (`NSDate`) und Zahlen (`NSNumber`) zeigen. Bei Auflistungsobjekten (also Arrays und Dictionarys) müssen alle Member und Schlüssel ebenfalls Eigenschaftslistenobjekte sein, also Daten, Strings, Zahlen, Datumsangaben oder eingebettete Arrays und Dictionarys.

Das mag zwar als Einschränkung erscheinen, doch können Sie die meisten Strukturen und Objekte in Strings umwandeln und umgekehrt. Beispielsweise können Sie dazu die eingebauten Funktionen `NSStringFromCGPoint()` und `NSStringFromClass()` verwenden oder Ihre eigenen schreiben. Das folgende Methodenpaar erweitert die Klasse `UIColor` und bietet den Funktionsumfang, um Farbinformationen als Strings über GameKit-Verbindungen zu senden.

```
@implementation UIColor (utilities)
- (NSString *) stringFromColor
{
    // Ruft den Farbraum ab und speichert eine RGB- oder monochrome Farbe
    const CGFloat *c = CGColorGetComponents(self.CGColor);
    CGColorSpaceModel csm =
        CGColorSpaceGetModel(CGColorGetColorSpace(self.CGColor));
    return (csm == kCGColorSpaceModelRGB) ?
```

```
        [NSString stringWithFormat:@"%0.2f %0.2f %0.2f %0.2f",
            c[0], c[1], c[2], c[3]] :
        [NSString stringWithFormat:@"%0.2f %0.2f %0.2f %0.2f",
            c[0], c[0], c[0], c[1]];
}

+ (UIColor *) colorWithString: (NSString *) colorString
{
    // Liest die Farbe aus einem String
    const CGFloat c[4];
    sscanf([colorString cStringUsingEncoding:NSUTF8StringEncoding],
        "%f %f %f %f", &c[0], &c[1], &c[2], &c[3]);
    return [UIColor colorWithRed:c[0] green:c[1] blue:c[2] alpha:c[3]];
}
@end
```

Liegen die Daten erst einmal in Form einer Eigenschaftenliste vor, können Sie sie serialisieren und in einem Stück senden. Nach dem Empfang sind die deserialisierten Daten zur weiteren Verwendung bereit. *Rezept 12.3* zeigt die Methoden `transmit` und `receivedData:`, die diesen Vorgang erledigen. Der Code stammt aus einer Beispielanwendung, die eine Folge von gezeichneten Punkten (wie in *Rezept* 8.9) und die dazu verwendete Farbe in einem `NSDictionary`-Objekt speichert. Außer der hier gezeigten Klasse `NSPropertyListSerialization` können Sie auch `NSKeyedArchiver` und `NSKeyedUnarchiver` verwenden.

Da die Punkte und Farben als Strings gespeichert werden, können die Daten leicht in eine Form umgewandelt werden, die sich besser für die Übertragung via GameKit eignet. Der Quellcode zu diesem Kapitel veranschaulicht diese Methoden am Beispiel eines Programms für gemeinschaftliches Zeichnen, bei dem die Übertragung von Eigenschaftenlisten genutzt wird.

```
- (void) transmit
{
    if (![GameKitHelper sharedInstance].isConnected) return;
    NSString *errorString;

    // Sendet die Punkte, wenn vorhanden
    if (!self.points)
        [GameKitHelper sendData:[@"clear"
            dataUsingEncoding:NSUTF8StringEncoding]];
    else
    {
        // Sendet eine Kopie der lokalen Punkte an den Peer, indem die
        // Eigenschaftenliste in eine Datenfolge serialisiert wird
        NSData *plistdata = [NSPropertyListSerialization
            dataFromPropertyList:self.points
            format:NSPropertyListXMLFormat_v1_0
            errorDescription:&errorString];
```

```
        if (plistdata)
            [GameKitHelper sendData:plistdata];
        else
            CFShow(errorString);
    }
}

- (void) receivedData: (NSData *) thedata
{
    // Überprüft auf "clear"
    NSString *string = [[[NSString alloc] initWithData:thedata
        encoding:NSUTF8StringEncoding] autorelease];
    if ([string isEqualToString:@"clear"])
    {
        self.foreignPoints = [NSMutableArray array];
        self.points = [NSMutableArray array];
        [self setNeedsDisplay];
        return;
    }

    // Deserialisiert die Daten zurück in eine Eigenschaftenliste
    CFStringRef errorString;
    CFPropertyListRef plist =
        CFPropertyListCreateFromXMLData(kCFAllocatorDefault,
        (CFDataRef)thedata, kCFPropertyListMutableContainers,
        &errorString);

    if (!plist)
    {
        CFShow(errorString);
        return;
    }

    // Weist die empfangenen Daten zu foreignPoints zu
    self.foreignPoints = (NSArray *)plist;

    // Behandelt den Fall einer ungeeigneten Löschung
    if (self.foreignPoints.count == 0)
        self.points = [NSMutableArray array];
    [self setNeedsDisplay];
}
```

► *Rezept 12.3: Eigenschaftenlisten serialisieren und deserialisieren*

Den Rezeptcode finden

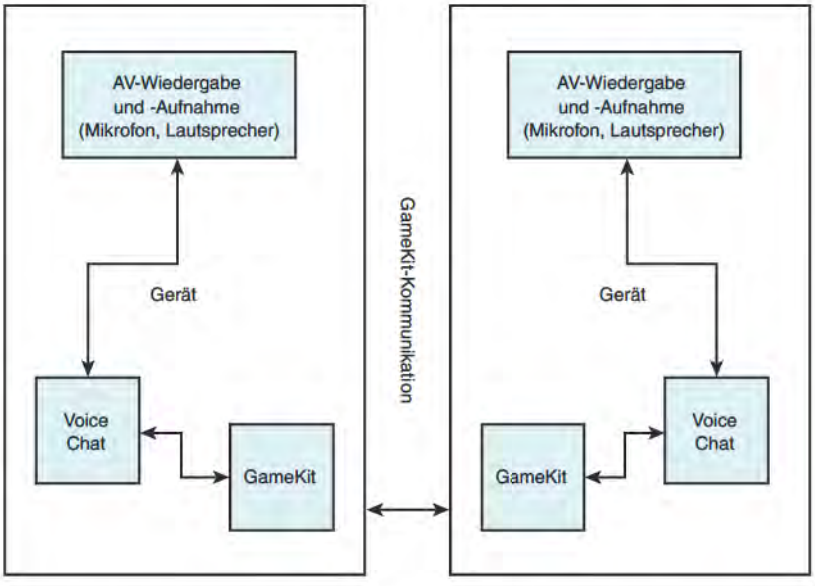
Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 12 und öffnen das Projekt zu diesem Rezept.

12.4 Rezept: Der Voice Chat-Dienst von GameKit

Mit dem In-Game-Voice-Chat-Dienst von GameKit können Anwendungen zwischen zwei Geräten einen Sprachübertragungskanal aufbauen, der der Kommunikation über Walkie-Talkies ähnelt. Diesen Dienst können Sie unter Verwendung des eingebauten Lautsprechers und Mikrofons auf einem iPhone nutzen und nach Anschluss eines externen Mikrofons auch auf iPod touch-Geräten der zweiten Generation. Die Standardkopfhörer des iPhones mit dem eingebauten Mikrofon eignen sich auch sehr gut für den iPod touch. Hierbei wird der Ton in die Kopfhörer geleitet und die Spracheingabe vom Mikrofon abgegriffen.

12.4.1 GameKit als Vermittler

Die in *Rezept 12.1* vorgestellte Klasse `GameKitHelper` bietet den üblichen Funktionsumfang für GameKit-Verbindungen. Um diesen Code für Voice Chat anzupassen, müssen Sie sich die GameKit-Kommunikation als Sprachübertragung über einen Vermittler vorstellen. GameKit kümmert sich um den Datendurchsatz sowohl beim Empfang als auch beim Senden.



► *Abbildung 12.5: Voice Chat fügt eine Schicht außerhalb der normalen GameKit-Kommunikation hinzu, um die Live-Übertragung von Audiodaten zwischen den Peers zu ermöglichen.*

Die Ergänzungen zur Sprachübertragung, die die Klasse `GKVoiceChatService` bietet, befinden sich außerhalb des normalen GameKits. Die Chat-Dienste klinken sich in das System des iPhones zur Audiowiedergabe und -aufnahme ein, sodass sie Ton empfangen und abspielen können. Voice Chat sendet seine Daten über GameKit und gibt die gleichfalls über GameKit empfangenen Daten wieder. *Abbildung 12.5* zeigt, wie die Verantwortlichkeiten aufgeteilt sind.

Leider können Sie den Voice Chat-Dienst von GameKit nur über eine Bluetooth-Verbindung nutzen. `GKVoice` erwartet eine `GKSession`-Sitzung mit `GKPeers`, um Daten übertragen zu können. Wenn Sie eine Sprachübertragung für andere Arten von Verbindungen brauchen, müssen Sie die Schicht selbst schreiben.

12.4.2 Voice Chat implementieren

Bei der Sprachübertragung fangen Sie genauso an wie bei anderen GameKit-Verbindungen: Sie zeigen einen Peer-Picker an und handeln wie üblich die Verbindung aus. Die Unterschiede zeigen sich erst, wenn die Peers Verbindung aufnehmen. Jetzt müssen Sie den Voice Chat einrichten und die Daten zu und von diesem Dienst umleiten.

Bei der Verbindung mit einem Peer müssen Sie die Grundlagen für den Voice Chat einrichten. Die Peer-Verbindungsmethode aus *Rezept 12.4* aktiviert eine Audiositzung mit Aufnahme und Wiedergabe, legt den Standardclient für den Chat-Dienst fest und startet einen neuen Voice Chat mit dem Peer. Über die Eigenschaft `client` legen Sie fest, dass die Klasse die erforderlichen Voice Chat-Callbacks für die Aushandlung der Datenübertragung empfängt.

Dazu muss die Hauptklasse das Protokoll `GKVoiceChatClient` deklarieren. Wenn der Chat-Dienst Daten über das Mikrofon erfasst, löst er den Callback `voiceChatService:sendData:toParticipantID:` aus. An dieser Stelle können Sie die Sprachübertragungsdaten zur normalen GameKit-Sitzung umleiten. Bei einer reinen Sprachverbindung senden Sie einfach nur die Daten. Verarbeitet Ihre Anwendung aber sowohl Sprach- als auch andere Daten, erstellen Sie ein Dictionary und kennzeichnen die Daten mit einem Schlüssel wie `@"voice"`. Wenn Ihre Klasse Daten über den normalen Callback `receiveData:fromPeer:inSession:context:` empfängt, gilt derselbe Ansatz.

Für die reine Sprachübertragung verwenden Sie `receivedData:fromParticipantID:`, um Daten zum Chat-Dienst zu senden. Bei Voice Chat können Sie den Ton eines Spiels mit einer Sprachübertragung während des Spiels mischen. Für Hybridanwendungen mit Sprach- und Datenübertragung deserialisieren Sie die Daten, überprüfen, ob das Paket Sprach- oder reguläre Daten enthält, und leiten die Daten dann an den entsprechenden Empfänger weiter.

```
- (void)voiceChatService:(GKVoiceChatService *)voiceChatService
   sendData:(NSData *)data toParticipantID:(NSString *)participantID
{
   // Sendet die nächste Datenmenge an die Peers
   [self.session sendData: data toPeers:[NSArray arrayWithObject:
      participantID] withDataMode: GKSendDataReliable error: nil];
}
```

```
- (void) receiveData:(NSData *)data fromPeer:(NSString *)peer inSession:
   (GKSession *)session context:(void *)context
{
   // Leitet alle Sprachdaten an den Voice Chat-Dienst weiter
   [[GKVoiceChatService defaultVoiceChatService]
      receivedData:data fromParticipantID:peer];
}

- (NSString *)participantID
{
   // Stellt die Teilnehmer-ID der Sitzung für den Chat bereit
   return self.session.peerID;
}

- (void)peerPickerController:(GKPeerPickerController *)picker
   didConnectPeer:(NSString *)peerID toSession: (GKSession *) session
{
   // Schließt beim Herstellen der Verbindung den Picker und richtet den
   // Datenhandler ein
   [picker dismiss];
   [picker release];
   isConnected = YES;
   [self.session setDataReceiveHandler:self withContext:nil];

   // Startet die Audiositzung
   NSError *error;
   AVAudioSession *audioSession = [AVAudioSession sharedInstance];

   if (![audioSession setCategory:AVAudioSessionCategoryPlayAndRecord
      error:&error])
   {
      NSLog(@"Error setting the AV play/record category: %@", [error
      localizedDescription]);
      showAlert(@"Could not establish an Audio Connection. Sorry!");
      return;
   }

   if (![audioSession setActive: YES error: &error])
   {
      NSLog(@"Error activating the audio session: %@", [error
         localizedDescription]);
      showAlert(@"Could not establish an Audio Connection. Sorry!");
      return;
   }
```

```
    // Legt den Voice Chat-Client fest und startet den Voice Chat
    [GKVoiceChatService defaultVoiceChatService].client = self;
    if (![[GKVoiceChatService defaultVoiceChatService]
       startVoiceChatWithParticipantID: peerID error: &error])
    {
       showAlert(@"Could not start voice chat. Sorry!");
       NSLog(@"Error starting voice chat");
    }

}
```

► *Rezept 12.4: Voice Chat-Dienste für die Sprachübertragung während eines Spiels hinzufügen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 12 und öffnen das Projekt zu diesem Rezept.

12.5 Rezept: Einen iPhone-Server mit Bonjour erstellen

GameKit stützt sich zwar auf Bonjour, doch manchmal müssen Sie Bonjour auch direkt einsetzen, beispielsweise dann, wenn Sie einen iPhone-Dienst schreiben möchten, der mit einem Client auf einem Macintosh Verbindung aufnimmt. (Bonjour wurde auch auf Windows und Linux portiert, doch werden diese beiden Plattformen in diesem Buch nicht behandelt.)

Apple stellt einen reichhaltigen Fundus an Bonjour-Beispielcode zur Verfügung, der auch in *Rezept 12.5* genutzt wird. Dort werden die vorgefertigten Apple-Klassen `TCPServer` und `TCPConnection` verwendet, um einen Bonjour-Dienst zu veröffentlichen und auf externe Verbindungen zu reagieren.

Der Code baut auf dem einfachen Beispiel zur Bildauswahl aus *Rezept 7.1* auf. Hierbei wird jedoch nicht nur ein Bild ausgewählt, sondern auch über eine Bonjour-Datenverbindung bereitgestellt.

Der Handshake-Vorgang beginnt damit, dass ein neuer `TCPServer` und seine Delegierung eingerichtet werden. Die Methode `viewDidLoad` startet den Dienst mit der aktuellen Ausführungsschleife und kündigt den Dienst `PictureThrow` an. Wenn ein externer Client Kontakt aufnimmt, akzeptiert der Callback `server:didOpenConnection:` diese Verbindung und sendet seine `TCPConnection`-Delegierung. Der Unterschied zwischen der Server- und der Verbindungsdelegierung besteht darin, dass der Server dazu da ist, auf neue Verbindungen zu lauschen, während die Verbindung die Verantwortung für das Senden und Empfangen der Daten hat.

Wie bei GameKit entscheidet der Benutzer, ob er die neue Verbindung akzeptieren möchte. Die Verbindungs-Delegierungsmethode `server:shouldAcceptConnectionFromAddress:` gibt einen booleschen Wert zurück, der bestimmt, ob die Verbindung zugelassen oder verweigert wird. *Abbildung 12.6* zeigt das Dialogfeld, das beim Vorschlagen einer neuen Verbindung in *Rezept 12.5* angezeigt wird.

▸ *Abbildung 12.6: Rezept 12.5 emuliert GameKit und ermöglicht dem Benutzer die Entscheidung, ob er die Remoteverbindung annehmen oder ablehnen will.*

Nach der Annahme der Verbindung empfängt die Verbindungsdelegierung einen `connectionDid➥Open:`-Callback. Hier sendet die Anwendung letzten Endes die Daten an den Client und schließt dann mit `invalidate` die Verbindung. Dadurch ist auf dem Client eine Schaltfläche zum Anfordern von Daten möglich. Jedes Mal, wenn der Benutzer auf diese Schaltfläche tippt, wird eine neue Verbindung und damit eine neue Datenanforderung initialisiert.

Bei den gesendeten Daten handelt es sich um das zurzeit ausgewählte Bild. Der Benutzer kann die Auswahl ändern, indem er auf **Choose Image** tippt (um die normale Bildauswahl zu verwenden) oder auf **Camera** (um ein Foto aufzunehmen).

Wie Sie sehen, ist im Code dieses Rezepts die Handhabung der Bildauswahl weit aufwendiger als der eigentliche Bonjour-Dienst. Für eine vollständige Datenserver-Anbindung sind lediglich einige wenige Methoden und Callbacks erforderlich.

Hinweis

Der Code für die von Apple vorgefertigten Klassen ist im Beispielcode zu diesem Kapitel enthalten.

```
@interface TestBedViewController : UIViewController
   <UINavigationControllerDelegate, UIImagePickerControllerDelegate,
   TCPServerDelegate, TCPConnectionDelegate>
{
   UIImage *image;
   TCPServer *server;
}
```

```
@property (retain) UIImage *image;
@property (retain) TCPServer *server;
@end

@implementation TestBedViewController
@synthesize image;
@synthesize server;

- (void) baseButtons
{
   // Ermöglicht es dem Benutzer, ein Bild auszuwählen oder aufzunehmen
   self.navigationItem.leftBarButtonItem = BARBUTTON(@"Choose Image",
      @selector(pickImage:));
   if ([UIImagePickerController
      isSourceTypeAvailable:UIImagePickerControllerSourceTypeCamera])
      self.navigationItem.rightBarButtonItem = BARBUTTON(@"Camera",
      @selector(snapImage:));
}

- (void)imagePickerController:(UIImagePickerController *)picker
   didFinishPickingMediaWithInfo:(NSDictionary *)info
{
   // Zeigt das ausgewählte Bild und entfernt den Picker
   self.image = [info objectForKey:
      @"UIImagePickerControllerOriginalImage"];
   [(UIImageView *)[self.view viewWithTag:101] setImage:self.image];
   [self dismissModalViewControllerAnimated:YES];
   [picker release];
   [self baseButtons];
}

- (void) imagePickerControllerDidCancel:
   (UIImagePickerController *)picker
{
   // Entfernt beim Abbrechen den Picker
   [self dismissModalViewControllerAnimated:YES];
   [picker release];
   [self baseButtons];
}

- (void) requestImageOfType: (NSString *) type
{
   // Zeigt je nach angetippter Schaltfläche entweder die Kamera oder den
   // Foto-Picker
```

```
    UIImagePickerController *ipc = [[UIImagePickerController alloc]
        init];
    ipc.sourceType = [type isEqualToString:@"Camera"] ?
        UIImagePickerControllerSourceTypeCamera :
        UIImagePickerControllerSourceTypePhotoLibrary;
    ipc.delegate = self;
    ipc.allowsImageEditing = NO;
    [self presentModalViewController:ipc animated:YES];
}

- (void) pickImage: (id) sender
{
    // Startet eine Auswahlsitzung in der Fotobibliothek
    self.navigationItem.leftBarButtonItem = nil;
    self.navigationItem.rightBarButtonItem = nil;
    [self performSelector:@selector(requestImageOfType:)
        withObject:@"Library" afterDelay:0.5f];
}

- (void) snapImage: (id) sender
{
    // Beginnt eine Kamerasitzung
    self.navigationItem.leftBarButtonItem = nil;
    self.navigationItem.rightBarButtonItem = nil;
    [self performSelector:@selector(requestImageOfType)
        withObject:@"Camera" afterDelay:0.5f];
}

- (NSString *) hostname
{
    // Stellt den Hostnamen für das iPhone bereit
    char baseHostName[255];
    int success = gethostname(baseHostName, 255);
    if (success != 0) return nil;
    baseHostName[255] = '\0';
    return [NSString stringWithCString:baseHostName
        encoding:NSUTF8StringEncoding];
}

- (BOOL) server:(TCPServer*)server
    shouldAcceptConnectionFromAddress:(const struct sockaddr*)address
{
    // Erlaubt den Benutzern, Anforderungen abzulehnen. Um alle
    // Verbindungen
```

```
    // zuzulassen, ersetzen Sie diese Stelle durch "return YES"
    return [ModalAlert ask:@"Accept remote connection?"];
}

- (void) connectionDidOpen:(TCPConnection*)connection
{
    // Sendet beim Öffnen der Verbindung die aktuellen Bilddaten an den
    // Client
    printf("Connection did open\n");
    if ([connection sendData:
        UIImageJPEGRepresentation(self.image, 0.75f)])
        printf("Data sent\n");
    [connection invalidate];
}

- (void) server:(TCPServer*)server
    didOpenConnection:(TCPServerConnection*)connection
{
    // Sendet die Delegierung der Verbindung, um den Callback beim Öffnen
    // zu empfangen
    [connection setDelegate:self];
}

- (void) viewDidLoad
{
    // Sucht nach einer WiFi-Verbindung, bevor fortgefahren wird
    NetReachability *nr = [[[NetReachability alloc]
        initWithDefaultRoute:YES] autorelease];
    if (![nr isReachable] || ([nr isReachable] && [nr isUsingCell]))
    {
        [ModalAlert performSelector:@selector(say:) withObject:
            @"This application requires WiFi. Please enable WiFi in\
            Settings and run this application again." afterDelay:0.5f];
        return;
    }

    // Erstellt eine Serverinstanz, die den Bonjour-Dienst bereitstellt
    self.server = [[[TCPServer alloc] initWithPort:0] autorelease];
    [self.server setDelegate:self];
    [self.server startUsingRunLoop:[NSRunLoop currentRunLoop]];
    [self.server enableBonjourWithDomain:@"local"
        applicationProtocol:@"PictureThrow" name:[self hostname]];
```

```
    // Richtet die Standardschaltflächen und das Standardbild ein

    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
    [self baseButtons];
    self.image = [UIImage imageNamed:@"cover320x416.png"];
}
@end
```

▶ *Rezept 12.5: Einen Bonjour-Dienst bereitstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 12 und öffnen das Projekt zu diesem Rezept.

12.6 Rezept: Einen Mac-Client für einen Bonjour-Dienst des iPhones erstellen

Mit nur wenigen Änderungen funktioniert der Bonjour-Beispielcode von Apple sowohl in iPhone- als auch in Macintosh-Anwendungen. Für den Einsatz auf dem Mac müssen Sie alle Verweise auf das Framework `CFNetwork` entfernen und durch das AppKit-Framework für den Mac ersetzen. *Rezept 12.6* erstellt einen Macintosh-Client, um zu zeigen, wie Sie die Clientseite des Bonjour-Beispielcodes verwenden können. Dieser Client kann mit dem Server aus *Rezept 12.5* zusammenarbeiten.

Die Rollen der einzelnen Plattformen sind nicht festgelegt. Es ist auch möglich, dass der Mac einen Dienst bereitstellt und das iPhone als Client fungiert. Während sich ein Server/Client-Paar aus iPhone und iPhone am besten mit GameKit einrichten lässt, zeigt dieses Rezept, wie Sie plattformübergreifend vorgehen und Bonjour für die Kommunikation zwischen Mobil- und Desktop-Geräten nutzen.

Da dieses Buch die iPhone- und nicht die Macintosh-Entwicklung zum Thema hat, ist *Rezept 12.6* auf die Methoden für die Bonjour-Kommunikation beschränkt. Wenn Sie neugierig sind, wie der Rest der Anwendung aufgebaut ist (vor allem, wie Sie die `UIImage`-Daten in ein sauber ausgerichtetes `NSImage` umwandeln), sehen Sie sich den Beispielcode zu diesem Buch an. Studieren Sie auch *Rezept 12.8*, in dem sowohl Server als auch Client für das iPhone implementiert werden. *Abbildung 12.7* zeigt die Clientanwendung auf dem Macintosh, die zur Aufnahme des Screenshots aus *Abbildung 12.6* verwendet wurde.

Als Erstes sucht der Bonjour-Client nach Diensten. Da der Dienst auf dem iPhone von Bonjour und nicht von GameKit bereitgestellt wird, ist der Name schon vor der Kompilierung und den Tests bekannt, nämlich `@"PictureThrow"`. Dieser Name wird auch von dem Server in *Rezept 12.5* verwendet.

Die Klasse `NSNetServiceBrowser` macht es möglich, einen Dienst eines gegebenen Typs zu finden. Ihre Delegierung empfängt den Callback `netServiceBrowser:didFindService:moreComing:`, wenn es eine Übereinstimmung gibt. Die Delegierung kann dann den Dienstbrowser anhalten und mit der Auflösung des Dienstes beginnen.

Dabei wird der Name des Dienstes in eine IP-Adresse umgewandelt. Die schon in *Rezept 12.5* verwendete Klasse `TCPConnection` ermöglicht es dem Bonjour-Client, Daten vom Server abzurufen. Ihr Callback `connection:didReceiveData:` stellt die Daten zu.

Verbindungen können aus drei Gründen geschlossen werden. Erstens kann es sein, dass die Daten erfolgreich übertragen wurden und der Host-Dienst die Verbindung absichtlich beendet. Zweitens kann der Benutzer die Verbindung abgelehnt haben, und drittens kann die Verbindung verloren gegangen sein, weil sie beendet wurde oder das Gerät außerhalb der Reichweite geraten ist. Ein einziger `conectionDidClose:`-Callback behandelt alle drei Fälle.

▸ *Abbildung 12.7: Der hier gezeigte Bonjour-Client für den Macintosh wurde mit dem Bonjour-Server für das iPhone aus Rezept 12.5 verwendet, um einige der Screenshots in diesem Buch erstellen zu können.*

In diesem *Rezept* legt der Callback einen booleschen Wert für `success` fest. Wenn die Verbindung endet, überprüft die Callback-Methode `connectionDidClose:` diesen Wert. Verlief die Datenübertragung nicht erfolgreich, wird dem Benutzer mitgeteilt, dass die Verbindung abgelehnt wurde oder verloren ging.

```
// Empfängt das Bild und aktualisiert die Oberfläche
- (void) connection:(TCPConnection*)connection
   didReceiveData:(NSData*)data;
{
```

```
    // Aktualisiert beim Empfang des Bildes die Bildansicht
    success = YES;
    self.imageData = data;
    NSImage *image = [self imageFromData:data];
    [imageView setImage:image];

    // Sie können jetzt ein anderes Bild speichern oder aufnehmen
    [saveItem setEnabled:YES];
    [button setEnabled:YES];
    [progress stopAnimation:nil];

    // Aktualisiert den Status
    ANNOUNCE(@"Received JPEG image (%d bytes).\n\nUse File > Save\
        to save the received image to disk.", data.length);
}

// Meldet einen Misserfolg und stellt die Benutzeroberfläche
// wieder her
- (void) connectionDidClose:(TCPConnection*)connection
{
    // Fehlgeschlagene oder abgeschlossene Verbindung. Prüft auf Erfolg.
    if (success) return;
    ANNOUNCE(@"Connection denied or lost. Sorry.");

    // Bereitet bei einer fehlgeschlagenen Verbindung die nächste
    // Aufnahme vor
    self.imageData = nil;
    [saveItem setEnabled:NO];
    [imageView setImage:nil];
    [button setEnabled:YES];
    [progress stopAnimation:nil];
}

// Erstellt beim Auflösen der Adresse eine Verbindung dorthin und
// fordert Daten an
- (void)netServiceDidResolveAddress:(NSNetService *)netService
{
    // Erfasst die Adressen und versucht, eine Verbindung herzustellen
    NSArray* addresses = [netService addresses];
    if (addresses && [addresses count]) {
        struct sockaddr* address = (struct sockaddr*)[[addresses
            objectAtIndex:0] bytes];
        TCPConnection *connection = [[TCPConnection alloc]
            initWithRemoteAddress:address];
```

```
        [connection setDelegate:self];
        [statusText setTitleWithMnemonic:@"Requesting data..."];
        [progress startAnimation:nil];
        [netService release];
        [connection receiveData];
    }
}

// Meldet einen Fehlschlag beim Auflösen des Dienstes
- (void)netService:(NSNetService *)sender didNotResolve:
    (NSDictionary *)errorDict {
    [statusText setTitleWithMnemonic:
        @"Error resolving service. Sorry."];
}

// Hält den Dienstbrowser an, wenn ein Dienst gefunden wurde, und
// löst den Dienst auf
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didFindService:(NSNetService *)netService
    moreComing:(BOOL)moreServicesComing
{
    [self.browser stop];
    self.browser = nil;
    [statusText setTitleWithMnemonic:@"Resolving service."];
    [[netService retain] setDelegate:self];
    [netService resolveWithTimeout:0.0f];
}

// Beginnt mit der Anforderung einer Aufnahme, startet den Dienstbrowser
// und aktualisiert die Benutzeroberfläche
- (IBAction) catchPlease: (id) sender
{
    success = NO;
    [statusText setTitleWithMnemonic:@"Scanning for service"];

    // Erstellt einen neuen Dienstbrowser
    self.browser = [[[NSNetServiceBrowser alloc] init] autorelease];
    [self.browser setDelegate:self];
    NSString *type = [TCPConnection
        bonjourTypeFromIdentifier:@"PictureThrow"];
    [self.browser searchForServicesOfType:type inDomain:@"local"];
```

```
    // Deaktiviert die interaktiven Funktionen beim Warten und
    // setzt sie zurück
    [button setEnabled:NO];
    self.imageData = nil;
    [saveItem setEnabled:NO];
    [imageView setImage:nil];
}
```

► *Rezept 12.6: Einen Bonjour-Client bereitstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 12 und öffnen das Projekt zu diesem Rezept.

12.7 Rezept: Die Einschränkungen von GameKit umgehen

GameKit baut zwar auf Bonjour auf, was aber nicht bedeutet, dass es die gleichen Möglichkeiten für die allgemeine Datenübertragung bietet, wie Sie sie in den beiden letzten reinen Bonjour-Rezepten gesehen haben. Für GameKit-Bluetooth-Verbindungen sind kleine Datenpakete zu bevorzugen, am besten unter jeweils 1000 Byte. `GKSession`-Objekte können nicht mehr als 95 Kbyte Daten senden. Wenn Sie dies versuchen, schlägt die Methode `sendDataToAllPeers:error:` fehl und gibt den booleschen Wert `NO` zurück.

Rezept 12.7 geht dieses Problem an und prüft die Datenmenge, bevor die Sendeanforderungen in die Warteschlange eingereiht werden. Kurze Daten können übertragen werden, lange Daten werden abgelehnt. Um etwas zum Testen zu haben, greift dieses Rezept auf die Zwischenablage des iPhones zurück.

In der Praxis sollten Sie lange Datenelemente in kürzere aufteilen und mit zuverlässiger Übertragung senden. Dadurch wird sichergestellt, dass die Daten in derselben Reihenfolge ankommen, in der sie abgeschickt wurden. Damit die Daten korrekt ankommen, können Sie Prüfsummentests und andere in Netzwerken übliche Lösungen umsetzen. (Für anspruchsvollere Bedürfnisse bei der Datenübertragung können Sie auch einen eigenen Bonjour-WiFi-Dienst programmieren oder Internet-Serververbindungen nutzen.)

Dieses Rezept bietet einen Ausgangspunkt für den Test von Dateigrößenelementen in GameKit. Sie können diesen Code gern erweitern, um die Zerlegung und Rekonstruktion von Dateien selbst auszuprobieren.

12.7.1 Die Zwischenablage des iPhones nutzen

Die *Zwischenablage* ist ein zentrales Merkmal des Betriebssystems, das die gemeinsame Nutzung von Daten in verschiedenen Anwendungen erlaubt. Die Benutzer können die Daten in einer Anwendung in die Zwischenablage kopieren, zu einer anderen Aufgabe wechseln und die Daten in eine

andere Anwendung einfügen. Funktionen zum Ausschneiden, Kopieren und Einfügen lassen sich in den meisten Betriebssystemen finden und sind seit der Firmware-Version 3.0 auch auf dem iPhone verfügbar.

Die Klasse `UIPasteboard` bietet Zugriff auf die gemeinsame iPhone-Zwischenablage und deren Inhalt. Wie auf Mac- und Windows-Computern können Sie mit der Zwischenablage Daten innerhalb einer Anwendung, aber auch zwischen verschiedenen Anwendungen austauschen. Neben der allgemeinen Zwischenablage für die gemeinsame Nutzung gibt es auf dem iPhone auch eine Zwischenablage für die Namenssuche und anwendungsspezifische Zwischenablagen, um die Daten vertraulicher zu halten. Die folgende Codezeile gibt die allgemeine Systemzwischenablage zurück, die sich für die meisten der üblichen Kopier- und Einfügeaufgaben eignet:

```
UIPasteboard *pb = [UIPasteboard generalPasteboard];
```

12.7.2 Daten speichern

In der Zwischenablage können Sie auch mehr als einen Eintrag auf einmal speichern. Die Einträge können verschiedene Typen aufweisen, wobei ein UTI (Uniform Type Identifier) angibt, um was für eine Art Daten es sich jeweils handelt. Zu den üblichen Datentypen auf dem iPhone gehören z. B. `public.text` (oder genauer `public.utf8-plain-text`), `public.url` und `public.jpeg`. Das Dictionary, das die Typen und die zugehörigen Daten speichert, heißt `item`, und Sie können ein Array aller verfügbaren Elemente über die Eigenschaft `items` der Zwischenablage abrufen.

Um die verfügbaren Typen in einer Zwischenablage abzurufen, senden Sie ihr die Nachricht `pasteboardTypes`. Dadurch wird ein Array der zurzeit in der Zwischenablage gespeicherten Typen zurückgegeben.

```
NSArray *types = [pb pasteboardTypes];
```

Für verschiedene Datentypen, nämlich Farben, Bilder, Strings und URLs, gibt es spezialisierte Zwischenablagen. Die Klasse `UIPasteboard` enthält besondere Get- und Set-Methoden, um die Handhabung dieser Art von Elementen zu vereinfachen. Da *Rezept 12.7* ein allgemeines Einfügewerkzeug erstellt, wird nur die Handhabung von Strings mit einem besonderen Aufruf gezeigt, nämlich `setString`.

12.7.3 Daten abrufen

Daten rufen Sie mit `dataForPasteboardType:` ab. Dadurch werden die Daten des ersten Elements zurückgegeben, dessen Typ mit dem als Parameter übergebenen Typ übereinstimmt. Alle anderen passenden Elemente in der Zwischenablage werden ignoriert. Wenn Sie alle passenden Daten lesen möchten, rufen Sie ein passendes Element mit `itemSetWithPasteboardType:` ab und iterieren durch die Menge, um alle Dictionarys zu erfassen. Rufen Sie für jedes Element den Datentyp aus dem Dictionary-Schlüssel und die Daten aus dem Wert ab.

Bei `UIPasteboard` haben Sie zwei Möglichkeiten, um etwas in die Zwischenablage aufzunehmen. Für Eigenschaftenlistenobjekte (die weiter vorn in diesem Kapitel erläutert wurden) verwenden Sie `setValueForPasteboardType:`, für allgemeine Daten dagegen wie in dem Rezept `setData:`

`forPasteboardType:`. Wenn sich eine Zwischenablage ändert, gibt sie die Nachricht `UIPasteboardChangedNotification` aus, die Sie über einen standardmäßigen `NSNotificationCenter`-Beobachter abhören können.

12.7.4 Verantwortungsvoller Umgang mit der Zwischenablage

Rezept 12.7 führt mehrere Tests durch, bevor Zwischenablagen-Daten gesendet, abgerufen oder kopiert werden. Die Benutzer müssen bestätigen, dass sie Daten eines gegebenen Typs für andere freigeben möchten. Beim Empfang von Daten müssen sie die Anwendung dazu autorisieren, die Daten in die allgemeine Systemzwischenablage zu kopieren. Dadurch wird sichergestellt, dass der Benutzer aktiv werden muss, bevor diese Vorgänge ausgeführt werden.

```
@implementation TestBedViewController

- (void) sharePasteboard
{
   // Konstruiert ein Dictionary aus Zwischenablagentypen und -daten
   NSMutableDictionary *md = [NSMutableDictionary dictionary];
   UIPasteboard *pb = [UIPasteboard generalPasteboard];
   NSString *type = [[pb pasteboardTypes] lastObject];
   NSData *data = [pb dataForPasteboardType:type];
   [md setObject:type forKey:@"type"];
   [md setObject:data forKey:@"data"];

   // Lehnt alle zu umfangreichen Anforderungen ab
   if (data.length > (95000))
   {
      [ModalAlert say:@"Too much data in pasteboard (%0.2f \
         Kilobytes). GameKit can only send up to approx 90 \
         Kilobytes at a time.", ((float) data.length) / 1000.0f];
      return;
   }

   // Der Benutzer muss die Freigabe der Daten bestätigen
   NSString *confirmString = [NSString stringWithFormat:
      @"Share %d bytes of type %@?", data.length, type];
   if (![ModalAlert ask:confirmString]) return;

   // Serialisiert und sendet die Daten
   NSString *errorString;
   NSData *plistdata = [NSPropertyListSerialization
      dataFromPropertyList:md format:NSPropertyListXMLFormat_v1_0
      errorDescription:&errorString];
   if (plistdata)
      [GameKitHelper sendData:plistdata];
```

```
    else
        CFShow(errorString);
}

- (void) sentData:(NSString *) errorString
{
    // Prüft, ob beim Senden der Daten ein Problem aufgetreten ist
    if (errorString)
    {
        [ModalAlert say:@"Error sending data: %@", errorString];
        return;
    }

    [ModalAlert say:@"Pasteboard data successfully queued for\
        transmission."];
}

// Erlaubt dem Benutzer beim Aufbau der Verbindung, die Zwischenablage
// zur gemeinsamen Nutzung freizugeben
- (void) connectionEstablished
{
    UIPasteboard *pb = [UIPasteboard generalPasteboard];
    NSArray *types = [pb pasteboardTypes];
    if (types.count == 0) return;

    self.navigationItem.leftBarButtonItem = BARBUTTON(
        @"Share Pasteboard", @selector(sharePasteboard));
}

// Blendet die Freigabeoption aus, wenn die Verbindung getrennt ist
- (void) connectionLost
{
    self.navigationItem.leftBarButtonItem = nil;
}

-(void) receivedData: (NSData *) data
{
    // Deserialisiert die Übertragung
    CFStringRef errorString;
    NSDictionary *dict =
        (NSDictionary *) CFPropertyListCreateFromXMLData(
        kCFAllocatorDefault, (CFDataRef)data,
        kCFPropertyListMutableContainers, &errorString);
    if (!dict)
```

```
    {
        CFShow(errorString);
        return;
    }

    // Ruft Typ und Daten ab
    NSString *type = [dict objectForKey:@"type"];
    NSData *sentdata = [dict objectForKey:@"data"];
    if (!type || !sentdata) return;

    // Es wird nichts in die Zwischenablage kopiert, bevor der Benutzer
    // den Vorgang zulässt
    NSString *message = [NSString stringWithFormat:
        @"Received %d bytes of type %@. Copy to pasteboard?",
        sentdata.length, type];
    if (![ModalAlert ask:message]) return;

    // Führt den Kopiervorgang in die Zwischenablage durch
    UIPasteboard *pb = [UIPasteboard generalPasteboard];
    if ([type isEqualToString:@"public.text"])
    {
        NSString *string = [[[NSString alloc] initWithData:sentdata
            encoding:NSUTF8StringEncoding] autorelease];
        [pb setString:string];
    }
    else [pb setData:sentdata forPasteboardType:type];
}

- (void) viewDidLoad
{

    self.navigationController.navigationBar.tintColor =

        COOKBOOK_PURPLE_COLOR;

    Self.title = @"Pasteboard Share";
    // Richtet die Hilfsklasse ein
    [GameKitHelper sharedInstance].sessionID = @"Pasteboard Share";
    [GameKitHelper sharedInstance].dataDelegate = self;
    [GameKitHelper assignViewController:self];
}
@end
```

▶ *Rezept 12.7: Gemeinsame Nutzung der iPhone-Zwischenablage über GameKit*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 12 und öffnen das Projekt zu diesem Rezept.

12.8 Rezept: Gemeinsame Spiele auf iPhone-Geräten mithilfe von BonjourHelper

Wenn Sie die Bluetooth-Option von GameKit aufgeben und mit WiFi arbeiten möchten, können Sie viele der GameKit-Funktionen auf allen iPhones nachahmen, auch auf Geräten der ersten Generation. *Rezept 12.8* stellt die Klasse `BonjourHelper` vor, die `GameKitHelper` aus *Rezept 12.1* imitiert. In jenem Rezept wurde die Verbindung dadurch hergestellt, dass eine Sitzungs-ID und eine Datendelegierung festgelegt und ein Ansichtscontroller zugewiesen wird.

```
[GameKitHelper sharedInstance].sessionID = @"Typing Together";
[GameKitHelper sharedInstance].dataDelegate = self;
[GameKitHelper assignViewController:self];
```

Um `GameKitHelper` durch `BonjourHelper` zu ersetzen, sind nur wenige Änderungen im Programm nötig. Es werden die gleichen Initialisierungsschritte durchgeführt, und auch die Datendelegierung empfängt den gleichen Satz von Callbacks. Sie müssen jedoch die Leerzeichen in der Sitzungs-ID entfernen, was in GameKit nicht notwendig ist, da GameKit die IDs so verschlüsselt, dass eine garantiert leerzeichenfreie, ordentliche Bonjour-ID dabei herauskommt. Der Klartextansatz von `BonjourHelper` bedeutet, dass Leerzeichen unzulässig sind. Schränken Sie die Namen für Sitzungs-IDs auf einfachen alphanumerischen Text mit maximal 14 Zeichen ein. Einzelheiten finden Sie im RFC 2782 zu Diensttypen (*http://www.dns-sd.org/ServiceTypes.html*). Der Code von `BonjourHelper` wandelt die Sitzungs-ID in eine standardmäßige Bonjour-ID um (d. h. in `_typingtogether._tcp`).

```
[BonjourHelper sharedInstance].sessionID = @"TypingTogether";
[BonjourHelper sharedInstance].dataDelegate = self;
[BonjourHelper assignViewController:self];
```

Dass nur wenige Änderungen erforderlich sind, heißt jedoch nicht, dass die Funktionsweise und Implementierung identisch sind. Bei `BonjourHelper` müssen sich beide Geräte im selben Netzwerk befinden. Außerdem können Sie nicht auf die ansprechend gestaltete Abfolge der GameKit-Controller für die Peer-Auswahl zurückgreifen, die Sie in den *Abbildung*en 12.2 bis 12.4 sehen. Stattdessen verwendet `BonjourHelper` die einfache Benachrichtigung aus *Abbildung 12.8*. Darüber hinaus jedoch bietet `BonjourHelper` grundlegend die gleiche Art von Peer-to-Peer-Anbindung und Datenübertragung wie GameKit.

12.8.1 Bonjour-Namen und Ports registrieren

Alle Bonjour-Namen, die Sie für die kommerzielle Veröffentlichung einsetzen möchten, sollten Sie bei der Organisation *DNS Service Discovery* registrieren. Dadurch stellen Sie sicher, dass die Namen und Protokolle Ihrer Dienste nicht mit denen anderer Anbieter in Konflikt geraten. Eine Liste der zurzeit registrierten Dienste wird unter *http://www.dns-sd.org/ServiceTypes.html* geführt.

Die Namen müssen dem Standard von RFC 2782 entsprechen. Reichen Sie den Protokollnamen unter *srv_type_request@dns-sd.org* ein, und fügen Sie den maximal 14-stelligen Namen des Bonjour-Dienstes, einen längeren beschreibenden Namen, die Kontaktinformationen (Name und E-Mail-Adresse) der Person, die den Dienst registriert, und den URL einer Informationsseite bei. Geben Sie das Transportprotokoll (also `_tcp` oder `_udp`) und eine Liste aller verwendeten TXT-Datensatzschlüssel an. (Am Ende dieses Kapitels finden Sie ein Beispiel, in dem TXT-Daten verwendet und angezeigt werden.)

▸ *Abbildung 12.8: Die benutzerdefinierte Klasse* `BonjourHelper` *zeigt eine einfacher gestaltete Verbindungsschnittstelle als GameKit.*

Es kann einige Zeit dauern, bis die ehrenamtlichen Mitarbeiter der Website *dns-sd.org* Ihre Anfrage bearbeitet haben und darauf antworten. Wartezeiten in der Größenordnung von Wochen sind nicht ungewöhnlich. Unter Umständen müssen Sie den Antrag erneut einreichen, weshalb Sie eine Kopie aller Informationen bereithalten sollten.

Wenn Sie einen festen Port verwenden möchten (die meisten Bonjour-Implementierungen wählen zur Laufzeit einen zufälligen Port aus), müssen Sie außerdem bei der IANA (*Internet Assigned Numbers Association*) einen Antrag auf Registrierung der Portnummer einreichen. Die IANA unterhält eine zentrale Portregistrierung und wird eines Tages mit der DNS-SD-Registrierung verschmolzen. Zur Registrierung neuer Protokollportnummern braucht die IANA häufig ein Jahr oder länger.

Hinweis

Im Dokument *Technical Q&A AQ1312* führt Apple eine Liste der offiziellen Bonjour-Diensttypen von OS X. Dieses Dokument finden Sie unter *http://developer.apple.com/mac/library/qa/qa2001/qa1312.html*.

12.8.2 Duplexverbindungen

Der Einfachheit halber baut `BonjourHelper` eine Duplexverbindung auf, wobei sich auf jedem Gerät sowohl ein Client als auch ein Host befindet. Dadurch werden alle Schwierigkeiten vermieden, die sich ergeben können, wenn die beiden Peers miteinander verhandeln und die Server- und Clientrollen festlegen müssen, ohne dass am Ende beide als Server oder als Client fungieren.

Beim Auflösen von Adressen achtet die Hilfsklasse darauf, dass sich ein Gerät nicht mit sich selbst verbindet. Sie verlangt eine eindeutige IP-Adresse, die nicht mit der lokalen identisch ist. Wenn die eingehende Adresse mit der lokalen übereinstimmt, sucht die Klasse einfach weiter. Der Host muss eine solche Überprüfung nicht durchführen, da ausgehende Clientverbindungen ohnehin auf fremde Adressen beschränkt sind.

Wenn die Hilfsklasse eine ausgehende Verbindung eingerichtet und eine eingehende akzeptiert hat, hört sie auf, nach weiteren Peers zu suchen, und betrachtet sich selbst als vollständig verbunden. Wenn ein Ansichtscontroller eingerichtet wurde, aktualisiert die Hilfsklasse die Schaltfläche **Connect** und zeigt sie als **Disconnect** an.

12.8.3 Daten lesen

Anders als in *Rezept 12.6* kann in *Rezept 12.8* keine einfache Leseschleife verwendet werden, bei der Daten angefordert und gelesen werden und der Vorgang dann wiederholt wird. Das Lesen von Daten ist ein blockierender Vorgang, weshalb eine Leseschleife eine Anwendung daran hindert, gleichzeitig ihre Aufgaben als Server und als Client zu erfüllen.

Stattdessen verwendet diese Klasse den nicht blockierenden Test `hasDataAvailable`, bevor sie neue Daten abfragt. Ein verzögerter Selektor führt in den Abrufvorgang eine Wartezeit ein, um jedem Host Zeit einzuräumen, sich zu aktualisieren und neue Daten vorzubereiten, bevor er durch eine neue Anforderung blockiert wird.

12.8.4 Verbindungen schließen

Verbindungen können auf verschiedene Weisen getrennt werden. Benutzer können die Anwendung beenden, sie können auf die Schaltfläche **Disconnect** tippen oder sich außerhalb der Reichweite der Verbindung bewegen. `BonjourHelper` überprüft solche Verbindungsabbrüche ausschließlich vom Server aus. Dadurch wird die Implementierung vereinfacht, denn ein getrennter Client wird einem getrennten Server gleichgesetzt, sodass für die beiden Enden der Duplexverbindungen keine unterschiedlichen Benachrichtigungen (»Verbindung zum Server getrennt« und »Verbindung zum Client getrennt«) erforderlich sind.

Hinweis

Aus Platzgründen wurden in dem Listing von *Rezept 12.8* mehrere grundlegende Hilfskonstruktionen für IP-Adressen ausgelassen, darunter `stringFromAddress:`, `addressFromSTring:address:` und `localAddress`. Diese Methoden befinden sich aber in dem Beispielcode zu diesem Kapitel und werden in *Kapitel 13, Netzwerke*, ausführlicher besprochen.

```
#define DO_DATA_CALLBACK(X, Y) if (sharedInstance.dataDelegate && \
   [sharedInstance.dataDelegate respondsToSelector:@selector(X)]) \
   [sharedInstance.dataDelegate performSelector:@selector(X) \
   withObject:Y];
#define BARBUTTON(TITLE, SELECTOR) [[[UIBarButtonItem alloc] \
   initWithTitle:TITLE style:UIBarButtonItemStylePlain \
   target:[BonjourHelper class] action:SELECTOR] autorelease]

@implementation BonjourHelper
@synthesize server;
@synthesize browser;
@synthesize inConnection;
@synthesize outConnection;

@synthesize dataDelegate;
@synthesize viewController;
@synthesize sessionID;
@synthesize isConnected;
@synthesize hud;

static BonjourHelper *sharedInstance = nil;
BOOL inConnected;
BOOL outConnected;

+ (BonjourHelper *) sharedInstance
{
   if(!sharedInstance) sharedInstance = [[self alloc] init];
   return sharedInstance;
}

#pragma mark Class utilities

+ (void) assignViewController: (UIViewController *) aViewController
{
   // Diese Klasse kümmert sich durch Zuweisung eines optionalen
   // Ansichtscontrollers um die Schaltfläche Connect/Disconnect
```

```
    sharedInstance.viewController = aViewController;
    if (sharedInstance.viewController)
        sharedInstance.viewController.navigationItem.rightBarButtonItem
        = BARBUTTON(@"Connect", @selector(connect));
}

#pragma mark Handshaking

- (void) updateStatus
{
    // Um fortfahren zu können, muss eine Verbindung bestehen
    if (!(self.inConnection && self.outConnection) ||
        !(inConnected && outConnected))
    {
        self.isConnected = NO;
        return;
    }

    // Sendet einen Callback, entfernt das Schwebefenster und
    // aktualisiert die Leistenschaltfläche
    self.isConnected = YES;
    DO_DATA_CALLBACK(connectionEstablished, nil);
    [self.hud dismissWithClickedButtonIndex:1 animated:YES];
    if (self.viewController)
        self.viewController.navigationItem.rightBarButtonItem =
            BARBUTTON(@"Disconnect", @selector(disconnect));
}

// Erstellt beim Auflösen einer Adresse eine Verbindung dorthin und
// fordert Daten an
- (void)netServiceDidResolveAddress:(NSNetService *)netService
{
    NSArray* addresses = [netService addresses];
    if (addresses && addresses.count)
    {
        for (int i = 0; i < addresses.count; i++)
        {
            // Die Implementierungen der IP-Hilfsmethoden finden Sie im
            // Beispielcode zu diesem Kapitel. Sie wurden  hier aus
            // Platzgründen ausgelassen.
            struct sockaddr* address =
                (struct sockaddr*)[[addresses objectAtIndex:i] bytes];
            NSString *addressString =
                [BonjourHelper stringFromAddress:address];
            if (!addressString) continue;
```

```
            if ([addressString hasPrefix:
               [BonjourHelper localIPAddress]])
            {
               printf("Will not resolve with self. \
                  Continuing to browse.\n");
               continue;
            }

            printf("Found a matching external service\n");
            printf("My address: %s\n",
               [[BonjourHelper localIPAddress] UTF8String]);
            printf("Remote address: %s\n", [addressString UTF8String]);

            // Beendet die Suche nach Diensten
            [self.browser stop];
            [netService release];

            // Erstellt eine ausgehende Verbindung zum neuen Dienst
            self.outConnection = [[[TCPConnection alloc]
               initWithRemoteAddress:address] autorelease];
            [self.outConnection setDelegate:self];
            [self performSelector:@selector(checkForData)];
            [self updateStatus];
            return;
         }
      }

      [netService stop];
}

- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
   didFindService:(NSNetService *)netService
   moreComing:(BOOL)moreServicesComing
{
   // Beginnt den gefundenen Dienst aufzulösen
   [[netService retain] setDelegate:self];
   [netService resolveWithTimeout:0.0f];
}

+ (void) startBrowsingForServices
{
   // Sucht nach übereinstimmenden Bonjour-Diensten. Die doppelte
   // Beibehaltung wurde aus Sicherheitsgründen hinzugefügt. Sie können
   // höchstwahrscheinlich darauf verzichten.
```

```
    sharedInstance.browser =
        [[[NSNetServiceBrowser alloc] init] retain];
    [sharedInstance.browser setDelegate:sharedInstance];
    NSString *type = [TCPConnection
        bonjourTypeFromIdentifier:sharedInstance.sessionID];
    [sharedInstance.browser searchForServicesOfType:type
        inDomain:@"local"];
}

+ (void) publish
{
    // Kündigt den Peers den Dienst an
    sharedInstance.server =
        [[[TCPServer alloc] initWithPort:0] autorelease];
    [sharedInstance.server setDelegate:sharedInstance];
    [sharedInstance.server startUsingRunLoop:
        [NSRunLoop currentRunLoop]];
    [sharedInstance.server enableBonjourWithDomainnil
        applicationProtocol:sharedInstance.sessionID
        name:[self localHostname]];
}

+ (void) initConnections
{
    // Kehrt zum einfachen, unverbundenen Zustand zurück
    [sharedInstance.browser stop];
    [sharedInstance.server stop];

    sharedInstance.inConnection = nil;
    sharedInstance.outConnection = nil;
    sharedInstance.isConnected = NO;
    inConnected = NO;
    outConnected = NO;
}

- (void)alertView:(UIAlertView *)alertView
    clickedButtonAtIndex:(NSInteger)buttonIndex
{
    // Handhabt Benutzeranforderungen zum Abbrechen der Verbindung
    if (buttonIndex) return; // wenn Abbrechen nicht gewählt wurde
    [BonjourHelper disconnect];
}

+ (void) connect
{
    if (sharedInstance.viewController)
```

```
        sharedInstance.viewController.navigationItem.rightBarButtonItem
            = nil;
    if (!sharedInstance.sessionID)
        sharedInstance.sessionID = @"Sample Session";

    // Bereitet Duplexverbindung vor
    [self initConnections];
    [self startBrowsingForServices];
    [self publish];

    // Erstellt Aktivitätsansicht mit Cancel-Schaltfläche
    sharedInstance.hud = [[[UIAlertView alloc]
        initWithTitle:
        @"Searching for connection peer on your local network"
        message:@"\n\n" delegate:sharedInstance
        cancelButtonTitle:@"Cancel" otherButtonTitles:nil]
        autorelease];
    [sharedInstance.hud show];

    // Fügt die Fortschrittsanzeige hinzu
    UIActivityIndicatorView *aiv = [[[UIActivityIndicatorView alloc]
        initWithActivityIndicatorStyle:
        UIActivityIndicatorViewStyleWhiteLarge] autorelease];
    [aiv startAnimating];
    aiv.center = CGPointMake(
        sharedInstance.hud.bounds.size.width / 2.0f,
        sharedInstance.hud.bounds.size.height/2.0f);
    [sharedInstance.hud addSubview:aiv];
}

+ (void) disconnect
{
    // Deaktiviert aktuelle Verbindungen
    [sharedInstance.inConnection invalidate];
    [sharedInstance.outConnection invalidate];
    [self initConnections];

    // Hält den Server an
    [sharedInstance.server stop];
    [sharedInstance updateStatus];

    // Zurücksetzen
    [sharedInstance.hud dismissWithClickedButtonIndex:1 animated:YES];
    if (sharedInstance.viewController)
```

```
        sharedInstance.viewController.navigationItem.rightBarButtonItem
           = BARBUTTON(@"Connect", @selector(connect));
}

#pragma mark Data Handling

- (void) checkForData
{
   // Führt einen blockierenden Empfangsvorgang nur dann durch, wenn
   // Daten vorliegen
   if (!self.outConnection) return;
   if ([self.outConnection hasDataAvailable])
      [self.outConnection receiveData];
   [self performSelector:@selector(checkForData)
      withObject:self afterDelay:0.1f];
}

+ (void) sendData: (NSData *) data
{
   if (!sharedInstance.outConnection) return;
   BOOL success = [sharedInstance.outConnection sendData:data];
   if (success) {
      DO_DATA_CALLBACK(sentData:, nil); }
   else {
      DO_DATA_CALLBACK(sentData:, @"Data could not be sent.");}
}

- (void) connection:(TCPConnection*)connection
   didReceiveData:(NSData*)data;
{
   // Leitet den Daten-Callback um
   DO_DATA_CALLBACK(receivedData:, data);
}

#pragma mark Connection Handlers

- (BOOL) server:(TCPServer*)server
   shouldAcceptConnectionFromAddress:(const struct sockaddr*)address
{
   // Akzeptiert Verbindungen nur, wenn noch keine besteht
   return !self.isConnected;
}
```

```
- (void) connectionDidFailOpening:(TCPConnection*)connection
{
   // Handhabt einen Fehlschlag beim Öffnen der Verbindung
   if (!connection) return;
   NSString *addressString = [BonjourHelper
      stringFromAddress:connection.remoteSocketAddress];
   [BonjourHelper disconnect];

   if (addressString)
      [ModalAlert say:@"Error while opening %@ connection (from %@).\
         Wait a few seconds or relaunch before trying to connect\n \
         again.", (connection == self.inConnection) ? @"incoming" :
         @"outgoing", addressString];
   else
      printf("Failed to open connection from unknown address\n");
}

- (void) server:(TCPServer*)server
   didCloseConnection:(TCPServerConnection*)connection
{
   // Handhabt eine kürzlich geschlossene Verbindung
   if (!connection) return;
   NSString *addressString = [BonjourHelper
      stringFromAddress:connection.remoteSocketAddress];
   if (!addressString) return;

   BOOL wasConnected = self.isConnected;

   [BonjourHelper disconnect];
   printf("Lost connection from %s\n", [addressString UTF8String]);

   if (wasConnected)
      [ModalAlert say:@"Disconnected from peer (%@). You are no \
         longer connected to another device.", addressString];
   else
      [ModalAlert say:@"Peer was lost before full connection could \
         be established."];
}

- (void) server:(TCPServer*)server
   didOpenConnection:(TCPServerConnection*)connection
{
```

```
    // Richtet die Verbindung ein, wartet aber mit dem Öffnen
    self.inConnection = connection;
    [self updateStatus];
    [connection setDelegate:self];
}

- (void) connectionDidOpen: (TCPConnection *) connection
{
    // Vollständig geöffnete Verbindung
    printf("Connection did open: %s\n", (connection ==
        self.inConnection) ? "incoming" : "outgoing");
    if (connection == self.inConnection) inConnected = YES;
    if (connection == self.outConnection) outConnected = YES;
    [self updateStatus];
}

- (void) connectionDidClose: (TCPConnection *)connection
{
    // Geschlossene Verbindung
    printf("Connection did close: %s\n", (connection ==
        self.inConnection) ? "incoming" : "outgoing");
    if (connection == self.inConnection) inConnected = NO;
    if (connection == self.outConnection) outConnected = NO;

}
@end
```

► *Rezept 12.8: GameKit-artige WiFi-Anbindung mithilfe von BonjourHelper*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 12 und öffnen das Projekt zu diesem Rezept.

12.9 »Online«-Verbindungen mit GameKit erstellen

In der GameKit-Terminologie ist mit »online« zurzeit jede andere gültige Verbindungsart außer Bluetooth gemeint. Beispielsweise können Sie ein lokales WLAN-Netzwerk nutzen, um Verbindung mit einem anderen Gerät im selben Netzwerk aufzunehmen oder um über WWAN (z. B. einen Zellfunkdienst) oder WiFi einen Host im Internet zu erreichen. GameKit hilft Ihnen dabei nur bis zu dem Dialogfeld aus *Abbildung 12.9*. Wenn der Benutzer **Online** wählt, sind Sie als Entwickler gefordert, um eine eigene Verbindung zu einem anderen Dienst oder Gerät herzustellen.

▸ *Abbildung 12.9: Die Online-Verbindung von GameKit bedeutet, dass Sie Ihre eigene Netzwerktechnologie mitbringen müssen. Beachten Sie, dass die Schaltfläche* **Send** *auf der Tastatur die standardmäßige* **Return***-Schaltfläche ist. In diesem Rezept werden die Daten bei der Eingabe gesendet.*

Dieses Dialogfeld mit zwei Auswahlmöglichkeiten erstellen Sie, indem Sie der Maske des Peer-Pickers die Online-Option hinzufügen. Ansonsten gibt es keine Änderung gegenüber der Art und Weise, wie Sie den standardmäßigen Peer-Picker-Controller von GameKit erstellen und anzeigen.

```
- (void) startConnection
{
   if (!self.isConnected)
   {
      GKPeerPickerController *picker = [[GKPeerPickerController
         alloc] init];
      picker.delegate = self;
      picker.connectionTypesMask = GKPeerPickerConnectionTypeNearby |
         GKPeerPickerConnectionTypeOnline;
      [picker show];
      if (self.viewController)
         self.viewController.navigationItem.rightBarButtonItem =
            nil;
   }
}
```

Die Auswahl, die der Benutzer getroffen hat, erfassen Sie im Callback `peerPickerController:didSelectConnectionType:`. Wenn der Benutzer **Nearby** wählt, können Sie davon ausgehen, dass alle Handshake-Dialogfelder für Sie erstellt werden. Bei der Auswahl von **Online** dagegen ist es Ihre Aufgabe, die nächsten Schritte durchzuführen. Sie müssen den Picker entfernen und die nächste Phase des Verbindungsvorgangs anzeigen. An dieser Stelle gibt der Peer-Picker die Steuerung auf.

Die Klasse `BonjourHelper` aus *Rezept 12.8* wird initialisiert und die Verbindung begonnen. Anstelle des grauen Peer-Picker-Dialogfelds erscheint die standardmäßige blaue Benachrichtigung von `BonjourHelper`.

```
- (void)peerPickerController:(GKPeerPickerController *)picker
   didSelectConnectionType:(GKPeerPickerConnectionType)type
{
   if(type == GKPeerPickerConnectionTypeOnline)
   {
      [picker dismiss];
      [picker release];
      [BonjourHelper sharedInstance].sessionID = self.sessionID;
      [BonjourHelper sharedInstance].viewController =
         self.viewController;
      [BonjourHelper sharedInstance].dataDelegate =
         self.dataDelegate;
      [BonjourHelper connect];
   }

   else printf("Selected nearby type\n");
}
```

Wie *Rezept 12.9* zeigt, sind auf der Seite von `BonjourHelper` kaum Änderungen erforderlich. Die Schaltfläche **Connect** in der Navigationsleiste muss wieder auf die Verbindungsmethode von GameKit zurückverweisen, nicht auf die von `BonjourHelper`. Dadurch kann der Benutzer beim Beenden der Bonjour-Verbindung wieder eine Bluetooth-Verbindung herstellen, ohne die Anwendung neu starten zu müssen.

```
#define GBARBUTTON(TITLE, SELECTOR) [[[UIBarButtonItem alloc] \
initWithTitle:TITLE style:UIBarButtonItemStylePlain \
target:[GameKitHelper class] action:SELECTOR] autorelease]

if (sharedInstance.viewController)
   sharedInstance.viewController.navigationItem.rightBarButtonItem =
      GBARBUTTON(@"Connect", @selector(connect));
```

▸ *Rezept 12.9: Den Makrocode aktualisieren um die GameKit-Version von* `connect` *zu verwenden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 12 und öffnen das Projekt zu diesem Rezept.

Vielleicht glauben Sie, dass Sie den Schlüssel `UIRequiresPersistentWiFi` in der Datei `Info.plist` auf den booleschen Wert `true` setzen müssen. Tun Sie dies jedoch nicht, sondern führen Sie die Überprüfung auf eine WiFi-Anbindung nur dann durch, wenn Sie dazu bereit sind, eine WiFi-Verbindung herzustellen, also wenn der Benutzer auf **Online** tippt. Im Gegensatz zu Bonjour-Verbindungen brauchen Bluetooth-Verbindungen von GameKit keine ständige WiFi-Anbindung. Verlangen Sie von den Benutzern nicht, eine Verbindung zu einem WiFi-Dienst aufzunehmen (die möglicherweise gar nicht zur Verfügung steht), wenn Bluetooth für gemeinsame Spiele im Umgebungsmodus ausreicht.

12.10 Ein letzter Punkt: Dienste suchen

Die Klasse `NSNetServiceBrowser` ist nicht auf einen einzigen vordefinierten Dienst beschränkt. Sie können den Browsercode so anpassen, dass nach allen verfügbaren Diensten gesucht wird, mit denen das iPhone kommunizieren kann. *Rezept 12.10* erweitert die in diesem Kapitel vorgestellten Beispiele zur Suche und Auflösung von Diensten, um alle aktiven Bonjour-Dienstanbieter zu finden und wie in *Abbildung 12.10* als Liste darzustellen. Wenn der Benutzer auf eine Zelle der Liste tippt, wird eine Seite mit näheren Angaben zu dem Dienst eingeblendet.

▶ *Abbildung 12.10: Das iPhone kann nach allen lokalen Bonjour-Diensten suchen und sie erkennen. Wenn Sie mit den eingebauten Protokollen vertraut sind, können Sie Anwendungen erstellen, die mit diesen Diensten kommunizieren.*

In diesem Rezept wird die DNS-Diensterkennung genutzt und nach `_services._dns-sd._udp` gesucht. Dadurch wird eine Liste von Einträgen mit Diensttypen zurückgegeben. Eine zweite Phase der Diensterkennung, die Iteration durch diese Liste, zeigt wie in *Abbildung 12.10* die eigentlichen Dienste an.

Wird ein Dienst gefunden, so wird er aufgelöst, um eine Liste der IP-Adressen und der TXT-Eintragsdaten zu erstellen, die (über `startMonitoring`) angefordert werden, um zusätzliche Informationen über den Dienst einzuholen. Versuchen Sie beim Ausprobieren dieses Rezepts, die Daten für den Dienst eines angeschlossenen Druckers (wie des Brother HL-5040 in *Abbildung 12.10*) anzuzeigen, um besonders ausführliche Dienstangaben zu bekommen.

Abgesehen von dem TXT-Callback, der in den vorhergehenden Rezepten nicht eingesetzt wurde, spiegeln die Methoden in diesem Rezept die bisherige Verwendung wider.

```
- (void)netServiceDidResolveAddress:(NSNetService *)netService
{
   NSMutableDictionary *md = [self dictionaryForService:netService];
   if (!md) return;

   NSArray* addresses = [netService addresses];
   if ([addresses count] > 0)
   {
      // Iteriert durch die einzelnen Adressen
      NSMutableArray *naddresses = [NSMutableArray array];
      for (int i = 0; i < addresses.count; i++)
      {
         struct sockaddr* address =
            (struct sockaddr*)[[addresses objectAtIndex:i] bytes];
         NSString *addressString = [self stringFromAddress:address];
         if (!addressString) continue;
         [naddresses addObject:addressString];
      }

      [md setObject:naddresses forKey:@"addresses"];
   }

   [netService release];
}

- (void)netService:(NSNetService *)netService
   didUpdateTXTRecordData:(NSData *)data
{
   // Ruft die TXT-Eintragsdaten ab
   NSDictionary *dict = [NSNetService
      dictionaryFromTXTRecordData:data];
   NSMutableDictionary *md = [self dictionaryForService:netService];
   if (!md) return;
   if ([[dict allKeys] count] == 0) return;
   [md setObject:[dict description] forKey:@"other"];
}
```

```
- (void) netServiceBrowser:(NSNetServiceBrowser *) netServiceBrowser
   didFindService:(NSNetService *) netService
   moreComing:(BOOL) moreServicesComing
{
   // Sucht nach den Diensttypelementen statt nach den eigentlichen
   // Diensten
   if (![netService hostName] && [[netService name] hasPrefix:@"_"])
      [self.services addObject:[netService name]];
   else
   {
      // Dies ist ein echter Dienst, weshalb ein Info-Dictionary
      // erstellt wird
      NSMutableDictionary *md = [NSMutableDictionary dictionary];
      [md setObject:[netService type] forKey:@"type"];
      [md setObject:[netService name] forKey:@"name"];
      [md setObject:[netService domain] forKey:@"domain"];
      [netService startMonitoring];
      [[netService retain] setDelegate:self];
      [netService resolveWithTimeout:0.0f];
      [self.results addObject:md];
      [self.tableView reloadData];
   }

   if (!moreServicesComing)
   {
      // Schließt den Suchvorgang ab
      [self.browser stop];
      self.title = @"Services";

      // Iteriert durch alle verbleibenden Dienste
      if ([self.services count] > 0)
      {
         NSString *type = [self.services objectAtIndex:0];
         [self.services removeObject:type];
         type = [type stringByAppendingString:@"._tcp."];
         [self.browser searchForServicesOfType:type inDomain:@""];
      }
      else
         self.navigationItem.rightBarButtonItem =
            BARBUTTON(@"Rescan", @selector(scan:));
   }
}
```

```
- (void)netServiceBrowser:(NSNetServiceBrowser *)netServiceBrowser
    didNotSearch:(NSDictionary *)errorInfo
{
    // Meldet jegliche Suchfehler
    NSLog (@"Error %@", errorInfo);
}

- (void) scan: (UIBarButtonItem *) bbi
{
    // Deaktiviert die Oberfläche während der Suche
    self.navigationItem.rightBarButtonItem = nil;
    self.title = @"Scanning...";

    // Zeigt eine Liste der noch zu untersuchenden Dienste an
    self.services = [NSMutableArray array];

    // Zeigt eine Liste der vollständig untersuchten Dienste an
    self.results = [NSMutableArray array];

    // Startet die Dienstsuche   self.browser = [[[NSNetServiceBrowser
alloc] init] retain];
    [self.browser setDelegate:self];
    [self.browser searchForServicesOfType:
       @"_services._dns-sd._udp." inDomain:@""];
}
```

► *Rezept 12.10: Bonjour-Dienstsuche*

12.11 Zusammenfassung

GameKit ist ein spannender neuer Baustein für die iPhone-Entwicklung. Aufgrund der leicht zu verwendenden Ad-hoc-Verbindungen über Bluetooth können Sie damit auf einfache Weise Anwendungen bereitstellen, die außerhalb herkömmlicher Netzwerke miteinander kommunizieren. In diesem Kapitel haben Sie gesehen, wie Sie solche Verbindungen erstellen und wie Daten in Echtzeit übertragen werden, sodass Spiele und andere Anwendungen Informationen auf verschiedenen Geräten koordinieren können. Außerdem haben Sie Bonjour-Anwendungen für das iPhone und den Macintosh kennengelernt, die sich nicht auf die proprietären Verbindungen von GameKit stützen, und Sie haben Beispiele für die Online-Verbindungen von GameKit gesehen, für die Sie die Netzwerktechnologie selbst mitbringen müssen. Beachten Sie auch noch folgende zusätzliche Punkte:

- Apple hat zwar noch kein GameKit für den Macintosh ausgeliefert, aber das ist wahrscheinlich nur eine Frage der Zeit. GameKit ist eine bemerkenswerte neue Technologie, die sicherlich erweitert wird.
- Vollständig internet-gestützte GameKit-Verbindungen gehören zwar nicht zum Thema dieses Buches, doch erwarte ich großartige Ergebnisse von den Entwicklern von Spielen, die solche Verbindungen nutzen. Unter Verwendung proprietärer Netzwerktechnologie können Benutzer iPhones miteinander verbinden, um zusammen zu spielen, und das unabhängig davon, ob sich die Spieler in der näheren Umgebung aufhalten oder online angebunden sind. Für Sie als Entwickler ist dies eine echte Revolution auf dem Gebiet der Spiele für Handheld-Geräte – und für die Benutzer wahrscheinlich auch.
- Denken Sie bei der Arbeit mit Voice Chat daran, dass Benutzer in der näheren Umgebung eine akustische Rückkopplung herbeiführen können, sofern sie keine Kopfhörer verwenden. Abgesehen davon können sich Personen, die keine drei Meter voneinander entfernt sind, auch sehr gut ohne technische Hilfsmittel miteinander unterhalten.
- Bonjour läuft auch unter Windows. Nähere Informationen dazu finden Sie, wenn Sie im Internet nach `mDNSResponder` suchen.

13 Netzwerke

Als ein mit dem Internet verbundenes Gerät eignet sich das iPhone besonders zum Abrufen von Daten aus dem Netzwerk und zum Abonnieren webbasierter Dienste. Apple hat diese Plattform großzügig mit einem soliden Grundstock aller möglichen Arten der Rechnervernetzung und unterstützender Technologien ausgestattet. Das iPhone SDK beherrscht unter anderem Sockets, Verschlüsselung für Passwörter (Schlüsselbundverwaltung), die XML-Verarbeitung usw. Dieses Kapitel gibt einen Überblick über verbreitete Techniken zur Rechnervernetzung und enthält Rezepte, die alltägliche Aufgaben vereinfachen. Sie erfahren hier, wie Sie den Netzwerkstatus prüfen, wie Sie den Status auf Veränderungen prüfen und wie Sie die Erreichbarkeit einer Website feststellen. Außerdem lernen Sie, wie Sie Ressourcen asynchron herunterladen und wie Sie auf Authentifizierungsanforderungen reagieren. Wenn Sie dieses Kapitel durchgearbeitet haben, wissen Sie, wie Sie einen FTP-Client, einen eigenen Webbrowser für das iPhone und viele verwandte Programme erstellen.

13.1 Rezept: Den Netzwerkstatus überprüfen

Netzwerkanwendungen benötigen eine aktive Verbindung, um mit dem Internet oder mit Geräten in der näheren Umgebung zu kommunizieren. Bevor ein Programm versucht, Daten zu senden oder zu empfangen, sollte es wissen, ob eine solche Verbindung besteht. Wenn eine Anwendung den Netzwerkstatus überprüft, kann sie den Benutzern Rückmeldung darüber geben, warum bestimmte Funktionen möglicherweise nicht verfügbar sind.

Apple lehnt Anwendungen ab, die den Benutzern Download-Optionen anbieten, ohne vorher den Netzwerkstatus zu bestimmen. Die Prüfer von Apple sind darin geschult, sich genau anzusehen, ob Sie die Benutzer angemessen informieren, vor allem im Fall von Netzwerkfehlern. Ermitteln Sie also stets den Netzwerkstatus, und benachrichtigen Sie die Benutzer entsprechend.

Überdies hat Apple auch schon Anwendungen aufgrund von »übermäßiger Datennutzung« abgelehnt. Wenn Sie vorhaben, umfangreiche Datenstreams in Ihre Anwendung einfließen zu lassen, z. B. Sprachübermittlung oder Daten, müssen Sie den aktuellen Verbindungstyp herausfinden. Stellen Sie für die Benutzer in einem Mobilfunknetzwerk Datenstreams geringerer Qualität bereit, für Benutzer mit Wi-Fi-Verbindungen dagegen solche von höherer Qualität. Apple ist gegenüber Anwendungen, die hohe Ansprüche für die Datenübertragung über ein Mobilfunknetzwerk stellen, nicht sehr tolerant.

Das iPhone kann zurzeit die folgenden Konfigurationszustände bestimmen: das Vorhandensein irgendeiner Form von Netzwerkverbindung, die Verfügbarkeit von Wi-Fi und die von Mobilfunkdiensten. Es gibt zurzeit keine APIs, mit denen das iPhone das Vorhandensein einer Bluetooth-Anbindung erkennen kann (allerdings können Sie Ihre Anwendung so einschränken, dass sie nur auf Bluetooth-fähigen Geräten läuft). Ebenso wenig ist es möglich, herauszufinden, ob ein Benutzer erreichbar ist, ohne einen Datenzugriff anzubieten.

Das Framework *System Configuration* umfasst viele Funktionen für das Netzwerk. Dazu gehört `SCNetworkReachabilityCreateWithAddress`, womit Sie prüfen können, ob eine IP-Adresse erreichbar ist. *Rezept 13.1* zeigt ein einfaches Beispiel für diesen Test.

Das Rezept erstellt einen einfachen Detektor, der ermittelt, ob das vorliegende Gerät über eine ausgehende Verbindung verfügt. Dazu muss es sowohl Zugriff als auch eine aktive Verbindung haben. Diese Methode, die auf Beispielcode von Apple beruht, gibt `YES` zurück, wenn das Netzwerk verfügbar ist, und anderenfalls `NO`. Die hier verwendeten Flags zeigen an, dass das Netzwerk erreichbar ist (`kSCNetworkFlagsReachable`) und dass keine weitere Verbindung erforderlich ist (`kSCNetworkFlagsConnectionRequired`). Darüber hinaus können Sie folgende weitere Flags verwenden:

- `kSCNetworkReachabilityFlagsIsWWAN` prüft, ob der Benutzer das Netzwerk des Betreibers verwendet oder eine lokale WiFi-Anbindung. Ist das Netzwerk zugänglich, kann es über EDGE, GPRS oder eine andere Mobilfunkverbindung erreicht werden. Aufgrund der beschränkten Bandbreite der Verbindung sollten Sie in diesem Fall eine schlankere Variante Ihrer Ressourcen verwenden (z. B. kleine Versionen von Bildern).
- `kSCNetworkReachabilityFlagsIsDirect` sagt Ihnen, ob der Netzwerkverkehr über ein Gateway läuft oder direkt ankommt.

Um auszuprobieren, ob der Code für die Anbindung funktioniert, testen Sie ihn am besten auf einem iPhone. Diese Geräte unterstützen sowohl Mobilfunk als auch Wi-Fi, sodass Sie damit überprüfen können, ob das Netzwerk erreichbar bleibt, wenn Sie eine WWAN-Verbindung nutzen. Testen Sie diesen Code, indem Sie im iPhone-Programm *Einstellungen* den Wi-Fi- und den Flugmodus an- und ausschalten. Denken Sie beim Entwurf Ihrer Anwendung daran, dass es beim Überprüfen der Erreichbarkeit eines Netzwerks zu kurzen Verzögerungen kommen an. Lassen Sie die Benutzer während dieses Vorgangs wissen, was das Programm gerade macht.

```
- (BOOL) connectedToNetwork
{
    // Erstellt Nulladresse
    struct sockaddr_in zeroAddress;
```

```
    bzero(&zeroAddress, sizeof(zeroAddress));
    zeroAddress.sin_len = sizeof(zeroAddress);
    zeroAddress.sin_family = AF_INET;

    // Ruft Erreichbarkeitsflags ab
    SCNetworkReachabilityRef defaultRouteReachability =
        SCNetworkReachabilityCreateWithAddress(NULL,
        (struct sockaddr *)&zeroAddress);
    SCNetworkReachabilityFlags flags;

    BOOL didRetrieveFlags =
        SCNetworkReachabilityGetFlags(
        defaultRouteReachability, &flags);
    CFRelease(defaultRouteReachability);

    if (!didRetrieveFlags)
    {
        printf("Error. Could not recover network flags\n");
        return NO;
    }

    BOOL isReachable = ((flags & kSCNetworkFlagsReachable) != 0);
    BOOL needsConnection = ((flags & kSCNetworkFlagsConnectionRequired)
        != 0);
    return (isReachable && !needsConnection) ? YES : NO;
}
```

▸ *Rezept 13.1: Netzwerkverbindungen prüfen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.2 Rezept: Die Klasse UIDevice um die Netzwerkanbindung erweitern

Die Klasse `UIDevice` bietet Informationen über das zurzeit verwendete Gerät, z. B. den Ladezustand des Akkus, das Modell, die Orientierung usw. Angaben zur Erreichbarkeit von Netzwerken hinzuzufügen, erscheint für eine Klasse, die den Gerätestatus wiedergibt, nur konsequent. *Rezept 13.2* definiert die `UIDevice`-Kategorie `Reachability`, die Aufrufe des Frameworks *System Configuration* kapselt und eine einfache Möglichkeit bietet, den aktuellen Netzwerkstatus zu prüfen. Sie können fragen, ob das Netzwerk aktiv ist und ob es Mobilfunkdienste oder Wi-Fi verwendet.

Die meisten Programme, die die Netzwerkanbindung prüfen, setzen voraus, dass ein verbundenes Gerät, das keine WWAN-Mobilfunkdienste nutzt, über eine Wi-Fi-Anbindung verfügt. Sollte Apple jedoch eines Tages im SDK den Zugriff auf Bluetooth-Dienste ermöglichen, gilt diese Voraussetzung nicht mehr. In *Rezept 13.2* wird daher direkt nach dem Vorhandensein einer Wi-Fi-Verbindung gesucht. Dieser Ansatz stammt von Matt Brown, einem Softwareentwickler und Fan der ersten Auflage dieses Buchs. Hierbei wird die lokale IP-Adresse für Wi-Fi durch maschinennahe (aber SDK-kompatible) Aufrufe ermittelt. Gibt es eine, so gibt die Klasse ein positives Ergebnis für Wi-Fi zurück.

Beachten Sie, dass in dieser Klasse eine andere Netzwerküberprüfung verwendet wird als in *Rezept 13.1*, wobei auch diese vom Apple-Beispielcode inspiriert ist. Mit dem booleschen Wert `ignoresAdHocWiFi` beschränken Sie die Netzwerkprüfung. Ist dieses Flag aktiviert, gibt der Rezeptcode beim Erkennen einer Ad-hoc-Wi-Fi-Verbindung keine Erfolgsmeldung zurück.

```
@implementation UIDevice (Reachability)
SCNetworkConnectionFlags connectionFlags;

// Lokale WiFiIP-Adresse ermitteln
+ (NSString *) localWiFiIPAddress
{
    BOOL success;
    struct ifaddrs * addrs;
    const struct ifaddrs * cursor;

    success = getifaddrs(&addrs) == 0;
    if (success) {
        cursor = addrs;
        while (cursor != NULL) {

            if (cursor->ifa_addr->sa_family == AF_INET &&
                (cursor->ifa_flags & IFF_LOOPBACK) == 0)
            {
                NSString *name = [NSString stringWithUTF8String:
                    cursor->ifa_name];
                if ([name isEqualToString:@"en0"]) // Wi-Fi adapter
                    return [NSString stringWithUTF8String:
                        inet_ntoa(((struct sockaddr_in *)
                        cursor->ifa_addr)->sin_addr)];
            }
            cursor = cursor->ifa_next;
        }
        freeifaddrs(addrs);
    }
    return nil;
}

#pragma mark Checking Connections
```

```
+ (void) pingReachabilityInternal
{
   BOOL ignoresAdHocWiFi = NO;
   struct sockaddr_in ipAddress;
   bzero(&ipAddress, sizeof(ipAddress));
   ipAddress.sin_len = sizeof(ipAddress);
   ipAddress.sin_family = AF_INET;
   ipAddress.sin_addr.s_addr = htonl(
      ignoresAdHocWiFi ? INADDR_ANY : IN_LINKLOCALNETNUM);

   // Erreichbarkeitsflags abrufen
   SCNetworkReachabilityRef defaultRouteReachability =
      SCNetworkReachabilityCreateWithAddress(
      kCFAllocatorDefault, (struct sockaddr *)&ipAddress);
   BOOL didRetrieveFlags = SCNetworkReachabilityGetFlags(
      defaultRouteReachability, &connectionFlags);

   CFRelease(defaultRouteReachability);
   if (!didRetrieveFlags)
      printf("Error. Could not recover flags\n");
}

+ (BOOL) networkAvailable
{
   [self pingReachabilityInternal];
   BOOL isReachable = ((connectionFlags &
      kSCNetworkFlagsReachable) != 0);
   BOOL needsConnection = ((connectionFlags &
      kSCNetworkFlagsConnectionRequired) != 0);
   return (isReachable && !needsConnection) ? YES : NO;
}

+ (BOOL) activeWWAN
{
   if (![self networkAvailable]) return NO;
   return ((connectionFlags &
      kSCNetworkReachabilityFlagsIsWWAN) != 0);
}

+ (BOOL) activeWLAN
{
   return ([UIDevice localWiFiIPAddress] != nil);
}
@end
```

▸ *Rezept 13.2:* `UIDevice` *um den Netzwerkstatus erweitern*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.3 Rezept: Änderungen des Verbindungszustands erkennen

Der Status der Netzwerkanbindung kann sich ändern, während die Anwendung läuft. Eine einzige Prüfung beim Start des Programms reicht für eine Anwendung, die während ihres gesamten Lebenszyklus auf Datenverbindungen angewiesen ist, nicht aus. Wenn eine Netzwerkverbindung ausfällt – oder wenn es endlich möglich ist, sie aufzubauen –, sollten Sie die Benutzer darüber benachrichtigen.

Rezept 13.3 löst diese Aufgabe mithilfe einer weiteren `UIDevice`-Kategorie, die die Erreichbarkeit überwacht. Sie finden hier ein Paar von Methoden, mit denen Sie Erreichbarkeitsbeobachter (*Reachability Watcher*) vorsehen und deaktivieren können. Diese Beobachter wiederum müssen bei Änderungen des Anbindungsstatus benachrichtigt werden. Der Rezeptcode erstellt einen Callback, der bei einer Zustandsänderung ein Beobachterobjekt informiert. Die Überwachung ist in der aktuellen Ausführungsschleife geplant und läuft asynchron. Beim Erkennen einer Änderung wird die Callback-Funktion ausgelöst.

Die Callback-Funktion von *Rezept 13.3* leitet sich selbst zu der besonderen Delegierungsmethode `rechabilityChanged` um, die von einem mit dem Protokoll `ReachabilityWatcher` kompatiblen Objekt implementiert werden muss. Das Beobachterobjekt kann dann (über die Kategorie `Reachability`) die Klasse `UIDevice` nach den aktuellen Flags und dem Netzwerkstatus abfragen.

Die Methode, die den Beobachter einplant, weist die Delegierung als ihren Parameter zu.

```
- (void) reachabilityChanged
{
   [self showAlert:@"Reachability has changed."];
}

- (void) viewDidLoad
{
   self.navigationController.navigationBar.tintColor =
      COOKBOOK_PURPLE_COLOR;
   [UIDevice scheduleReachabilityWatcher:self];
}
```

Im Allgemeinen empfängt Ihre Anwendung für jede Art von Statusänderung einen Callback, also wenn die Mobilfunk- oder die Wi-Fi-Verbindung aufgebaut oder getrennt wird. Die Verbindungseinstellungen der Benutzer (vor allem, ob sich das Gerät bekannte Wi-Fi-Netzwerke merken und daran anmelden soll) haben Einfluss darauf, wie viele Callbacks welcher Art Sie handhaben müssen.

13.3 Rezept: Änderungen des Verbindungszustands erkennen

Informieren Sie die Benutzer, wenn sich die Anbindung ändert, und aktualisieren Sie die Benutzeroberfläche, sodass sie den jeweiligen Status widerspiegelt. Beispielsweise sollten Sie Schaltflächen oder Menüelemente für den Netzwerkzugriff deaktivieren, wenn es keine Möglichkeiten mehr für diesen Zugriff gibt. Mit einer Benachrichtigung irgendeiner Art können Sie die Benutzer wissen lassen, warum sich die Oberfläche geändert hat.

```
// Das Protokoll ReachabilityWatcher definiert den Callback
@protocol ReachabilityWatcher <NSObject>
- (void) reachabilityChanged;
@end

// Aktiviert und deaktiviert Beobachterobjekte mithilfe dieser
// Kategorie
@interface UIDevice (ReachabilityCallback)
+ (BOOL) scheduleReachabilityWatcher: (id) watcher;
+ (void) unscheduleReachabilityWatcher;
@end

@implementation UIDevice (ReachabilityCallback)
SCNetworkConnectionFlags connectionFlags;
SCNetworkReachabilityRef reachability;

#pragma mark Checking Connections

// Aktualisiert die Erreichbarkeits-Flags
+ (void) pingReachabilityInternal
{
   if (!reachability)
   {
      BOOL ignoresAdHocWiFi = NO;
      struct sockaddr_in ipAddress;
      bzero(&ipAddress, sizeof(ipAddress));
      ipAddress.sin_len = sizeof(ipAddress);
      ipAddress.sin_family = AF_INET;
      ipAddress.sin_addr.s_addr =
         htonl(ignoresAdHocWiFi ? INADDR_ANY : IN_LINKLOCALNETNUM);

      reachability = SCNetworkReachabilityCreateWithAddress(
         kCFAllocatorDefault, (struct sockaddr *)&ipAddress);
      CFRetain(reachability);
   }

   // Ruft die Erreichbarkeits-Flags ab
   BOOL didRetrieveFlags = SCNetworkReachabilityGetFlags(reachability,
      &connectionFlags);
```

```
    if (!didRetrieveFlags)
        NSLog(@"Error. Could not recover reachability flags");
}

#pragma mark Monitoring reachability

// Der eigentliche Callback wird zur Delegierung umgeleitet. Der Parameter
// info wird beim Einrichten des Callbacks durch den übergebenen Kontext
// definiert.
static void ReachabilityCallback(SCNetworkReachabilityRef target,
    SCNetworkConnectionFlags flags, void* info)
{
    NSAutoreleasePool *pool = [NSAutoreleasePool new];
    [(id)info performSelector:@selector(reachabilityChanged)];
    [pool release];
}

// Plant einen Beobachter
+ (BOOL) scheduleReachabilityWatcher: (id) watcher
{
    // Muss dem Protokoll entsprechen
    if (![watcher conformsToProtocol:@protocol(ReachabilityWatcher)])
    {
        NSLog(@"Watcher doesn't conform to protocol.");
        return NO;
    }

    [self pingReachabilityInternal];

    // An dieser Stelle wird der Beobachter für den Parameter info
    // eingerichtet
    SCNetworkReachabilityContext context =
        {0, watcher, NULL, NULL, NULL};

    // Richtet den Callback ein
    if(SCNetworkReachabilitySetCallback(reachability,
        ReachabilityCallback, &context))
    {
        if(!SCNetworkReachabilityScheduleWithRunLoop(reachability,
            CFRunLoopGetCurrent(), kCFRunLoopCommonModes))
        {
            NSLog(@"Error Could not schedule reachability");
            SCNetworkReachabilitySetCallback(reachability, NULL, NULL);
            return NO;
        }
```

```
    }
    else
    {
        NSLog(@"Error Could not set reachability callback");
        return NO;
    }

    return YES;
}

+ (void) unscheduleReachabilityWatcher
{
    // Deaktiviert den Callback
    SCNetworkReachabilitySetCallback(reachability, NULL, NULL);

    // Entfernt Beobachter aus der Ausführungsschleife
    if (SCNetworkReachabilityUnscheduleFromRunLoop(reachability,
        CFRunLoopGetCurrent(), kCFRunLoopCommonModes))
        NSLog(@"Unscheduled reachability");
    else
        NSLog(@"Error Could not unschedule reachability");

    CFRelease(reachability);
    reachability = nil;
}
@end
```

▸ *Rezept 13.3: Änderungen der Anbindung überwachen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.4 Rezept: IP- und Hostinformationen abrufen

Bei der iPhone-Netzwerkprogrammierung stellen sich immer wieder dieselben Aufgaben, vor allem der Abruf der Angaben zur lokalen IP-Adresse des iPhones und der Umgang mit Adressstrukturen. *Rezept 13.4* bietet eine Reihe von Hilfsprogrammen, die zum Teil auf Beispielcode von Apple beruhen und mit denen Sie diese Aufgaben meistern können.

Wie in den *Rezepten* 13.2 und 13.3 werden diese Methoden als Kategorie-Erweiterung zur Klasse `UIDevice` hinzugefügt. Sie sind wiederum als Klassenmethoden implementiert, da sie nicht an irgendeine bestimmte Objektinstanz gebunden sind. In diesem Rezept werden folgende Methoden verwendet:

- Mit den beiden Methoden `stringFromAddress:` und `addressFromString:` können Sie Adressstrukturen in eine Stringdarstellung umwandeln und umgekehrt. Diese Methoden wurden im `BonjourHelper`-Rezept aus Kapitel 12, *Verbindungen mit GameKit und Bonjour*, häufig verwendet. Sie arbeiten sehr gut mit der Klasse `NSNetService` zusammen, sodass Sie damit auch `sockaddr`-Strukturen in `NSString`-Instanzen umwandeln können und umgekehrt.
- Die Methode `hostname` gibt den Hostnamen des aktuellen Geräts zurück, wobei sie eine kleine Merkwürdigkeit des iPhones ausbügelt. Der Simulator hängt normalerweise die Domäne `.local` an den aktuellen Hostnamen an, das iPhone jedoch nicht. Diese Routine erzwingt einen Macintosh-kompatiblen Hostnamen. Beachten Sie, dass `hostname` sowie einige andere der folgenden Methoden in einigen Versionen des iPhone-Simulators fehlschlagen können, vor allem unter Snow Leopard (im Gegensatz zu Leopard). Auf dem iPhone funktionieren sie jedoch nach wie vor.
- Mit `getIPAddressForHost:` schlagen Sie die Adresse zu einem gegebenen Hostnamen nach. Im Beispielcode zu diesem Kapitel werden mit dieser Routine die IP-Adressen von *www.google.com* und *www.mobile-dev.de* abgerufen. Es handelt sich hierbei um blockierende Aufrufe, die bis zur Rückgabe der Steuerung eine gewisse Zeit in Anspruch nehmen (vor allem, wenn es den gesuchten Host gar nicht gibt). Setzen Sie sie daher mit Bedacht ein, bevorzugt in einem zweiten Thread oder über eine `NSOperationQueue`.

```
[self doLog:@" Google IP Addy: %@", [UIDevice
   getIPAddressForHost:@"www.google.com"]];
[self doLog:@" Mobile-Dev IP Addy: %@", [UIDevice
   getIPAddressForHost:@"www.mobile-dev.de"]];
```

- Die Methode `localIPAdresse` ermittelt die Adresse des Hosts und gibt sie als String zurück. Wie `getIPAddressForHost:` verwendet sie `getHostByName()`, um den Hostnamen in eine IP-Adresse umzuwandeln.
- Die letzte Methode, `whatIsMyIPDotCom`, hilft dabei, über die Grenzen des LANs hinauszugehen und eine IP-Adresse für Kabel, DSL oder eine ähnliche Verbindung herauszufinden. Sie senden einen Aufruf an die Website *whatismyip.com*, die eine IP-Adresse für die Verbindung zurückgibt. Diese Methode läuft synchron, blockiert also die Steuerung. Stellen Sie stets sicher, dass eine Netzwerkverbindung besteht, bevor Sie diese Methode aufrufen.

```
@implementation UIDevice (IP)

// Erstellt eine Stringdarstellung einer IP-Adresse
+ (NSString *) stringFromAddress: (const struct sockaddr *) address
{
   if(address && address->sa_family == AF_INET) {
```

```
        const struct sockaddr_in* sin = (struct sockaddr_in*) address;
        return [NSString stringWithFormat:@"%@:%d", [NSString
           stringWithUTF8String:inet_ntoa(sin->sin_addr)],
           ntohs(sin->sin_port)];
    }

    return nil;
}

// Erstellt eine Adresse aus einem NSString
+ (BOOL)addressFromString:(NSString *)IPAddress
    address:(struct sockaddr_in *)address
{
    if (!IPAddress || ![IPAddress length]) return NO;

    memset((char *) address, sizeof(struct sockaddr_in), 0);
    address->sin_family = AF_INET;
    address->sin_len = sizeof(struct sockaddr_in);

    int conversionResult = inet_aton([IPAddress UTF8String],
       &address->sin_addr);
    if (conversionResult == 0) return NO;
    return YES;
}

// Gibt den aktuellen Hostnamen zurück
+ (NSString *) hostname
{
    char baseHostName[256];
    int success = gethostname(baseHostName, 255);
    if (success != 0) return nil;
    baseHostName[255] = '\0';


    #if !TARGET_IPHONE_SIMULATOR
        return [NSString stringWithFormat:@"%s.local", baseHostName];
    #else
        return [NSString stringWithFormat:@"%s", baseHostName];
    #endif
}

// Gibt die IP-Adresse (in Stringform) für einen gegebenen Host zurück
+ (NSString *) getIPAddressForHost: (NSString *) theHost
{
    struct hostent *host = gethostbyname([theHost UTF8String]);
```

```
    if (!host) {herror("resolv"); return NULL; }
    struct in_addr **list = (struct in_addr **)host->h_addr_list;
    NSString *addressString = [NSString
        stringWithCString:inet_ntoa(*list[0])
        encoding:NSUTF8StringEncoding];
    return addressString;
}

// Gibt die lokale IP-Adresse zurück
+ (NSString *) localIPAddress
{
    struct hostent *host = gethostbyname([[self hostname] UTF8String]);
    if (!host) {herror("resolv"); return nil;}
    struct in_addr **list = (struct in_addr **)host->h_addr_list;
    return [NSString stringWithCString:inet_ntoa(*list[0])
        encoding:NSUTF8StringEncoding];
}

// Fragt http://whatismyip.com nach der IP-Adresse
+ (NSString *) whatismyipdotcom
{
    // Dies ist ein blockierender Aufruf, weshalb Sie ihn vorsichtig
    // einsetzen müssen
    NSError *error;
    NSURL *ipURL = [NSURL URLWithString:
        @"http://www.whatismyip.com/automation/n09230945.asp"];
    NSString *ip = [NSString stringWithContentsOfURL:ipURL
        encoding:NSUTF8StringEncoding error:&error];
    return ip ? ip : [error localizedDescription];
}
@end
```

▸ *Rezept 13.4: Hilfsmethoden für IP-Adressen und Hostnamen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.5 Rezept: Verfügbarkeit von Websites überprüfen

Nachdem Sie die IP-Adresse einer Website ermittelt haben, prüfen Sie mit der Funktion `SCNetworkReachabilityCreateWithAddress()` ihre Verfügbarkeit. Übergeben Sie einen `sockaddr`-Datensatz mit der IP-Adresse der Site, und prüfen Sie das Flag `kSCNetworkFlagsReachable`, wenn die Funktion die Steuerung zurückgibt. *Rezept 13.5* zeigt die Methode `hostAvailable:` zum Prüfen von Sites. Sie gibt `YES` oder `NO` zurück.

Diese Art von Prüfung erfolgt synchron und blockiert die Interaktion, bis die Methode die Steuerung zurückgibt. In *Rezept 13.5* sieht der Benutzer die Netzwerkaktivitätsanzeige von `UIApplication`, aber in der Praxis sollten Sie diese Tests in einem zweiten Thread durchführen und während der Wartezeit irgendeine Form von Rückmeldung geben. (In den noch folgenden Rezepten in diesem Kapitel wird die Verwendung mehrerer Threads mit den Objekten `NSOperation` und `NSOperationQueue` vereinfacht.) Der Code in diesem Rezept hat bei Tests etwa 30 Sekunden für alle sechs Überprüfungen gebraucht, einschließlich der Suche nach *notverylikely.com*, die dazu dient, einen Fehler zu provozieren.

```
- (BOOL) hostAvailable: (NSString *) theHost
{
    // Ruft den Adressstring für den Host ab
    NSString *addressString = [self getIPAddressForHost:theHost];
    if (!addressString)
    {
        printf("Error recovering IP address from host name\n");
        return NO;
    }

    // Wandelt die Struktur in eine Adresse um
    struct sockaddr_in address;
    BOOL gotAddress = [self addressFromString:addressString
        address:&address];

    if (!gotAddress)
    {
        printf("Error recovering sockaddr address from %s\n",
            [addressString UTF8String]);
        return NO;
    }

    // Prüft Erreichbarkeits-Flags
    SCNetworkReachabilityRef defaultRouteReachability =
        SCNetworkReachabilityCreateWithAddress(NULL,
    (struct sockaddr *)&address);
    SCNetworkReachabilityFlags flags;

    BOOL didRetrieveFlags =
```

```
        SCNetworkReachabilityGetFlags(defaultRouteReachability,
        &flags);
    CFRelease(defaultRouteReachability);

    if (!didRetrieveFlags)
    {
        printf("Error. Could not recover flags\n");
        return NO;
    }

    BOOL isReachable = flags & kSCNetworkFlagsReachable;
    return isReachable ? YES : NO;;
}

#define CHECK(SITE) [self doLog:@"• %@ : %@", SITE, \
    [self hostAvailable:SITE] ? @"available" : @"not available"];

- (void) action: (UIBarButtonItem *) bbi
{
    [[UIApplication sharedApplication]
        setNetworkActivityIndicatorVisible:YES];
    self.log = [NSMutableString string];
    CHECK(@"www.google.com");
    CHECK(@"www.ericasadun.com");
    CHECK(@"www.notverylikely.com");
    CHECK(@"192.168.0.108");
    CHECK(@"mobile-dev.de");
    CHECK(@"www.pearson.com");
    [[UIApplication sharedApplication]
        setNetworkActivityIndicatorVisible:NO];

}
```

► *Rezept 13.5: Die Erreichbarkeit von Websites prüfen*

13.6 Rezepte: Synchrone Downloads

Bei synchronen Downloads fordern Sie Daten vom Internet an, warten, bis diese Daten angekommen sind, und fahren dann mit dem nächsten Schritt Ihrer Anwendung fort. Beispielsweise wurden in *Rezept* 7.1 von Kapitel 7, *Mit Bildern arbeiten*, synchrone Downloads verwendet, um eine Bildansicht mit den von einem URL abgerufenen Inhalten zu initialisieren. Im Folgenden sehen Sie noch einmal den dazu verwendeten Aufruf. Beachten, Sie dass es sich um eine synchrone und blockierende Methode handelt. Sie gibt die Steuerung erst zurück, wenn alle Daten angekommen sind.

```
+ (UIImage *) imageFromURLString: (NSString *) urlstring
{
   // Dies ist ein blockierender Aufruf
   return [UIImage imageWithData:[NSData
      dataWithContentsOfURL:[NSURL URLWithString:urlstring]]];
}
```

Die Klasse `NSURLConnection` ermöglich einen allgemeineren Ansatz für Downloads als die klassenspezifische URL-Initialisierung. Dabei sind sowohl synchrone als auch asynchrone Downloads möglich, wobei für Letztere eine Folge von Delegierungs-Callbacks eingesetzt wird. In *Rezept 13.6* geht es um die einfachere Variante, nämlich um die synchronen Downloads. Als Erstes wird ein `NSMutableURLRequest` mit dem gewünschten URL erstellt. Diese Anforderung wird dann mithilfe der Klasse `NSURLConnection` synchron gesendet.

```
NSMutableURLRequest *theRequest =
   [NSMutableURLRequest requestWithURL:url];
NSData* result = [NSURLConnection sendSynchronousRequest:
   theRequest returningResponse:&response error:&error];
```

Dieser Aufruf blockiert die Steuerung, bis entweder die Anforderung fehlschlägt (wobei `nil` zurückgegeben und ein Fehler hervorgerufen wird) oder die Daten vollständig heruntergeladen sind.

In *Rezept 13.6* wird die synchrone Anforderung in einem zweiten Thread durchgeführt. Dazu wurde die Methode `doLog:`, die Aktualisierungen während des Download-Vorgangs ermöglicht, in threadsicherer Form umgeschrieben. Anstatt die Textansicht direkt zu aktualisieren, führt sie den Selektor `setText:` auf dem Hauptthread (für die grafische Benutzeroberfläche) durch.

```
[textView performSelectorOnMainThread:
   @selector(setText:) withObject:self.log waitUntilDone:NO];
```

Dieses Beispiel kann mit drei vordefinierten URLs getestet werden. Von einem URL wird ein kurzer Film heruntergeladen (3 Mbyte), vom zweiten ein längerer (23 Mbyte). Der dritte URL dagegen existiert nicht und dient dazu, das Verhalten bei Fehlern zu demonstrieren. Die Quelle der Filme ist das *Internet Archive* (*archive.org*), das einen reichen Fundus an Public-Domain-Daten bietet.

Manche Internetprovider zeigen auch dann eine gültige Webseite an, wenn ihnen ein falscher URL übergeben wird. Die im Parameter `response` zurückgegebenen Daten lassen erkennen, ob dies der Fall ist. Dieser Parameter zeigt auf ein `NSURLResponse`-Objekt, das Informationen über die von der URL-Verbindung zurückgegebenen Daten enthält. Dazu gehören auch die erwartete Länge des Inhalts und ein vorgeschlagener Dateiname. Sollte die erwartete Inhaltslänge weniger als null betragen, ist dies ein Hinweis darauf, dass der Anbieter Daten zurückgegeben hat, die nicht mit der Anforderung übereinstimmen.

```
[self doLog:@"Response expects %d bytes",
   [response expectedContentLength]];
```

Wie Sie in *Rezept 13.6* sehen, ist es unsauber und langsam, umfangreiche Downloads in die Hauptbenutzeroberfläche der Anwendung aufzunehmen, selbst unter Verwendung eines zweiten Threads. In *Rezept 13.7* werden diese beiden Probleme durch eine straffer gestaltete Einzweck-Klasse gelöst.

```
- (void) doLog: (NSString *) formatstring, ...
{
   // Hilfsmethode zur Protokollierung
   va_list arglist;
   if (!formatstring) return;
   va_start(arglist, formatstring);
   NSString *outstring = [[[NSString alloc]
      initWithFormat:formatstring arguments:arglist]
      autorelease];
   va_end(arglist);
   [self.log appendString:outstring];
   [self.log appendString:@"\n"];
   [textView performSelectorOnMainThread:
      @selector(setText:) withObject:self.log waitUntilDone:NO];
}

// URLs zum Datenabruf
#define SMALL_URL @"http://www.archive.org/download/Drive-\
   inSaveFreeTv/Drive-in--SaveFreeTv_512kb.mp4"
#define BIG_URL @"http://www.archive.org/download/\
   BettyBoopCartoons/Betty_Boop_More_Pep_1936_512kb.mp4"
#define FAKE_URL @"http://www.idontbelievethisisvalidurlforthisexample.
com"

// Ruft Daten aus dem Netz ab
- (void) getData: (NSNumber *) which
{
   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
   self.log = [NSMutableString string];
   [self doLog:@"Download data now...\n"];
   NSDate *date = [NSDate date];

   // Bestimmt, welche Ressource heruntergeladen werden soll
   NSArray *urlArray = [NSArray arrayWithObjects:
      SMALL_URL, BIG_URL, FAKE_URL, nil];
   NSURL *url = [NSURL URLWithString:
      [urlArray objectAtIndex:[which intValue]]];

   // Bereitet die Anforderung vor und beginnt den Download
   NSMutableURLRequest *theRequest =
      [NSMutableURLRequest requestWithURL:url];
```

```
    NSURLResponse *response;
    NSError *error;
    NSData* result = [NSURLConnection
       sendSynchronousRequest:theRequest
       returningResponse:&response error:&error];

    // Zeigt beim Abschluss den Parameter response an
    [self doLog:@"Response expects %d bytes",
       [response expectedContentLength]];
    [self doLog:@"Response suggested file name: %@",
       [response suggestedFilename]];

    if ([response suggestedFilename])
       self.savePath = [DEST_PATH stringByAppendingString:
            [response suggestedFilename]];

    // Sucht nach Fehlern oder speichert die Daten
    if (!result)
       [self doLog:@"Error downloading data: %@.",
          [error localizedDescription]];
    else if ([response expectedContentLength] < 0)
       [self doLog:@"Error with download. Carrier redirect?"];
    else
    {
       [self doLog:@"Download succeeded."];
       [self doLog:@"Read %d bytes", result.length];
       [self doLog:@"Elapsed time: %0.2f seconds.",
          -1.0f * [date timeIntervalSinceNow]];
       [result writeToFile:DEST_PATH atomically: YES];
       [self doLog:@"Data written to file: %@.", self.savePath];
    }

    // Räumt nach dem Download auf
    [self performSelectorOnMainThread:
       @selector(finishedGettingData)
       withObject:nil waitUntilDone:NO];
    [pool release];
}

- (void) action: (UIBarButtonItem *) bbi
{
    // Beginnt den Download in einem neuen Thread
    NSNumber *which = [NSNumber numberWithInt:
       [(UISegmentedControl *)self.navigationItem.titleView
       selectedSegmentIndex]];
```

```
    self.navigationItem.rightBarButtonItem = nil;
    [(UISegmentedControl *)self.navigationItem.titleView setEnabled:NO];
    [[UIApplication sharedApplication]
       setNetworkActivityIndicatorVisible:YES];
    [NSThread detachNewThreadSelector:@selector(getData:)
       toTarget:self withObject:which];
}
```

▸ *Rezept 13.6: Synchrone Downloads*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.7 Rezept: Asynchrone Downloads

Mit asynchronen Downloads können Anwendungen Daten im Hintergrund herunterladen, wodurch der Code nicht blockiert wird, während er auf den Abschluss des Vorgangs wartet. Wenn Sie beispielsweise asynchrone Downloads für Tabellenansichten verwenden, können Sie Platzhalterbilder anzeigen, während Sie Vorschaubilder von Diensten wie YouTube herunterladen. *Rezept 13.7* zeigt die asynchrone Verwendung von `NSURLConnection`-Verbindungen. Sie finden dort die Hilfsklasse `DownloadHelper`, die alle Einzelheiten beim Herunterladen von Daten kapselt. Dies funktioniert auf folgende Weise: Anstatt eine synchrone Anforderung zu senden, wird die Verbindung initialisiert und eine Delegierung zugewiesen.

```
NSURLConnection *theConnection = [[NSURLConnection alloc]
   initWithRequest:theRequest delegate:sharedInstance];
```

Wenn Sie die Verbindung auf diese Weise einrichten, werden die Daten asynchron heruntergeladen, aber die grafische Benutzeroberfläche kann trotzdem nicht ohne Blockierung aktualisiert werden. Um dies zu erreichen, müssen Sie die Verbindung in der aktuellen Ausführungsschleife einplanen. Achten Sie darauf, die Verbindung nach dem Abschluss des Downloads wieder dort herauszunehmen. Ein Download ist abgeschlossen, wenn alle angeforderten Daten empfangen wurden oder wenn er mit einem Fehler abgebrochen wird.

```
[self.urlconnection scheduleInRunLoop:[NSRunLoop currentRunLoop]
   forMode:NSRunLoopCommonModes];
```

Mit Delegierungsmethoden können Sie den Lebenszyklus des Downloads nachverfolgen. Dabei erhalten Sie Aktualisierungen, wenn neue Daten verfügbar sind, wenn die Daten komplett heruntergeladen sind und wenn der Download fehlschlägt. Zur Unterstützung dieser Callbacks definiert die Klasse `DownloadHelper` die folgenden wichtigen Eigenschaften:

- Die Eigenschaft `urlString` zeigt auf die angeforderte Ressource. Sie wird zur Initialisierung der URL-Anfrage verwendet, mit der der Download-Vorgang beginnt (`requestWithURL:`).
- Die Eigenschaft `response` verzeichnet die erwartete Inhaltslänge und den Dateinamen für das Download-Objekt. Diese Eigenschaft wird im Delegierungs-Callback `connection:didReceiveResponse:` zurückgegeben.
- Die Eigenschaft `data` speichert die beim Download empfangenen Daten. Es handelt sich dabei um eine Instanz der Klasse `NSMutableData`. Wenn neue Daten ankommen (`connection:didReceiveData:`), fügt die Hilfsklasse sie hinter den bereits vorhandenen Daten an.
- Die Eigenschaft `delegate` zeigt auf das Clientobjekt. Die Delegierung, die das Protokoll `DownloadHelperDelegate` implementieren muss, wird mit optionalen Callbacks aktualisiert, während der Download voranschreitet. Diese externe Delegierung unterscheidet sich von der internen, die für das `NSURLConnection`-Objekt verwendet wird. Externe Callbacks treten auf, wenn der Download erfolgreich abgeschlossen wird (`connection:didFinishLoading:`), wenn er fehlschlägt (`connection:didFailWithError:`), wenn der Dateiname bekannt wird (`connection:didReceiveResponse:`) und jeweils dann, wenn ein Datenabschnitt ankommt (`connection:didReceiveData:`). Wenn Sie über den optionalen Callback `dataDownloadAtPercent:` den Prozentsatz der empfangenen Daten übergeben, kann der Datenkonsument ein Fortschrittsfenster aktualisieren, um dem Benutzer zu zeigen, welcher Anteil der Daten schon heruntergeladen ist.
- Die Eigenschaft `urlconnection` speichert das aktuelle `NSURLConnection`-Objekt. Sie wird vorgehalten, damit die Methode `cancel` der Klasse `DownloadHelper` einen laufenden Download anhalten kann (`[sharedInstance.urlconnection cancel]`).

Der Client beginnt den Vorgang, indem er eine `DownloadHelper`-Delegierung zuweist (wahrscheinlich sich selbst) und wie folgt einen Download anfordert. Wie Sie hier sehen, hat diese Hilfsklasse eine extrem einfache Entwicklerschnittstelle:

```
[DownloadHelper sharedInstance].delegate = self;
[DownloadHelper download:urlString];
```

Zwar sind alle Delegierungsmethoden von `DownloadHelper` optional, doch sollte die Delegierung mindestens `didReceiveData:` implementieren, die nach dem Herunterladen sämtlicher Daten aufgerufen wird.

Hinweis

Rezept 13.7 setzt voraus, dass Ihnen der Datenanbieter eine erwartete Inhaltslänge zusichert. Wenn der Server die Antwort in mehreren Datenabschnitten zurückgibt (`Transfer-Encoding: ➥chunked`), ist die Inhaltslänge in der Antwort nicht angegeben. *Rezept 13.7* funktioniert nicht bei abschnittsweise übermittelten Daten, da der Code die Inhaltslänge prüft und fehlschlägt, wenn dieser Wert unbekannt ist (also bei `NSURLResponsseUnknownLength`).

```
@implementation DownloadHelper
@synthesize response;
@synthesize data;
@synthesize delegate;
@synthesize urlString;
@synthesize urlconnection;
@synthesize isDownloading;

- (void) start
{
   self.isDownloading = NO;

   // Wandelt den URL-String in einen URL um
   NSURL *url = [NSURL URLWithString:self.urlString];
   if (!url)
   {
      NSString *reason = [NSString stringWithFormat:
         @"Could not create URL from string %@", self.urlString];
      DELEGATE_CALLBACK(dataDownloadFailed:, reason);
      return;
   }

   // Erstellt die Anforderung
   NSMutableURLRequest *theRequest = [NSMutableURLRequest
      requestWithURL:url];
   if (!theRequest)
   {
      NSString *reason = [NSString stringWithFormat:
         @"Could not create URL request from string %@",
         self.urlString];
      DELEGATE_CALLBACK(dataDownloadFailed:, reason);
      return;
   }

   // Erstellt die Verbindung
      self.urlconnection = [[NSURLConnection alloc]
   initWithRequest:theRequest delegate:self];
   if (!self.urlconnection)
   {
      NSString *reason = [NSString stringWithFormat:
         @"URL connection failed for string %@", self.urlString];
      DELEGATE_CALLBACK(dataDownloadFailed:, reason);
      return;
   }
```

```
    self.isDownloading = YES;

    // Erstellt das neue Datenobjekt
    self.data = [NSMutableData data];
    self.response = nil;

    [self.urlconnection scheduleInRunLoop:
        [NSRunLoop currentRunLoop] forMode:NSRunLoopCommonModes];
}

- (void) cleanup
{
    // Räumt alle Eigenschaften auf
    self.data = nil;
    self.response = nil;
    self.urlconnection = nil;
    self.urlString = nil;
    self.isDownloading = NO;
}

- (void)connection:(NSURLConnection *)connection
    didReceiveResponse:(NSURLResponse *)aResponse
{
    // Speichert die Antwortinformationen
    self.response = aResponse;

    // Prüft, ob die Verbindung schlecht ist
    if ([aResponse expectedContentLength] < 0)
    {
        NSString *reason = [NSString stringWithFormat:
            @"Invalid URL [%@]", self.urlString];
        DELEGATE_CALLBACK(dataDownloadFailed:, reason);
        [connection cancel];
        [self cleanup];
        return;
    }

    if ([aResponse suggestedFilename])
        DELEGATE_CALLBACK(didReceiveFilename:,
        [aResponse suggestedFilename]);
}

- (void)connection:(NSURLConnection *)connection
    didReceiveData:(NSData *)theData
{
```

```
    // Hängt die neuen Daten an und aktualisiert die Delegierung
    [self.data appendData:theData];

    // Hier wird vorausgesetzt, dass eine Inhaltslänge zugesichert ist

    if (self.response)
    {
        float expectedLength = [self.response expectedContentLength];
        float currentLength = self.data.length;
        float percent = currentLength / expectedLength;
        DELEGATE_CALLBACK(dataDownloadAtPercent:, NUMBER(percent));
    }
}

- (void)connectionDidFinishLoading:(NSURLConnection *)connection
{
    // Räumt nach dem Abschluss des Downloads auf
    self.response = nil;

    // Die Delegierung ist für die Freigabe der Daten verantwortlich
    if (self.delegate)
    {
        NSData *theData = [self.data retain];
        DELEGATE_CALLBACK(didReceiveData:, theData);
    }

    [self.urlconnection unscheduleFromRunLoop:[NSRunLoop
        currentRunLoop] forMode:NSRunLoopCommonModes];
    [self cleanup];
}

- (void)connection:(NSURLConnection *)connection
    didFailWithError:(NSError *)error
{
    self.isDownloading = NO;
    NSLog(@"Error Failed connection, %@",
        [error localizedDescription]);
    DELEGATE_CALLBACK(dataDownloadFailed:, @"Failed Connection");

    [self cleanup];
}

+ (DownloadHelper *) sharedInstance
{
    if(!sharedInstance) sharedInstance = [[self alloc] init];
```

```
    return sharedInstance;
}

+ (void) download:(NSString *) aURLString
{
    // Beginnt einen neuen Download
    if (sharedInstance.isDownloading)
    {
        NSLog(@"Error Cannot start new download yet.");
        DELEGATE_CALLBACK(dataDownloadFailed:, @"");
        return;
    }
    sharedInstance.urlString = aURLString;
    [sharedInstance start];
}

+ (void) cancel
{
    if (sharedInstance.isDownloading)
        [sharedInstance.urlconnection cancel];
}
@end
```

▸ *Rezept 13.7: Hilfsklasse für Download-Vorgänge*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.8 Rezept: Authentifizierungsanforderungen verarbeiten

Manche Websites sind dadurch geschützt, dass die Eingabe eines Benutzernamens und eines Passworts verlangt wird. Mit `NSURLConnection` können Sie auf solche Sites zugreifen, indem Sie auf die Authentifizierungsanforderung reagieren. *Rezept 13.8* erweitert `DownloadHelper` um diese Möglichkeit. Dazu erstellt die Klasse ein neues `NSURLCredential`-Objekt und initialisiert es mit einem Benutzernamen und einem Passwort. Dieses Objekt wird dem Absender der Anforderung übergeben, der dann entscheidet, ob er es akzeptiert.

Um die Authentifizierung zu testen, nehmen Sie Verbindung mit *http://ericasadun.com/Private* auf. Diese Site wurde eigens für die Verwendung mit diesem Rezept eingerichtet. Für den Testordner werden der Benutzername `PrivateAccess` und das Passwort `tuR7!mZ#eh` verwendet. Um `DownloadHelper` so einzurichten, dass die Klasse auf eine Authentifizierungsanforderung reagieren kann, benötigen Sie folgende Aufrufe:

```
NSString *urlString = @"http://ericasadun.com/Private/";
[DownloadHelper sharedInstance].username = @"PrivateAccess";
[DownloadHelper sharedInstance].password = @"tuR7!mZ#eh";
[DownloadHelper sharedInstance].delegate = self;
[DownloadHelper download:urlString];
```

Um eine unautorisierte Verbindung zu testen – bei der Ihnen der Zugriff verweigert wird –, setzen Sie Benutzername und Passwort auf `nil` oder auf einen falschen String. Bei `nil` erhält die anfordernde Site Anmeldeinformationen mit `nil`, was unmittelbar zu einem Fehler führt. Wenn Sie einen falschen String eingeben, schlägt die Authentifizierung fehl, sobald der Absender die Anmeldeinformationen zurückweist.

```
- (void)connection:(NSURLConnection *)connection
   didReceiveAuthenticationChallenge:
   (NSURLAuthenticationChallenge *)challenge
{
   if (!username || !password)
   {
      [[challenge sender] useCredential:nil
         forAuthenticationChallenge:challenge];
      return;
   }
   NSURLCredential *cred = [[[NSURLCredential alloc]
      initWithUser:username password:password
      persistence:NSURLCredentialPersistenceNone] autorelease];
   [[challenge sender] useCredential:cred
      forAuthenticationChallenge:challenge];
}

- (void)connection:(NSURLConnection *)connection
   didCancelAuthenticationChallenge:
   (NSURLAuthenticationChallenge *)challenge
{
   NSLog(@"Challenge cancelled");
}
```

▸ *Rezept 13.8: Authentifizierung mit* `NSURLCredential`*-Instanzen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.9 Rezept: Den Schlüsselbund zum Speichern sensibler Daten verwenden

Im iPhone-Schlüsselbund können Sie die Anmeldeinformationen von Benutzern auf sichere Weise speichern. »Sichere« Passwortfelder, deren Daten in regulären Dateien gespeichert werden, sind nicht besonders sicher. Die Eingabe wird zwar auf dem Bildschirm verschleiert, doch wenn Dateien auf Festplatte geschrieben werden, enthalten sie Klartext. Dies gilt auch für Voreinstellungsdateien. Zwar könnten Sie die Daten selbst verschlüsseln, doch dadurch beschwören Sie für Ihre Anwendung eine Reihe von Exportproblemen herauf. Mit dem Schlüsselbund bietet Apple einen eingebauten Dienst für sichere Passwörter, der die Arbeit für Sie erledigt.

13.9.1 Der Schlüsselbund-Wrapper

Der Schlüsselbund verwendet das Framework Security, das ausschließlich für das iPhone-Gerät zur Verfügung steht. Auf dem Simulator können Sie Anwendungen, die den Schlüsselbund nutzen, weder programmieren noch testen. Wenn Sie das Framework zu Ihrem Projekt hinzufügen, müssen Sie dieses auf dem Gerät bereitstellen.

Die Programmierung für den Schlüsselbund ist kompliziert. Zum Glück bietet Apple die einfache Klasse `KeychainItemWrapper` an, die die grundlegende Arbeit mit dem Schlüsselbund für Sie erledigt. Um diesen Wrapper zu verwenden, erstellen Sie eine Instanz und initialisieren sie mit einer ID und einer Zugriffsgruppe. Erstellen Sie für jedes Anmeldepaar, das Sie verwenden wollen, eine eindeutige ID. Dadurch können Sie Anmeldeinformationen für jede Art von Konto hinzufügen.

```
self.wrapper = [[KeychainItemWrapper alloc]
   initWithIdentifier:@"Twitter" accessGroup:nil];
```

Sofern Sie den Schlüsselbund nicht über mehrere Anwendungen hinweg nutzen müssen, setzen Sie die Zugriffsgruppe auf `nil` (*Rezept 13.12* zeigt anwendungsübergreifende Schlüsselbunde.)

Um Elemente in den Schlüsselbund zu schreiben, speichern Sie Benutzername und Passwort mit den vordefinierten Schlüsseln für Konto und Daten, wie Sie es auch in einem Dictionary tun würden. Der Wrapper aktualisiert automatisch den Schlüsselbund, ohne dass Sie noch irgendetwas tun müssten, um die Speicherung zu bestätigen.

```
[self.wrapper setObject:uname forKey:(id)kSecAttrAccount];
[self.wrapper setObject:pword forKey:(id)kSecValueData];
```

Zum Abrufen der Daten verwenden Sie `objectForKey:`

```
NSString* uname = [self.wrapper objectForKey:(id)kSecAttrAccount];
NSString* pword = [self.wrapper objectForKey:(id)kSecValueData];
```

Rezept 13.9 stellt die modale Klasse `SettingsViewController` vor, die einen Benutzernamen und ein Passwort in ihr Textfeld lädt, wenn sie angezeigt wird, und beim Ausblenden alle Änderungen speichert. Dazu nutzt sie den Schlüsselbund-Wrapper.

Achten Sie in diesem Rezept auf den Umgang mit den Leistenschaltflächen. Zuerst gibt es eine **Zurück**-Schaltfläche, bis das Textfeld angesprochen wird. Sobald der Benutzer mit der Bearbeitung beginnt, erscheinen zwei neue Schaltflächen zum Speichern und zum Abbrechen. Durch diese Einbeziehung des Kontextes stehen Schaltflächen mit aussagekräftigeren Bezeichnungen zur Verfügung als einfach nur **Done** (**Fertig**).

13.9.2 Dauerhafte Speicherung von Schlüsselbunddaten

Schlüsselbunddaten bestehen auch nach der Deinstallation Ihrer Anwendung fort. Die Apple-Abteilung für den Kontakt mit Entwicklern schreibt dazu: »Schlüsselbundelemente, die von einer beliebigen Anwendung erstellt wurden, bestehen auch nach der Deinstallation einfach aufgrund der Tatsache fort, dass die Speicherung des Schlüsselbunds nicht im Anwendungsbundle stattfindet und dass es keine Möglichkeit gibt, durch die das System benachrichtigt werden kann, bei einer Deinstallation auch alle zugehörigen Schlüsselbundelemente zu entfernen. Es muss auch zwischen der Gefahr, sensible Passwörter bei arglistiger Deinstallation zu verlieren, und dem Anliegen abgewogen werden, sensible Passwörter für den oder die Benutzer unversehrt und möglicherweise außer Reichweite gesichert zu bewahren.«

Dieses Verhalten ermöglicht es Ihnen, den Schlüsselbund dazu zu verwenden, Informationen dauerhaft auf dem iPhone zu erhalten. Sie können die Benutzerregistrierung nachverfolgen oder die Verwendung im Demo-Betrieb einschränken. Beständigkeit bedeutet, dass einzig eine Firmware-Neuinstallation (ohne Speicherung einer Sicherheitskopie) die Daten löschen kann.

```
@implementation SettingsViewController

@synthesize wrapper;

- (void)textFieldDidBeginEditing:(UITextField *)textField
{
   // Erlaubt dem Benutzer bei der Eingabe, zu speichern und abzubrechen
   self.navigationItem.rightBarButtonItem =
      BARBUTTON(@"Save", @selector(dismiss:));
   self.navigationItem.leftBarButtonItem =
      BARBUTTON(@"Cancel", @selector(dismissCancel:));
}

- (void) dismiss: (id) sender
{
   // Ruft die Daten ab, speichert sie und wird ausgeblendet
   NSString *uname = [username text];
   NSString *pword = [password text];

   if (uname) [self.wrapper setObject:uname
      forKey:(id)kSecAttrAccount];
   if (pword) [self.wrapper setObject:pword
```

```
        forKey:(id)kSecValueData];
    [self.parentViewController dismissModalViewControllerAnimated:YES];
}

- (void) dismissCancel: (id) sender
{
    // Der Controller wird ohne zu speichern entfernt
    [self.parentViewController dismissModalViewControllerAnimated:YES];
}

- (void) viewDidLoad
{
    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
    self.navigationItem.leftBarButtonItem =
        BARBUTTON(@"Back", @selector(dismissCancel:));

    // Die ID bezieht sich auf nachfolgende Rezepte, die auf diesem
    // Beispiel aufbauen
    self.wrapper = [[KeychainItemWrapper alloc]
        initWithIdentifier:@"Twitter" accessGroup:nil];
    [self.wrapper release];

    // Ruft einen gespeicherten Benutzernamen und ein Passwort ab
    NSString *uname = [self.wrapper objectForKey:(id)kSecAttrAccount];
    NSString *pword = [self.wrapper objectForKey:(id)kSecValueData];

    if (uname) username.text = uname;
    if (pword) password.text = pword;

    username.delegate = self;
    password.delegate = self;
}

- (void) dealloc
{
    username = nil;
    password = nil;
    self.wrapper = nil;
    [super dealloc];
}
@end
```

▸ *Rezept 13.9: Zugriff auf den iPhone-Schlüsselbund über eine modale Einstellungsansicht*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.10 Daten mit POST hochladen

`NSURLRequest`-Instanzen sind weit vielseitiger, als die Rezepte in diesem Kapitel bis jetzt erkennen ließen. *Rezept 13.10* erstellt eine Anforderung, die Daten an den Social-Networking-Dienst Twitter sendet. Dazu wird eine POST-Anforderung im HTTP-Stil eingerichtet und mit einer Nachricht versehen.

```
[urlRequest setHTTPMethod:@"POST"];
[urlRequest setHTTPBody:
   [body dataUsingEncoding:NSUTF8StringEncoding]];
[urlRequest setValue:@"application/x-www-form-urlencoded"
   forHTTPHeaderField:@"Content-Type"];
```

In der URL-Anforderung können Sie Headerfelder, einen Nachrichtentext und weitere Elemente festlegen. Im Grunde handelt es sich um eine internet-geeignete Klasse, die sich sehr gut für die Kommunikation mit Online-Diensten einsetzen lässt.

In diesem Beispiel werden Benutzername und Passwort als Teil des URLs gesendet:

```
http://username:password@twitter.com/statuses/update.xml
```

Das ist bei vielen Diensten jedoch nicht der Normalfall (siehe *Rezept 13.11*). Da Twitter diesen Ansatz verfolgt, bietet gerade dieser Dienst ein gutes Beispiel für die einfachste Art von POST-API.

13.10.1 NSOperationQueue

In diesem Beispiel erfolgt der Upload über eine synchrone Anforderung, was bis zu einer Minute in Anspruch nehmen kann. Um zu verhindern, dass die Aktualisierung der Benutzeroberfläche blockiert wird, ist der gesamte Übertragungsvorgang in `TwitterOperation` eingebettet, eine Unterklasse von `NSOperation`. Operationen kapseln Code und Daten für eine einzelne Aufgabe, sodass Sie diese Aufgabe asynchron ausführen können.

`NSOperation`-Objekte können Sie an eine asynchrone `NSOperationQueue` übermitteln. Operationswarteschlangen verwalten die Ausführung der einzelnen Operationen, wobei jede einzelne mit einer Priorität versehen und in der Warteschlange platziert wird. Dort werden die Operationen nach ihrer Priorität abgearbeitet. Wenn Sie eine Operation in eine Warteschlange stellen, können Sie in der Benutzeroberfläche Elemente darstellen (z. B. eine Aktivitätsanzeige oder einen Fortschrittsbalken), ohne dass diese durch die Ausführung der Operation blockiert werden.

```
TwitterOperation *operation = [[[TwitterOperation alloc] init]
   autorelease];
operation.delegate = self;
operation.theText = text;

NSOperationQueue *queue = [[[NSOperationQueue alloc] init]
   autorelease];
[queue addOperation:operation];
```

Da die Operation asynchron ausgeführt wird, muss der Hauptansichtscontroller auf irgendeine Weise bestimmen können, wann der Upload abgeschlossen ist. Während des Vorgangs ist die Benutzeroberfläche deaktiviert, und stattdessen wird ein `UIActivityIndicatorView` angezeigt. In diesem Beispiel sendet die Operation eine Delegierungs-Callback-Methode (`doneTweeting:`). Der Callback informiert die Hauptoberfläche, wann sie wieder in den normalen interaktiven Modus wechseln muss.

Wenn Sie Unterklassen von `NSOperation` erstellen, müssen Sie stets die Methode `main` implementieren, die bei Ausführung der Operation aufgerufen wird. Wenn `main` die Steuerung zurückgibt, ist die Operation abgeschlossen.

```
@implementation TwitterOperation
@synthesize wrapper;
@synthesize theText;
@synthesize delegate;

#define NOTIFY_AND_LEAVE(X) {[self cleanup:X]; return;}

- (void) cleanup: (NSString *) output
{
   // Räumt nach Erfolg oder Fehlschlag auf
   self.theText = nil;
   self.wrapper = nil;
   if (self.delegate && [self.delegate
      respondsToSelector:@selector(doneTweeting:)])
      [self.delegate doneTweeting:output];
}

- (void) main
{
   if (!theText || ![theText length])
      NOTIFY_AND_LEAVE(@"You cannot tweet an empty message.");

   // Ruft Benutzeranmeldeinformationen ab
   self.wrapper = [[KeychainItemWrapper alloc]
      initWithIdentifier:@"Twitter" accessGroup:nil];
   [self.wrapper release];
   NSString *uname = [self.wrapper objectForKey:(id)kSecAttrAccount];
   NSString *pword = [self.wrapper objectForKey:(id)kSecValueData];
```

```
    if (!uname || !pword || (!uname.length) || (!pword.length))
       NOTIFY_AND_LEAVE(@"Please enter your account credentials");

    // Verarbeitet Benutzeranmeldeinformationen
    NSString *unpwraw = [NSString stringWithFormat:@"%@:%@",
       uname, pword];
    NSString *unpw = ENCODE(unpwraw);
    NSString *theTweet = ENCODE(theText);
    NSString *body = [NSString stringWithFormat:
       @"source=iTweet&status=%@", theTweet];

    // Richtet Twitter-API-Anforderungen ein
    NSString *baseurl = [NSString stringWithFormat:
       @"http://%@@twitter.com/statuses/update.xml", unpw];
    NSURL *url = [NSURL URLWithString:baseurl];
    NSMutableURLRequest *urlRequest =
       [NSMutableURLRequest requestWithURL:url];

    if (!urlRequest)
       NOTIFY_AND_LEAVE(@"Error creating the URL Request");

    [urlRequest setHTTPMethod: @"POST"];
    [urlRequest setHTTPBody:
       [body dataUsingEncoding:NSUTF8StringEncoding]];
    [urlRequest setValue:@"application/x-www-form-urlencoded"
       forHTTPHeaderField:@"Content-Type"];
    [urlRequest setValue:@"iTweet" forHTTPHeaderField:@"X-Twitter-Client"];

    NSLog(@"Contacting Twitter. This can take a minute or so...");

    // Lanciert die Anforderung und wartet auf Antwort
    NSError *error;
    NSURLResponse *response;
    NSData *tw_result = [NSURLConnection
       sendSynchronousRequest:urlRequest returningResponse:&response
       error:&error];
    NSString *tw_output = [NSString stringWithFormat:
       @"Submission error: %@", [error localizedDescription]];
    if (!tw_result) NOTIFY_AND_LEAVE(tw_output);

    // Räumt auf und benachrichtigt die Delegierung
    [self cleanup:[[[NSString alloc] initWithData:tw_result
       encoding:NSUTF8StringEncoding] autorelease]];
}
@end
```

▸ *Rezept 13.10: Twitter-Meldungen mit POST hochladen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.11 Rezept: Daten hochladen

In *Rezept 13.10* wurde zur Übertragung der Daten ein einfaches Formular mit URL-Kodierung versendet. *Rezept 13.11* geht noch einen Schritt weiter und führt die Datenübertragung mit einem mehrteiligen Formular vor. Damit können Sie mithilfe der Twitter-Anmeldeinformationen Bilder zum Dienst TwitPic.com hochladen. Die TwitPic-API finden Sie unter *http://twitpic.com/api/uploadAndPost*. Dazu brauchen Sie einen Benutzernamen, ein Passwort und binäre Bilddaten.

Die Herausforderung bei *Rezept 13.11* besteht darin, einen korrekt formatierten Nachrichtenrumpf zu erstellen, der von TwitPic genutzt werden kann. Der Code implementiert eine Methode, die aus einem Dictionary von Schlüsseln und Werten Formulardaten generiert. In diesem Beispiel sind die Objekte in dem Dictionary auf Strings und Bilder beschränkt, doch Sie können auch weitere Datentypen verwenden, indem Sie im String für den Inhaltstyp andere MIME-Typen angeben.

```
#define NOTIFY_AND_LEAVE(X) {[self cleanup:X]; return;}
#define DATA(X) [X dataUsingEncoding:NSUTF8StringEncoding]

#define IMAGE_CONTENT @"Content-Disposition: form-data; name=\"%@\";\
   filename=\"image.jpg\"\r\nContent-Type: image/jpeg\r\n\r\n"
#define STRING_CONTENT @"Content-Disposition: form-data; \
   name=\"%@\"\r\n\r\n"
#define MULTIPART @"multipart/form-data; boundary=------------\
   0x0x0x0x0x0x0x0x"

@implementation TwitPicOperation
@synthesize wrapper;
@synthesize theImage;
@synthesize delegate;

- (void) cleanup: (NSString *) output
{
   self.theImage = nil;
   self.wrapper = nil;
   if (self.delegate &&
      [self.delegate respondsToSelector:@selector(doneTweeting:)])
      [self.delegate doneTweeting:output];
}

- (NSData*)generateFormDataFromPostDictionary:(NSDictionary*)dict
```

```
{
   // Legt den Rahmen fest
   id boundary = @"------------0x0x0x0x0x0x0x0x";

   // Richtet ein Dictionary ein
   NSArray* keys = [dict allKeys];

   // Richtet die Ausgabedaten ein
   NSMutableData* result = [NSMutableData data];

   for (int i = 0; i < [keys count]; i++)
   {
      // Ruft den nächsten Schlüssel ab
      id value = [dict valueForKey: [keys objectAtIndex:i]];

      // Fügt die Separatordaten hinzu
      [result appendData:[[NSString stringWithFormat:@"--%@\r\n",
         boundary] dataUsingEncoding:NSUTF8StringEncoding]];

      if ([value isKindOfClass:[NSData class]])
      {
         // Verarbeitet Bilddaten
         NSString *formstring =
            [NSString stringWithFormat:IMAGE_CONTENT,
            [keys objectAtIndex:i]];
         [result appendData: DATA(formstring)];
         [result appendData:value];

      }
      else
      {
         // Es wird angenommen, dass es sich bei allen Feldern, die keine
         // Bilder sind, um Strings handelt
         NSString *formstring =
            [NSString stringWithFormat:STRING_CONTENT,
            [keys objectAtIndex:i]];
         [result appendData: DATA(formstring)];
         [result appendData:DATA(value)];
      }

      // Ende des Teils
      NSString *formstring = @"\r\n";
      [result appendData:DATA(formstring)];
   }

   // Alle Daten sind hinzugefügt, weshalb ein weiterer Rahmen
```

```
    // angehängt wird
    NSString *formstring =[NSString stringWithFormat:@"--%@--\r\n",
        boundary];
    [result appendData:DATA(formstring)];
    return result;
}

- (void) main
{
    if (!self.theImage)
        NOTIFY_AND_LEAVE(@"Please set image before uploading.");

    // Verwendet die Twitter-Anmeldeinformationen für TwitPic
    self.wrapper = [[KeychainItemWrapper alloc]
        initWithIdentifier:@"Twitter" accessGroup:nil];
    [self.wrapper release];

    NSString *uname = [self.wrapper objectForKey:(id)kSecAttrAccount];
    NSString *pword = [self.wrapper objectForKey:(id)kSecValueData];

    if (!uname || !pword || (!uname.length) || (!pword.length))
        NOTIFY_AND_LEAVE(@"Please enter your account credentials.");

    NSMutableDictionary* post_dict =
        [[NSMutableDictionary alloc] init];
    [post_dict setObject:uname forKey:@"username"];
    [post_dict setObject:pword forKey:@"password"];
    [post_dict setObject:@"Posted from iTweet" forKey:@"message"];
    [post_dict setObject:UIImageJPEGRepresentation(self.theImage,
        0.75f) forKey:@"media"];

    // Erstellt die Post-Daten aus dem Post-Dictionary
    NSData *postData = [self
        generateFormDataFromPostDictionary:post_dict];
    [post_dict release];

    // Richtet die API-Anforderung ein
    NSString *baseurl = @"http://twitpic.com/api/uploadAndPost";
    NSURL *url = [NSURL URLWithString:baseurl];
    NSMutableURLRequest *urlRequest = [NSMutableURLRequest
        requestWithURL:url];
    if (!urlRequest)
        NOTIFY_AND_LEAVE(@"Error creating the URL Request");

    [urlRequest setHTTPMethod: @"POST"];
    [urlRequest setValue:MULTIPART forHTTPHeaderField:
```

```
        @"Content-Type"];
    [urlRequest setHTTPBody:postData];

    // Übermittelt Ergebnisse und ruft sie ab
    NSError *error;
    NSURLResponse *response;
    NSLog(@"Contacting TwitPic....");
    NSData* result = [NSURLConnection sendSynchronousRequest:
        urlRequest returningResponse:&response error:&error];
    if (!result)
    {
        [self cleanup:[NSString stringWithFormat:
            @"Submission error: %@", [error localizedDescription]]];
        return;
    }

    // Gibt die Ergebnisse zurück
    NSString *outstring = [[[NSString alloc] initWithData:result
        encoding:NSUTF8StringEncoding] autorelease];
    [self cleanup: outstring];
}
@end
```

► *Rezept 13.11: Bilder zu TwitPic hochladen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.12 Rezept: Schlüsselbunde in mehreren Anwendungen verwenden

Leider ist zur gemeinsamen Verwendung von Schlüsselbunden mehr notwendig, als einfach nur einen Gruppennamen zuzuweisen. Damit ein Schlüsselbund in mehreren Anwendungen gültig wird, müssen Sie mehrere Hürden überwinden. Wie Sie in *Rezept 13.12* sehen, gibt es in der Anwendung selbst nicht viel zu tun. Die meiste Arbeit verrichten Sie in Xcode. Die folgende Übersicht zeigt den Vorgang Schritt für Schritt:

1. Ermitteln Sie die Anwendungs-ID.
   a. Öffnen Sie in Xcode den Organizer (**Window ▸ Organizer**), und klicken Sie auf **iPhone Development ▸ Provisioning Profiles**. Wählen Sie Ihr normales Profil für die Entwicklung oder – falls Sie die Anwendung schon bereitstellen wollen – das Profil für die Bereitstellung aus. Die Anwendungs-ID erscheint als *App Identifier* in der Profilübersicht.

b. Die ID sollte in beiden Profilen identisch sein, vorausgesetzt, dass Sie der Namenskonvention `com.ihre_firma.*` gefolgt sind. Unmittelbar vor der ID sollte ein zehnstelliges Präfix stehen. Kopieren Sie die gesamte ID. In meinem Fall lautet sie `Y93A4XLA79.com.sadun.*`.

3. Richten Sie die Schlüsselbund-Zugriffsgruppe ein.

   Aktualisieren Sie in Ihrer Anwendung alle Wrapper-Initialisierungen, sodass Sie die folgende Zugriffsgruppe nutzen. Ersetzen Sie das schließende Sternchen in der ID durch `GenericKeychainSuite`, aber verwenden Sie ansonsten die ID, die Sie aus der Profilübersicht kopiert haben – also *nicht* `Y93A4XLA79`, denn diese ID ist `com.sadun` zugewiesen und nicht Ihrer Firma.

4. Erstellen Sie ein neues Entitlement.

   Wählen Sie in Xcode **File ▸ New File ▸ Code Signing ▸ Entitlements**, und klicken Sie auf **Next**. Nennen Sie das neue Entitlement `KeychainEntitlement.plist` (auch dies ist ein willkürlicher Name), und klicken Sie auf **Finish**. Xcode fügt die neue Datei dem aktiven Projekt hinzu.

5. Bearbeiten Sie das Entitlement.

   Löschen Sie den Eintrag `get-task-allow` in der neuen Eigenschaftenliste, und fügen Sie den neuen Eintrag `keychain-access-groups` hinzu. Übernehmen Sie diesen Namen ganz genau! Setzen Sie den Typ auf `Array`, und fügen Sie einen Eintrag hinzu. Der Standardname lautet `Item 1`. Setzen Sie den Stringwert dieses Eintrags auf den Namen Ihrer Zugriffsgruppe, in meinem Fall also z. B. auf `Y93A4XLA79.comsadun.GenericKeychainSuite`. Nehmen Sie auch hier wieder Ihre eigene Firmen- und Profil-ID. Speichern Sie die Datei, und schließen Sie sie.

6. Aktualisieren Sie das Ziel.

   a. Wählen Sie im Projektfenster **Groups & Files ▸ Targets ▸** *Anwendungsname*. Klicken Sie oben im Projektfenster auf die blaue Info-Schaltfläche, und öffnen Sie den Titel **Build**.

   b. Scrollen Sie im Titel **Build** nach unten zum Abschnitt **Code Signing ▸ Code Signing Entitlements**. Doppelklicken Sie, um den Entitlement-Editor zu öffnen. Geben Sie in das Textfeld `KeychainEntitlement.plist` ein, und klicken Sie auf **OK**. Der Dateiname muss genau mit dem der Eigenschaftenliste übereinstimmen, die Sie im vorherigen Schritt bearbeitet haben.

   c. Schließen Sie das Zielfenster, wenn Sie fertig sind.

Mit diesen Schritten aktualisieren Sie Ihr Projekt so, dass es ein und denselben Schlüsselbund in mehreren Anwendungen verwendet. Als Test kopieren Sie das Projekt, ändern die Anwendungs-ID in der Datei `Info.plist` und führen die neue Anwendung auf dem iPhone aus. Sie sollte denselben Zugriff auf Daten haben wie die alte. Die Benutzer können ihre Anmeldeinformationen für beliebte Dienste dann in irgendeiner Ihrer Anwendungen aktualisieren, um sie in allen Anwendungen zur Verfügung zu haben.

Hinweis

Für mehrere Anmeldeelemente innerhalb einer oder in mehreren Anwendungen können Sie ein einziges Entitlement und eine einzige Zugriffsgruppe verwenden. Sie müssen nur unterschiedliche IDs für die einzelnen Anmeldeelemente angeben.

```
self.wrapper = [[KeychainItemWrapper alloc]
   initWithIdentifier:@"SharedTwitter"
   accessGroup:@"Y93A4XLA79.com.sadun.GenericKeychainSuite"];
[self.wrapper release];
```

► *Rezept 13.12: Wrapper-Initialisierung für die gemeinsame Nutzung von Schlüsselbunden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.13 Rezept: XML-Daten in Baumstrukturen umwandeln

Die im iPhone SDK enthaltene Klasse `NSXMLParser` durchsucht XML-Code und ruft Callbacks hervor, wenn neue Elemente verarbeitet und abgeschlossen werden (mit der typischen Logik eines SAX-Parsers). Diese Klasse ist fantastisch, wenn Sie einfache Datenfeeds herunterladen und sich nur ein kleines Stück der relevanten Information abschaben wollen. Sie ist weniger fantastisch, wenn Sie professionelle Arbeit mit Fehlerprüfung, Statusinformationen und Datenflusssteuerung in beiden Richtungen zu leisten haben.

Baumstrukturen sind eine bessere Möglichkeit zur Darstellung von XML-Daten. Sie erlauben Ihnen, Suchpfade durch die Daten zu erstellen, um genau die Daten zu finden, die Sie benötigen. Dabei können Sie alle »Einträge« abrufen, nach einem Erfolgswert suchen usw. Durch solche Bäume wird XML-Text wieder in eine mehrdimensionale Struktur umgewandelt.

Um die Lücke zwischen `NSXMLParser` und baumgestützter Analyse zu schließen, können Sie eine auf `NSXMLParser` aufbauende Hilfsklasse verwenden, um Baumdaten zurückzugeben, die eher dem Standard entsprechen. Dazu ist nur ein einfacher Baumknoten wie der hier gezeigte erforderlich, der über eine doppelte Verlinkung auf seine Eltern- und Kindknoten zugreift, um ein Durchlaufen des Baums in beide Richtungen zu ermöglichen.

```
@interface TreeNode : NSObject
{
   TreeNode *parent;
   NSMutableArray *children;
```

```
    NSString *key;
    NSString *leafvalue;
}

@property (nonatomic, retain)   TreeNode          *parent;
@property (nonatomic, retain)   NSMutableArray    *children;
@property (nonatomic, retain)   NSString          *key;
@property (nonatomic, retain)   NSString          *leafvalue;
@end
```

13.13.1 Einen Parserbaum erstellen

Rezept 13.13 stellt die Klasse `XMLParser` vor. Ihre Aufgabe besteht darin, einen Parserbaum zu erstellen, während sich die Klasse `NSXMLParser` durch die XML-Quelle vorarbeitet. Die drei standardmäßigen `NSXML`-Routinen (»Element gestartet«, »Element abgeschlossen« und »Zeichen gefunden«) führen eine rekursive, absteigende Tiefensuche im Baum durch.

Die Klasse fügt neue Knoten hinzu, wenn sie neue Elemente erreicht (`parser:didStartElement:qualifiedName:attributes:`), und Blattwerte (letztes Element am Ast eines Baumes), wenn sie auf Text stößt (`parser:foundCharacters:`). Da bei XML Geschwisterelemente auf derselben Baumhöhe zulässig sind, verwendet dieser Code einen Stack, um den aktuellen Pfad zur Baumwurzel festzuhalten. In `parser:didEndElement:` führen Geschwister stets zum selben Elternelement zurück, sodass sie auf der richtigen Ebene hinzugefügt werden können.

Nach der XML-Analyse kehrt die Methode `parseXMLFile:` zum Wurzelknoten zurück.

```
@implementation XMLParser
// Der Parser kehrt zur Baumwurzel zurück. Sie müssen einen Knoten nach
// unten gehen, um die Ergebnisse zu bekommen
- (TreeNode *) parse: (NSXMLParser *) parser
{
    stack = [NSMutableArray array];

    TreeNode *root = [TreeNode treeNode];
    root.parent = nil;
    root.leafvalue = nil;
    root.children = [NSMutableArray array];

    [stack addObject:root];

    [parser setDelegate:self];
    [parser parse];
    [parser release];

    // Springt zur Wurzel zurück
    TreeNode *realroot = [[root children] lastObject];
```

```
    root.children = nil;
    root.parent = nil;
    root.leafvalue = nil;
    root.key = nil;

    realroot.parent = nil;
    return realroot;
}

// Steigt zu einem neuen Element hinab
- (void)parser:(NSXMLParser *)parser
    didStartElement:(NSString *)elementName
    namespaceURI:(NSString *)namespaceURI
    qualifiedName:(NSString *)qName
    attributes:(NSDictionary *)attributeDict
{
    if (qName) elementName = qName;

    TreeNode *leaf = [TreeNode treeNode];
    leaf.parent = [stack lastObject];
    [(NSMutableArray *)[[stack lastObject] children] addObject:leaf];

    leaf.key = [NSString stringWithString:elementName];
    leaf.leafvalue = nil;
    leaf.children = [NSMutableArray array];

    [stack addObject:leaf];
}

// Springt nach Abschluss eines Elements zurück
- (void)parser:(NSXMLParser *)parser
    didEndElement:(NSString *)elementName
    namespaceURI:(NSString *)namespaceURI
    qualifiedName:(NSString *)qName
{
    [stack removeLastObject];
}

// Erreicht ein Blatt
- (void)parser:(NSXMLParser *)parser foundCharacters:(NSString *)string
{
    if (![[stack lastObject] leafvalue])
    {
        [[stack lastObject] setLeafvalue:[NSString
            stringWithString:string]];
```

```
        return;
    }

    [[stack lastObject] setLeafvalue:
        [NSString stringWithFormat:@"%@%@",
            [[stack lastObject] leafvalue], string]];
}
@end
```

▸ *Rezept 13.13: Die Hilfsklasse* `XMLParser`

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.13.2 Die Baumergebnisse verwenden

Listing 13.1 zeigt einen Konsumenten für einen XML-Parserbaum, der die Daten aus *Rezept 13.13* verarbeitet. In diesem Beispiel sehen Sie eine Reihe von Tabellenansichtscontrollern, die sich von der Wurzel des Baums bis zu den Blättern vorarbeiten. Bei Erreichen eines Blattes wird dessen Wert in einer Benachrichtigung angezeigt. Unterbäume führen zu zusätzlichen Ansichtscontroller-Bildschirmen.

Dieses Beispiel stellt eine triviale Anwendung der Klasse `TreeNode` dar, bei der nur die Blattwerte und Kindknoten beachtet werden. Die Klasse kann jedoch weit mehr, z. B. Blätter und Objekte zurückgeben, die mit einem gegebenen Schlüssel übereinstimmen. Damit können Sie Informationen abrufen, ohne den genauen Pfad zu einem Kindknoten zu kennen, solange Sie nur wissen, wie der Knoten heißt, z. B. »entry« oder »published«. (Diese beiden Namen werden tatsächlich in der Twitter-API verwendet.) Die Suchmöglichkeiten von `TreeNode` werden ausführlicher in *Rezept* 16.3 von Kapitel 16, *Push-Benachrichtigungen*, vorgestellt. Darin werden einzelne Tweets und ihr Veröffentlichungsdatum abgerufen.

```
@implementation TreeBrowserController
@synthesize root;

// Jede Instanz dieses Controllers hat eine eigene Wurzel, da beim
// Absteigen durch den Baum neue Wurzeln erstellt werden.
- (id) initWithRoot:(TreeNode *) newRoot
{
    if (self = [super init])
    {
        self.root = newRoot;
        if (newRoot.key) self.title = newRoot.key;
```

```
    }
    return self;
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}

// Die Anzahl der Zeilen entspricht der Anzahl der Kinder eines Knotens
- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section
{
    return [self.root.children count];
}

// Versieht die Zellen, die durchlaufen werden können, mit einem
// Farbcode
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    UITableViewCell *cell = [tableView
        dequeueReusableCellWithIdentifier:@"generic"];
    if (!cell) cell = [[[UITableViewCell alloc]
        initWithFrame:CGRectZero reuseIdentifier:@"generic"]
        autorelease];
    TreeNode *child = [[self.root children]
        objectAtIndex:[indexPath row]];

    // Legt den Text fest
    if (child.hasLeafValue)
        cell.textLabel.text = [NSString stringWithFormat:@"%@:%@",
        child.key, child.leafvalue];
    else
        cell.textLabel.text = child.key;

    // Legt die Farbe fest
    if (child.isLeaf)
        cell.textLabel.textColor = [UIColor darkGrayColor];
    else
        cell.textLabel.textColor = [UIColor blackColor];

    return cell;
}
```

```
// Bei der Auswahl wird entweder ein neuer Controller oder der Blattwert
// angezeigt
- (void)tableView:(UITableView *)tableView
   didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
   TreeNode *child =
      [self.root.children objectAtIndex:[indexPath row]];
   if (child.isLeaf)
   {
      showAlert(@"%@", child.leafvalue);
      return;
   }
   TreeBrowserController *tbc = [[[TreeBrowserController alloc]
      initWithRoot:child] autorelease];
   [self.navigationController pushViewController:tbc animated:YES];
}

// Diese Controller sind flüchtig und müssen freigegeben werden
- (void) dealloc
{
   self.root = nil;
   [super dealloc];
}
@end
```

▸ *Listing 13.1: Den Parserbaum durchlaufen*

13.13.3 Bäume einreißen

Der in diesem Rezept verwendete XML-Parsercode baut einen Baum aus beidseitig verlinkten Objekten auf. Die Elternelemente besitzen ihre Kinder und umgekehrt. Um den Arbeitsspeicher korrekt freizugeben, wenn Sie diese Struktur nicht mehr brauchen, müssen Sie den Baum »einreißen« und alle diese Verknüpfungen entfernen, damit die Beibehaltungszähler der einzelnen Knoten auf null sinken.

Rufen Sie die folgende `TreeNode`-Methode an der Wurzel des verwendeten Baums auf, kurz bevor Sie diese Wurzel freigeben. Sie dürfen den Baum erst einreißen, wenn Sie dazu bereit sind, die gesamte Struktur freizugeben.

```
- (void) teardown
{
   for (TreeNode *node in [[self.children copy] autorelease])
      [node teardown];
   [self.parent.children removeObject:self];
   self.parent = nil;
}
```

13.14 Rezept: Einen einfachen Webserver erstellen

Ein Webserver bietet einen der saubersten Wege, Daten von Ihrem iPhone für einen anderen Rechner bereitzustellen. Dazu benötigen Sie keine besondere Clientsoftware, denn jeder Browser kann Dateien im Web auflisten und auf sie zugreifen. Was das Beste ist: Ein Webserver erfordert nur wenige wichtige Routinen. Sie müssen den Dienst einrichten, indem Sie eine Schleife erstellen, die auf eine Anfrage wartet (`startServer`) und diese Anfrage dann an einen Handler weitergibt (`handleWebRequest:`), der mit den abgefragten Daten antwortet. *Rezept 13.14* zeigt diese drei Hauptmethoden zum Einrichten und Steuern eines Webdienstes.

Die Schleifenroutine verwendet systemnahe Socketprogrammierung, um einen lauschenden Port einzurichten und Anfragen des Clients abzufangen. Wenn der Client einen `GET`-Befehl ausgibt, fängt der Server diese Anfrage ab und gibt sie an den Handler für Webabfragen. Dieser Handler zerlegt sie, um den Namen der gewünschten Datei herauszufinden. Die hier gezeigte Standardversion der Klasse `WebHelper` setzt voraus, dass Sie Ihre eigene Handler-Methode mithilfe einer Kategorie hinzufügen (statt in einer Unterklasse). Durch dieses Rezept wird unabhängig von der genauen `GET`-Anforderung eine einzelne, einfache Antwortseite hervorgerufen. Sie können diese Klasse erweitern, um von Ihrer Anwendung aus auf Dateien oder Dienste zuzugreifen. Eine Beispielkategorie für einen Dateidienst befindet sich im Beispielcode zu diesem Kapitel.

```
@implementation WebHelper
@synthesize cwd;
@synthesize isServing;
@synthesize delegate;
@synthesize chosenPort;

static WebHelper *sharedInstance = nil;

+ (WebHelper *) sharedInstance
{
   if(!sharedInstance) sharedInstance = [[self alloc] init];
   return sharedInstance;
}

- (NSString *) getRequest: (int) fd
{
   // Liest die Anforderung und wandelt sie in einen NSString um
   static char buffer[BUFSIZE+1];
   int len = read(fd, buffer, BUFSIZE);
   buffer[len] = '\0';
   return [NSString stringWithCString:buffer
      encoding:NSUTF8StringEncoding];
}
```

```
// Stellt Dateien aufgrund von GET-Anforderungen bereit
- (void) handleWebRequest:(int) fd
{
   // Ruft Anforderungen ab
   NSString *request = [self getRequest:fd];

   // Hier muss eine Kategorie erstellt und sinnvoll implementiert werden.
   // Dieses Beispiel ist nur ein Platzhalter.
   NSMutableString *outcontent = [NSMutableString string];
   [outcontent appendString:
      @"HTTP/1.0 200 OK\r\nContent-Type: text/html\r\n\r\n"];
   [outcontent appendString:
      @"<html><h1>iPhone Developer's Cookbook</h1><h3>Notice</h3>"];
   [outcontent appendString:
      @"<p>Please add a WebHelper category that responds "];
   [outcontent appendString:
      @"to the following request:</p>"];
   [outcontent appendFormat:@"<pre>%@</pre></html>", request];
   write (fd, [outcontent UTF8String], [outcontent length]);
   close(fd);
}

// Lauscht auf externe Anforderungen
- (void) listenForRequests
{
   NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];
   static struct sockaddr_in cli_addr;
   socklen_t length = sizeof(cli_addr);

   // Liest Daten endlos, bis die Eigenschaft isServing auf NO gesetzt
   // wird oder bis ein Socket-Annahmefehler auftritt
   while (1 > 0) {
      if (!self.isServing) return;

      if ((socketfd = accept(listenfd,
         (struct sockaddr *)&cli_addr, &length)) < 0)
      {
         self.isServing = NO;
         DO_CALLBACK(serviceWasLost, nil);
         return;
      }

      // Gibt die Verantwortung für das Lesen der Socketdaten und die
      // Reaktion darauf ab
```

```
            [self handleWebRequest:socketfd];
        }

        [pool release];
    }

    // Beginnt mit der Bereitstellung der Daten. Diese private Methode wird
    // von startService aufgerufen.
    - (void) startServer
    {
        static struct sockaddr_in serv_addr;

        // Richtet das Socket ein
        if((listenfd = socket(AF_INET, SOCK_STREAM,0)) < 0)
        {
            self.isServing = NO;
            DO_CALLBACK(serviceCouldNotBeEstablished, nil);
            return;
        }

        // Stellt Daten über einen zufälligen Port bereit
        serv_addr.sin_family = AF_INET;
        serv_addr.sin_addr.s_addr = htonl(INADDR_ANY);
        serv_addr.sin_port = 0;

        // Richtet die Bindung ein
        if(bind(listenfd, (struct sockaddr *)&serv_addr,
            sizeof(serv_addr)) <0)
        {
            self.isServing = NO;
            DO_CALLBACK(serviceCouldNotBeEstablished, nil);
            return;
        }

        // Findet die gewählte Portnummer heraus
        int namelen = sizeof(serv_addr);
        if (getsockname(listenfd, (struct sockaddr *)&serv_addr,
            (void *) &namelen) < 0)
        {
            close(listenfd);
            self.isServing = NO;
            DO_CALLBACK(serviceCouldNotBeEstablished, nil);
            return;
        }
```

```
    chosenPort = ntohs(serv_addr.sin_port);

    // Lauscht
    if(listen(listenfd, 64) < 0)
    {
        self.isServing = NO;
        DO_CALLBACK(serviceCouldNotBeEstablished, nil);
        return;
    }

    DO_CALLBACK(serviceWasEstablished, nil);
    [NSThread detachNewThreadSelector:
        @selector(listenForRequests) toTarget:self withObject:NULL];
}

- (void) startService
{
    if (self.isServing) return; // Es wird bereits gelauscht
    if (![UIDevice  networkAvailable])
    {
        showAlert(@"You are not connected to the network. Please \
            do so before running this application.");
        return;
    }
    [self startServer];
    self.isServing = YES;
}
@end
```

► *Rezept 13.14: iPhone-Dateien über einen Webdienst bereitstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 13 und öffnen das Projekt zu diesem Rezept.

13.15 Ein letzter Punkt: FTPHelper

FTP (File Transfer Protocol) ist vor allem deshalb so bequem zu nutzen, weil dieser Standard auf sehr vielen Plattformen gültig ist. Der Core Foundation-Beispielcode von Apple zu FTP ist einfach zu verwenden, wenn Sie gewillt sind, über den uneleganten Programmierstil hinwegzusehen. Den Quellcode finden Sie unter *http://developer.apple.com/mac/library/samplecode/CFFTPSample*. Um die Upload- und Download-Funktionen zum Laufen zu bekommen, müssen Sie nicht viel tun. Sobald Sie den Benutzernamen, das Passwort und Angaben zum Host gespeichert haben, können Sie die Datenübertragung per FTP auf einfache Weise automatisieren.

Auf die Bitte mehrerer Leser hin füge ich dem Beispielcode zu diesem Kapitel die Hilfsklasse `FTPHelper` bei. Sie bildet zwar bei Weitem keine elegante Lösung, doch ihr Funktionsumfang reicht als Ausgangspunkt für alle Entwickler aus, die in ihren Anwendungen einen FTP-Zugriff bereitstellen wollen. Code, der auf dieser Hilfsklasse beruht, sollten Sie lieber nicht im App Store bereitzustellen versuchen, bevor Sie ihn ausführlich getestet und optimiert haben. Sie können daran aber ablesen, wie der FTP-Zugriff funktioniert.

Die Klasse `FTPHelper` ermöglicht das Hoch- und Herunterladen von Dateien und das Auflisten von Verzeichnissen. Die Delegierungen für diese drei Operationen richten Sie folgendermaßen ein:

```
[FTPHelper sharedInstance].delegate = self;
[FTPHelper sharedInstance].uname = BASE_USERNAME;
[FTPHelper sharedInstance].pword = BASE_PASSWORD;
[FTPHelper sharedInstance].urlString = BASE_URL;

// Auflistung
[FTPHelper list:BASE_URL];

// Download
[FTPHelper download:FILE_TO_MOVE];

// Upload
[FTPHelper upload:FILE_TO_MOVE];
```

Verwenden Sie für den Benutzernamen und das Passwort einfache Stringkonstanten. Der grundlegende URL-Pfad zeigt nicht auf eine bestimmte Ressource. Stattdessen geben Sie darin eine allgemeine FTP-Adresse wie `@"ftp://MySystem.local"`, `@"ftp://somehost.com"` oder sogar `@"ftp://somehost.com/ftp/UploadArea"` an. Denken Sie an das Präfix `ftp://`!

Beim Hoch- und Herunterladen geben Sie den Dateinamen als eigenen Parameter an. In der derzeitigen Implementierung werden alle Dateien im Hauptdokumentordner der Sandbox abgelegt bzw. von dort entnommen.

Listing 13.2 zeigt die Schnittstelle der Klasse `FTPHelper` und das Protokoll für die Delegierung. Der Funktionsumfang wird über einfache Klassenmethoden bereitstellt.

```
@protocol FTPHelperDelegate <NSObject>
@optional
// Erfolg
- (void) receivedListing: (NSDictionary *) listing;
- (void) downloadFinished;
- (void) dataUploadFinished: (NSNumber *) bytes;
- (void) progressAtPercent: (NSNumber *) aPercent;

// Fehlschlag
- (void) listingFailed;
- (void) dataDownloadFailed: (NSString *) reason;
- (void) dataUploadFailed: (NSString *) reason;
- (void) credentialsMissing;
@end

@interface FTPHelper : NSObject
{
   NSString *urlString;
   id <FTPHelperDelegate> delegate;
   BOOL isBusy;
   NSString *uname;
   NSString *pword;
   NSMutableArray *fileListings;
   NSString *filePath;
}

@property (retain) NSString *urlString;
@property (retain) id delegate;
@property (assign) BOOL isBusy;
@property (retain) NSString *uname;
@property (retain) NSString *pword;
@property (retain) NSMutableArray *fileListings;
@property (retain) NSString *filePath; // valid after download

+ (FTPHelper *) sharedInstance;
+ (void) download:(NSString *) anItem;
+ (void) upload: (NSString *) anItem;
+ (void) list: (NSString *) aURLString;

+ (NSString *) textForDirectoryListing: (CFDictionaryRef) dictionary;
@end
```

13.16 Zusammenfassung

Dieses Kapitel behandelte eine große Vielzahl von Technologien für die Netzwerkunterstützung. Sie haben gelernt, wie Sie die Netzanbindung prüfen, mit Schlüsselbunden arbeiten, um eine sichere Authentifizierung zu ermöglichen, wie Sie Daten über `NSURLConnection` und FTP hoch- und herunterladen usw.

Folgende Gedanken sollten Sie aus diesem Kapitel mitnehmen:

- Die Netzwerkunterstützung von Apple wird größtenteils durch sehr systemnahe C-Routinen bereitgestellt. Wenn Sie einen komfortablen Objective-C-Wrapper finden, um die Programmierung zu vereinfachen, sollten Sie ihn benutzen. Dies bietet nur dann einen Nachteil, wenn Sie auf der grundlegendsten Ebene Ihrer Anwendung eine sehr genaue Netzwerksteuerung benötigen.
- In diesem Kapitel war kein Platz, um Authentifizierungsverfahren für Daten-APIs ausführlich zu besprechen. Wenn Sie beispielsweise Zugriff auf `OAuth` benötigen, suchen Sie nach bestehenden Cocoa-Implementierungen. In Open-Source-Archiven stehen einige davon zur Verfügung, die sich auch leicht auf Cocoa Touch portieren lassen. Brauchen Sie einfache Routinen für Prüfsummen, Digest und Kodierung, sehen Sie sich in Ihrem Browser *http://www.cocoadev.com/index.pl?NSDataCategory* an. Die äußerst praktische Kategorie `NSData` bietet Lösungen für MD5, SHA1, Base32 usw.
- Viele Datendienste bieten einfach zu verwendende APIs an, z. B. Twitter und TwitPic. Diese APIs sind häufig stärker eingeschränkt als die vollständig autorisierten Entwickler-APIs, für die gewöhnlich Entwickler-Anmeldeinformationen und erweiterte Rechte erforderlich sind. Allerdings bieten sie einfache Lösungen für Aufgaben, die Sie häufig durchführen müssen, vor allem wenn Sie keinen vollständigen Client für einen bestimmten Dienst schreiben wollen.
- Die gemeinsame Nutzung von Schlüsselbunden in mehreren Anwendungen ist an das Profil gekoppelt, mit dem diese Anwendungen signiert sind. Sie können die Anmeldeinformationen von Benutzern in Ihren eigenen Anwendungen gemeinsam nutzen, aber nicht mit anderen Entwicklern teilen. Führen Sie alle Schritte sorgfältig durch, wenn Sie neue Dateien für Schlüsselbund-Entitlements erstellen, damit Sie die Anwendung später auch erfolgreich kompilieren können.
- Zwar verfügt Apple über Objective-C-Wrapper wie zum Beispiel `NSXMLParser`, doch handelt es sich dabei nicht unbedingt um die Klasse, die Sie gesucht oder sich erhofft haben. Die Anpassung von Klassen macht einen großen Teil der iPhone-Programmierung aus. In diesem Kapitel wurden viele maßgeschneiderte Klassen vorgestellt, die den Zugriff auf wichtige Cocoa Touch-Objekte vereinfachen.

14 Gerätefähigkeiten

Jedes iPhone weist eine Reihe von Eigenschaften auf – momentane und dauerhafte, nur für das betreffende Gerät selbst geltende und solche, die auch auf andere zutreffen. Dazu gehören die aktuelle physische Orientierung, der Modellname, der Ladezustand des Akkus und der Zugriff auf die eingebaute Hardware. In diesem Kapitel sehen wir uns das Gerät von der Hardwarekonfiguration bis zu den aktiven Sensoren an. Sie finden hier Rezepte, die eine breite Palette von Informationen über das Gerät zurückgeben, und erfahren, wie Sie die Hardwarevoraussetzungen zur Laufzeit prüfen und in der Datei `Info.plist` der Anwendung festlegen. Außerdem lernen Sie, wie Sie Meldungen von den Sensoren anfordern und wie Sie Benachrichtigungen abonnieren, um bei der Änderung des Sensorstatus Callbacks zu erstellen. In diesem Kapitel geht es um die Hardware, das Dateisystem und die Sensoren auf dem iPhone und darum, wie Sie diese Merkmale in Ihren Programmen nutzen können.

14.1 Rezept: Elementare Geräteinformationen abrufen

Die Klasse `UIDevice` ermöglicht es Ihnen, wichtige gerätespezifische Informationen abzurufen, z. B. das Modell des jeweiligen iPhone- oder iPod touch-Modells, den Gerätenamen sowie den Namen und die Version des Betriebssystems. Wie *Rezept 14.1* zeigt, können Sie hiermit die Angaben zum System auf einen Schlag zum Vorschein bringen. Alle Methoden sind Instanzmethoden, die mit dem `UIDevice`-Singleton über `[UIDevice currentDevice]` aufgerufen werden.

Von `UIDevice` können Sie folgende Informationen abrufen:

- `System Name` Hiermit wird der Name des zurzeit verwendeten Betriebssystems zurückgegeben. Bei den jetzigen Generationen von iPhones gibt es nur ein mögliches Betriebssystem, nämlich iPhone OS.
- `System Version` Dieser Wert gibt die Version der Firmware an, die zurzeit auf dem Gerät installiert ist, z. B. 2.2.1, 3.0, 3.1, 3.2 usw.

- `Unique ID` Die eindeutige ID eines iPhones ist eine hexadezimale Zahl, die für jedes iPhone und jeden iPod touch garantiert eindeutig ist. Laut Apple wird diese ID dadurch erstellt, dass an bestimmte Hardwarebezeichner, darunter die Gerätenummer, ein interner Hashwert angehängt wird. Diese eindeutige ID wird verwendet, um Geräte im iPhone-Portal für die Profilerstellung und Ad-hoc-Bereitstellung zu registrieren.
- `Model` Das Modell des iPhones wird in Form eines Strings angegeben, der die Plattform beschreibt, also iPhone, iPod touch oder iPad. Sollte iPhone OS in Zukunft auch auf anderen Arten von Geräten zu Einsatz kommen, werden Strings zur Beschreibung dieser Modelle hinzugefügt.
- `Name` Dieser String steht für den Namen, den der Benutzer dem iPhone in iTunes gegeben hat, z. B. »Ericas iPhone« oder »Binky«. Dieser Name wird auch als lokaler Hostname des Geräts verwendet. Einzelheiten über den Abruf des lokalen Hostnamens finden Sie in Kapitel 13, *Netzwerke*.

```
- (void) action: (UIBarButtonItem *) bbi
{

   self.log = [NSMutableString string];
   [self doLog:@"System Name: %@",
      [[UIDevice currentDevice] systemName]];
   [self doLog:@"System Version: %@",
      [[UIDevice currentDevice] systemVersion]];
   [self doLog:@"Unique ID: %@",
      [[UIDevice currentDevice] uniqueIdentifier]];
   [self doLog:@"Model %@", [[UIDevice currentDevice] model]];
   [self doLog:@"Name %@", [[UIDevice currentDevice] name]];
}
```

▸ *Rezept 14.1: Die Klasse* `UIDevice` *verwenden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 14 und öffnen das Projekt zu diesem Rezept.

14.2 Gerätevoraussetzungen angeben

Wenn Sie in iTunes eine Anwendung für Version 3.0 einreichen, müssen Sie nicht mehr angeben, mit welchen Plattformen sie kompatibel ist, sondern welche Gerätefunktionen sie braucht.

Jedes iPhone und jeder iPod touch weist seinen eigenen Satz von Funktionen auf. Manche Geräte haben eine Kamera und können GPS-Informationen verarbeiten, andere nicht. Einige unterstützen OpenGL ES 2.0, andere sind auf OpenGL ES 1.1 beschränkt. Seit der Firmware-Version 3.0 können Sie angeben, welche Funktionen Ihre Anwendung braucht.

14.2 Gerätevoraussetzungen angeben

Wenn Sie in die Datei `Info.plist` den Schlüssel `UIRequiredDeviceCapabilities` aufnehmen, beschränkt iTunes die Installation der Anwendung auf Geräte, die die verlangten Eigenschaften aufweisen. Die erforderlichen Fähigkeiten geben Sie als Array aus Strings an. Eine Übersicht über die möglichen Werte finden Sie in *Tabelle 14.1*. Führen Sie nur die Funktionen auf, die die Anwendung tatsächlich benötigt. Wenn Ihre Anwendung selbst eine Notlösung für ein fehlendes Merkmal bietet, fügen Sie die betreffende Einschränkung nicht hinzu.

Schlüssel	Verwendung
`telephony`	Die Anwendung erfordert die Anwendung *Telefon* oder verwendet `tel://`-URLs.
`sms`	Die Anwendung erfordert die Anwendung *Nachrichten* oder verwendet `sms://`-URLs.
`still-camera`	Die Anwendung greift für den Controller der Bildauswahl auf den Kameramodus zurück.
`auto-focus-camera`	Die Anwendung braucht eine erhöhte Schärfe für Makrofotografie oder für die Datenerkennung innerhalb von Bildern.
`video-camera`	Die Anwendung greift für den Controller der Bildauswahl auf den Videomodus zurück.
`wifi`	Die Anwendung erfordert Zugriff auf ein lokales 802.11-Netzwerk.
`accelerometer`	Die Anwendung braucht Meldungen der Beschleunigungssensoren, die über einfache `UIViewController`-Orientierungsereignisse hinausgehen.
`location-services`	Die Anwendung verwendet Core Location.
`gps`	Die Anwendung verwendet Core Location und benötigt eine erhöhte Genauigkeit bei der GPS-Ortung.
`magnetometer`	Die Anwendung verwendet Core Location und benötigt Richtungsereignisse, also Messwerte zur Bewegungsrichtung. (Beim Magnetometer handelt es sich um den eingebauten Kompass.)
`microphone`	Die Anwendung greift entweder auf das eingebaute Mikrofon oder auf (zugelassene) Zubehörgeräte mit Mikrofon zurück.
`opengles-1`	Die Anwendung nutzt OpenGL ES 1.1.
`opengles-2`	Die Anwendung nutzt OpenGL ES 2.0.
`armv6`	Die Anwendung ist *ausschließlich* für den Befehlssatz `armv6` kompiliert (Version 3.1 und höher).

`armv7`	Die Anwendung ist *ausschließlich* für den Befehlssatz `armv7` kompiliert (Version 3.1 und höher).
`peer-peer`	Die Anwendung verwendet die Peer-to-Peer-Anbindung von GameKit über Bluetooth (Version 3.1 und höher).

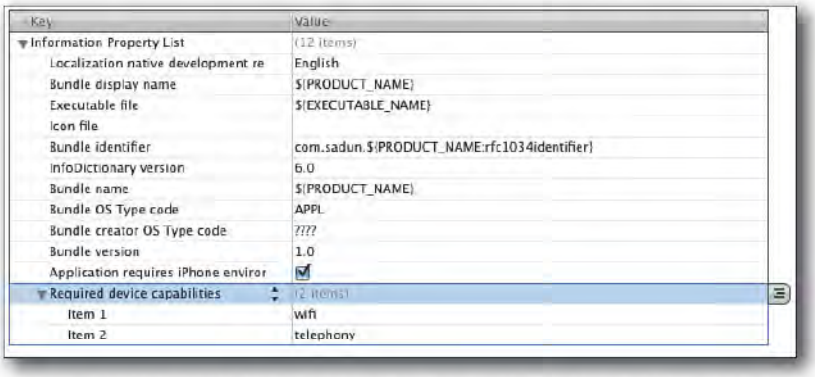
▸ *Tabelle 14.1: Erforderliche Gerätemerkmale*

Stellen Sie sich z. B. eine Anwendung vor, die optional die Möglichkeit bietet, Bilder aufzunehmen, wenn sie auf einem Gerät mit eingebauter Kamera ausgeführt wird. Wenn dieses Programm bis auf diese Option auch auf iPod touch-Geräten läuft, dürfen Sie die Einschränkung `still-camera` nicht angeben. Prüfen Sie stattdessen innerhalb der Anwendung, ob eine Kamera vorhanden ist, und bieten Sie die Kameraoption nur dann an, wenn das Gerät entsprechend ausgestattet ist. Durch die Angabe der Einschränkung `still-camera` würden Sie alle Besitzer von iPods der ersten, zweiten und dritten Generation sowie von den aktuellen iPads der ersten Generation aus Ihrem Kreis möglicher Kunden ausschließen.

14.2.1 Geräteanforderungen hinzufügen

Um zu `Info.plist` Geräteanforderungen hinzuzufügen, öffnen Sie die Datei im Xcode-Editor. Markieren Sie die letzte Zeile (gewöhnlich `Applicaton Requires iPhone Environment`), und drücken Sie [Return]. Daraufhin erscheint ein neuer Eintrag, der zur Bearbeitung bereit ist. Wenn Sie **Req** eingeben, macht Xcode daraus automatisch `Required device capabilities`. Dies ist die »für Menschen lesbare« Variante des Schlüssels `UIRequiredDeviceCapabilities`. Den normalen Schlüsselnamen können Sie sehen, wenn Sie auf einen Eintrag darunter rechtsklicken (bzw. bei gedrückter [Ctrl]-Taste klicken) und **Show Raw Key/Values** auswählen.

Xcode legt als Eintragstyp automatisch ein Array fest und fügt einen neuen Eintrag namens `Item 1` hinzu. Bearbeiten Sie diesen Eintrag, um das erste erforderliche Gerätemerkmal anzugeben. Um weiter Einträge hinzuzufügen, markieren Sie einen beliebigen Eintrag und drücken [Return], woraufhin Xcode ein weiteres Schlüssel-Wert-Paar einfügt. *Abbildung* 14.1 zeigt den Editor während des Vorgangs.



▸ *Abbildung 14.1: In Xcode erforderliche Gerätemerkmale zur Datei* `Info.plist` *hinzufügen*

14.3 Rezept: Zusätzliche Geräteinformationen abrufen

Sowohl mit `sysctl()` als auch mit `sysctlbyname()` können Sie Systeminformationen abrufen. Diese beiden standardmäßigen UNIX-Funktionen fragen Angaben zur Hardware und zum Betriebssystem selbst beim Betriebssystem ab. Um einen Eindruck davon zu bekommen, was für eine Vielfalt an Informationen darüber zur Verfügung steht, sehen Sie sich die Datei `/usr/include/sys/sysctl.h` auf dem Macintosh an. Darin finden Sie eine erschöpfende Menge von Konstanten, die Sie als Parameter für diese Funktionen verwenden können.

Mit diesen Konstanten können Sie wichtige Informationen wie die CPU-Frequenz des Systems, den Betrag an verfügbarem Arbeitsspeicher usw. abfragen, wie *Rezept 14.2* zeigt. Dort wird eine Kategorie für `UIDevice` vorgestellt, die Systeminformationen erfasst und über eine Folge von Methodenabrufen zurückgibt.

Vielleicht fragen Sie sich, warum in dieser Kategorie die Methode `platform` enthalten ist, wenn doch die Klasse `UIDevice` bereits das Gerätemodell zurückgibt. Die Antwort lautet, dass wir hier eine Möglichkeit brauchen, um die verschiedenen Arten von iPhone-, iPod touch- und iPad-Geräten zu unterscheiden.

Als Modell eines iPhone 3GS wird in `UIDevice` schlicht »iPhone« zurückgegeben, was aber auch für ein iPhone 3G und ein einfaches iPhone stehen kann. Dagegen wird in diesem Rezept für ein iPhone 3GS der Plattformwert `iPhone2,1` zurückgeben. Damit können Sie das Gerät im Programm von einem iPhone der ersten Generation (`iPhone1,1`) und einem iPhone 3G (`iPhone1,2`) unterscheiden.

Jedes Modell weist von Haus aus seine eigenen Merkmale auf. Wenn Sie genau wissen, mit welcher Art von iPhone Sie es zu tun haben, können Sie bestimmen, ob auf diesem Gerät Merkmale wie Barrierefreiheit, GPS und Magnometer vorhanden sind.

```
@implementation UIDevice (Hardware)
+ (NSString *) getSysInfoByName:(char *)typeSpecifier
{
    // Ruft sysctl-Informationen nach den Namen ab
    size_t size;
    sysctlbyname(typeSpecifier, NULL, &size, NULL, 0);
    char *answer = malloc(size);
    sysctlbyname(typeSpecifier, answer, &size, NULL, 0);
    NSString *results = [NSString stringWithCString:answer
        encoding: NSUTF8StringEncoding];
    free(answer);
    return results;
}

+ (NSString *) platform
{
    return [self getSysInfoByName:"hw.machine"];
}
```

```
+ (NSUInteger) getSysInfo: (uint) typeSpecifier
{
    size_t size = sizeof(int);
    int results;
    int mib[2] = {CTL_HW, typeSpecifier};
    sysctl(mib, 2, &results, &size, NULL, 0);
    return (NSUInteger) results;
}

+ (NSUInteger) cpuFrequency
{
    return [self getSysInfo:HW_CPU_FREQ];
}

+ (NSUInteger) busFrequency
{
    return [self getSysInfo:HW_BUS_FREQ];
}

+ (NSUInteger) totalMemory
{
    return [self getSysInfo:HW_PHYSMEM];
}

+ (NSUInteger) userMemory
{
    return [self getSysInfo:HW_USERMEM];
}

+ (NSUInteger) maxSocketBufferSize
{
    return [self getSysInfo:KIPC_MAXSOCKBUF];
}
@end
```

▶ *Rezept 14.2: Geräteinformationen über* `sysctl()` *und* `sysctlbyname()` *abrufen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 14 und öffnen das Projekt zu diesem Rezept.

14.4 Rezept: Den Ladezustand des Akkus überwachen

Mit der API der Version 3.0 und höher können Sie den Ladezustand des Akkus im iPhone überprüfen und feststellen, in welcher Phase des Ladevorgangs sich das Gerät jeweils befindet. Der Ladezustand wird dabei als Fließkommawert zwischen 1,0 (vollständig geladen) und 0,0 (vollständig entladen) dargestellt. Damit können Sie ungefähr den Ladezustand bestimmen, bevor Sie Operationen ausführen, die das Gerät ungewöhnlich stark belasten.

Beispielsweise können Sie Ihre Benutzer warnen, wenn Sie eine umfangreiche Folge mathematischer Rechenoperationen durchführen, und vorschlagen, das Gerät an eine Stromquelle anzuschließen. Den Akkuladezustand rufen Sie über einen Aufruf von `UIDevice` ab. Der zurückgegebene Wert wird in Schritten von 5 % angegeben.

```
NSLog(@"Battery level: %0.2f%",
   [[UIDevice currentDevice] batteryLevel] * 100);
```

Für die Phase des Ladevorgangs gibt es vier mögliche Werte. Das Gerät wird gerade geladen (»charging«, d. h., es ist an eine Stromquelle angeschlossen), es kann voll (»full«) oder nicht an eine externe Stromquelle angeschlossen (»unplugged«) sein. Für alle anderen Zustände gibt es den Wert »unbekannt« (»unknown«). Die Ladephase rufen Sie mit der Eigenschaft `batteryState` von `UIDevice` ab:

```
NSArray *stateArray = [NSArray arrayWithObjects:
   @"Battery state is unknown",
   @"Battery is not plugged into a charging source",
   @"Battery is charging",
   @"Battery state is full", nil];

NSLog(@"Battery state: %@",
   [stateArray objectAtIndex:
   [[UIDevice currentDevice] batteryState]]);
```

Denken Sie daran, dass es sich hierbei nicht um dauerhafte Zustände handelt, sondern um Momentaufnahmen dessen, was gerade mit dem Gerät geschieht. Diese Angaben sind keine Flags, und sie können auch nicht mit einer ODER-Verknüpfung verarbeitet werden, um eine allgemeine Beschreibung des Akkuzustands zu bilden. Stattdessen zeigen diese Werte nur die jüngste Statusänderung an.

Rezept 14.3 überwacht solche Statusänderungen. Nur wenn der Code entdeckt, dass sich der Akkuzustand geändert hat, schaut er nach, was diese Statusänderung bedeutet. Dadurch können Sie momentane Ereignisse erfassen, z. B. die Tatsache, dass der Akku wieder vollständig geladen ist, dass der Benutzer das Gerät zum Laden an eine Stromquelle angeschlossen hat oder dass er die Verbindung zu einer Stromquelle getrennt hat.

Um die Überwachung zu starten, setzen Sie die Eigenschaft `batteryMonitoringEnabled` auf `YES`. Während der Überwachung gibt die Klasse `UIDevice` Benachrichtigungen aus, wenn sich der Ladezustand oder die Ladephase des Akkus ändern. Der Code in *Rezept 14.3* abonniert beide Benachrichti-

gungen. Sie können diese Werte aber auch direkt abrufen, ohne auf eine Benachrichtigung zu warten. Apple gibt keine Garantie dafür, wie häufig Aktualisierungen von Statusänderungen erfolgen, doch beim Testen dieses Rezepts werden Sie bemerken, dass dies ziemlich regelmäßig geschieht.

```
- (void) checkBattery: (id) sender
{
   NSArray *stateArray = [NSArray arrayWithObjects:
      @"Battery state is Unknown",
      @"Battery is unplugged",
      @"Battery is charging",
      @"Battery state is full", nil];
   self.log = [NSMutableString string];
   [self doLog:@"Battery level: %0.2f%",
      [[UIDevice currentDevice] batteryLevel] * 100];
   [self doLog:@"Battery state: %@", [stateArray
      objectAtIndex:[[UIDevice currentDevice] batteryState]]];
}

- (void) viewDidLoad
{
   self.navigationController.navigationBar.tintColor =
      COOKBOOK_PURPLE_COLOR;

   // Aktiviert die Akku-Überwachung
   [[UIDevice currentDevice] setBatteryMonitoringEnabled:YES];

   // Registriert sich als Beobachter für Änderungen des Ladezustands
   // und der Ladephase
   [[NSNotificationCenter defaultCenter] addObserver:self
      selector:@selector(checkBattery)
      name:UIDeviceBatteryStateDidChangeNotification
      object:nil];
   [[NSNotificationCenter defaultCenter] addObserver:self
      selector:@selector(checkBattery)
      name:UIDeviceBatteryLevelDidChangeNotification
      object:nil];
}
```

► *Rezept 14.3: Den iPhone-Akku überwachen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 14 und öffnen das Projekt zu diesem Rezept.

14.5 Rezept: Den Annäherungssensor aktivieren und deaktivieren

Wenn Sie nicht gerade zwingende Gründe dafür haben, ein iPhone gegen Körperteile zu drücken (oder umgekehrt), bewirkt die Aktivierung des Annäherungssensors nur wenig. Aktiviert macht er nur eines: Er spürt auf, ob es direkt vor ihm ein großes Objekt gibt. In diesem Fall schaltet er den Bildschirm ab. Schieben Sie das im Weg stehende Objekt zur Seite, schaltet sich der Bildschirm wieder an. Damit wird verhindert, dass Sie versehentlich mit Ihrem Ohr Schaltflächen betätigen oder eine andere Nummer wählen, wenn Sie einen Anruf durchführen. Manche schlecht entworfenen Schutzhüllen verhindern, dass der Annäherungssensor des iPhones korrekt funktioniert.

Die Anwendung *Google Mobile* aus dem App Store nutzt diese Funktion, um eine Sitzung zur Sprachaufzeichnung zu starten. Wenn Sie das Telefon an Ihren Kopf halten, wird Ihre Anfrage aufgezeichnet, und wenn Sie es wieder wegnehmen, wird diese Anfrage gesendet, um am Ziel interpretiert zu werden. Den Entwicklern war es dabei gleichgültig, dass der Bildschirm während dieses Vorgangs abgeschaltet wird, da die Sprachaufzeichnung auch ohne sichtbare Benutzeroberfläche funktioniert.

Rezept 14.4 veranschaulicht die Arbeit mit dem Annäherungssensor auf dem iPhone. Es verwendet die Klasse `UIDevice`, um den Annäherungssensor ein- und auszuschalten, und abonniert `UIDeviceProximityStateDidChangeNotification`, um Zustandsänderungen zu erfassen, wobei die beiden möglichen Zustände »An« und »Aus« sind. Wenn die `UIDevice`-Eigenschaft `proximityState` den Wert `YES` zurückgibt, ist der Annäherungssensor aktiviert.

Hinweis

Vor der Firmware-Version 3.0 wurde der Annäherungssensor von der Klasse `UIApplication` gesteuert, doch dieser Ansatz ist jetzt veraltet und sollte nicht mehr verwendet werden. Beachten Sie auch, dass die Methode `setProximityState:` zwar dokumentiert ist, aber in Wirklichkeit gar nicht existiert. Die Eigenschaft `proximityState` ist schreibgeschützt. Beachten Sie, das es den Annäherungssensor derzeit nur auf den iPhone-Modellen gibt.

```
- (void) toggle: (id) sender
{
   // Bestimmt den jetzigen Zustand des Annäherungssensors und
   // schaltet ihn um
   BOOL isIt = [UIDevice currentDevice].proximityMonitoringEnabled;
   NSString *title = isIt ? @"Enable" : @"Disable";
   self.navigationItem.rightBarButtonItem = BARBUTTON(title,
      @selector(toggle:));
   [UIDevice currentDevice].proximityMonitoringEnabled = !isIt;

   self.log = [NSMutableString string];
```

```
    [self doLog:@"You have %@ the Proximity sensor.", isIt ? @"disabled"
        : @"enabled"];
    if (!isIt) [self doLog:
        @"View state changes - the screen is not readable when
            blanked."];}

- (void) stateChange: (NSNotificationCenter *) notification
{
    // Protokolliert die Benachrichtigungen
    NSLog(@"The proximity sensor %@",
        [UIDevice currentDevice].proximityState ?
        @"will now blank the screen" :
        @"will now restore the screen");
}

- (void) viewDidLoad
{
    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
    self.navigationItem.rightBarButtonItem =
        BARBUTTON(@"Enable", @selector(toggle));

    // Fügt einen Beobachter für den Status des Annäherungssensors hinzu
    [[NSNotificationCenter defaultCenter]
        addObserver:self selector:@selector(stateChange:)
        name:@"UIDeviceProximityStateDidChangeNotification"
        object:nil];
}
```

► *Rezept 14.4: Den Annäherungssensor aktivieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 14 und öffnen das Projekt zu diesem Rezept.

14.6 Rezept: Wo ist »oben«?

Das iPhone hat drei eingebaute Sensoren, die die Beschleunigung entlang der drei Achsen des Geräts messen, also entlang der x-Achse (links/rechts), der y-Achse (oben/unten) und der z-Achse (vorne/hinten). Diese Werte zeigen die Kräfte an, die durch die Erdanziehung und die Bewegungen des Benutzers auf das iPhone einwirken. Ganz außerordentliche Messwerte zu den Kräften können Sie

bekommen, wenn Sie das iPhone im Kreis schwingen (Zentripetalkraft) oder von einem hohen Gebäude fallen lassen (freier Fall). Leider werden Sie kaum noch in der Lage sein, die Daten abzurufen, wenn Sie aus dem iPhone ein kleines Häuflein Schrott gemacht haben.

Damit ein Objekt Daten vom Beschleunigungsmesser abonnieren kann, müssen Sie es als Delegierung festlegen, wobei das Objekt das Protokoll `UIAccelerometerDelegate` implementieren muss.

```
[[UIAccelerometer sharedAccelerometer] setDelegate:self]
```

Ist diese Delegierung zugewiesen, empfängt sie `accelerometer:didAccelerate:`-Nachrichten, die Sie nach verfolgen und beantworten können. Normalerweise weisen Sie die Delegierung dem Hauptansichtscontroller zu, aber Sie können auch eine eigene Hilfsklasse verwenden.

Das an die Delegierungsmethode gesendete `UIAcceleration`-Objekt gibt Fließkommawerte für die x-, y- und z-Achse zurück, deren Wertebereich jeweils von -1,0 bis 1,0 reicht.

```
float x = [acceleration x];
float y = [acceleration y];
float z = [acceleration z];
```

In *Rezept 14.5* wird anhand dieser Werte bestimmt, wo oben ist. Der Code berechnet den Arkustangens zwischen den Beschleunigungsvektoren x und y und gibt den Winkel von der Richtung nach oben an. Wenn neue Nachrichten über die Beschleunigung eingehen, dreht der Rezeptcode eine `UIImageView` mit der Darstellung eines nach oben weisenden Pfeils, wie Sie in *Abbildung* 14.2 sehen. Die Echtzeitreaktion auf die Benutzeraktion sorgt dafür, dass der Pfeil stets nach oben zeigt, unabhängig davon, wie der Benutzer das Gerät dreht.

▸ *Abbildung 14.2: Mithilfe von etwas Mathematik lässt sich ermitteln, wo »oben« ist, indem eine Arkustangens-Funktion unter Verwendung der Kraftvektoren x und y ausgeführt wird. In diesem Beispiel deutet der Pfeil immer nach oben, unabhängig davon, wie der Benutzer das iPhone hält.*

```
- (void)accelerometer:(UIAccelerometer *)accelerometer
    didAccelerate:(UIAcceleration *)acceleration
{
    // Bestimmt aus den Beschleunigungskomponenten in x- und y-Richtung, wo
    // oben ist
    float xx = -[acceleration x];
    float yy = [acceleration y];
    float angle = atan2(yy, xx);
    [arrow setTransform:
        CGAffineTransformMakeRotation(angle)];
}

- (void) viewDidLoad
{
    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;

    // Initialisiert die Delegierung, um mit der Erfassung von
    // Beschleunigungsereignissen zu beginnen
    [[UIAccelerometer sharedAccelerometer] setDelegate:self];
}
```

▸ *Rezept 14.5: Beschleunigungsereignisse erfassen*

14.7 Rezept: Objekte anhand von Beschleunigungsmesswerten auf dem Bildschirm verschieben

Mit geschickter Programmierung kann der in das iPhone eingebaute Beschleunigungsmesser Objekte auf dem Bildschirm »bewegen« und dabei in Echtzeit darauf reagieren, wie der Benutzer das Gerät neigt. *Rezept 14.6* erstellt einen animierten Schmetterling, den Benutzer über den Bildschirm verschieben können.

Das Geheimnis dabei besteht darin, dem Programm etwas hinzuzufügen, das ich einen »Physik-Timer« nenne. Anstatt wie in *Rezept 14.15* direkt auf Änderungen in der Beschleunigung zu reagieren, macht der Callback des Beschleunigungsmessers nichts weiter, als die aktuellen Kräfte zu messen. Es liegt ganz bei der Timer-Routine, diese Kräfte auf den Schmetterling zu übertragen, indem sie dessen Rahmen ändert.

- Solange die Richtung der Krafteinwirkung gleich bleibt, wird der Schmetterling beschleunigt. Seine Geschwindigkeit nimmt zu, und zwar entsprechend dem Ausmaß der Beschleunigungskraft in Richtung der x- oder y-Achse.
- Die Routine `tick`, die der Timer aufruft, bewegt den Schmetterling, indem er den Geschwindigkeitsvektor zum Ursprung des Schmetterlingsbilds addiert.

> Der Freiraum des Schmetterlings ist begrenzt. Daher hört er auf, in eine Richtung zu wandern, wenn er an eine Kante stößt. Dadurch bleibt der Schmetterling stets auf dem Bildschirm sichtbar. Die etwas sonderbare, verschachtelte `if`-Struktur in der Methode `tick` überprüft die Randbedingungen. Wenn der Schmetterling gegen eine Begrenzung in vertikaler Richtung stößt, kann er sich immer noch horizontal bewegen.

```
- (void)accelerometer:(UIAccelerometer *)accelerometer
   didAccelerate:(UIAcceleration *)acceleration
{
   // Extrahiert die Beschleunigungskomponenten
   float xx = -[acceleration x];
   float yy = [acceleration y];

   // Hat sich die Richtung geändert?
   float accelDirX = SIGN(xvelocity) * -1.0f;
   float newDirX = SIGN(xx);
   float accelDirY = SIGN(yvelocity) * -1.0f;
   float newDirY = SIGN(yy);

   // Beschleunigt das Objekt. Um die Zähigkeit zu erhöhen, verringern
   // Sie den addierten Wert.
   if (accelDirX == newDirX)
      xaccel = (abs(xaccel) + 0.85f) * SIGN(xaccel);
   if (accelDirY == newDirY)
      yaccel = (abs(yaccel) + 0.85f) * SIGN(yaccel);

   // Wendet Beschleunigungsänderungen auf die aktuelle Geschwindigkeit an
   xvelocity = -xaccel * xx;
   yvelocity = -yaccel * yy;
}

- (CGRect) offsetButterflyBy: (float) dx and: (float) dy
{
   CGRect rect = [self.butterfly frame];
   rect.origin.x += dx;
   rect.origin.y += dy;
   return rect;
}

- (void) tick
{
```

```
    // Verschiebt den Schmetterling um den aktuellen Geschwindigkeitsvektor
    CGRect rect;

    // Freie Bewegung
    if (CGRectContainsRect(self.view.bounds,
        rect = [self offsetButterflyBy:xvelocity and:yvelocity]));

    // Kante in vertikaler Bewegungsrichtung
    else if (CGRectContainsRect(self.view.bounds,
        rect = [self offsetButterflyBy:xvelocity and:0.0f]));

    // Kante in horizontaler Bewegungsrichtung
    else if (CGRectContainsRect(self.view.bounds,
        rect = [self offsetButterflyBy:0.0f and:yvelocity]));

    // Ecke
    else return;

    [butterfly setFrame:rect];
}

- (void) initButterfly
{
    // Lädt die Animationszellen
    NSMutableArray *bflies = [NSMutableArray array];
    for (int i = 1; i <= 17; i++)
        [bflies addObject:[UIImage imageNamed:
            [NSString stringWithFormat:@"bf_%d.png", i]]];

    // Erstellt den Schmetterling und beginnt die Animation
    self.butterfly = [[[UIImageView alloc] initWithFrame:
        CGRectMake(0.0f, 0.0f, 150.0f, 76.5f)] autorelease];
    [self.butterfly setAnimationImages:bflies];
    self.butterfly.animationDuration = 0.75f;
    [self.butterfly startAnimating];
    self.butterfly.center = CGPointMake(160.0f, 100.0f);
    [self.view addSubview:butterfly];

    // Legt die Anfangsgeschwindigkeit und -beschleunigung des
    // Schmetterlings fest
    xaccel = 2.0f;
    yaccel = 2.0f;
```

```
    xvelocity = 0.0f;
    yvelocity = 0.0f;

    // Aktiviert den Beschleunigungsmesser
    [[UIAccelerometer sharedAccelerometer] setDelegate:self];

    // Startet den Physik-Timer
    [NSTimer scheduledTimerWithTimeInterval: 0.03f
       target: self selector: @selector(tick)
       userInfo: nil repeats: YES];
}
```

► *Rezept 14.6: Ein Objekt auf dem Bildschirm aufgrund von Meldungen des Beschleunigungsmessers verschieben*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 14 und öffnen das Projekt zu diesem Rezept.

14.8 Rezept: Die Geräteorientierung erkennen

Mit »Orientierung« ist gemeint, wie herum der Benutzer das Gerät hält. Diese Information können Sie jederzeit ablesen, indem Sie `[UIDevice currentDevice].orientation` abrufen. Diese Eigenschaft gibt eine Geräteorientierungsnummer zurück, die einem der folgenden Zustände entspricht:

```
typedef enum {
    UIDeviceOrientationUnknown,
    UIDeviceOrientationPortrait,
    UIDeviceOrientationPortraitUpsideDown,
    UIDeviceOrientationLandscapeLeft,
    UIDeviceOrientationLandscapeRight,
    UIDeviceOrientationFaceUp,
    UIDeviceOrientationFaceDown
} UIDeviceOrientation;
```

Die Orientierungen `Portrait` (Hochformat) und `Landscape` (Querformat) sind selbsterklärend. Bei `FaceUp` liegt das Gerät flach mit dem Bildschirm nach oben auf einer Oberfläche, bei `FaceDown` mit dem Bildschirm nach unten. Die Orientierungen werden vom SDK mithilfe des eingebauten Beschleunigungsmessers und durch Berechnungen ermittelt, die denen aus dem vorherigen Rezept ähneln.

Meistens ist es am wichtigsten, zu erkennen, ob sich das Gerät zurzeit im Hoch- oder im Querformat befindet. Um dies zu bestimmen, stellt Apple zwei mitgelieferte Hilfsmakros zur Verfügung, denen Sie eine Orientierung übergeben, wie Sie im folgenden Codefragment sehen. Die Makros geben den booleschen Wert `YES` oder `NO` zurück, um anzuzeigen, ob das Gerät im entsprechenden Format (hoch bzw. quer) gehalten wird oder nicht.

```
- (BOOL) shouldAutorotateToInterfaceOrientation:
   (UIInterfaceOrientation) anOrientation
{
   printf("Is Portrait?: %s\n",
      UIDeviceOrientationIsPortrait(anOrientation)
      ? "Yes" : "No");
   printf("Is Landscape?: %s\n",
      UIDeviceOrientationIsLandscape(anOrientation)
      ? "Yes" : "No");
   return YES;
}
```

Wenn Sie die Orientierung außerhalb des `shouldAutorotate...`-Callbacks für den Ansichtscontroller bestimmen möchten, wird der Code komplizierter und weist viele Wiederholungen auf. *Rezept 14.7* erstellt die Kategorie `Orientation` für die Klasse `UIDevice` und stellt die Eigenschaften `isLandscape` und `isPortrait` bereit. Außerdem wird in diesem Rezept die Eigenschaft `orientationString` angelegt, die eine Beschreibung der Orientierung in Textform liefert.

Hinweis

Zurzeit gibt das iPhone seine Orientierung nicht richtig an, wenn es eingeschaltet wird. Die Orientierung wird erst aktualisiert, nachdem der Benutzer das iPhone in eine andere Position bewegt hat. Eine Anwendung im Hochformat wird nicht als hochformatig erkannt, bevor der Benutzer die Orientierung ändert und dann wiederherstellt. Dieser Bug zeigt sich sowohl im Simulator als auch auf einem iPhone-Gerät und lässt sich mit *Rezept 14.7* sehr leicht sichtbar machen. Als Lösung können Sie die Winkelorientierung aus *Rezept 14.5* verwenden. Dieser Bug beeinträchtigt jedoch nicht die korrekte Darstellung der Oberfläche über die Klasse `UIViewController`.

```
@implementation UIDevice (Orientation)
- (BOOL) isLandscape
{
   return (self.orientation == UIDeviceOrientationLandscapeLeft)
      || (self.orientation == UIDeviceOrientationLandscapeRight);
}

- (BOOL) isPortrait
{
   return (self.orientation == UIDeviceOrientationPortrait)
      || (self.orientation == UIDeviceOrientationPortraitUpsideDown);
}

- (NSString *) orientationString
{
```

```
    switch ([[UIDevice currentDevice] orientation])
    {
        case UIDeviceOrientationUnknown: return @"Unknown";
        case UIDeviceOrientationPortrait: return @"Portrait";
        case UIDeviceOrientationPortraitUpsideDown:
            return @"Portrait Upside Down";
        case UIDeviceOrientationLandscapeLeft:
            return @"Landscape Left";
        case UIDeviceOrientationLandscapeRight:
            return @"Landscape Right";
        case UIDeviceOrientationFaceUp: return @"Face Up";
        case UIDeviceOrientationFaceDown: return @"Face Down";
        default: break;
    }
    return nil;
}
@end
```

► *Rezept 14.7: Die Kategorie* `Orientation` *für* `UIDevice`

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 14 und öffnen das Projekt zu diesem Rezept.

14.9 Schüttelbewegungen mit Bewegungsereignissen erkennen

Wenn das iPhone ein Bewegungsereignis erkennt, übergibt es dieses an den jeweiligen First Responder, also das erste Objekt in der Responderkette. Responder sind Objekte, die Ereignisse handhaben können. Alle Ansichten und Fenster und auch das Anwendungsobjekt sind Responder.

Die Responderkette bildet eine Hierarchie von Objekten, die alle auf Ereignisse reagieren können. Wenn ein Objekt am Anfang der Kette ein Ereignis empfängt, geht dieses Ereignis nicht auch noch an die Objekte weiter hinten in der Kette, sondern wird von dem betreffenden Objekt verarbeitet. Wenn dieses Objekt dazu nicht in der Lage ist, kann das Objekt zum nächsten Responder übergehen.

Objekte werden häufig dadurch zum ersten Responder, dass sie sich über `becomeFirstResponder` selbst dazu deklarieren. In dem folgenden Codefragment macht sich der `UIViewController` selbst zum First Responder, sobald seine Ansicht auf dem Bildschirm erscheint. Beim Ausblenden der Ansicht gibt er diese Position wieder auf.

```
- (BOOL)canBecomeFirstResponder {
    return YES;
}
// Bekommt den ersten Responder, wenn die Ansicht erscheint
- (void)viewDidAppear:(BOOL)animated {
    [super viewDidAppear:animated];
    [self becomeFirstResponder];
}

// Gibt den ersten Responder zurück, wenn die Ansicht verschwindet
- (void)viewWillDisappear:(BOOL)animated {
    [super viewWillDisappear:animated];
    [self resignFirstResponder];
}
```

Der First Responder empfängt alle Berührungs- und Bewegungsereignisse. Die Bewegungs-Callbacks sind das Gegenstück zu den Berührungs-Callbacks aus Kapitel 8, *Gesten und Berührungen*. Es handelt sich um folgende:

- `motionBegan:withEvent:` Dieser Callback zeigt den Beginn eines Bewegungsereignisses an. Zurzeit wird nur eine Art von Bewegungsereignis erkannt, nämlich eine Schüttelbewegung. Da sich dies in Zukunft ändern mag, sollten Sie in Ihrem Code den Bewegungstyp überprüfen.
- `motionEnded:withEvent:` Diesen Callback empfängt der First Responder am Ende des Bewegungsereignisses.
- `motionCancelled:withEvent:` Wie Berührungen können auch Bewegungen durch eingehende Telefonanrufe und andere Systemereignisse abgebrochen werden. Apple empfiehlt, im Produktionscode alle drei Callbacks für Bewegungsereignisse zu implementieren (und ebenso alle Callbacks für Berührungsereignisse).

Rezept 14.8 zeigt zwei Beispiele für Bewegungs-Callbacks. Wenn Sie diesen Code auf einem Gerät ausprobieren, werden Ihnen mehrere Dinge auffallen. Erstens erfolgen die Ereignisse für Beginn und Ende der Bewegung für den Benutzer nahezu gleichzeitig. Töne für beide Ereignistypen abzuspielen ist des Guten zu viel. Zweitens ist die Erkennung von Bewegungen nicht in allen Richtungen gleich gut. Das iPhone kann Seitwärtsbewegungen besser erkennen als Bewegungen nach oben und unten oder nach vorn und hinten. Drittens weist die Apple-Implementierung von Bewegungsereignissen ein leichtes Sperrverhalten auf. Ein neues Bewegungsereignis können Sie erst etwa eine Sekunde nach der Verarbeitung des vorhergehenden erstellen. Dies ähnelt dem Sperrverhalten bei der schüttelgesteuerten Bearbeitung und dem schüttelgesteuerten Widerruf.

```
- (void)motionBegan:(UIEventSubtype)motion
    withEvent:(UIEvent *)event {
    // Spielt beim Beginn einer Schüttelbewegung einen Ton ab
    if (motion != UIEventSubtypeMotionShake) return;
    [self playSound:startSound];
}
```

```
- (void)motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event
{
    // Spielt am Ende einer Schüttelbewegung einen Ton ab
    if (motion != UIEventSubtypeMotionShake) return;
    [self playSound:endSound];
}
```

► *Rezept 14.8: Bewegungsereignisse im First Reponder erfassen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 14 und öffnen das Projekt zu diesem Rezept.

14.10 Rezept: Schüttelbewegungen direkt vom Beschleunigungsmesser ablesen

Rezept 14.9 imitiert das Bewegungsermittlungssystem von Apple, ohne dass dazu ein Ereigniskonsument als First Responder erforderlich ist. Der Code baut auf zwei Hauptparametern auf: einem Empfindlichkeitsgrad als Schwellenwert, der erreicht werden muss, bevor eine Schüttelbewegung erkannt wird, und einer Sperrzeit, die festlegt, wie oft ein neues Schüttelereignis generiert werden kann.

Die Klasse `AccelerometerHelper` speichert eine Dreiergruppe von Beschleunigungswerten, wobei jeder Wert für eine Komponente des Kraftvektors im dreidimensionalen Raum steht. Jeweils zwei aufeinander folgende Dreiergruppen können analysiert werden, um den Winkel zwischen den beiden Vektoren zu bestimmen. In diesem Beispiel wird anhand des Winkels zwischen den ersten beiden Einträgen und den folgenden beiden bestimmt, ob eine Schüttelbewegung vorliegt. Der Code sucht nach einem Paar, bei dem der zweite Winkel größer ist als der erste. Wenn sich die Bewegung um den Winkel vergrößert (wenn also eine Erhöhung der Winkelgeschwindigkeit vorliegt, was auf ein »Reißen« hindeutet), wird eine Schüttelbewegung erkannt.

Die Hilfsklasse erstellt keine Delegierungs-Callbacks, bevor eine zweite Hürde genommen ist. Eine Sperre verhindert, dass vor dem Verstreichen einer gewissen Zeit neue Callbacks auftreten. Dies wird durch die Speicherung der Zeit umgesetzt, zu der das letzte Schüttelereignis ausgelöst wurde. Alle Schüttelbewegungen, die vor dem Ablauf der Sperrzeit auftreten, werden ignoriert. Neue Schüttelereignisse werden erst wieder danach generiert.

Bei der eingebauten Schüttelerkennung von Apple erfolgt eine anspruchsvollere Analyse der Daten aus dem Beschleunigungsmesser. Nach den Informationen eines Technikers, der sich auf diesem Gebiet auskennt, wird dabei nach Schwankungen zwischen etwa acht bis zehn aufeinander folgenden Datenpunkten gesucht. *Rezept 14.9* verfolgt einen weniger komplizierten Ansatz und zeigt, wie Sie die Rohdaten des Beschleunigungsmessers nutzen und daraus Ergebnisse berechnen können.

```
@implementation AccelerometerHelper
- (id) init
{
   if (!(self = [super init])) return self;

   self.triggerTime = [NSDate date];

   // Aktueller Kraftvektor
   cx = UNDEFINED_VALUE;
   cy = UNDEFINED_VALUE;
   cz = UNDEFINED_VALUE;

   // Letzter Kraftvektor
   lx = UNDEFINED_VALUE;
   ly = UNDEFINED_VALUE;
   lz = UNDEFINED_VALUE;

   // Vorhergehender Kraftvektor
   px = UNDEFINED_VALUE;
   py = UNDEFINED_VALUE;
   pz = UNDEFINED_VALUE;

   self.sensitivity = 0.5f;
   self.lockout = 0.5f;

   // Startet den Beschleunigungsmesser
   [[UIAccelerometer sharedAccelerometer] setDelegate:self];

   return self;
}

- (void) setX: (float) aValue
{
   px = lx;
   lx = cx;
   cx = aValue;
}

- (void) setY: (float) aValue
{
   py = ly;
   ly = cy;
   cy = aValue;
}
```

```
- (void) setZ: (float) aValue
{
   pz = lz;
   lz = cz;
   cz = aValue;
}

- (float) dAngle
{
   if (cx == UNDEFINED_VALUE) return UNDEFINED_VALUE;
   if (lx == UNDEFINED_VALUE) return UNDEFINED_VALUE;
   if (px == UNDEFINED_VALUE) return UNDEFINED_VALUE;

   // Berechnet das Punktprodukt des ersten Paars
   float dot1 = cx * lx + cy * ly + cz * lz;
   float a = ABS(sqrt(cx * cx + cy * cy + cz * cz));
   float b = ABS(sqrt(lx * lx + ly * ly + lz * lz));
   dot1 /= (a * b);

   // Berechnet das Punktprodukt des zweiten Paars
   float dot2 = lx * px + ly * py + lz * pz;
   a = ABS(sqrt(px * px + py * py + pz * pz));
   dot2 /= a * b;

   // Gibt den Unterschied zwischen den Vektorwinkeln zurück
   return acos(dot2) - acos(dot1);
}

- (BOOL) checkTrigger
{
   if (lx == UNDEFINED_VALUE) return NO;

   // Prüft, ob die neue Daten ausgelöst werden können
   if ([[NSDate date] timeIntervalSinceDate:self.triggerTime]
      < self.lockout) return NO;

   // Ruft die aktuelle Winkeländerung ab
   float change = [self dAngle];

   // Gibt NO zurück, wenn noch nicht zwei Werte erfasst sind
   if (change == UNDEFINED_VALUE) return NO;

   // Übersteigt das Punktprodukt den Trigger?
   if (change > self.sensitivity)
   {
```

```
            self.triggerTime = [NSDate date];
            return YES;
        }
        else return NO;
    }

    - (void)accelerometer:(UIAccelerometer *)accelerometer
        didAccelerate:(UIAcceleration *)acceleration
    {
        // Passt die Werte an ein Standardkoordinatensystem an
        [self setX:-[acceleration x]];
        [self setY:[acceleration y]];
        [self setZ:[acceleration z]];

        // Alle Beschleunigungsmesser-Ereignisse
        if (self.delegate &&
            [self.delegate respondsToSelector:@selector(ping)])
            [self.delegate performSelector:@selector(ping)];

        // Alle Schüttelereignisse
        if ([self checkTrigger] && self.delegate &&
            [self.delegate respondsToSelector:@selector(shake)])
        {
            [self.delegate performSelector:@selector(shake)];
        }
    }
    @end
```

► *Rezept 14.9: Schüttelbewegungen mit der Klasse* `AccelerometerHelper` *erkennen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 14 und öffnen das Projekt zu diesem Rezept.

14.11 Ein letzter Punkt: Den verfügbaren Festplattenspeicher ermitteln

Mit der Klasse `NSFileManager` können Sie bestimmen, wie viel Platz auf dem iPhone insgesamt zur Verfügung steht und wie viel davon noch frei ist. *Listing 14.1* zeigt, wie Sie diese Werte ermitteln und wie Sie sie als gut lesbaren, durch Kommata getrennten String anzeigen. Die zurückgegebenen Werte zeigen den Speicherplatz in Bytes an.

```
- (NSString *) commasForNumber: (long long) num
{
   // Gibt einen sauber formatierten String aus Zahlen zurück.
   // Alternativ können Sie NSNumberFormatter verwenden.
   if (num < 1000) return [NSString stringWithFormat:@"%d", num];
   return [[self commasForNumber:num/1000]
      stringByAppendingFormat:@",%03d", (num % 1000)];
}

- (void) action: (UIBarButtonItem *) bbi
{
   self.log = [NSFileManager defaultManager];
   NSFileManager *fm = [NSFileManager defaultManager];
   NSDictionary *fattributes =
      [fm fileSystemAttributesAtPath:NSHomeDirectory()];
   [self doLog:@"System space: %@",
      [self commasForNumber:[[fattributes
      objectForKey:NSFileSystemSize] longLongValue]]];
   [self doLog:@"System free space: %@",
      [self commasForNumber:[[fattributes
      objectForKey:NSFileSystemFreeSize] longLongValue]]];
}
```

▸ *Listing 14.1: Gesamtgröße und freien Speicherplatz des Dateisystems abrufen*

14.12 Zusammenfassung

In diesem Kapitel haben Sie elementare Möglichkeiten zur Interaktion mit dem iPhone-Gerät kennengelernt. Sie haben gesehen, wie Sie Geräteinformationen abrufen, den Ladezustand des Akkus prüfen und Ereignisse des Annäherungssensors abonnieren. Außerdem wurden der Beschleunigungsmesser und seine Verwendung in mehreren Beispielen vorgestellt: von der einfachen Möglichkeit, herauszufinden, wo oben ist, bis zu einem anspruchsvollen Algorithmus zur Erkennung von Schüttelbewegungen. Des Weiteren haben Sie gelernt, wie Sie einen iPod touch von einem iPhone unterscheiden und bestimmen können, mit was für einem Modell Sie es zu tun haben. Bedenken Sie auch die folgenden Schlussbemerkungen zu den hier vorgestellten Rezepten:

- Der Beschleunigungsmesser bietet eine neuartige Ergänzung zur Berührungssteuerung der Oberfläche. Mit Beschleunigungsdaten können Sie neben einfacher Berührung auch Schüttelbewegungen zur Benutzerinteraktion nutzen.
- Systemnahe Aufrufe können sich für die SDK-Programmierung eignen, da sie sich nicht auf Apple-APIs stützen, die sich möglicherweise mit der nächsten Version der Firmware ändern. UNIX-Systemaufrufe mögen abschreckend erscheinen, allerdings werden viele davon auf dem iPhone unterstützt.
- Denken Sie an die Einschränkungen der Geräte. Prüfen Sie lieber erst auf den verfügbaren Festplattenplatz, bevor Sie dateiintensive Aufgaben durchführen, und sehen Sie sich den Ladezustand des Akkus an, bevor Sie die CPU mit Volldampf arbeiten lassen.
- Wenn Sie Anwendungen in iTunes einreichen, geben Sie ab Version 3.0 nicht mehr an, für welche Geräte sie sich eignen, sondern legen in der `Info.plist`-Datei fest, welche Gerätemerkmale erforderlich sind. Anhand dieser Liste bestimmt iTunes, ob eine Anwendung auf ein gegebenes Gerät heruntergeladen und dort korrekt ausgeführt werden kann.

15 Audio, Video und MediaKit

Das iPhone beherrscht die Nutzung von Medien meisterhaft. Die eingebauten iPod-Funktionen können sowohl Bild als auch Ton korrekt verarbeiten. Über das iPhone SDK erhalten Entwickler Zugriff auf diese Funktionen. Eine breite Palette an Klassen vereinfacht die Wiedergabe, Suche und Aufnahme von Medien. In diesem Kapitel finden Sie Rezepte, in denen diese Klassen verwendet werden, um den Benutzern Medien anzuzeigen und ihnen zu ermöglichen, mit diesen Medien zu arbeiten. Sie lernen hier, wie Sie Objekte zur Anzeige und zur Aufnahme von Ton und Bild erstellen, wie Sie die iPod-Bibliothek durchsuchen und wie Sie auswählen, welche Elemente wiedergegeben werden sollen. Die Rezepte, denen Sie hier begegnen, zeigen Schritt für Schritt, wie Sie Ihren Anwendungen Medienfunktionen hinzufügen.

15.1 Rezept: Audiowiedergabe mit AVAudioPlayer

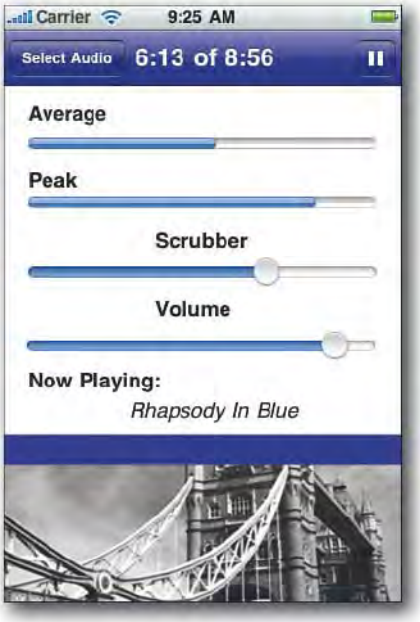
Wie der Name schon andeutet, dient die Klasse `AVAudioPlayer` zur Wiedergabe von Audiodaten. Diese einfach zu verwendende Klasse bietet viele Möglichkeiten, von denen einige in Abbildung 15.1 zu erkennen sind. Mit dieser Klasse können Sie Audiodaten laden, wiedergeben, die Wiedergabe anhalten und beenden, den durchschnittlichen und den Spitzen-Audiopegel bestimmen, die Wiedergabelautstärke anpassen und die Stelle anzeigen und festlegen, die gerade wiedergegeben wird. All diese Funktionen lassen sich mit nur geringem Entwicklungsaufwand nutzen. Wie Sie noch sehen werden, verfügt die Klasse `AVAudioPlayer` über eine solide API.

15.1.1 Einen Audioplayer initialisieren

Die Funktion zur Audiowiedergabe von `AVAudioPlayer` lässt sich mit geringem Aufwand in Ihrem Code implementieren. Apple bietet eine unkomplizierte Klasse, die für das Laden und Abspielen von Dateien optimiert ist.

Als Erstes erstellen Sie den Player und initialisieren ihn mit Daten oder mit den Inhalten, die an einem lokalen URL zu finden sind. Im folgenden Codefragment zeigt ein Datei-URL auf eine Audiodatei. Jeglicher Fehler beim Erstellen und Einrichten des Players wird gemeldet. Mit `initWithData:error:` können Sie den Player auch mit Daten initialisieren, die sich bereits im Arbeitsspeicher befinden. Das ist praktisch, wenn Sie Daten bereits in den Speicher geladen haben (z. B. bei einem Audio-Chat) und nicht mehr von einer Datei auf dem Gerät laden müssen.

```
self.player = [[AVAudioPlayer alloc] initWithContentsOfURL:
   [NSURL fileURLWithPath:self.path] error:&error];
if (!self.player)
{
   NSLog(@"Error %@", [error localizedDescription]);
   return;
}
```



▸ *Abbildung 15.1: Die in diesem Screenshot dargestellten Funktionen wurden einzig und allein mit der Klasse `AVAudioPlayer` eingerichtet. Sie ermöglicht die Anzeige der Wiedergabezeit (in der Mitte der Titelleiste), die Anzeige der Pegel (Durchschnitts- und Spitzenwert – **Average** und **Peak**), Schieberegler für die Wahl der Wiedergabeposition (**Scrubber**) und der Lautstärke (**Volume**) sowie Schaltflächen für Wiedergabe und Pause (rechts in der Titelleiste).*

Nachdem Sie den Player initialisiert haben, bereiten Sie ihn für die Wiedergabe vor. Mit dem Aufruf von `prepareToPlay` sorgen Sie dafür, dass die Wiedergabe so schnell wie möglich beginnt, wenn Sie dazu bereit sind, sie mit `play` zu starten. Dieser Aufruf lädt vorab den Puffer des Players und initialisiert die Hardware zur Audiowiedergabe.

```
[self.player prepareToPlay];
```

Mit dem Aufruf von `pause` können Sie die Wiedergabe jederzeit anhalten. Dies hat keine Auswirkungen auf die Eigenschaft `currentTime` des Players. Wenn Sie erneut `play` aufrufen, können Sie die Wiedergabe an derselben Stelle wieder aufnehmen.

Wenn Sie die Wiedergabe dagegen vollständig beenden wollen, verwenden Sie `stop`. Dadurch wird die im Puffer abgelegte Einrichtung entfernt, die Sie mit `prepareToPlay` festgelegt haben. Die aktuelle Zeit wird jedoch nicht auf 0,0 zurückgesetzt, sodass Sie wie bei `pause` die Wiedergabe an derselben Stelle wieder aufnehmen können, indem Sie `play` aufrufen. Da der Player seine Puffer neu lädt, können beim Start jedoch Verzögerungen auftreten.

15.1.2 Audiopegel überwachen

Wenn Sie die Audiopegel überwachen möchten, richten Sie als Erstes die Eigenschaft `meteringEnabled` ein. Bei aktivierter Messeinrichtung können Sie während der Audiowiedergabe und -aufnahme die Pegel bestimmen.

```
self.player.meteringEnabled = YES;
```

Die Klasse `AVAudioPlayer` stellt Informationen über den Durchschnitts- und den Spitzenwert der Aussteuerung zur Verfügung, die Sie kanalweise abrufen können. Fragen Sie dazu die Anzahl der verfügbaren Kanäle ab (über die Eigenschaft `numberOfChannels`), und ermitteln Sie dann die einzelnen Audiopegel, indem Sie den Kanalindex angeben. Ein Monosignal läuft über Kanal 0, der auch für die Stereoaufnahme verwendet wird.

Sie müssen nicht nur die Messung allgemein einrichten, sondern auch jedes Mal, wenn Sie die Pegel bestimmen möchten, die AV-Player-Methode `updateMeters` aufrufen, die die Pegelstände aktualisiert. Anschließend können Sie die Pegelwerte mit `peakPowerForChannel:` und `averagePowerForChannel:` ablesen. In *Rezept 15.7* erfahren Sie genau, was im Player hinter den Kulissen geschieht, wenn Sie diese Aussteuerungspegel abfragen. Der Code fordert die Messwerte an und entnimmt daraus den Spitzen- bzw. den Durchschnittspegel. Die Klasse `AVAudioPlayer` kapselt all diese Einzelheiten, was den Zugriff auf diese Werte vereinfacht.

`AVAudioPlayer` misst die Aussteuerung in Dezibel, und zwar im Fließkommaformat. Dezibel ist eine logarithmische Maßeinheit des Signalpegels. Aussteuerungswerte reichen von 0 dB als Maximum zu negativen Werten für eine unter dem Maximum liegende Aussteuerung. Je geringer der Wert ist (alle sind negativ), umso schwächer ist das Signal.

```
int channels = self.player.numberOfChannels;
[self.player updateMeters];
for (int i = 0; i < channels; i++)
{
   // Zeichnet Spitzen- und Durchschnittsaussteuerung auf
   NSLog(@"%d %0.2f %0.2f",
      [self.player peakPowerForChannel:i],
      [self.player averagePowerForChannel:i];
}
```

Die Verstärkung des Audioplayers (also letztlich die »Lautstärke«) ermitteln Sie über die Eigenschaft `volume`, die eine Fließkommazahl zwischen 0,0 und 1,0 zurückgibt. Sie gilt nicht für die Lautstärke des Systems, sondern des Players. Diese Eigenschaft können Sie nicht nur ablesen, sondern auch

festlegen. In dem folgenden Codefragment wird die Lautstärke mithilfe eines Target-Action-Paars angepasst, wenn der Benutzer einen Regler auf dem Bildschirm betätigt.

```
- (void) setVolume: (id) sender
{
   // Setzt die Lautstärke des Audioplayers auf den aktuellen Reglerwert
   if (self.player) self.player.volume = volumeSlider.value;
}
```

15.1.3 Wiedergabeposition bestimmen und wählen

Mit den beiden Eigenschaften `currentTime` und `duration` können Sie bestimmen, wie weit die Wiedergabe fortgeschritten ist. Um herauszufinden, wie viel Prozent bereits abgespielt sind, dividieren Sie die aktuelle Wiedergabeposition durch die Gesamtdauer.

```
progress = self.player.currentTime / self.player.duration;
```

Wenn der Benutzer die Abspielposition innerhalb der Audiospur frei wählen darf, sollten Sie die Wiedergabe während dieses Vorgangs anhalten, denn die Klasse `AVAudioPlayer` kann dem Benutzer keine akustischen Hinweise geben, um eine Wunschposition zu finden. Warten Sie stattdessen, bis der Benutzer eine Position festgelegt hat, und starten Sie dann von dort aus die Wiedergabe.

Wenn Sie für diese Positionswahl einen standardmäßigen `UISlider`-Regler verwenden möchten, müssen Sie mindestens zwei Target-Action-Paare implementieren. Für das erste müssen Sie `UIControlEventTouchDown` mit `UIControlEventValueChanged` maskieren. Mit diesem Ereignistyp können Sie erfassen, wann der Benutzer beginnt, die Abspielposition zu verschieben, und wann sich der Wert ändert. Als Reaktion auf diese Ereignisse sollten Sie den Audioplayer anhalten und einen optischen Hinweis auf die neue Wiedergabeposition geben.

```
- (void) scrub: (id) sender
{
   // Hält den Player an
   [self.player pause];

   // Berechnet die neue Wiedergabeposition
   self.player.currentTime = scrubber.value * self.player.duration;

   // Aktualisiert den Titel mit der neuen Wiedergabeposition
   self.title = [NSString stringWithFormat:@"%@ of %@",
      [self formatTime:self.player.currentTime],
      [self formatTime:self.player.duration]];
}
```

Im zweiten Target-Action-Paar können Sie mit einer Maske aus den drei Werten `UIControlEventTouchUpInside`, `UIControlEventTouchUpOutside` und `UIControlEventCancel` das gewollte Ende und den Abbruch von Berührungen erfassen. Beendet der Benutzer die Berührung, starten Sie die Wiedergabe an der vom Benutzer gewählten Position.

```
- (void) scrubbingDone: (id) sender
{
   // Nehmen Sie hier die Wiedergabe wieder auf
   [self play:nil];
}
```

15.1.4 Das Ende der Wiedergabe erkennen

Um das Ende der Wiedergabe zu erkennen, richten Sie die Delegierung des Players ein und erfassen den Delegierungs-Callback `audioPlayerDidFinishPlaying:successfully:`. Diese Methode ist der geeignete Ort, um Aufräumarbeiten zu erledigen, also z. B., um die Pausen-Schaltfläche wieder in eine Wiedergabe-Schaltfläche umzuwandeln. Apple hält die folgenden Systemschaltflächen eigens für die Medienwiedergabe vor:

- `UIBarButtonSystemItemPlay`
- `UIBarButtonSystemItemPause`
- `UIBarButtonSystemItemRewind`
- `UIBarButtonSystemItemFastForward`

Die Schaltflächen zum Zurückspulen und schnellen Vorspulen zeigen Doppelpfeile, also die üblichen Symbole für den Vorgang, die Abspielposition zum vorhergehenden oder nächsten Element einer Wiedergabeliste zu verschieben. Damit können Sie auch zum Anfang oder Ende der Spur springen. Leider zeigt das Systemelement für Stopp ein X, wie es für Schaltflächen zum Abbrechen eines laufenden Ladevorgangs üblich ist, und nicht das gewohnte gefüllte Quadrat, das auf physischen Geräten auf den Tasten zum Beenden der Wiedergabe oder Aufnahme zu sehen ist.

Rezept 15.1 kombiniert all diese Elemente, um die einheitliche Oberfläche zu erstellen, die Sie bereits in *Abbildung 15.1* gesehen haben. Hier kann der Benutzer eine Audiodatei auswählen, sie wiedergeben, die Wiedergabe anhalten, die Lautstärke regeln, die Abspielposition willkürlich wählen usw.

Der XMAX-Trick, den Sie hier sehen, gehört schon fast ins Reich der Hacks. Dabei wird ein willkürlicher Maximalwert verwendet, um den Dynamikbereich der Eingangspegel abzuschätzen. Im Gegensatz zu direkten Audio-Queue-Aufrufen (die einen Fließkommawert zwischen 0,0 und 1,0 zurückgeben) werden die Pegelwerte hier angenähert, um einen Fortschrittswert für die Rückmeldung in Echtzeit zu bekommen. Testen Sie bei der Entwicklung die XMAX-Werte, und richten Sie sie so ein, wie sie am besten für Ihre Anwendung passen.

```
- (void) updateMeters
{
   // Ruft die Daten der Audiometer ab und aktualisiert die Darstellung auf
   // dem Bildschirm
   [self.player updateMeters];
   float avg = -1.0f * [self.player averagePowerForChannel:0];
   float peak = -1.0f * [self.player peakPowerForChannel:0];
```

```
    meter1.progress = (XMAX  avg) / XMAX;
    meter2.progress = (XMAX  peak) / XMAX;

    // Zeigt aktuelle Wiedergabeposition und aktualisiert die
    // Positionswahlleiste
    self.title = [NSString stringWithFormat:@"%@ of %@",
        [self formatTime:self.player.currentTime],
        [self formatTime:self.player.duration]];
    scrubber.value = (self.player.currentTime / self.player.duration);
}

- (void) pause: (id) sender
{
    // Hält die Wiedergabe an und aktualisiert die Wiedergabe/Pause-
    // Schaltfläche
    if (self.player) [self.player pause];
    self.navigationItem.rightBarButtonItem =
        SYSBARBUTTON(UIBarButtonSystemItemPlay, self,
        @selector(play:));

    // Deaktiviert die Audiometer und den Überwachungs-Timer
    meter1.progress = 0.0f;
    meter2.progress = 0.0f;
    [timer invalidate];

    // Deaktiviert den Lautstärkeregler
    volumeSlider.enabled = NO;

    // Deaktiviert die Positionswahlleiste
    scrubber.enabled = NO;
}

- (void) play: (id) sender
{
    // Beginnt die Wiedergabe oder nimmt sie wieder auf
    if (self.player) [self.player play];

    // Aktualisiert und aktiviert den Lautstärkeregler
    volumeSlider.value = self.player.volume;
    volumeSlider.enabled = YES;

    // Aktualisiert die Wiedergabe/Pause-Schaltfläche
    self.navigationItem.rightBarButtonItem =
        SYSBARBUTTON(UIBarButtonSystemItemPause, self,
        @selector(pause:));
```

```
    // Beginnt mit der Pegelüberwachung
    timer = [NSTimer scheduledTimerWithTimeInterval:0.1f
        target:self selector:@selector(updateMeters)
        userInfo:nil repeats:YES];

    // Aktiviert die Positionswahlleiste während der Wiedergabe
    scrubber.enabled = YES;
}

- (void) setVolume: (id) sender
{
    // Reagiert auf Lautstärkeänderungen durch den Benutzer
    if (self.player) self.player.volume = volumeSlider.value;
}

- (void) scrubbingDone: (id) sender
{
    // Beginnt die Wiedergabe an der gewählten Position
    [self play:nil];
}

- (void) scrub: (id) sender
{
    // Hält den Player an
    [self.player pause];

    // Berechnet die neue Wiedergabeposition
    self.player.currentTime = scrubber.value * self.player.duration;

    // Aktualisiert Titel und Navigationsleiste
    self.title = [NSString stringWithFormat:
        @"%@ of %@", [self formatTime:self.player.currentTime],
        [self formatTime:self.player.duration]];
    self.navigationItem.rightBarButtonItem =
        SYSBARBUTTON(UIBarButtonSystemItemPlay, self,
        @selector(play:));
}

- (BOOL) prepAudio
{
    // Prüft, ob die Audiodatei vorhanden ist
    NSError *error;
    if (![[NSFileManager defaultManager]
        fileExistsAtPath:self.path]) return NO;

    // Initialisiert den Player
```

```
    self.player = [[AVAudioPlayer alloc] initWithContentsOfURL:
        [NSURL fileURLWithPath:self.path] error:&error];
    if (!self.player)
    {
        NSLog(@"Error %@", [error localizedDescription]);
        return NO;
    }

    // Bereitet Player, Audiometer usw. vor
    [self.player prepareToPlay];
    self.player.meteringEnabled = YES;
    meter1.progress = 0.0f;
    meter2.progress = 0.0f;
    self.player.delegate = self;

    // Richtet die Wiedergabe/Pause-Schaltfläche ein
    self.navigationItem.rightBarButtonItem =
        SYSBARBUTTON(UIBarButtonSystemItemPlay, self,
        @selector(play:));
    scrubber.enabled = NO;

    return YES;
}

- (void)audioPlayerDidFinishPlaying:(AVAudioPlayer *)player
    successfully:(BOOL)flag
{
    // Setzt am Ende der Wiedergabe die Benutzeroberfläche zurück
    self.navigationItem.rightBarButtonItem = nil;
    scrubber.value = 0.0f;
    scrubber.enabled = NO;
    volumeSlider.enabled = NO;

    // Bereitet die erneute Wiedergabe vor
    [self prepAudio];
}
```

▸ *Rezept 15.1: Audiowiedergabe mit* `AVAudioPlayer`

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.2 Rezept: Schleifenwiedergabe

Durch Schleifenwiedergabe können Sie Hintergrundgeräusche gestalten, indem Sie einen Klang mehrmals hintereinander oder kontinuierlich abspielen. *Rezept 15.2* zeigt eine Audioschleife, die während der Anzeige eines bestimmten Videocontrollers als akustischer Hintergrund wiedergegeben werden könnte.

Sie können festlegen, wie oft eine Audiodatei abgespielt wird, bevor die Wiedergabe endet. Eine sehr hohe Zahl (etwa 999.999) ergibt in der Praxis eine Endlosschleife. Beispielsweise würde ein Klang von vier Sekunden bei einer so hohen Anzahl von Wiederholungen mehr als 1000 Stunden lang abgespielt werden.

```
// Bereitet den Player vor und richtet die Schleifenwiedergabe ein
[self.player prepareToPlay];
[self.player setNumberOfLoops:999999];
```

In *Rezept 15.2* wird eine Tonschleife für den Hauptansichtscontroller verwendet. Solange sich diese Ansicht auf dem Bildschirm befindet, wird die Schleife im Hintergrund abgespielt. Sie sollten hier Klänge wählen, die nicht stören, die zur Anwendung passen und bei denen das Ende einer Wiedergabe nahtlos in den Anfang der nächsten übergeht.

Am Anfang und Ende der Gesamtwiedergabe wird hier ein Fading-Effekt verwendet. Wenn die Ansicht erscheint, wird der Ton eingeblendet, und beim Verschwinden der Ansicht wird er wieder ausgeblendet. Dies wird mit einem ganz einfachen Ansatz erreicht. Die Schleife iteriert durch verschiedene Lautstärkepegel – von 0,0 bis 1,0 beim Einblenden und von 1,0 bis 0,0 beim Ausblenden. Ein Aufruf der integrierten Standby-Funktion von `NSThread` richtet die Zeitabstände zwischen den einzelnen Lautstärkeänderungen ein (jeweils eine Zehntelsekunde), ohne die Audiowiedergabe zu beeinträchtigen.

```
@implementation TestBedViewController
@synthesize player;

- (BOOL) prepAudio
{
    // Prüft das Vorhandensein der Audiodatei
    NSError *error;
    NSString *path = [[NSBundle mainBundle]
        pathForResource:@"loop" ofType:@"mp3"];
    if (![[NSFileManager defaultManager] fileExistsAtPath:path])
        return NO;

    // Initialisiert den Player
    self.player = [[AVAudioPlayer alloc] initWithContentsOfURL:
        [NSURL fileURLWithPath:path] error:&error];
    if (!self.player)
    {
        NSLog(@"Error %@", [error localizedDescription]);
```

```
        return NO;
    }

    // Bereitet den Player vor und setzt die Schleife praktisch auf
    // unendliche Wiedergabe
    [self.player prepareToPlay];
    [self.player setNumberOfLoops:999999];

    return YES;
}

- (void) viewDidAppear: (BOOL) animated
{
    // Beginnt die Wiedergabe bei der Lautstärke null
    self.player.volume = 0.0f;
    [self.player play];

    // Blendet den Ton im Verlauf einer Sekunde ein
    for (int i = 1; i <= 10; i++)
    {
        self.player.volume = i / 10.0f;
        [NSThread sleepUntilDate:
            [NSDate dateWithTimeIntervalSinceNow:0.1f]];
    }

    // Fügt die Schaltfläche Push hinzu
    self.navigationItem.rightBarButtonItem =
        BARBUTTON(@"Push", @selector(push));
}

- (void) viewWillDisappear: (BOOL) animated
{
    // Blendet den Ton im Verlauf einer Sekunde aus
    for (int i = 9; i >= 0; i--)
    {
        self.player.volume = i / 10.0f;
        [NSThread sleepUntilDate:
            [NSDate dateWithTimeIntervalSinceNow:0.1f]];
    }

    [self.player pause];
}

- (void) push
```

```
{
    // Erstellt einen einfachen neuen Ansichtscontroller
    UIViewController *vc = [[UIViewController alloc] init];
    vc.view.backgroundColor = [UIColor whiteColor];
    vc.title = @"No Sounds";

    // Deaktiviert die betätigte rechte Schaltfläche
    self.navigationItem.rightBarButtonItem = nil;

    // Bringt den neuen Ansichtscontroller auf den Bildschirm
    [self.navigationController
        pushViewController:[vc autorelease] animated:YES];
}

- (void) viewDidLoad
{
    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
    self.navigationItem.rightBarButtonItem =
        BARBUTTON(@"Push", @selector(push));
    self.title = @"Looped Sounds";
    [self prepAudio];
}
@end
```

► *Rezept 15.2: Hintergrundgeräusche durch Schleifenwiedergabe hervorrufen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.3 Unterbrechungen der Audiowiedergabe handhaben

Wenn der Benutzer während der Audiowiedergabe einen Telefonanruf erhält, wird der Ton ausgeblendet. Stattdessen erscheint der normale Bildschirm, auf dem der Benutzer auswählen kann, ob er den Anruf entgegennehmen oder ablehnen will. Wenn dies geschieht, empfangen die `AVAudioPlayer`-Delegierungen den Callback `audioPlayerBeginInteruption:`, den Sie in *Rezept 15.3* sehen. Die Audiositzung wird deaktiviert, und der Player wird angehalten. Sie können die Wiedergabe erst dann wieder starten, wenn die Unterbrechung beendet ist.

Wenn der Benutzer den Anruf entgegennimmt, wird die Anwendung beendet, wobei die Anwendungsdelegierung den Callback `applicatonWillResignActive:` empfängt. Ist das Telefongespräch beendet, wird die Anwendung (mit dem Callback `applicationDidBecomeActive:`) erneut gestartet. Lehnt der Benutzer den Anruf ab oder endet der Anruf, bevor der Benutzer antworten konnte, wird der Delegierung stattdessen `audioPlayerEndInterruption:` gesendet. Von dieser Methode aus können Sie die Wiedergabe wieder aufnehmen.

Wenn die Anwendung neu startet und es wichtig ist, die Wiedergabe nach der Annahme eines Anrufs wiederaufzunehmen, können Sie die aktuelle Abspielposition wie in *Rezept 15.3* speichern. Die Methode `viewDidLoad` in diesem Rezept sucht in den Benutzervoreinstellungen nach einem gespeicherten Wert über die Unterbrechung. Wird sie dort fündig, verwendet sie den Wert, um die letzte Abspielposition für die erneute Wiedergabe wiederherzustellen.

Dieser Ansatz berücksichtigt die Tatsache, dass die Anwendung nach der Beendigung des Anrufs neu gestartet und nicht einfach nur wiederaufgenommen wird. Wenn der Benutzer einen Anruf entgegennimmt, wird kein Callback für das Ende der Unterbrechung gesendet.

```
- (void)audioPlayerBeginInterruption:(AVAudioPlayer *)player
{
    // Handhabt die Unterbrechung
    printf("Interruption Detected\n");
    [[NSUserDefaults standardUserDefaults]
        setFloat:[self.player currentTime]
        forKey:@"Interruption"];
}

- (void)audioPlayerEndInterruption:(AVAudioPlayer *)player
{
    // Nimmt bei Beendigung der Unterbrechung die Wiedergabe erneut auf
    printf("Interruption ended\n");
    [self.player play];

    // Entfernt den Schlüssel "Interruption", der nicht gebraucht wird
    [[NSUserDefaults standardUserDefaults]
        removeObjectForKey:@"Interruption"];
}

- (void) viewDidLoad
{
    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
    [self prepAudio];

    // Sucht nach vorhergehender Unterbrechung
    if ([[NSUserDefaults standardUserDefaults]
        objectForKey:@"Interruption"])
    {
```

```
        self.player.currentTime =
            [[NSUserDefaults standardUserDefaults]
            floatForKey:@"Interruption"];
        [[NSUserDefaults standardUserDefaults]
            removeObjectForKey:@"Interruption"];
    }

    // Startet die Wiedergabe
    [self.player play];
}
```

► *Rezept 15.3: Den Zeitpunkt der Unterbrechung für die Wiederaufnahme speichern*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.4 Rezept: Wiedergabe auch im Standby-Modus

Wenn der Benutzer das iPhone oder den iPod sperrt, indem er die Standby-Taste betätigt, führt das zu den gleichen Unterbrechungsereignissen wie bei einem eingehenden Telefonanruf. Wird das Gerät gesperrt, gibt `AVAudioPlayer` einen Unterbrechungs-Callback aus, woraufhin der Ton ausgeblendet und die Wiedergabe beendet wird. Beim Entsperren wird der Callback `audioPlayerEndInterruption:` ausgelöst und die Wiedergabe dort fortgesetzt, wo sie aufgehört hat. Um dieses Verhalten zu beobachten, verwenden Sie *Rezept 15.3* und schalten das iPhone zwischendurch ein und aus.

Soll die Audiowiedergabe auch dann fortgesetzt werden, wenn der Benutzer sein Gerät sperrt, müssen Sie die Audiositzung ändern. Audiositzungen legen den Kontext für die Audiowiedergabe einer Anwendung fest und bieten direkten Zugriff auf die Wiedergabehardware.

Um die Audiowiedergabe fortzusetzen, müssen Sie eine Sitzung einrichten, die nicht auf die automatische Sperrung reagiert. Beispielsweise können Sie eine Abspiel- und Aufnahmesitzung verwenden:

```
if (![[AVAudioSession sharedInstance]
    setCategory:AVAudioSessionCategoryPlayAndRecord error:&error])
{
    // Fehler beim Einrichten einer Abspiel- und Aufnahmesitzung
    NSLog(@"Error %@", [error localizedDescription]);
    return NO;
}
```

Fügen Sie diese Zeilen zu Ihrem Code hinzu, bevor Sie einen neuen Player zuweisen. Bei der Wiedergabe werden dann Sperrereignisse ignoriert. Wenn Sie die Standby-Taste drücken, wird der Bildschirm schwarz, aber der Ton wird weiterhin abgespielt.

Es gibt jedoch ein Problem. Wenn Sie eine Wiedergabe- und Aufnahmesitzung verwenden, senkt das iPhone automatisch die Lautstärke für die Lautsprecherausgabe. Das ist so gewollt, denn dadurch sollen Rückkopplungsschleifen vermieden werden, wenn der Benutzer etwas aufnimmt und gleichzeitig auch etwas wiedergibt. Für einen Voice-Chat in zwei Richtungen ist das ideal, aber für die Wiedergabe, bei der Sie den gesamten Audiopegel haben wollen, ist es äußerst schlecht.

Rezept 15.4 bietet eine Lösung, um den Dynamikbereich zu erhalten und gleichzeitig Sperrereignisse zu ignorieren. Dieser Code ruft die systemnahen C-Funktionen für Audiositzungen auf, um die Sitzungskategorie festzulegen. Die Wiedergabekategorie `media` steht nicht als Standardkonstante von `AVAudioPlayer` zur Verfügung, weshalb dieser alternative Ansatz erforderlich ist. Wie eine Wiedergabe- und Aufnahmesitzung ignoriert eine Mediensitzung Ereignisse der Standby-Taste und fährt mit der Wiedergabe fort, allerdings mit voller Lautstärke.

Wenn Sie die Audiositzung auf diese Weise initialisieren, übergeben Sie eine Callback-Funktion statt einer Methode. Dies wird in *Rezept 15.4* durch die Implementierung der Skelettfunktion `interruptionListenerCallback()` gezeigt. Da alle Unterbrechungen bereits im Delegierungscode von *Rezept 15.3* erfasst werden, fügt diese Funktion einfach nur ein paar `print`-Anweisungen hinzu, die Sie auch gern weglassen können.

Geht ein Telefonanruf ein, kümmern sich die Delegierungs-Callbacks aus *Rezept 15.3* um die Unterbrechung und den möglichen Neustart der Anwendung. Allerdings reagiert die Anwendung nie auf Sperr- und Entsperrereignisse. Dies können Sie beobachten, wenn Sie den Beispielcode ausführen und dabei die fünf grundlegenden Arten von Unterbrechungen ausprobieren: entgegengenommener Anruf, abgelehnter Anruf, ignorierter Anruf, Sperren und Entsperren. Durch die Änderung des Typs der Audiositzung werden diese Callbacks nicht mehr hervorgerufen, sodass eine Betätigung der Standby-Taste keine Auswirkung mehr auf die Wiedergabe hat.

```
void interruptionListenerCallback (void *userData,
   UInt32 interruptionState)
{
   if (interruptionState == kAudioSessionBeginInterruption)
      printf("(ilc) Interruption Detected\n");
   else if (interruptionState == kAudioSessionEndInterruption)
      printf("(ilc) Interruption ended\n");
}

- (BOOL) prepAudio
{
   NSError *error;
   NSString *path = [[NSBundle mainBundle]
      pathForResource:@"MeetMeInSt.Louis1904" ofType:@"mp3"];
   if (![[NSFileManager defaultManager] fileExistsAtPath:path])
      return NO;

   /* Nicht verwenden: Wiedergabe ist zu leise!
   if (![[AVAudioSession sharedInstance]
```

```
        setCategory:AVAudioSessionCategoryPlayAndRecord error:&error])
    {
        NSLog(@"Error %@", [error localizedDescription]);
        return NO;
    }
    */

    // Erfasst Unterbrechungen über einen Callback
    AudioSessionInitialize(NULL, NULL,
        interruptionListenerCallback, self);
    AudioSessionSetActive(true);
    UInt32 sessionCategory = kAudioSessionCategory_MediaPlayback;
    AudioSessionSetProperty( kAudioSessionProperty_AudioCategory,
        sizeof(sessionCategory), &sessionCategory);

    // Initialisiert den Player
    self.player = [[AVAudioPlayer alloc] initWithContentsOfURL:
        [NSURL fileURLWithPath:path] error:&error];
    self.player.volume = 1.0f;
    self.player.delegate = self;
    if (!self.player)
    {
        NSLog(@"Error %@", [error localizedDescription]);
        return NO;
    }

    [self.player prepareToPlay];

    return YES;
}
```

▸ *Rezept 15.4: Audiowiedergabe mit voller Lautstärke auch bei gesperrtem Gerät*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.5 Audioaufnahme

Die Klasse `AVAudioRecorder` vereinfacht die Audiowiedergabe in Ihren Anwendungen. Sie bietet den gleichen API-Komfort wie `AVAudioPlayer` mit ähnlichen Eigenschaften für die Rückmeldung. Diese beiden Klassen erleichtern die Entwicklung vieler üblicher Audioaufgaben von Anwendungen.

Um eine Aufnahme zu beginnen, richten Sie zunächst eine `AVAudioSession` ein. Wenn Sie innerhalb einer Anwendung zwischen Wiedergabe und Aufnahme wechseln möchten, verwenden Sie eine Wiedergabe- und Aufnahmesitzung, ansonsten eine einfache Aufnahmesitzung (`AVAudioSessionCategoryRecord`). Nachdem Sie die Sitzung eingerichtet haben, rufen Sie deren Eigenschaft `inputIsAvailable` ab, die anzeigt, ob das jeweilige Gerät über ein Mikrofon verfügt.

```
- (BOOL) startAudioSession
{
    // Bereitet die Audiositzung vor
    NSError *error;
    self.session = [AVAudioSession sharedInstance];

    if (![self.session
        setCategory:AVAudioSessionCategoryPlayAndRecord
        error:&error])
    {
        NSLog(@"Error %@", [error localizedDescription]);
        return NO;
    }

    // Aktiviert die Sitzung
    if (![self.session setActive:YES error:&error])
    {
        NSLog(@"Error %@", [error localizedDescription]);
        return NO;
    }

    return self.session.inputIsAvailable;
}
```

Rezept 15.5 zeigt den Schritt, der nach dem Erstellen der Sitzung folgt. Es richtet den Recorder ein und stellt Methoden zum Anhalten, Wiederaufnehmen und Beenden der Wiedergabe bereit.

Um mit der Aufzeichnung zu beginnen, wird ein Dictionary aus Einstellungen erstellt und mit Schlüsseln und Werten gefüllt, die beschreiben, wie die Aufnahme erfasst werden soll. In diesem Beispiel wird ein Mono-Linear-PCM mit einer Sample-Rate von 8000 Abtastvorgänge pro Sekunde verwendet, was ziemlich wenig ist. Beachten Sie die folgenden Hinweise zum Anpassen der Formate. Leider gibt es zurzeit keine Richtlinien für Audioeinstellungen von Apple.

- Setzen Sie `AVNumberOfChannelsKey` für Monoaufnahmen auf 1, für Stereo auf 2.
- Für das iPhone gut geeignete Audioformate (`AVFormatIDKey`) sind `kAudioFormatLinearPCM` (sehr große Dateien) und `kAudioFormatAppleIMA4` (kompakte Dateien).
- Die gebräuchlichen Sample-Raten (`AVSampleRateKey`) betragen 8000, 11.025, 22.050 und 44.100 Abtastvorgänge pro Sekunde.
- Als Bit-Tiefe für den linearen PCM (`AVLinearPCMBitDepthKey`) verwenden Sie 16 oder 32 Bit.

Der Code weist einen neuen AV-Recorder zu und initialisiert ihn mit einem Datei-URL und dem Dictionary der Einstellungen. Anschließend werden auch die Delegierung des Recorders eingerichtet und die Pegelmessung aktiviert. Letztere funktioniert in `AVAudioRecorder`-Instanzen ebenso wie in *Rezept 15.3* für `AVAudioPlayer`-Instanzen. Bevor Sie den Durchschnitts- und den Spitzenwert des Aussteuerungspegels abrufen, müssen Sie erst die Audiometer aktualisieren.

In der folgenden Methode wird der aus *Rezept 15.1* bekannte XMAX-Trick verwendet, um einen angenäherten Dynamikbereich für die Anzeige in den Audiometern zu bestimmen. Passen Sie den XMAX-Wert an den tatsächlichen Dynamikbereich Ihrer Anwendung an.

```
- (void) updateMeters
{
   // Zeigt die aktuellen Aussteuerungspegel
   [self.recorder updateMeters];
   float avg = [self.recorder averagePowerForChannel:0];
   float peak = [self.recorder peakPowerForChannel:0];
   meter1.progress = (XMAX + avg) / XMAX;
   meter2.progress = (XMAX + peak) / XMAX;

   // Aktualisiert die Angabe der aktuellen Aufnahmeposition
   self.title = [NSString stringWithFormat:@"%@",
      [self formatTime:self.recorder.currentTime]];
}
```

Der Code bestimmt auch die aktuelle zeitliche Position in der Aufnahme. Wenn Sie die Aufnahme anhalten, bleibt die Position bestehen, bis Sie wieder fortfahren. Im Grunde genommen zeigt die Zeitangabe der Aufnahmeposition an, wie lang die Aufnahme gerade ist.

Wenn Sie zur Aufnahme bereit sind, verwenden Sie `prepareToRecord` und starten die Aufzeichnung mit `record`. Rufen Sie `pause` auf, um die Aufnahme zu unterbrechen, und setzen Sie sie mit einem weiteren Aufruf von `record` fort. Dabei wird die Aufzeichnung an der Stelle weitergeführt, an der Sie sie angehalten haben. Um die Aufnahme zu beenden, verwenden Sie `stop`, was einen Callback an `audioRecorderDidFinishRecording:successfully:` hervorruft. Darin können Sie auch die Oberfläche aufräumen und jegliche Aspekte der Aufnahme finalisieren.

```
- (void) stopRecording
{
   // Löst die Delegierungsmethode didFinishRecording aus
   [self.recorder stop];
}

- (void) continueRecording
{
   // Nimmt eine angehaltene Aufnahme wieder auf
   [self.recorder record];
   self.navigationItem.rightBarButtonItem =
      BARBUTTON(@"Done", @selector(stopRecording));
```

```
    self.navigationItem.leftBarButtonItem =
        SYSBARBUTTON(UIBarButtonSystemItemPause, self,
        @selector(pauseRecording));
}

- (void) pauseRecording
{
    // Hält eine laufende Aufnahme an
    [self.recorder pause];
    self.navigationItem.leftBarButtonItem =
        BARBUTTON(@"Continue", @selector(continueRecording));
    self.navigationItem.rightBarButtonItem = nil;
}

- (BOOL) record
{
    NSError *error;

    // Aufnahmeeinstellungen
    NSMutableDictionary *settings = [NSMutableDictionary dictionary];
    [settings setValue:
        [NSNumber numberWithInt:kAudioFormatLinearPCM]
        forKey:AVFormatIDKey];
    [settings setValue:
        [NSNumber numberWithFloat:8000.0]
        forKey:AVSampleRateKey];
    [settings setValue:
        [NSNumber numberWithInt: 1]
        forKey:AVNumberOfChannelsKey]; // Mono
    [settings setValue:
        [NSNumber numberWithInt:16]
        forKey:AVLinearPCMBitDepthKey];
    [settings setValue:
        [NSNumber numberWithBool:NO]
        forKey:AVLinearPCMIsBigEndianKey];
    [settings setValue:
        [NSNumber numberWithBool:NO]
        forKey:AVLinearPCMIsFloatKey];

    // Datei-URL
    NSURL *url = [NSURL fileURLWithPath:FILEPATH];

    // Erstellt den Recorder
    self.recorder = [[AVAudioRecorder alloc]
```

```
        initWithURL:url settings:settings error:&error];
    if (!self.recorder)
    {
        NSLog(@"Error %@", [error localizedDescription]);
        return NO;
    }

    // Initialisiert Delegierung, Pegelmessung usw.
    self.recorder.delegate = self;
    self.recorder.meteringEnabled = YES;
    meter1.progress = 0.0f;
    meter2.progress = 0.0f;
    self.title = @"0:00";

    if (![self.recorder prepareToRecord])
    {
        NSLog(@"Error Prepare to record failed");
        [ModalAlert say:@"Error while preparing recording"];
        return NO;
    }

    if (![self.recorder record])
    {
        NSLog(@"Error Record failed");
        [ModalAlert say:@"Error while attempting to record audio"];
    return NO;
    }

    // Richtet einen Timer zur Beobachtung der Pegel und der Aufnahme-
    // position ein
    timer = [NSTimer scheduledTimerWithTimeInterval:0.1f
        target:self selector:@selector(updateMeters)
        userInfo:nil repeats:YES];

    // Aktualisiert die Navigationsleiste
    self.navigationItem.rightBarButtonItem =
        BARBUTTON(@"Done", @selector(stopRecording));
    self.navigationItem.leftBarButtonItem =
        SYSBARBUTTON(UIBarButtonSystemItemPause, self,
        @selector(pauseRecording));

    return YES;
}
```

▶ *Rezept 15.5: Audioaufnahme mit* `AVAudioRecorder`

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.6 Audioaufnahme mit Audio-Queue

Außer der Klasse `AVAudioPlayer` können auch Audio-Queues die Wiedergabe und Aufnahme in Ihren Anwendungen handhaben. Bevor es die Klasse `AVAudioRecorder` gab, waren sie für die Aufzeichnung auch erforderlich. Bei der direkten Verwendung von Audio-Queues können Sie beobachten, was hinter den Kulissen der Klasse `AVAudioRecorder` geschieht.

Rezept 15.6 führt eine Audioaufzeichnung mit Audio-Queue durch und zeigt die C-Funktionen und -Callbacks, die dafür verwendet werden. Dieser Code stützt sich zum Großteil auf Beispielcode von Apple und veranschaulicht vor allem die Funktionen, die der Wrapper `AVAudioRecorder` kapselt.

Die Einstellungen in der Methode `setupAudioFormat:` von *Rezept 15.6* wurden getestet und funktionieren zuverlässig auf einem iPhone. Bei dem Versuch, die Tonqualität anzupassen, kann man die Parameter jedoch ziemlich schnell durcheinanderbringen. Wenn diese Parameter nicht richtig eingestellt sind, kann die Audio-Queue fehlschlagen, ohne dass Sie darüber ausführliche Meldungen erhalten. Im Internet finden Sie einen reichen Fundus an Beispielen für Einstellungen.

Hinweis

Möchten Sie noch eine andere Vorgehensweise für Audiofunktionen auf dem iPhone kennenlernen? Der iPhone-Entwickler Philipp Homann *http://www.mobile-dev.de/openal-auf-dem-iPhone.html* eine schöne Einführung in die Verwendung von OpenAL-Audio auf dem iPhone veröffentlicht. OpenAL bietet eine API für die Audiowiedergabe im virtuellen 3D-Raum für räumliche Soundeffekte.

```
#define SAMPLES_PER_SECOND 8000.0f

// Schreibt die aktuellen Pakete, wenn der Eingabepuffer voll ist
static void HandleInputBuffer (void *aqData,
   AudioQueueRef inAQ, AudioQueueBufferRef inBuffer,
   const AudioTimeStamp *inStartTime,
   UInt32 inNumPackets,
   const AudioStreamPacketDescription *inPacketDesc)
{
   RecordState *pAqData = (RecordState *) aqData;

   if (inNumPackets == 0 &&
```

```
        pAqData->dataFormat.mBytesPerPacket != 0)
        inNumPackets = inBuffer->mAudioDataByteSize /
            pAqData->dataFormat.mBytesPerPacket;

    if (AudioFileWritePackets(pAqData->audioFile, NO,
        inBuffer->mAudioDataByteSize, inPacketDesc,
        pAqData->currentPacket, &inNumPackets,
        inBuffer->mAudioData) == noErr)
    {
        pAqData->currentPacket += inNumPackets;
        if (pAqData->recording == 0) return;
        AudioQueueEnqueueBuffer (pAqData->queue, inBuffer,
            0, NULL);
    }
}

@implementation Recorder

// Legt Mono-AIFF geringer Qualität als Aufnahmeformat fest
- (void)setupAudioFormat:(AudioStreamBasicDescription*)format
{
    format->mSampleRate = SAMPLES_PER_SECOND;
    format->mFormatID = kAudioFormatLinearPCM;
    format->mFormatFlags = kLinearPCMFormatFlagIsBigEndian |
        kLinearPCMFormatFlagIsSignedInteger |
        kLinearPCMFormatFlagIsPacked;

    format->mChannelsPerFrame = 1; // Mono
    format->mBitsPerChannel = 16;
    format->mFramesPerPacket = 1;
    format->mBytesPerPacket = 2;
    format->mBytesPerFrame = 2;
    format->mReserved = 0;
}

// Beginnt die Aufnahme
- (BOOL) startRecording: (NSString *) filePath
{
    // Viele dieser Aufrufe entsprechen dem Vorgang bei AVAudioRecorder

    // Legt das Audioformat und den URL des Aufnahmeziels fest
    [self setupAudioFormat:&recordState.dataFormat];
    CFURLRef fileURL = CFURLCreateFromFileSystemRepresentation(
        NULL, (const UInt8 *) [filePath UTF8String],
        [filePath length], NO);
    recordState.currentPacket = 0;
```

```
// Initialisiert die Warteschlange mit dem gewählten Format
OSStatus status;
status = AudioQueueNewInput(&recordState.dataFormat,
   HandleInputBuffer, &recordState,
   CFRunLoopGetCurrent(),kCFRunLoopCommonModes, 0,
   &recordState.queue);
if (status) {
   CFRelease(fileURL);
   printf("Could not establish new queue\n");
   return NO;
}

// Erstellt die Ausgabedatei
status = AudioFileCreateWithURL(fileURL,
   kAudioFileAIFFType, &recordState.dataFormat,
   kAudioFileFlags_EraseFile, &recordState.audioFile);
if (status)
{
   printf("Could not create file to record audio\n");
   return NO;
}

// Richtet die Puffer ein
DeriveBufferSize(recordState.queue, recordState.dataFormat,
   0.5, &recordState.bufferByteSize);
for(int i = 0; i < NUM_BUFFERS; i++)
{
   status = AudioQueueAllocateBuffer(recordState.queue,
      recordState.bufferByteSize, &recordState.buffers[i]);
   if (status) {
      printf("Error allocating buffer %d\n", i);
      return NO;
   }
   status = AudioQueueEnqueueBuffer(recordState.queue,
      recordState.buffers[i], 0, NULL);
   if (status) {
      printf("Error enqueuing buffer %d\n", i);
      return NO;
   }
}

// Aktiviert die Pegelmessung
UInt32 enableMetering = YES;
status = AudioQueueSetProperty(recordState.queue,
   kAudioQueueProperty_EnableLevelMetering,
```

```
            &enableMetering,sizeof(enableMetering));
        if (status)
        {
            printf("Could not enable metering\n");
            return NO;
        }

        // Startet die Aufzeichnung
        status = AudioQueueStart(recordState.queue, NULL);
        if (status)
        {
            printf("Could not start Audio Queue\n");
            return NO;
        }

        recordState.currentPacket = 0;
        recordState.recording = YES;
        return YES;
}

// Gibt den durchschnittlichen Aussteuerungspegel zurück
- (float) averagePower
{
        AudioQueueLevelMeterState state[1];
        UInt32 statesize = sizeof(state);
        OSStatus status;
        status = AudioQueueGetProperty(recordState.queue,
            kAudioQueueProperty_CurrentLevelMeter, &state, &statesize);
        if (status)
        {
            printf("Error retrieving meter data\n");
            return 0.0f;
        }
        return state[0].mAveragePower;
}

// Gibt den Spitzenwert des Aussteuerungspegels zurück
- (float) peakPower
{
        AudioQueueLevelMeterState state[1];
        UInt32 statesize = sizeof(state);
        OSStatus status;
        status = AudioQueueGetProperty(recordState.queue,
            kAudioQueueProperty_CurrentLevelMeter, &state, &statesize);
        if (status)
        {
```

```
        printf("Error retrieving meter data\n");
        return 0.0f;
    }
    return state[0].mPeakPower;
}

// Vor der vollständigen Entleerung des Puffers gibt es im Allgemeinen
// eine Verzögerung von einer Sekunde
- (void) reallyStopRecording
{
    AudioQueueFlush(recordState.queue);
    AudioQueueStop(recordState.queue, NO);
    recordState.recording = NO;

    for(int i = 0; i < NUM_BUFFERS; i++)
    {
        AudioQueueFreeBuffer(recordState.queue,
            recordState.buffers[i]);
    }
    AudioQueueDispose(recordState.queue, YES);
    AudioFileClose(recordState.audioFile);
}

// Beendet die Aufnahme nach einer Wartezeit von einer Sekunde
- (void) stopRecording
{
    [self performSelector:@selector(reallyStopRecording)
        withObject:NULL afterDelay:1.0f];
}

// Hält an, nachdem allen Puffern Gelegenheit zum Aufholen geboten wurde
- (void) reallyPauseRecording
{
    if (!recordState.queue) {printf("Nothing to pause\n"); return;}
    OSStatus status = AudioQueuePause(recordState.queue);
    if (status) {printf("Error pausing audio queue\n"); return;}
}

// Hält die Aufnahme nach einer Wartezeit von einer halben Sekunde an
- (void) pause
{
    [self performSelector:@selector(reallyPauseRecording)
        withObject:NULL afterDelay:0.5f];
}
```

```
// Nimmt die Aufnahme von einer angehaltenen Warteschlange wieder auf
- (BOOL) resume
{
   if (!recordState.queue)
   {
      printf("Nothing to resume\n");
      return NO;
   }

   OSStatus status = AudioQueueStart(recordState.queue, NULL);
   if (status)
   {
      printf("Error restarting audio queue\n");
      return NO:
   }

   return YES;
}

// Gibt die aktuelle Aufnahmedauer zurück
- (float) currentTime
{
   AudioTimeStamp outTimeStamp;
   OSStatus status = AudioQueueGetCurrentTime (
      recordState.queue, NULL, &outTimeStamp, NULL);
   if (status)
   {
      printf("Error: Could not retrieve current time\n");
      return 0.0f;
   }

   // Sample-Rate von 8000 Abtastvorgängen pro Sekunde
   return outTimeStamp.mSampleTime / SAMPLES_PER_SECOND;
}

// Meldet, ob die Aufnahme aktiv ist
- (BOOL) isRecording
{
   return recordState.recording;
}
@end
```

▸ *Rezept 15.6: Aufnahme mit Audio-Queue: Die Implementierung* `Recorder.m`

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.7 Rezept: Videowiedergabe mit dem Media Player

Die Klasse `MPMoviePlayerController` vereinfacht die Videodarstellung in Ihren Anwendungen. Sie gehört zum Framework MediaPlayer, gehorcht aber ihren eigenen Regeln. Sie bringen sie nicht auf den Navigationsstack, und Sie rufen sie auch nicht modal auf, sondern Sie erstellen sie und weisen sie an, mit der Wiedergabe zu beginnen. Daraufhin übernimmt sie die Kontrolle über den Bildschirm und zeigt die Steuerelemente aus *Abbildung 15.2* an.

Damit Ihre Anwendung die Kontrolle zurückerhalten kann, muss sie die Benachrichtigung `MPMoviePlayerPlaybackDidFinishNotification` abonnieren, die in zwei Fällen gesendet wird, nämlich wenn die Wiedergabe am Ende des Films aufhört und wenn der Benutzer auf **Done** (**Fertig**) tippt. Die Verwendung dieser Klasse könnte nicht einfacher sein. Weisen Sie eine neue Instanz der Klasse `MPMoviePlayer` zu, initialisieren Sie sie mit einem URL, und weisen Sie sie mit `play` zur Wiedergabe an. Ist die Wiedergabe beendet, geben Sie den Player frei.

Rezept 15.7 umfasst zwei Methoden, von denen die eine die Wiedergabe startet und die andere nach Beendigung der Wiedergabe aufräumt. Mit diesem Code können Sie ohne jegliche Änderungen sowohl Video als auch einfach nur Audio abspielen. Sie müssen nur den URL zu einer Quelldatei eines möglichen Typs angeben. Verwendbar sind u. a. die Dateitypen MOV, MP4, MPV, M4V und 3GP sowie MP3, AIFF und M4A.

▸ *Abbildung 15.2: Die bildschirmfüllende Oberfläche des Media Players erlaubt dem Benutzer eine umfassende Steuerung der Videowiedergabe. Hierbei handelt es sich um die gleiche Videooberfläche, die auch in den Anwendungen iPod und YouTube verwendet wird. (Dieser Screenshot zeigt eine Szene aus einem lizenzfreien Betty-Boop-Zeichentrickfilm mit freundlicher Genehmigung des Internet Archives unter archive.org.)*

In diesem Code wird über einen externen URL eine Quelle verwendet, die sich nicht auf dem Gerät selbst befindet. Bedenken Sie, dass solche Verbindungen langsam sein können (oder möglicherweise gar nicht zustandekommen), weshalb Sie für Verzögerungen während der Wiedergabe vorsorgen müssen. Lokale Datei-URLs (wie Sie sie in *Rezept 15.1* gesehen haben) ermöglichen eine zuverlässigere Wiedergabe von Video- und Audioquellen.

Leidet bietet das `MPMoviePlayer`-Objekt nur eine eingeschränkte API-Steuerung. Wenn Sie einen Film in einer Schleife wiedergeben möchten, können Sie ihn erneut abspielen, nachdem Sie die Benachrichtigung über den Abschluss der Wiedergabe erhalten haben:

```
-(void)myMovieFinishedCallback:(NSNotification*)aNotification
{
   MPMoviePlayerController* theMovie=[aNotification object];
   [theMovie play];
}
```

Dadurch entsteht ein sichtbarer Sprung zwischen dem Ende der ersten und dem Beginn der folgenden Wiedergabe. Dafür können Sie Vorsorge treffen, indem Sie z. B. die Hauptbenutzeroberfläche mit einer schwarzen Ansicht überlagern. Möglicherweise müssen die Benutzer erneut auf die Wiedergabe-Schaltfläche tippen. Die Interaktion der Benutzer mit dem Film können Sie einschränken, indem Sie `movieControlMode` im Player so einrichten, dass gar keine Interaktion oder nur eine Lautstärkeänderung möglich ist. In allen diesen Modi ist die Wiedergabe-Schaltfläche ausgeblendet.

Hinweis

Rezept 13.7 zeigt, wie Sie eine Filmdatei von einem Server im Netzwerk vollständig (und asynchron) herunterladen, bevor Sie sie abspielen.

```
// Nicht lokale Quelle "Betty Boop Cinderella" unter Archive.org
#define PATHSTRING \
   @"http://www.archive.org/download/bb_poor_cinderella/\
   bb_poor_cinderella_512kb.mp4"

@interface TestBedViewController : UIViewController
@end

@implementation TestBedViewController
-(void)myMovieFinishedCallback:(NSNotification*)aNotification
{
   // Räumt nach dem Abspielen des Films auf
   MPMoviePlayerController* theMovie=[aNotification object];
   [[NSNotificationCenter defaultCenter] removeObserver:self
      name:MPMoviePlayerPlaybackDidFinishNotification
      object:theMovie];
   [theMovie release];
```

```
    self.navigationItem.rightBarButtonItem =
        BARBUTTON(@"Play", @selector(play:));
    self.title = nil;
}

- (void) play: (UIBarButtonItem *) bbi
{
    // Blendet die Wiedergabe-Schaltfläche aus und zeigt den Player
    self.navigationItem.rightBarButtonItem = nil;
    self.title = @"Contacting Server";
    MPMoviePlayerController* theMovie=[[MPMoviePlayerController alloc]
        initWithContentURL:[NSURL URLWithString:PATHSTRING]];
    [[NSNotificationCenter defaultCenter] addObserver:self
        selector:@selector(myMovieFinishedCallback:)
        name:MPMoviePlayerPlaybackDidFinishNotification
        object:theMovie];
    [theMovie play];
}
```

▸ *Rezept 15.7: Videos mit* `MPMoviePlayer` *abspielen*

Hinweis

Beachten Sie, dass das iPad zwar reine iPhone-Applikation ausführen kann, aber derzeit bei der Verwendung des `MPMoviePlayerController` nicht in der Lage ist,Videos darzustellen. Es spielt nur deren Ton ab.

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.8 Rezept: Videoaufnahme

Bevor Sie Video aufzeichnen können, müssen Sie herausfinden, ob das Gerät überhaupt dafür ausgestattet ist. Die Suche nach einer eingebauten Kamera, wie sie auch iPhones der ersten Generation und des Modells 3G haben, reicht dabei nicht aus, denn nur das 3GS und jüngere Modelle haben die Fähigkeit zur Videoaufnahme. Daher müssen Sie zwei Überprüfungen durchführen: Erstens müssen Sie feststellen, ob eine Kamera vorhanden ist, und zweitens, ob unter den möglichen Aufnahmetypen auch die Videoaufzeichnung ist. Dazu verwenden Sie die folgende Methode:

```
- (BOOL) videoRecordingAvailable
{
   // Der Quelltyp muss verfügbar sein
   if (![UIImagePickerController isSourceTypeAvailable:
      UIImagePickerControllerSourceTypeCamera])
   return NO;

   // Unter den Medientypen muss sich movie befinden
   NSArray *mediaTypes = [UIImagePickerController
      availableMediaTypesForSourceType:
      UIImagePickerControllerSourceTypeCamera];

   return [mediaTypes containsObject:@"public.movie"];
}
```

Diese Methode sucht mit einem konstanten String nach dem Medientyp `public.movie`. In irgendeiner späteren Version des iPhone SKD werden Sie stattdessen wahrscheinlich nach der Konstante `kUTType Movie` suchen. Diese Typen werden im öffentlichen Framework *MobileCoreServices* in `UTCoreTypes.h` definiert.

Wie *Rezept 15.8* zeigt, ähnelt die Videoaufzeichnung der Aufnahme von Einzelbildern mit der eingebauten Kamera. Hier wie dort weisen Sie einen neuen Bild-Picker zu und initialisieren ihn, richten seine Delegierung ein, wählen aus, ob eine Bearbeitung möglich sein soll, und stellen ihn dar.

Im Gegensatz zur Aufnahme von statischen Bildern müssen Sie hier jedoch drei wichtige Eigenschaften festlegen. Die erste davon ist die Bildqualität. In *Rezept 15.8* wird eine mittlere Qualität verwendet, Sie können aber auch eine hohe oder niedrige wählen. Zweitens geben Sie die maximale Filmdauer in Sekunden an. *Rezept 15.8* erlaubt dem Benutzer, Material bis zu einer Länge von 30 Sekunden aufzunehmen. (Unter Verwendung des Video-Bild-Pickers kann der Maximalwert bis zu zehn Minuten betragen.) Schließlich müssen Sie auch noch als Medientyp-Array für den Picker eine Liste mit nur einem einzigen Objekt, nämlich dem Typ `movie`, festlegen. Zusätzlich können Sie noch den Typ `public.image` hinzufügen, um dem Benutzer die Gelegenheit zu geben, zwischen der Aufnahme von statischen Bildern und Filmen umzuschalten, wie es in *Abbildung 15.3* zu sehen ist.

Wenn Sie die Eigenschaft `allowsEditing` auf `YES` setzen, ermöglichen Sie es dem Benutzer, die Clips vor der Speicherung und sonstigen Weiterverarbeitung der Daten mithilfe des eingebauten Video-Editors zu beschneiden. Unabhängig davon, ob Sie eine solche Bearbeitung erlauben oder nicht, informieren die standardmäßigen Callbacks des Bild-Pickers die Delegierung, wenn der Benutzer die Videoaufnahme abgeschlossen hat.

Bei der Arbeit mit Video rufen Sie einen URL anstelle der tatsächlichen Daten ab. Wie *Rezept 15.8* zeigt, müssen Sie prüfen, ob der Film mit dem eingebauten Album kompatibel ist, bevor Sie ihn speichern. Wenn ja, können Sie die Speicherung mit der Funktion `UISaveVideoAtPathToSavedPhotosAlbum()` durchführen. Inkompatible Clips sind z. B. MPEG-4-Videos, die Sie vom Internet Archive (*http://archive.org*) heruntergeladen haben. Diese Dateien können Sie nicht dem Fotoalbum hinzufügen.

► *Abbildung 15.3: Als Entwickler können Sie einen oder mehrere Medientypen für den Bild-Picker-Controller angeben. Der linke Screenshot zeigt eine reine Videoaufnahme, der rechte einen Picker, der sowohl die Standbild- als auch die Videoaufnahme ermöglicht.*

Hinweis

Beachten Sie, dass die Eigenschaften und Methoden zum Aufnehmen und die Auswahl von Videos mithilfe des `UIImagePickerController` erst seit SDK 3.1 zur Verfügung stehen und dass Sie dementsprechend auch dieses SDK zum Ausführen der Rezepte benötigen.

```
- (void)video:(NSString *)videoPath
   didFinishSavingWithError:(NSError *)error
   contextInfo:(void *)contextInfo
{
   // Prüft auf Fehler beim Speichern
   if (!error)
      self.title = @"Saved!";
   else
      CFShow([error localizedDescription]);
}

- (void) imagePickerControllerDidCancel:
   (UIImagePickerController *) picker
{
```

```
    // Der Picker wird entfernt, da der Benutzer auf Cancel getippt hat
    [self dismissModalViewControllerAnimated:YES];
    [picker release];
}

- (void)imagePickerController:(UIImagePickerController *)picker
    didFinishPickingMediaWithInfo:(NSDictionary *)info
{
    // Ruft den Video-URL ab
    NSURL *url = [info objectForKey:UIImagePickerControllerMediaURL];

    // Prüft, ob das Videoformat mit dem Album kompatibel ist
    BOOL compatible =
        UIVideoAtPathIsCompatibleWithSavedPhotosAlbum([url path]);

    // Speichert
    if (compatible)
        UISaveVideoAtPathToSavedPhotosAlbum([url path], self,
            @selector(video:didFinishSavingWithError:contextInfo:),
            NULL);

    [self dismissModalViewControllerAnimated:YES];
    [picker release];
}

- (void) recordVideo: (id) sender
{
    // Zeigt den Videorecorder an
        UIImagePickerController *ipc =
    [[UIImagePickerController alloc] init];
    ipc.sourceType = UIImagePickerControllerSourceTypeCamera;
    ipc.delegate = self;
    ipc.allowsEditing = YES;
    ipc.videoQuality = UIImagePickerControllerQualityTypeMedium;
    ipc.videoMaximumDuration = 30.0f; // 30 Sekunden
    ipc.mediaTypes = [NSArray arrayWithObject:@"public.movie"];
    [self presentModalViewController:ipc animated:YES];
}
```

▸ *Rezept 15.8: Videoaufzeichnung mit* `UIImagePickerController`

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.9 Rezept: Filme auswählen und bearbeiten

Die in *Rezept 15.8* verwendete Eigenschaft `mediaTypes` hat nicht nur Einfluss auf die Medienaufnahme, sondern auch auf die Medienauswahl. Um einen Picker anzufordern, der nur Videos anzeigt, erstellen Sie einen Fotobibliotheks-Picker und geben im Medienarray nur den String `public.movie` an. Die folgende Methode erstellt einen solchen reinen Video-Picker:

```
- (void) pickVideo: (id) sender
{
    // Zeigt einen reinen Video-Picker an
    UIImagePickerController *ipc =
        [[UIImagePickerController alloc] init];
    ipc.sourceType = UIImagePickerControllerSourceTypePhotoLibrary;
    ipc.delegate = self;
    ipc.allowsEditing = NO;
    ipc.videoQuality = UIImagePickerControllerQualityTypeMedium;
    ipc.videoMaximumDuration = 30.0f; // 30 seconds
    ipc.mediaTypes = [NSArray arrayWithObject:@"public.movie"];
    [self presentModalViewController:ipc animated:YES];
}
```

Um einen bereits vorhandenen Film zu bearbeiten, prüfen Sie zunächst wie in *Rezept 15.9*, ob er sich überhaupt bearbeiten lässt. Dazu rufen Sie die Methode `canEditVideoAtPath:` der Klasse `UIVideoEditorController` auf. Sie gibt einen booleschen Wert zurück, der anzeigt, ob der Film mit dem Editor-Controller kompatibel ist.

Wenn ja, können Sie einen neuen Editor zuweisen, seine Eigenschaften `delegate` und `videoPath` festlegen und ihn anzeigen. Der Editor verwendet eine Reihe von Delegierungs-Callbacks, die denen der Klasse `UIImagePickerController` ähneln, aber nicht mit ihnen identisch sind. Bei diesen Callbacks handelt es sich um die Methoden für den Erfolg und den Fehlschlag des Vorgangs sowie den Abbruch durch den Benutzer.

Wenn ein Benutzer die Bearbeitung beendet hat, speichert der Controller den Film in einem temporären Pfad im Ordner `tmp` der Anwendungs-Sandbox und ruft `videoEditorController:didSaveEditedVideoToPath:` auf. Wenn Sie mit den Daten nichts weiter anstellen, werden sie bei dem nächsten Neustart des iPhones gelöscht. Sie können sie jedoch entweder lokal im Dokumentenordner der Sandbox oder in einem gemeinsam genutzten iPhone-Fotoalbum speichern. Folgen Sie dazu dem Beispiel von *Rezept 15.8*. Unabhängig davon, welcher Callback gesendet wird, liegt es in Ihrer Verantwortung, den Editor zu entfernen und freizugeben.

```
- (void)videoEditorController:(UIVideoEditorController *)editor
   didSaveEditedVideoToPath:(NSString *)editedVideoPath
{
   CFShow(editedVideoPath);

   // Hier kann eine Speicherung erfolgen. Die Daten sind noch *nicht*
   // im Fotoalbum gespeichert!
   [self dismissModalViewControllerAnimated:YES];
   [editor release];
}

- (void)videoEditorControllerDidCancel:
   (UIVideoEditorController *)editor
{
   // Blendet den Picker aus, wenn der Benutzer den Vorgang abbricht
   [self dismissModalViewControllerAnimated:YES];
   [editor release];
}

- (void)videoEditorController:(UIVideoEditorController *)editor
   didFailWithError:(NSError *)error
{
   // Reagiert auf einen Fehlschlag des Editors
   [self dismissModalViewControllerAnimated:YES];
   [editor release];

   NSLog(@"Fail! %@", [error localizedDescription]);
}

- (void) doEdit
{
   // Überprüft, ob eine Bearbeitung möglich ist
   if (![UIVideoEditorController canEditVideoAtPath:self.vpath])
   {
      self.title = @"Cannot Edit Video";
      printf("Cannot edit vid at path\n");
      return;
   }

   // Wenn ja, wird der Editor angezeigt
   UIVideoEditorController *vec =
      [[UIVideoEditorController alloc] init];
   vec.videoPath = self.vpath;
   vec.delegate = self;
   [self presentModalViewController:vec animated:YES];
}
```

► *Rezept 15.9: Den Video-Editor-Controller verwenden*

Hinweis

Der iPod touch besitzt zwar die Fähigkeit, synchronisierte Videos darzustellen, unterstützt aber zumindest im Moment noch nicht den Medientyp `public.movie` und ist daher nicht in der Lage, dem Anwender die Möglichkeit der Videoeditierung zu bieten. Sie sollten also als Entwickler immer vorab eine Überprüfung auf diesen Typ durchführen.

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.10 Rezept: Audioauswahl mit dem MPMediaPickerController

Die Klasse `MPMediaPickerController` bietet das Gegenstück zu den Bildauswahlmöglichkeiten von `UIImagePickerController`. Damit können die Benutzer Elemente wie Musikstücke, Podcasts und Hörbücher aus ihrer Musikbibliothek auswählen. Die standardmäßige Oberfläche im iPod-Stil ermöglicht es den Benutzern, Wiedergabelisten, Listen von Künstlern und Songs, Alben usw. zu durchsuchen.

Um diese Klasse nutzen zu können, müssen Sie einen neuen Picker auswählen und mit dem gewünschten Medientyp initialisieren, wobei `MPMediaTypeMusic`, `MPMediaTypePodcast`, `MPMediaTypeAudioBook`, `MPMediaTypeAnyAudio` und `MPMediaTypeAny` zur Verfügung stehen. Es handelt sich dabei um Flags, die Sie nicht über eine ODER-Verknüpfung zu einer Maske kombinieren können.

```
MPMediaPickerController *mpc = [[MPMediaPickerController alloc]
   initWithMediaTypes:MPMediaTypeAny];
mpc.delegate = self;
mpc.prompt = @"Please select an item";
mpc.allowsPickingMultipleItems = NO;
[self presentModalViewController:mpc animated:YES];
```

Legen Sie als Nächstes eine Delegierung und optional den Text für eine Benutzeraufforderung an, die oben im Medien-Picker erscheinen soll (siehe *Abbildung 15.4*). Wenn Sie die Auswahl mehrerer Elemente ermöglichen, wird die Schaltfläche **Cancel (Abbrechen)** des standardmäßigen Pickers durch **Done (Fertig)** ersetzt. Normalerweise endet das Dialogfeld, wenn der Benutzer auf ein Element tippt, doch bei der Mehrfachauswahl können die Benutzer mehrere Elemente markieren, bis sie auf **Done** tippen. Die ausgewählten Elemente werden aktualisiert und mit grauer Beschriftung dargestellt.

▶ *Abbildung 15.4: In diesem Medien-Picker mit Mehrfachauswahl erscheinen die bereits ausgewählten Elemente grau (»ABC« und »Above It All«). Wenn die Benutzer fertig sind, tippen sie auf* **Done**. *Bei der normalen Picker-Auswahl gibt es anstelle von* **Done** *die Schaltfläche* **Cancel (Abbrechen)**, *mit der Benutzer den Vorgang abbrechen können, ohne ein Element auszuwählen. Eine optionale Benutzeraufforderung (hier »Please Select an Item«) erscheint oberhalb der normalen Picker-Elemente.*

Der Delegierungs-Callback `mediaPicker:didPickMediaItems:` kümmert sich darum, die Benutzerauswahl abzuschließen. Die `MPMediaItemCollection`-Instanz, die als Parameter übergeben wird, kann durch Zugriff auf ihre Elemente (`items`) aufgelistet werden. Jedes Element ist ein Member der Klasse `MPMediaItem`, dessen Eigenschaften Sie abfragen können (siehe *Rezept 15.10*). In *Rezept 15.10* wird ein Medien-Picker zur Auswahl mehrerer Musikstücke erstellt, der die vom Benutzer gewählten Elemente nach Künstler und Titel protokolliert.

Tabelle 15.1 führt die verfügbaren Eigenschaften von Medienelementen mit ihrem Rückgabetyp und der Angabe auf, ob sie zur Konstruktion eines Prädikats aus Medieneigenschaften herangezogen werden können. Wie Sie Abfragen erstellen und dabei Prädikate nutzen, erfahren Sie in *Rezept 15.11*.

Eigenschaft	Typ	Verwendung in Filtern möglich
Allgemeine Medienelemente		
`MPMediaItemPropertyPersistentID`	`uint64_t`	Ja
`MPMediaItemPropertyMediaType`	`NSNumber Integer`	Ja
`MPMediaItemPropertyTitle`	`NSString`	Ja
`MPMediaItemPropertyAlbumTitle`	`NSString`	Ja
`MPMediaItemPropertyArtist`	`NSString`	Ja

MPMediaItemPropertyAlbumArtist	NSString	Ja
MPMediaItemPropertyGenre	NSString	Ja
MPMediaItemPropertyComposer	NSString	Ja
MPMediaItemPropertyPlaybackDuration	NSNumber NSTimeInterval	
MPMediaItemPropertyAlbumTrackNumber	NSNumber Integer	
MPMediaItemPropertyAlbumTrackCount	NSNumber Integer	
MPMediaItemPropertyDiscNumber	NSNumber Integer	
MPMediaItemPropertyDiscCount	NSNumber Integer	
MPMediaItemPropertyArtwork	MPMediaItemArtwork	
MPMediaItemPropertyLyrics	NSString	
MPMediaItemPropertyIsCompilation	NSNumber Boolean	Ja
Podcast-Elemente		
MPMediaItemPropertyPodcastTitle	NSString	
Benutzerdefinierte Elemente		
MPMediaItemPropertyPlayCount	NSNumber Integer	
MPMediaItemPropertySkipCount	NSNumber Integer	
MPMediaItemPropertyRating	NSNumber Integer zwischen 0 und 5	
MPMediaItemPropertyLastPlayedDate	NSDate	

▸ *Tabelle 15.1: Medienelementeigenschaften*

```
@implementation TestBedViewController
- (void)mediaPicker: (MPMediaPickerController *)mediaPicker
   didPickMediaItems:(MPMediaItemCollection *)mediaItemCollection
{
   // Zeigt die ausgewählten Elemente
   for (MPMediaItem *item in [mediaItemCollection items])
      NSLog(@"[%@] %@",
         [item valueForProperty:MPMediaItemPropertyArtist],
         [item valueForProperty:MPMediaItemPropertyTitle]);

   [self dismissModalViewControllerAnimated:YES];
   [mediaPicker release];
}

- (void)mediaPickerDidCancel:(MPMediaPickerController *)mediaPicker
{
   // Reagiert auf Abbruch durch den Benutzer
   [self dismissModalViewControllerAnimated:YES];
   [mediaPicker release];
}

- (void) action: (UIBarButtonItem *) bbi
{
   // Zeigt den Picker an und erlaubt die Mehrfachauswahl
   MPMediaPickerController *mpc = [[MPMediaPickerController alloc]
      initWithMediaTypes:MPMediaTypeMusic];
   mpc.delegate = self;
   mpc.prompt = @"Please select an item";
   mpc.allowsPickingMultipleItems = YES;
   [self presentModalViewController:mpc animated:YES];
}

- (void) viewDidLoad
{
   self.navigationController.navigationBar.tintColor =
      COOKBOOK_PURPLE_COLOR;
   self.navigationItem.rightBarButtonItem =
      BARBUTTON(@"Action", @selector(action:));
}
@end
```

▶ *Rezept 15.10: Musikelemente aus der iPod-Bibliothek auswählen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.11 Medienabfragen erstellen

Mit Medienabfragen können Sie die Inhalte einer iPod-Bibliothek filtern und den Suchbereich einschränken. *Tabelle 15.2* zeigt die neun Klassenmethoden von `MPMediaQuery` für vordefinierte Suchvorgänge, wobei die angegebenen Gruppierungstypen die Gliederung der zurückgegebenen Daten festlegen. Die einzelnen Sammlungen werden dadurch jeweils nach Album, nach Künstler, nach Hörbuch usw. geordnet.

Klassenmethode	Global?	Filter	Gruppierungstyp
`albumsQuery`	Nein	`MPMediaTypeMusic`	`MPMediaGroupingAlbum`
`artistsQuery`	Nein	`MPMediaTypeMusic`	`MPMediaGroupingArtist`
`audiobooksQuery`	Nein	`MPMediaTypeAudioBook`	`MPMediaGroupingTitle`
`compilations➥Query`	Nein	`MPMediaTypeAny | MPMediaItemProperty➥IsCompilation`	`MPMediaGroupingAlbum`
`composersQuery`	Ja	`MPMediaTypeAny`	`MPMediaGroupingComposer`
`genresQuery`	Ja	`MPMediaTypeAny`	`MPMediaGroupingGenre`
`playlistsQuery`	Ja	`MPMediaTypeAny`	`MPMediaGroupingPlaylist`
`podcastsQuery`	Nein	`MPMediaTypePodcast`	`MPMediaGroupingPodcastTitle`
`songsQuery`	Nein	`MPMediaTypeMusic`	`MPMediaGroupingTitle`

▸ *Tabelle 15.2: Abfragetypen*

Dieser Ansatz bildet die Funktionsweise von iTunes auf dem Desktop nach. In iTunes wählen Sie die Spalte aus, nach der Sie die Ergebnisse ordnen möchten, aber um eine Suche durchzuführen, geben Sie Text in das Suchfeld der Anwendung ein.

15.11.1 Eine Abfrage erstellen

Die Anzahl der Alben in einer Bibliothek ermitteln Sie mit einer Albumabfrage. Das folgende Codefragment erstellt die entsprechende Abfrage und ruft ein Array ab, dessen Elemente für die einzelnen Alben stehen. Die Albumelemente sind wiederum Sammlungen individueller Medienelemente. Eine Sammlung kann einen einzigen Titel umfassen, aber auch mehrere.

```
MPMediaQuery *query = [MPMediaQuery albumsQuery];
NSArray *collections = query.collections;
NSLog(@"You have %d albums in your library\n", collections.count);
```

Viele iPhone-Benutzer haben umfassende Mediensammlungen mit oft Hunderten oder Tausenden von Alben, ganz zu schweigen von einzelnen Titeln. Eine einfache Abfrage wie diese kann mehrere Sekunden in Anspruch nehmen und eine Datenstruktur zurückgeben, die die gesamte Bibliothek darstellt.

Eine Suche mit einem anderen Abfragetyp gibt Sammlungen zurück, die nach Typ geordnet sind. Mit einer ähnlichen Vorgehensweise können Sie die Anzahl der Künstler, Songs, Komponisten usw. abrufen.

15.11.2 Prädikate verwenden

Ein Prädikat aus Medieneigenschaften filtert wirkungsvoll die von einer Abfrage zurückgegebenen Elemente. Nehmen wir z. B. an, dass Sie nur die Songs finden möchten, in dessen Titel das Wort »road« vorkommt. Der folgende Code erstellt eine neue Songabfrage mit einem Filterprädikat, um nach diesem Begriff zu suchen. Das Prädikat wird aus einem Wert (dem Suchbegriff), einer Eigenschaft (Suche nach dem Songtitel) und einem Vergleichstyp (in diesem Fall `contains`, also »enthält«) zusammengesetzt. Für die Suche nach genauen Übereinstimmungen verwenden Sie `MPMediaPredicateComparisonEqualTo` und für Teilstring-Übereinstimmungen `MPMediaPredicateComparisonContains`.

```
MPMediaQuery *query = [MPMediaQuery songsQuery];

// Konstruiert ein Prädikat für den Titelvergleich
   MPMediaPropertyPredicate *mpp = [MPMediaPropertyPredicate
   predicateWithValue:@"road"
   forProperty:MPMediaItemPropertyTitle
   comparisonType:MPMediaPredicateComparisonContains];
[query addFilterPredicate:mpp];

// Ruft die Sammlungen ab
NSArray *collections = query.collections;
NSLog(@"You have %d matching tracks in your library\n",
   collections.count);

// Iteriert durch die einzelnen Elemente und protokolliert dabei Song und
// Künstler
```

```
for (MPMediaItemCollection *collection in collections)
{
   for (MPMediaItem *item in [collection items])
   {
      NSString *song = [item valueForProperty:
         MPMediaItemPropertyTitle];
      NSString *artist = [item valueForProperty:
         MPMediaItemPropertyArtist];
      NSLog(@"%@, %@", song, artist);
   }
}
```

Hinweis

Für den Fall, dass Sie statt Prädikaten aus Medieneigenschaften lieber reguläre Prädikate verwenden möchten, habe ich die Eigenschaftskategorie `MPMediaItem` erstellt (*http://github.com/erica/MPMediaItem-Properties*). Damit können Sie normale `NSPredicate`-Abfragen an Sammlungen durchführen, wie sie z. B. ein Picker zur Mehrfachauswahl zurückgibt.

15.11.3 Geschwindigkeitsprobleme

Das oben gezeigte Codefragment wird nur langsam ausgeführt, denn der Abruf der Eigenschaftswerte erfordert mehr Zeit, als Sie erwarten mögen, nämlich bis zu einer Sekunde bei einer umfangreichen Bibliothek auf einem modernen iPhone 3GS oder einem iPod touch der dritten Generation, den zurzeit schnellsten iPhone-Modellen. Es gibt keinen offensichtlichen Grund dafür, dass eine Eigenschaftsabfrage so viel Zeit in Anspruch nimmt oder warum der Abruf der Eigenschaften durch die Größe der Bibliothek beeinträchtigt wird.

Wegen dieser Verzögerungen eignen sich solche Arten von Abfragen nicht gut als Datenquelle für eine Tabellenansicht. *Rezept 15.11* führt dieses Problem vor. Dazu wird auf der Grundlage einer Suche des Benutzers eine Liste von Songtiteln erstellt. Selbst bei der Verwendung eines Caches, um bereits genutzte Zellentitel wiederzuverwenden, wird deutlich, dass die Konstruktion eigener Picker zurzeit aufgrund der Geschwindigkeitsprobleme nicht praktikabel ist.

Solange Apple den Datenabruf nicht dramatisch beschleunigt, sind Sie besser dran, wenn Sie einzelne `MPMediaItem`-Elemente mithilfe der eingebauten Klasse `MPMediaPickerController` abrufen.

```
- (void)searchBarSearchButtonClicked: (UISearchBar *) searchBar
{
   // Blendet die Tastatur aus
   [searchBar resignFirstResponder];

   // Setzt den Titelcache zurück
   self.titleCache = [NSMutableDictionary dictionary];
```

```
    // Erstellt eine neue Abfrage
    MPMediaQuery *query = [MPMediaQuery songsQuery];
    MPMediaPropertyPredicate *mpp = [MPMediaPropertyPredicate
        predicateWithValue:searchBar.text
        forProperty:MPMediaItemPropertyTitle
        comparisonType:MPMediaPredicateComparisonContains];
    [query addFilterPredicate:mpp];

    // Ruft die Ergebnisse ab und lädt die Tabellendaten neu
    self.songCollections = query.collections;
    [self.tableView reloadData];
}

- (NSInteger)numberOfSectionsInTableView:(UITableView *)aTableView
{
    // Fasst alle Daten in einem einzigen Abschnitt zusammen
    return 1;
}

- (NSInteger)tableView:(UITableView *)aTableView
    numberOfRowsInSection:(NSInteger)section
{
    // Die Anzahl der Zeilen wird durch den Vergleich der Sammlungen
    // bestimmt
    return [self.songCollections count];
}

- (UITableViewCell *)tableView:(UITableView *)tView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Dient dazu, einen Eindruck vom zeitlichen Ablauf zu vermitteln
    printf("Retrieving cell %d\n", indexPath.row);

    UITableViewCellStyle style = UITableViewCellStyleDefault;
    UITableViewCell *cell = [tView
        dequeueReusableCellWithIdentifier:@"BaseCell"];
    if (!cell) cell = [[[UITableViewCell alloc] initWithStyle:style
        reuseIdentifier:@"BaseCell"] autorelease];

    NSString *label = [titleCache objectForKey:NUMBER(indexPath.row)];

    if (!label)

    {

        MPMediaItem *item = [[[self.songCollections
```

```
            objectAtIndex:indexPath.row] items] lastObject];

        label = [item valueForProperty:MPMediaItemPropertyTitle];

        [titleCache setObject:label forKey:NUMBER(indexPath.row)];

    }
    cell.textLabel.text = label;
    return cell;
}

- (void) viewDidLoad
{
    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;

    // Richtet die Suchleiste ein
    UISearchBar *sb = [[[UISearchBar alloc]
        initWithFrame:CGRectMake(0.0f, 0.0f, 320.0f, 44.0f)]
        autorelease];
    sb.autocapitalizationType = UITextAutocapitalizationTypeNone;
    sb.autocorrectionType = UITextAutocorrectionTypeNo;
    sb.backgroundColor = [UIColor clearColor];
    sb.tintColor = COOKBOOK_PURPLE_COLOR;
    self.navigationItem.titleView = sb;
    sb.delegate = self;

    self.titleCache = [NSMutableDictionary dictionary];
}
```

▶ *Rezept 15.11: Zur Demonstration der Langsamkeit von Medienabfragen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.12 Rezept: MPMusicPlayerController verwenden

Cocoa Touch enthält eine einfach zu verwendende Klasse für einen Musik-Player, die sich problemlos für Mediensammlungen einsetzen lässt. Auch wenn ihr Name etwas anderes vermuten lässt, ist die Klasse `MPMusicPlayerController` kein Ansichtscontroller, denn sie bietet keine Bildschirmelemente für die Musikwiedergabe an. Stattdessen handelt es sich um einen abstrakten Controller, der für die Wiedergabe von Musik sorgt.

Wenn sich der Wiedergabestatus ändert, veröffentlicht diese Klasse optionale Benachrichtigungen. Sie bietet zwei gemeinsam genutzte Instanzen, nämlich `iPodMusicPlayer` und `applicationMusicPlayer`. Verwenden Sie immer Erstere! Sie gibt zuverlässigere Rückmeldungen über Statusänderungen, die Sie im Programmcode erfassen möchten.

Den Player-Controller initialisieren Sie, indem Sie `setQueueWithItemCollection:` mit einer `MPMediaItemCollection` initialisieren:

```
[[MPMusicPlayerController iPodMusicPlayer]
   setQueueWithItemCollection:self.songs];
```

Alternativ können Sie auch eine Liste mit einer Medienabfrage laden. Beispielsweise können Sie eine `playListQuery` festlegen, die nach einem bestimmten Begriff in einer Wiedergabeliste sucht, oder eine Künstlerabfrage, um Songs eines bestimmen Interpreten zu finden. Mit `setQueueWithQuery:` erstellen Sie aus einer `MPMediaQuery`-Instanz eine neue Liste.

Wenn Sie die Titel in zufälliger Reihenfolge wiedergeben möchten, weisen Sie der Eigenschaft `shuffleMode` des Controllers einen Wert zu. Zur Auswahl stehen `MPMusicShuffleModeDefault`, was der aktuellen Voreinstellung des Benutzers entspricht, `MPMusicShuffleModeOff` (keine Zufallswiedergabe), `MPMusicShuffleModeSongs` (Wiedergabe der Songs in zufälliger Reihenfolge) und `MPMusicShuffleModeAlbums` (Wiedergabe der Alben in zufälliger Reihenfolge). Ähnliche Optionen gibt es auch für den Wiederholungsmodus (`repeatMode`).

Nachdem Sie die Sammlung der Elemente eingerichtet haben, können Sie sie wiedergeben, die Wiedergabe anhalten, zum nächsten oder vorherigen Element in der Warteschlange springen usw. Um nicht zum vorherigen Element zu wechseln, sondern ganz an den Anfang zurückzuspulen, verwenden Sie `skipToBeginning`. Sie können auch innerhalb des zurzeit abgespielten Elements suchen und die Wiedergabeposition dabei vor- und zurückbewegen.

Rezept 15.12 stellt einen einfachen Medien-Player vor, der den zurzeit abgespielten Song anzeigt (zusammen mit einer ggf. dafür verfügbaren Grafik). Der Benutzer wählt eine Gruppe von Elementen aus einem `MPMediaPickerController` aus, wobei diese Elementsammlung dann zurückgegeben und dem Player zugewiesen wird, der mit der Wiedergabe der Gruppe beginnt.

Zwei Observer greifen auf das standardmäßige Benachrichtigungscenter zurück, um auf die beiden wichtigsten Änderungen zu warten, nämlich die Änderung des aktuellen Elements und des Wiedergabestatus. Um diese Änderungen zu erfassen, müssen Sie die Benachrichtigungen manuell anfordern. Dadurch können Sie die Oberfläche mit neuen Informationen über den zurzeit gespielten Titel aktualisieren, wenn ein anderes Element wiedergegeben wird.

```
[[MPMusicPlayerController iPodMusicPlayer]
   beginGeneratingPlaybackNotifications];
```

Diese Anforderung können Sie mit `endGenerationPlaybackNotifications` rückgängig machen, Sie können dem Programm aber auch einfach erlauben, alle Beobachter zu zerstören, wenn die Anwendung am Ende der Wiedergabe anhält. Da in diesem Rezept der iPod-Musik-Player verwendet wird, dauert die Wiedergabe noch an, wenn Sie die Anwendung verlassen, sofern Sie sie nicht ausdrücklich beenden. Das Beenden der Anwendung hat keinen Einfluss auf die Wiedergabe.

```
- (void) applicationWillTerminate: (UIApplication *) application
{
   // Beendet beim Schließen der Anwendung den Player
   [[MPMusicPlayerController iPodMusicPlayer] stop];
}
```

Rezept 15.2 veranschaulicht nicht nur die Wiedergabesteuerung, sondern zeigt auch, wie Sie beim Abspielen Grafiken aus dem Album anzeigen. Dazu wird der gleiche Abruf von `MPItem`-Eigenschaften verwendet wie in den vorherigen Rezepten. In diesem Fall wird `MPMediaItemPropertyArtwork` abgefragt, und falls es entsprechende Grafiken gibt, werden sie mithilfe der Klasse `MPMediaItemArtwork` in ein Bild der verlangten Größe umgewandelt.

```
#define PLAYER [MPMusicPlayerController iPodMusicPlayer]

#pragma mark PLAYBACK
- (void) pause
{
   // Hält die Wiedergabe an
   [PLAYER pause];
   toolbar.items = [self playItems];
}

- (void) play
{
   // Nimmt die Wiedergabe erneut auf
   [PLAYER play];
   toolbar.items = [self pauseItems];
}

- (void) fastforward
{
   // Springt zum nächsten Element vor
   [PLAYER skipToNextItem];
}

- (void) rewind
{
   // Springt zum vorherigen Element
   [PLAYER skipToPreviousItem];
}

#pragma mark STATE CHANGES
- (void) playbackItemChanged: (NSNotification *) notification
{
   // Aktualisiert Titel und Grafik
```

```
    self.title = [PLAYER.nowPlayingItem
        valueForProperty:MPMediaItemPropertyTitle];
    MPMediaItemArtwork *artwork = [PLAYER.nowPlayingItem
        valueForProperty: MPMediaItemPropertyArtwork];
    imageView.image = [artwork imageWithSize:[imageView frame].size];
}

- (void) playbackStateChanged: (NSNotification *) notification
{
    // Löscht beim Beenden Titel, Symbolleiste und Grafik
    if (PLAYER.playbackState == MPMusicPlaybackStateStopped)
    {
        self.title = nil;
        toolbar.items = nil;
        imageView.image = nil;
    }
}

#pragma mark MEDIA PICKING
- (void)mediaPicker: (MPMediaPickerController *)mediaPicker
    didPickMediaItems:(MPMediaItemCollection *)mediaItemCollection
{
    // Setzt die Songs auf die vom Benutzer ausgewählte Sammlung
    self.songs = mediaItemCollection;

    // Aktualisiert die Liste für die Wiedergabe
    [PLAYER setQueueWithItemCollection:self.songs];

    // Zeigt das wiedergegebene Element in der Symbolleiste an
    [toolbar setItems:[self playItems]];

    // Entfernt den Picker
    [self dismissModalViewControllerAnimated:YES];
    [mediaPicker release];
}

- (void)mediaPickerDidCancel:(MPMediaPickerController *)mediaPicker
{
    // Der Benutzer hat den Vorgang abgebrochen
    [self dismissModalViewControllerAnimated:YES];
    [mediaPicker release];
}

- (void) pick: (UIBarButtonItem *) bbi
{
```

```
    // Wählt die Songs für die Wiedergabe-Warteschlange aus
    MPMediaPickerController *mpc = [[MPMediaPickerController alloc]
        initWithMediaTypes:MPMediaTypeMusic];
    mpc.delegate = self;
    mpc.prompt = @"Please select items to play";
    mpc.allowsPickingMultipleItems = YES;

    [self presentModalViewController:mpc animated:YES];
}

#pragma mark INIT VIEW
- (void) viewDidLoad
{
    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
    self.navigationItem.rightBarButtonItem = BARBUTTON(@"Pick",
        @selector(pick:));
    toolbar.tintColor = COOKBOOK_PURPLE_COLOR;

    // Beendet die Wiedergabe
    [PLAYER stop];

    // Fügt Beobachter für Status- und Elementänderungen hinzu
    [[NSNotificationCenter defaultCenter] addObserver:self
        selector:@selector(playbackStateChanged:)
        name:MPMusicPlayerControllerPlaybackStateDidChangeNotification
        object:PLAYER];
    [[NSNotificationCenter defaultCenter] addObserver:self
        selector:@selector(playbackItemChanged:)
        name:MPMusicPlayerControllerNowPlayingItemDidChangeNotification
        object:PLAYER];
    [PLAYER beginGeneratingPlaybackNotifications];
}
@end
```

▸ *Rezept 15.12: Einfache Medienwiedergabe mit dem iPod-Musik-Player*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 15 und öffnen das Projekt zu diesem Rezept.

15.13 Ein letzter Punkt: Zusätzliche Eigenschaften des VideoPlayers

Die in diesem Kapitel vorgestellte Klasse `MPMoviePlayerController` weist noch einige nützliche Eigenschaften auf, die in den Rezepten bisher nicht behandelt wurden. Mit diesen Eigenschaften können Sie bestimmen, wie der Player auf dem Bildschirm angezeigt wird.

Mit der Eigenschaft `backgroundColor` legen Sie die Farbe des Hintergrunds fest. Normalerweise erscheint der Player in Schwarz, wobei der Film eingeblendet wird, wenn der Vorab-Ladevorgang beendet ist. Um die Videodarstellung enger in die normale Oberfläche einzubetten, können Sie diese Eigenschaft auf `[UIColor clearColor]` setzen, sodass Ihre Benutzeroberfläche im Player durchscheint.

Eine weitere Eigenschaft ist `scalingMode`, mit der Sie festlegen, wie das Videobild in den Rahmen des Players eingepasst wird. In der Voreinstellung versucht der Player, den Film so darzustellen, dass der Rahmen zumindest in einer Richtung ausgefüllt wird, ohne das Bild in der anderen Richtung beschneiden zu müssen. Dies entspricht der Konstante `MPMovieScalingModeAspectFit`. Die beiden anderen Skalierungsoptionen (neben `None`, wobei keine Einpassung erfolgt) sind `AspectFill` und `NormalFill`. Bei `MPMovieScalingModeAspectFill` wird der gesamte Bildschirm gefüllt, aber das Seitenverhältnis beibehalten, sodass Bildteile, die außerhalb des sichtbaren Bereichs fallen, abgeschnitten werden. Dagegen wird bei `MPMovieScalingModeNormalFill` das Seitenverhältnis aufgegeben, und die horizontale und die vertikale Achse werden unabhängig voneinander skaliert.

Wenn Sie festlegen möchten, an welcher Stelle die Wiedergabe beginnt, verwenden Sie die Eigenschaft `initialPlaybackTime`. Sie nimmt einen `NSTimeInterval`-Wert entgegen und verschiebt den Startpunkt der Wiedergabe um den angegebenen Betrag. Leider können Sie den Player nicht direkt nach der aktuellen Wiedergabeposition abfragen, um einen Fixpunkt für eine spätere Wiedergabe zu speichern. Hoffentlich wird sich Apple in zukünftigen Versionen des SDK dieses Mankos annehmen.

15.14 Zusammenfassung

In diesem Kapitel wurden viele Möglichkeiten zur Arbeit mit Audio- und Videomedien vorgestellt, vor allem für die Wiedergabe und Aufnahme. Sie haben hier Rezepte gesehen, in denen Objective-C-Klassen hoher Ebene verwendet wurden, und andere, in denen systemnahe C-Funktionen zum Einsatz kamen. Außerdem haben Sie etwas über lokale und entfernte Datenquellen, über Medien-Picker, Controller usw. gelesen. Von der Lektüre dieses Kapitels sollten Sie die folgenden wichtigen Punkte mitnehmen:

- Apple ist immer noch dabei, die Klassen für die AV-Medienwiedergabe zu erstellen. Viele dieser Klassen wirken zurzeit noch unfertig und skelettartig, doch können wir erwarten, dass sie bei der Einführung der nächsten Firmware-Version ausgereifter sind. `AVAudioPlayer` stand erst mit der Firmware-Version 2.2 zur Verfügung, `AVAudioRecorder` sogar noch später. Medienwiedergabe und -steuerung sind immer noch in der Entwicklung.

- Bedenken Sie bei der Arbeit mit Videodaten die Bandbreiteneinschränkungen. Lokale Netzwerkanbieter werden es wahrscheinlich nicht schätzen, wenn Sie ihre Einrichtungen mit übermäßigen Datenübertragungen überlasten, was auch zu Schwierigkeiten führen kann, wenn Sie versuchen, diese Anwendungen im App Store einzustellen.
- Audio-Queues sind leistungsfähige, systemnahe Audioroutinen, aber sie eignen sich nicht für Leute mit schwachen Nerven und Programmierer, die nur auf eine schnelle Lösung aus sind. Wenn Sie aber die Art von detaillierter Audiosteuerung benötigen, die mit Audio Queues möglich ist, finden Sie bei Apple eine ausführliche Dokumentation dazu, wie Sie damit die verschiedenen Ziele erreichen können.
- `MPMusicPlayerController` bietet einen wirklich einfachen Weg, um mit Musik der iTunes-Bibliothek auf dem Gerät zu arbeiten. Meines Wissens gibt es keine Möglichkeit, um direkt auf Musikdaten zuzugreifen, weshalb Sie sich sowohl mit `AVAudioPlayer` für lokale Datendateien als auch mit `MPMusicPlayerController` für iTunes vertraut machen sollten.

16 Push-Benachrichtigungen

Wenn Entwickler unmittelbar mit den Benutzern kommunizieren müssen, können sie dazu auf Push-Benachrichtigungen zurückgreifen. Dadurch werden Nachrichten über einen besonderen Apple-Dienst direkt auf dem Bildschirm des iPhones zugestellt. Bei einer Push-Benachrichtigung zeigt das iPhone ein Meldungsfeld an, spielt einen festgelegten Klang ab oder aktualisiert ein Anwendungs-Badge. Dadurch können Dienste außerhalb des iPhones Kontakt mit einem Client auf dem Gerät aufnehmen und ihn über neue Daten oder Aktualisierungen informieren. Anders als andere Bereiche der iPhone-Programmierung erfolgt die Erstellung von Push-Benachrichtigungen fast ausschließlich außerhalb des iPhones. Die Entwickler müssen Webdienste einrichten, um solche Aktualisierungen zu verwalten und bereitzustellen. In diesem Kapitel erfahren Sie, wie Push-Benachrichtigungen funktionieren und wie Sie ein eigenes System dafür erstellen.

16.1 Einführung in Push-Benachrichtigungen

Push-Benachrichtigungen (oder *Remote-Benachrichtigungen*) sind Nachrichten, die ein externer Dienst an das iPhone sendet. Solche Push-Dienste gibt es gewöhnlich für Anwendungen, die nach Aktualisierungen suchen. Beispielsweise kann ein solcher Dienst neue Direktnachrichten von Twitter abrufen oder auf die Sensoren eines Alarmsystems reagieren. Wenn neue Informationen für den Client zur Verfügung stehen, überträgt der Dienst sie über das Remote-Benachrichtigungssystem von Apple. Die Nachricht geht unmittelbar an das iPhone, das für den Empfang solcher Aktualisierungen registriert wurde.

Der wichtigste Punkt für das Verständnis dieses Vorgangs besteht darin, dass der Ursprung der Nachricht außerhalb des Geräts liegt. Wir haben es hier mit einem Client/Server-System zu tun, bei dem Serverkomponenten im Web über einen Apple-Dienst mit iPhone-Clients kommunizieren können. Mithilfe von Push-Benachrichtigungen können Entwickler praktisch im Nu Aktualisierungen

auf iPhones senden, ohne dass die Benutzer eine bestimmte Anwendung starten müssen. Stattdessen erfolgt die Verarbeitung auf der Serverseite. Wenn die Push-Nachricht eintrifft, reagiert der iPhone-Client darauf, indem er ein Badge anzeigt, einen Klang abspielt oder ein Benachrichtigungsfeld einblendet.

Laut Apple ist der wichtigste Grund für die Nutzung von Push-Benachrichtigungen die Akkukapazität. Wenn in Hintergrundprozessen viele Anwendungen auf einmal ausgeführt werden, kann dies den Akku übermäßig belasten, sodass er in kürzeren Zeitabständen neu geladen werden muss. Dank Push-Benachrichtigungen erhalten die Anwendungen auch dann Informationen über Aktualisierungen, wenn sie gar nicht laufen. Dadurch kann Apple seine strenge Richtlinie durchsetzen, immer nur eine Drittanbieter-Anwendung auf einmal auszuführen, während die Benutzer gleichzeitig Benachrichtigungen über Änderungen des Anwendungsstatus erhalten können.

Die Verschiebung der Anwendungslogik auf einen Server vereinfacht auch die Clientseite. Die externe Verarbeitung spart auch Energie für iPhone-Anwendungen, die nicht mehr auf die lokalen CPU-Ressourcen zurückgreifen müssen, sondern sich auf Push-Benachrichtigungen stützen können, um wichtige Änderungen von Informationen mitzubekommen und darauf zu reagieren.

Die Einsparung lokaler Ressourcen ist jedoch nicht die einzige Existenzberechtigung für Push-Benachrichtigungen. Dieses System bietet eine wertvolle Lösung für eine Kommunikation von Webdiensten, die über das einfache Muster von Abruf und Aktualisierung hinausgeht. Beispielsweise können Sie sich damit in einen Empfehlungsdienst einklinken, der Ihnen Restaurants vorschlägt, selbst wenn die entsprechende Anwendung nicht läuft, oder in einen Kalenderdienst, der Sie an die nächsten Termine erinnert. Betrachten Sie Push-Benachrichtigungen daher nicht nur als Akkuschoner, sondern auch als Kanal für Webdienste.

Mit Push-Benachrichtigungen bleiben die iPhone-Benutzer immer auf dem neuesten Stand asynchroner Daten-Feeds, von Social Networking-Anwendungen bis zum Abonnement von RSS-Feeds. Damit haben Sie eine leistungsfähige Lösung zur Hand, um iPhone-Clients mit webgestützten Systemen aller Art zu verbinden. Die Dienste, die Sie schreiben, können über Push-Benachrichtigungen Verbindung mit den iPhones aufnehmen, auf denen Ihre Anwendung installiert ist, und Aktualisierungen sauber und wirkungsvoll übertragen.

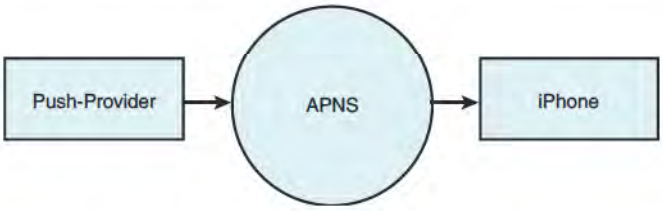
16.1.1 Wie funktionieren Push-Benachrichtigungen?

Push-Benachrichtigungen sind nicht einfach nur eine allgemeine Möglichkeit, um bei Bedarf direkt mit dem iPhone zu kommunizieren, sondern sie sind an bestimmte Anwendungen gebunden und erfordern verschiedene Sicherheitsprüfungen. Ein Push-Server kann nur mit den iPhones kommunizieren, auf denen seine Anwendung läuft, die online sind und deren Benutzer sich mit dem Empfang von Remote-Nachrichten einverstanden erklärt haben. Bei Push-Aktualisierungen hat der Benutzer das letzte Wort. Er kann diese Form der Kommunikation zulassen oder verweigern, und eine gut geschriebene Anwendung erlaubt ihm jederzeit, die Nutzung dieses Dienstes zu erlauben oder abzulehnen.

Die Kommunikation zwischen Server und Client läuft wie folgt ab. Push-Provider stellen ihren iPhone-Clients Nachrichtenanforderungen über einen zentralen Apple-Server zu. Im normalen Gebrauch wird der Server durch irgendein Ereignis ausgelöst (z. B. eine neue E-Mail oder einen fälligen Termin) und

generiert Benachrichtigungsdaten für ein bestimmtes iPhone-Gerät. Anschließend sendet er diese Nachrichtenanforderung an den *Apple Push Notification Service* (APNS). Diese Benachrichtigung liegt im JSON-Format vor und ist auf 256 Byte begrenzt, sodass Sie damit nur stark eingeschränkte Informationshäppchen übertragen können. Formatierung und Größe sorgen dafür, dass APNS nur die geringstmögliche Bandbreite in Anspruch nimmt.

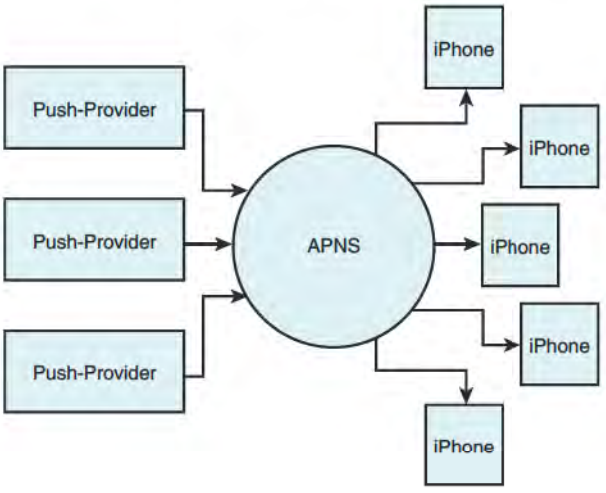
APNS ist ein zentrales System, das die Kommunikation mit den iPhones aushandelt. Der Dienst stellt die Nachricht auf dem vorgesehenen iPhone zu, wo ein Handler auf dem Gerät entscheidet, wie sie zu verarbeiten ist. Wie Sie in *Abbildung 16.1* sehen, kommunizieren die Push-Provider mit APNS und senden ihre Nachrichtenanforderungen an den Dienst, der wiederum mit den iPhones spricht und die Nachrichten an die Handler auf den Geräten weiterleitet.



▸ *Abbildung 16.1: Provider senden Nachrichten über den zentralen Push-Benachrichtigungsdienst von Apple an die iPhones.*

16.1.2 Unterstützung für mehrere Provider

APNS ist für mehrere Provider-Verbindungen eingerichtet, sodass viele Dienste auf einmal damit kommunizieren können. Der Apple-Dienst weist mehrere Gateways auf, sodass die einzelnen Push-Dienste nicht auf einen verfügbaren Zugriff warten müssen, bevor sie ihre Nachrichten senden können. *Abbildung 16.2* veranschaulicht diese m:n-Beziehung zwischen Providern und iPhones. APNS ermöglicht es den Providern dank seiner mehreren Gateways, sofort Verbindung aufzunehmen, und jeder Provider kann Nachrichten an verschiedene iPhones senden.



▸ *Abbildung 16.2: Der Push-Benachrichtigungsdienst von Apple bietet auf der Provider-Seite viele Gateways, sodass mehrere Provider gleichzeitig darauf zugreifen können. Jeder Provider wiederum kann mit einer beliebigen Anzahl von iPhone-Geräten Verbindung aufnehmen.*

16.1.3 Sicherheit

Die Sicherheit ist ein wichtiger Aspekt für Remote-Benachrichtigungen. Der Push-Provider muss sich für die Anwendung, mit der er zusammenarbeitet, für ein SSL-Zertifikat (*Secure Sockets Layer*) registrieren. Dienste können erst dann mit APNS kommunizieren, wenn sie sich mit einem solchen Zertifikat authentifiziert haben. Außerdem müssen sie ein sogenanntes *Token* (einen eindeutigen Schlüssel) vorweisen, der sowohl das anzusprechende Gerät als auch die zu benachrichtigende Anwendung bezeichnet.

Nach dem Eingang einer authentifizierten Nachricht und eines Geräte-Tokens nimmt APNS Kontakt mit dem fraglichen Gerät auf. Jedes iPhone- oder sonstige Gerät aus der iPhone-Familie (z. B. ein iPod touch) muss auf irgendeine Weise online erreichbar sein, um eine Benachrichtigung zu empfangen, wobei die Verbindung über Mobilfunk oder einen Wi-Fi-Hotspot erfolgen kann. APNS richtet die Verbindung mit dem Gerät ein und leitet die Benachrichtigungsanforderung weiter. Ist das Gerät offline, sodass der APNS-Server keinen Kontakt herstellen kann, wird die Benachrichtigung für die spätere Zustellung in eine Warteschlange gestellt.

Beim Empfang einer Anforderung führt das iPhone einige Überprüfungen durch. Die Anforderung wird ignoriert, wenn der Benutzer Push-Aktualisierungen für die betreffende Anwendung deaktiviert hat, was er im Programm *Einstellungen* auf dem iPhone tun kann. Sind Aktualisierungen dagegen zugelassen, schaut das iPhone nach, ob die entsprechende Anwendung zurzeit läuft oder nicht. Wenn ja, wird die Nachricht über eine Anwendungsdelegierung unmittelbar an das Programm gesendet. Anderenfalls wird der Benutzer auf irgendeine Weise benachrichtigt, z. B. durch die Anzeige von Text, das Abspielen eines Tons oder die Aktualisierung eines Badges.

Wird eine solche Benachrichtigung angezeigt, hat der Benutzer für gewöhnlich die Wahl, sie zu schließen oder auf eine Schaltfläche zum Anzeigen zu tippen. In letzterem Fall startet das iPhone die fragliche Anwendung und sendet ihr die Nachricht, wie in dem Fall, dass das Programm beim Empfang gerade läuft. Schließt der Benutzer dagegen das Benachrichtigungsfeld, wird die Nachricht ignoriert und die Anwendung nicht gestartet.

Dieser Weg vom Server über APNS zum iPhone und dort zur Anwendung ist der normale Kanal für Push-Benachrichtigungen. In jeder Phase wird die Nachricht ein Stück weiter auf dem Weg verschoben. Die Anzahl der Schritte mag übermäßig erscheinen, doch in der Praxis kommen Benachrichtigungen nahezu sofort an. Wenn Sie erst einmal die Zertifikate, Bezeichner und Verbindungen eingerichtet haben, ist die eigentliche Zustellung der Informationen trivial. Fast die gesamte Arbeit erschöpft sich darin, den Kanal aufzubauen und die Informationen zusammenzustellen, die zugestellt werden sollen.

Achten Sie darauf, alle Anwendungszertifikate und Geräte-Token als sensible Informationen zu behandeln. Wenn Sie diese Objekte auf Ihrem Server speichern, müssen Sie sicherstellen, dass sie nicht frei zugänglich sind. Sollten diese Informationen an die breite Öffentlichkeit geraten, könnten sie von Fremden missbraucht werden, was wahrscheinlich dazu führt, dass Apple Ihr SSL-Zertifikat für Push-Benachrichtigungen sperrt. Damit würden jegliche Remote-Benachrichtigungen für alle von Ihnen verkauften Apps ausgesetzt, und Sie müssten wahrscheinlich das Programm wieder aus dem App Store nehmen.

16.1.4 Einschränkungen von Push-Benachrichtigungen

Push-Benachrichtigungen sind alles andere als zuverlässig. Apple garantiert nicht, dass die Benachrichtigungen in der Reihenfolge zugestellt werden, in der sie eingehen. Senden Sie auch niemals wesentliche Informationen als Push-Benachrichtigungen, sondern reservieren Sie diese Funktion für nützliche Hinweise, über die Sie die Benutzer auf dem neuesten Stand halten können, deren Ausbleiben aber für die Benutzer folgenlos ist.

Die Elemente in der Warteschlange für die Push-Zustellung können durch neuere Benachrichtigungen ersetzt werden. Das bedeutet, dass die einzelnen Benachrichtigungen konkurrieren und verloren gehen können. Apple meldet zwar fehlgeschlagene Zustellungen (also Nachrichten, die nicht ordnungsgemäß über den Push-Dienst übertragen werden können, vor allem Mitteilungen an Anwendungen, die vom vorgesehenen Gerät entfernt wurden), doch erhalten Sie keine Informationen über verlorene Nachrichten. Der APNS sieht sie als erfolgreich zugestellt an.

16.1.5 Ein Push-System einrichten

Um mit der Push-Entwicklung zu beginnen, müssen Sie das Portal des iPhone-Entwicklerprogramms von Apple unter *http://developer.apple.com/iphone/manage/overview/index.action* aufsuchen und dort Ihre Anmeldeinformationen als iPhone-Entwickler angeben. In diesem Portal erstellen Sie einen neuen Anwendungsbezeichner für einen Push-Dienst.

Es gibt dabei sehr viele einzelne Schritte zu beachten, die Sie alle sorgfältig ausführen müssen. In den folgenden Abschnitten finden Sie eine ausführliche Anleitung dazu. Sie lernen dort, wie Sie eine neue ID erstellen, wie Sie ein Zertifikat gewinnen und ein besonderes Profil für push-fähige Anwendungen anfordern. Ohne ein solches Profil kann Ihre Anwendung keine Push-Benachrichtigungen empfangen.

16.1.6 Einen neuen Anwendungsbezeichner erstellen

Klicken Sie im Entwicklerportal auf die Option **App IDs**, die Sie in der Spalte auf der linken Seite finden. Dadurch wird eine Seite geöffnet, auf der Sie neue Anwendungsbezeichner erstellen können. Jeder Push-Dienst braucht einen eigenen Bezeichner, den Sie hier erstellen und dann so einrichten müssen, dass Remote-Benachrichtigungen zulässig sind. Für Push-Anwendungen können Sie keine Bezeichner mit Platzhaltern verwenden, da jedes push-fähige Programm einen eindeutigen Bezeichner benötigt.

Klicken Sie im Abschnitt **App IDs** auf die Schaltfläche **New App ID** oben rechts auf der Seite. Dadurch wird die neue Seite **Create App ID** geöffnet. Geben Sie einen Namen ein, der den neuen Bezeichner beschreibt, z. B. »Meine erste Push-Anwendung«, und einen neuen Bundle-Bezeichner.

Diese Bezeichner sind normalerweise in der umgekehrten Domänenschreibweise `com.`*`domänenname`*`.`*`anwendungsname`* aufgebaut, z. B. `com.sadun.firstpushapp`. Der Bezeichner muss eindeutig sein und darf nicht mit irgendeinem anderen registrierten Anwendungsbezeichner im Apple-System übereinstimmen. Der Bundle-Bezeichner für Ihre Anwendung (den Sie in `Info.plist` festlegen) muss genau mit dem letzten Teil dieses Strings übereinstimmen. Lautet der Bezeichner im Portal z. B. `XYZZYPLUGH.com.sadun.pushapp`, dann lautet der Bundle-Bezeichner des Programms `com.sadun.pushapp`.

Klicken Sie auf **Submit**, um den neuen Bezeichner zu erstellen. Dadurch wird er unwiderruflich zum Apple-System hinzugefügt und für Sie registriert. Anschließend kehren Sie zur Seite **App ID** mit der Liste der Bezeichner zurück und können jetzt den neuen Bezeichner für Push-Benachrichtigungen einrichten.

> **Hinweis**
>
> Apple bietet keine Möglichkeit, um einen einmal erstellten Anwendungsbezeichner wieder aus dem Portal zu entfernen.

16.1.7 Ein SSL-Zertifikat anfordern

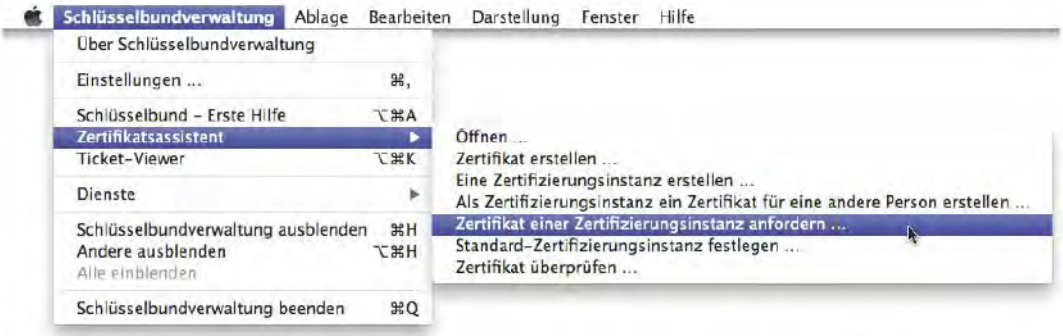
Auf der Seite **App IDs** können Sie in der Spalte **Apple Push Notificaton service** ablesen, für welche Bezeichner Push-Benachrichtigungen aktiviert sind. In dieser Spalte werden drei mögliche Werte angezeigt:

- **Unavailable** (nicht verfügbar, grau) für Bezeichner, die nicht mehr verfügbar sind bzw. nicht für eine Push-Benachrichtigung angelegt wurden.
- **Available** (verfügbar, gelb) für Anwendungen, für die Push-Benachrichtigungen verwendet werden könnten, die aber nicht dafür eingerichtet sind.
- **Enabled** (aktiviert, grün) für Anwendungen, die zum Empfang von Push-Benachrichtigungen bereit sind.

Neben jedem Anwendungsbezeichner finden Sie zwei farbige Punkte: einen für die Entwicklung und einen für die Produktion, denn die Push-Verfügbarkeit wird für diese beiden Bereiche getrennt eingerichtet. Für den neuen Bezeichner sollte die gelbe Verfügbarkeitsmarkierung für die Entwicklung angezeigt werden. Klicken Sie auf die Option **Configure** in der Spalte ganz rechts. Dadurch wird die neue Seite **Configure App ID** geöffnet, in der Sie den Bezeichner mit dem Push-Benachrichtigungsdienst verknüpfen können.

Auf halber Höhe der Seite finden Sie das Markierungsfeld **Enable for Apple Push Notificaton Service**. Aktivieren Sie dieses Feld, um damit zu beginnen, ein Zertifikat zu erstellen. Jetzt werden die beiden Konfigurationsschaltflächen auf der rechten Seite aktiv. Wenn Sie darauf klicken, wird eine Seite mit Anweisungen geladen, auf der Sie lesen können, wie Sie weiter vorgehen müssen. Hier erfahren Sie, wie Sie ein Sicherheitszertifikat erstellen, das Ihr Server verwendet, um die an APNS gesendeten Nachrichten zu signieren.

Öffnen Sie nach diesen Anweisungen das Programm **Schlüsselbundverwaltung**, das Sie auf Ihrem Macintosh im Ordner `/Programme/Dienstprogramme` finden, und wählen Sie **Schlüsselbundverwaltung ▸ Zertifikatsassistent ▸ Zertifikat einer Zertifizierungsinstanz anfordern** (siehe *Abbildung 16.3*). Diesen Schritt müssen Sie auch dann durchführen, wenn Sie bereits früher Entwickler- und Verteilungszertifikate angefordert haben. Durch die neue Anforderung wird das SSL-Zertifikat eindeutig bezeichnet.

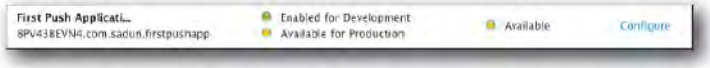


▸ *Abbildung 16.3: Selbst wenn Sie bereits Entwickler- und Verteilungszertifikate angefordert haben, müssen Sie eine neue Zertifikatanforderung erstellen.*

Wenn sich der Zertifikatsassistent öffnet, geben Sie Ihre E-Mail-Adresse und einen eindeutigen Namen wie `First Push App` ein. Dieser Name wird auch im weiteren Verlauf verwendet, weshalb Sie ihn so wählen sollten, dass er leicht erkennbar ist und Ihr Projekt gut beschreibt. Anhand dieses Namens können Sie in der Schlüsselbundverwaltung von OS X die ansonsten sehr ähnlichen Schlüsselbundelemente voneinander unterscheiden.

Nach der Angabe des Namens wählen Sie **Auf der Festplatte sichern** und **Fortfahren**. Der Zertifikatsassistent fordert Sie dazu auf, einen Speicherort anzugeben (der Schreibtisch ist dafür gut geeignet). Klicken Sie auf **Sichern**, warten Sie, bis das Zertifikat generiert ist, und klicken Sie anschließend auf **Fertig**. Kehren Sie zu Ihrem Webbrowser zurück, und klicken Sie auf **Continue**. Sie können jetzt die Signieranforderung einreichen.

Klicken Sie auf **Choose File**, und wechseln Sie zu der gerade erstellten Anforderung. Markieren Sie diese, und klicken Sie auf **Choose**. Klicken Sie auf **Generate**, um das neue SSL-Zertifikat für den Push-Dienst zu erstellen, was eine oder sogar zwei Minuten dauern kann. Haben Sie Geduld, und schließen Sie nicht die Webseite. Sobald das Zertifikat erstellt ist, klicken Sie auf **Continue**. Laden Sie das neue Zertifikat mit einem Klick auf **Download** herunter, und klicken Sie abschließend auf **Done**. Dadurch kehren Sie auf die Seite **App ID** zurück, wo jetzt ein neuer, grüner **Enabled**-Indikator neben dem Anwendungsbezeichner erscheint (siehe *Abbildung 16.4*). Apple sendet Ihnen auch per E-Mail eine Bestätigung, dass die Zertifikatsanforderung genehmigt wurde.



▸ *Abbildung 16.4: Die Bezeichnung* **Enabled** *erscheint neben Anwendungsbezeichnern, die für Push-Benachrichtigungen eingerichtet wurden. Für die Entwicklung und die Produktion müssen Sie jeweils eigene SSL-Zertifikate erstellen.*

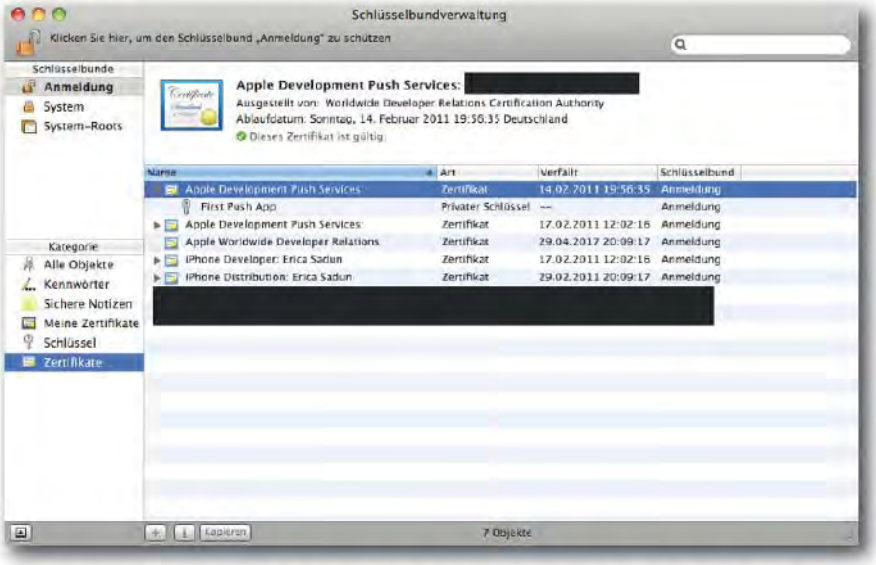
Wenn Sie den Push-Server auf Ihrem Macintosh ausführen möchten, müssen Sie das neue Zertifikat dem Schlüsselbund hinzufügen, indem Sie auf die heruntergeladene `.cer`-Datei doppelklicken. Dadurch wird sie zum Schlüsselbund **Anmeldung** hinzugefügt und erscheint unter **Zertifikate**. Wie Sie in *Abbildung 16.5* sehen, können Sie das Zertifikat identifizieren, indem Sie auf das kleine Dreieck daneben klicken, um den Namen einzublenden, den Sie beim Aufstellen der Zertifikatanforderung eingegeben haben.

16.1.8 Push-spezifische Profile

Für push-fähige Anwendungen können Sie keine Platzhalter-Profile verwenden, sondern müssen dafür jeweils ein eigenes, eindeutiges Profil anlegen. Wenn Sie also eine Entwicklungs-, eine Ad-hoc- und eine Verteilungsversion Ihres Programms erstellen möchten, müssen Sie neben den bereits vorhandenen drei neue Profildateien erstellen.

Wählen Sie im Abschnitt **Provisioning** des Entwicklerportals, ob Sie ein Entwicklungs- oder ein Verteilungsprofil erstellen möchten, indem Sie auf den Titel **Development** bzw. **Distribution** klicken. Klicken Sie auf **New Profile**, um ein neues Profil zu erstellen. Unabhängig davon, ob Sie ein Entwicklungs- oder ein Verteilungsprofil anlegen, öffnet sich die Seite **Create iPhone Provisioning Profile**.

- **Entwicklungsprofil** Geben Sie für die Entwicklung einen Profilnamen wie »My First Push App Development« ein. Prüfen Sie das zu verwendende Zertifikat, und wählen Sie den Anwendungsbezeichner aus dem Einblendmenü aus. Wählen Sie die Geräte, die Sie verwenden möchten, und klicken Sie auf **Submit**.
- **Verteilungsprofil** Für die Verteilung wählen Sie **App Store** oder **Ad Hoc** und geben einen Namen wie »My First Push App Distribution« oder »My First Push App Ad hoc« ein. Wählen Sie den Anwendungsbezeichner aus dem Einblendmenü aus. Nur bei der Ad-hoc-Verteilung wählen Sie die Geräte aus, die Sie in das Profil aufnehmen möchten. Klicken Sie abschließend auf **Submit**.



▶ *Abbildung 16.5: Sie können herausfinden, mit welchem SSL-Zertifikat für den Push-Dienst Sie es zu tun haben, indem Sie auf das Einblenddreieck klicken. Dadurch wird der Name angezeigt, den Sie bei der Zertifikatsanforderung gewählt haben.*

Es kann ein bis zwei Minuten dauern, das Profil zu erstellen. Warten Sie eine kurze Zeit, und laden Sie die Seite dann neu. Der Profilstatus sollte von **Pending** zu **Active** wechseln. Laden Sie das neue Profil herunter, und fügen Sie es in Xcode hinzu, indem Sie es auf das Programmsymbol von Xcode ziehen.

16.2 Eine Anwendung registrieren

Die Signierung einer Anwendung mit einem push-kompatiblen Profil ist lediglich der erste Schritt. Die Anwendung muss sich auch beim Remote-Benachrichtigungssystem des iPhones registrieren. Dazu verwenden Sie wie folgt einen einzelnen `UIApplication`-Aufruf. Die Delegierungsmethode `applicationDidFinishLaunching` ist der geeignete Ort für diesen Aufruf.

```
[[UIApplication sharedApplication]
   registerForRemoteNotificationTypes:types];
```

Dieser Aufruf teilt dem iPhone OS mit, dass die Anwendung Push-Nachrichten empfangen soll. Mit dem übergebenen Typ legen Sie fest, welche Art von Benachrichtigungen das Programm erhält. Auf dem iPhone stehen drei mögliche Typen zur Verfügung:

- `UIRemoteNotificationTypeBadge` Bei dieser Art von Benachrichtigung wird in SpringBoard ein rotes Badge zu der Anwendung hinzugefügt.
- `UIRemoteNotificationTypeSound` Bei Soundbenachrichtigungen können Sie eine Klangdatei aus dem Anwendungs-Bundle abspielen.
- `UIRemoteNotificationTypeAlert` Bei diesem Typ wird in SpringBoard oder einer anderen Anwendung eine Textmeldung mit einer von Ihnen gestalteten Nachricht angezeigt.

Wählen Sie die Typen, die Sie verwenden möchten, und kombinieren Sie sie mit einer ODER-Verknüpfung. Es handelt sich um Bit-Flags, die zusammengenommen dem Benachrichtigungsprozess anzeigen, wie Sie weiter vorgehen möchten. Mit den folgenden Flags erlauben Sie beispielsweise Badges und Meldungsfelder, aber keine Sounds:

```
types = UIRemoteNotificationTypeBadge | UIRemoteNotificationTypeAlert;
```

Durch die Registrierung werden die Benutzereinstellungen aktualisiert. Wie Sie in *Abbildung 16.6* sehen, wird dem Programm *Einstellungen* ein Abschnitt für Benachrichtigungen hinzugefügt, falls noch kein andere Anwendung einen solchen Bereich erstellt hat. Ihre Anwendung erscheint als Unterabschnitt, in dem der Benutzer die gewünschten Benachrichtigungstypen festlegen kann. Dabei erscheinen Schalter nur für die Arten von Benachrichtigungen, die Sie registriert haben. Nutzt Ihre Anwendung also zwei Formen von Benachrichtigungen, sieht der Benutzer auch zwei Schalter. *Abbildung 16.6* zeigt die Einstellungen für eine Anwendung, die für alle drei Typen registriert ist.

Um die aktive Teilnahme an Push-Benachrichtigungen für eine Anwendung wieder aufzuheben, senden Sie `unregisterForRemoteNotifications`. Dadurch wird die Registrierung der Anwendung für alle Benachrichtigungstypen entfernt. Dabei werden keine Argumente übergeben.

```
[[UIApplication sharedApplication] unregisterForRemoteNotifications];
```

16.2.1 Das Geräte-Token abrufen

Ihre Anwendung kann erst dann Push-Nachrichten empfangen, wenn sie ein Geräte-Token erstellt und an den externen Server gesendet hat, der die Benachrichtigungen ausgibt. In *Rezept 16.1*, das auf diesen Abschnitt folgt, werden die Serverfunktionen nicht implementiert, sondern nur die Clientsoftware.

Ein Token ist an ein Gerät gebunden. Zusammen mit dem SSL-Zertifikat bezeichnet es eindeutig das iPhone und wird dazu eingesetzt, Nachrichten von und zu dem fraglichen Gerät zu senden. Beachten Sie, dass sich Geräte-Token nach einer Wiederherstellung der iPhone-Firmware ändern können.

Geräte-Token werden im Rahmen der Registrierung erstellt. Beim Empfang einer Registrierungsanforderung nimmt das iPhone OS über SSL Kontakt mit dem *Apple Push Notification Service* (APNS) auf. Offensichtlich muss das Gerät dazu mit dem Internet verbunden sein. Das iPhone leitet die Anforderung an APNS weiter und wartet darauf, dass dieser Dienst mit einem Geräte-Token antwortet.

APNS erstellt das Geräte-Token und gibt es an das iPhone OS zurück, das es wiederum über das Anwendungsdelegierungs-Callback `application:didRegisterForRemoteNotificationsWithDeviceToken:` an die Anwendung übergibt.

► *Abbildung 16.6: Steuerelemente für Remote-Benachrichtigungen erscheinen für alle Anwendungen, die auf dem iPhone für die Push-Unterstützung registriert sind. Wird die Registrierung aufgehoben, werden die Steuerelemente entfernt.*

Ihre Anwendung muss dieses Token abrufen und der Providerkomponente Ihres Dienstes übergeben, wo es sicher gespeichert werden muss. Jeder, der Zugriff auf ein Geräte-Token und das SSL-Zertifikat der Anwendung hat, kann unerwünschte Nachrichten an das iPhone senden, weshalb Sie diese Angaben als sensible Informationen behandeln und entsprechend schützen müssen.

Hinweis

Manchmal kann es etwas länger dauern, das Token zu erstellen. Sehen Sie solche möglichen Verzögerungen beim Entwurf Ihrer Anwendungen vor, indem Sie beispielsweise die Registrierung jedes Mal durchführen, wenn die Anwendung läuft. Solange das Token noch nicht erstellt und auf Ihre Website hochgeladen ist, können Sie den Benutzern keine Remote-Benachrichtigungen zukommen lassen.

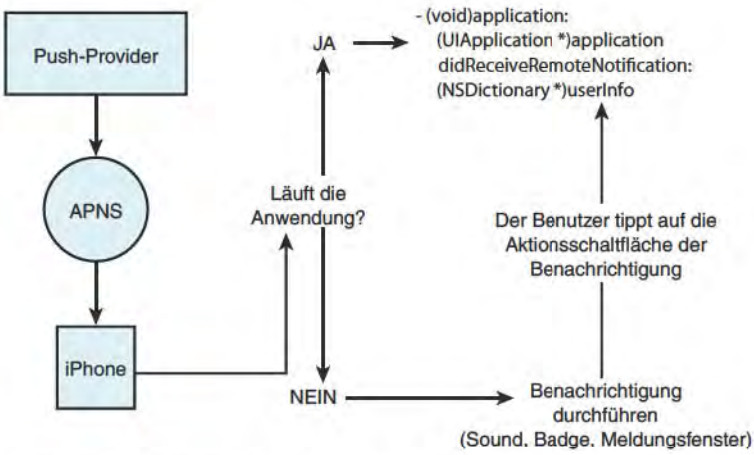
16.2.2 Auf Fehler bei der Token-Anforderung reagieren

Manchmal ist APNS nicht in der Lage, ein Token zu erstellen. Es mag auch vorkommen, dass das Gerät keine Anforderung senden kann. Beispielsweise können Sie im Simulator keine Tokens erzeugen. Mithilfe der `UIApplicationDelegate`-Methode `application:didFailToReigsterForRemoteNotificationsWithError:` können Sie auf solche Fehler reagieren. Meistens rufen Sie dabei die Fehlermeldung ab und zeigen sie dem Benutzer an.

```
// Zeigt dem Benutzer den Grund für den Fehlschlag der Registrierung an
- (void)application:(UIApplication *)application
    didFailToRegisterForRemoteNotificationsWithError:(NSError *)error
{
    UITextView *tv = (UITextView *)[[application keyWindow]
        viewWithTag:TEXTVIEWTAG];
    NSString *status = [NSString stringWithFormat:
        @"%@\nRegistration failed.\n\nError: %@", pushStatus(),
        [error localizedDescription]];
    tv.text = status;
}
```

16.2.3 Auf Benachrichtigungen reagieren

Zur Reaktion auf Push-Benachrichtigungen verwendet das iPhone eine vorgegebene Abfolge von Operationen (siehe *Abbildung 16.7*). Wenn die Anwendung läuft, werden die Benachrichtigungen direkt an die `UIApplicationDelegate`-Methode `application:didReceiveRemoteNotification:` gesendet. Die Nutzlast im JSON-Format wird automatisch in ein `NSDictionary` umgewandelt, wobei die Anwendung die darin enthaltenen Informationen auf beliebige Weise nutzen kann. Da die Anwendung bereits ausgeführt wird, werden keine Sounds, Badges und Meldungsfenster aufgerufen.



▸ *Abbildung 16.7: Sicht- und hörbare Benachrichtigungen werden nur dann angezeigt bzw. abgespielt, wenn die Anwendung nicht läuft. Tippt der Benutzer auf die Aktionsschaltfläche der Benachrichtigung (gewöhnlich* **View***), wird die Anwendung gestartet und die Nutzlast als Nachricht an* `UIApplicationDelegate` *gesendet.*

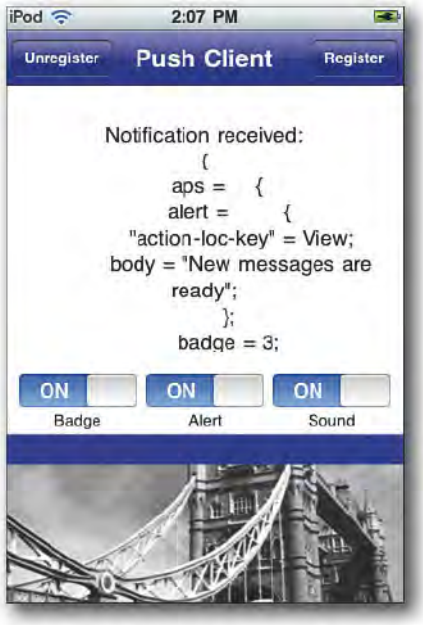
Läuft die Anwendung nicht, führt das iPhone alle angeforderten Benachrichtigungen durch, die in der Registrierung und den Benutzereinstellungen zulässig sind. Das kann bedeuten, einen Ton abzuspielen, das Anwendungssymbol mit einem Badge zu versehen oder ein Meldungsfeld anzuzeigen. Beim Abspielen eines Tons können Sie auch den Vibrationsalarm des iPhones auslösen.

▸ *Abbildung 16.8: Remote-Benachrichtigungen können in SpringBoard (links), aber auch in Anwendungen anderer Hersteller erscheinen (rechts). Die Benutzer können dieses Meldungsfeld schließen oder auf die Aktionsschaltfläche auf der rechten Seite tippen und damit zu der betroffenen Anwendung wechseln. In diesem Fall ist dies das Programm »HelloWorld«, dessen Name in der Benachrichtigung angezeigt wird. Den Text der Aktionsschaltfläche können Sie selbst festlegen.*

Wenn Sie ein Meldungsfeld anbieten, hat der Benutzer die Auswahl zwischen zwei Schaltflächen. Er kann auf **Close** klicken (die Schaltfläche links), um die Benachrichtigung auszublenden, oder auf die Aktionsschaltfläche (rechts), um die Anwendung zu starten. Beim Starten empfängt die Anwendungsdelegierung denselben Remote-Benachrichtigungs-Callback, den auch eine bereits laufende Anwendung erhält (siehe *Abbildung 16.8*). Wenn das iPhone gesperrt ist, erscheinen die Meldungsfelder über dem Sperrbildschirm.

16.3 Rezept: Das Grundgerüst eines Push-Clients

Rezept 16.1 erstellt einen elementaren Client, in dem die Benutzer die Registrierung für Push-Benachrichtigungen vornehmen und aufheben können. Die Benutzeroberfläche (dargestellt in *Abbildung 16.9*) zeigt drei Schalter, die steuern, für welche Arten von Benachrichtigungen das Programm registriert wird. Wenn die Anwendung startet, fragt sie nach, für welche Typen von Remote-Nachrichten sie registriert ist, und aktualisiert die Schalter entsprechend. Danach verfolgt der Client alle Registrierungen und Aufhebungen der Registrierung und passt die Schalter jeweils an, sodass sie mit den Einstellungen übereinstimmen.

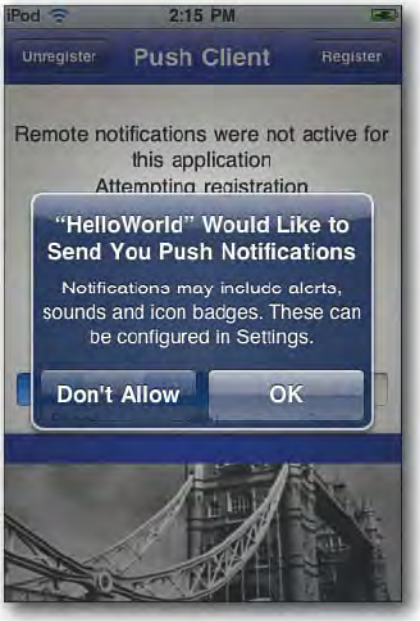


▸ *Abbildung 16.9: In dem in Rezept 16.1 vorgestellten Grundgerüst für einen Push-Client können die Benutzer festlegen, für welche Benachrichtigungstypen sie die Anwendung registrieren möchten.*

Mit zwei Schaltflächen oben links und oben rechts in der Oberfläche können die Benutzer die Anwendung registrieren und die Registrierung aufheben. Wie ich bereits erwähnt habe, werden beim Aufheben der Registrierung alle mit der Anwendung verknüpften Dienste deaktiviert. Das geht auf einen Schlag, wohingegen bei der Registrierung Flags gesetzt werden müssen, um anzuzeigen, welche Benachrichtigungstypen angefordert werden.

Bei der Registrierung für einen neuen Benachrichtigungstyp wird der Benutzer zur Bestätigung aufgefordert. Das dazu eingeblendete Dialogfeld sehen Sie in *Abbildung 16.10*. Hier muss der Benutzer ausdrücklich die Berechtigung erteilen. Wenn er dagegen auf **Don't Allow** tippt, verbleibt das betreffende Flag im bisherigen Zustand.

Leider ruft das Bestätigungsdialogfeld keinen Callback hervor, wenn es ausgeblendet wird (unabhängig davon, ob der Benutzer die Bestätigung abgegeben oder verweigert hat). Um dieses Ereignis zu erfassen, können Sie auf allgemeine Benachrichtigungen lauschen (`UIApplicationDidBecomeActiveNotification`), die ausgegeben werden, wenn ein Dialogfeld die Steuerung wieder an die Anwendung übergibt. Dies ist zwar ein Hack, der in Zukunft vielleicht nicht mehr funktioniert, aber zurzeit bietet Apple keine andere Möglichkeit, um herauszufinden, wann und wie der Benutzer reagiert hat. In *Rezept 16.1* fängt die Methode `confirmationWasHidden:` diese Benachrichtigung ab und aktualisiert die Schalter, sodass sie den neuen Registrierungseinstellungen entsprechen.



▸ *Abbildung 16.10: Die Benutzer müssen einer Anwendung ausdrücklich die Berechtigung erteilen, Remote-Benachrichtigungen zu empfangen.*

Da dieses System nur ein Grundgerüst ist, erschöpft sich die Reaktion des Push-Clients auf Push-Benachrichtigungen darin, dass er dem Benutzer die erhaltene Datennutzlast anzeigt. In *Abbildung 16.9* sehen Sie die tatsächlichen Nutzdaten, die in *Abbildung 16.10* gesendet wurden. Die Anzeige erfolgt in der Methode `applicatoin:didReceiveRemoteNotification:` der Anwendungsdelegierung.

Hinweis

Mit den drei Sounddateien im Online-Beispielprojekt (`ping1.caf`, `ping2.caf` und `ping3.caf`) können Sie Soundbenachrichtigungen testen.

```
#define TEXTVIEWTAG 11

NSString *pushStatus ()
{
   return [[UIApplication sharedApplication]
      enabledRemoteNotificationTypes] ?
      @"Remote notifications were active for this application" :
      @"Remote notifications were not active for this application";
}

@implementation TestBedController

// Ruft die aktuellen Schaltereinstellungen ab
- (NSUInteger) switchSettings
{
   NSUInteger which = 0;
   if ([(UISwitch *)[self.view viewWithTag:101] isOn])
      which = which | UIRemoteNotificationTypeBadge;
   if ([(UISwitch *)[self.view viewWithTag:102] isOn])
      which = which | UIRemoteNotificationTypeAlert;
   if ([(UISwitch *)[self.view viewWithTag:103] isOn])
      which = which | UIRemoteNotificationTypeSound;
   return which;
}

// Ändert die Schalter entsprechend der Registrierung
- (void) updateSwitches
{
   NSUInteger rntypes = [[UIApplication sharedApplication]
      enabledRemoteNotificationTypes];
   [(UISwitch *)[self.view viewWithTag:101] setOn:
      (rntypes & UIRemoteNotificationTypeBadge)];
   [(UISwitch *)[self.view viewWithTag:102] setOn:
      (rntypes & UIRemoteNotificationTypeAlert)];
   [(UISwitch *)[self.view viewWithTag:103] setOn:
      (rntypes & UIRemoteNotificationTypeSound)];
}

// Ein kleiner Hack, um festzustellen, ob das Bestätigungsdialogfeld
// ausgeblendet wird. Nachdem ich dies als technische Frage eingereicht
// habe, hat Apple diesen Trick zur Verwendung genehmigt.
- (void) confirmationWasHidden: (NSNotification *) notification
{
   [[UIApplication sharedApplication]
      registerForRemoteNotificationTypes: [self switchSettings]];
```

```
    [self updateSwitches];
}

// Registriert die Anwendung für die mit den Schaltern eingestellten
// Benachrichtigungstypen
- (void) doOn
{
    UITextView *tv = (UITextView *)[self.view viewWithTag:TEXTVIEWTAG];
    if (![self switchSettings])
    {
        tv.text = [NSString stringWithFormat:
            @"%@\nNothing to register. Skipping.\n\
            (Did you mean to press Unregister instead?)",
            pushStatus()];
        [self updateSwitches];
        return;
    }

    NSString *status = [NSString stringWithFormat:
        @"%@\nAttempting registration", pushStatus()];
    tv.text = status;
    [[UIApplication sharedApplication]
        registerForRemoteNotificationTypes:[self switchSettings]];
}

// Hebt die Registrierung der Anwendung für alle Arten von Push-
// Benachrichtigungen auf
- (void) doOff
{
    UITextView *tv = (UITextView *)[self.view viewWithTag:TEXTVIEWTAG];
    NSString *status = [NSString stringWithFormat:
        @"%@\nUnregistering.", pushStatus()];
    tv.text = status;

    [[UIApplication sharedApplication]
        unregisterForRemoteNotifications];
    [self updateSwitches];
}

- (void)loadView
{
    self.view = [[[NSBundle mainBundle] loadNibNamed:@"view" owner:self
        options:NULL] objectAtIndex:0];
    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
    self.title = @"Push Client";
```

```
    self.navigationItem.rightBarButtonItem = BARBUTTON(@"Register",
        @selector(doOn));
    self.navigationItem.leftBarButtonItem = BARBUTTON(@"Unregister",
        @selector(doOff));
    [self updateSwitches];
    [[NSNotificationCenter defaultCenter] addObserver:self
        selector:@selector(confirmationWasHidden:)
        name:@"UIApplicationDidBecomeActiveNotification" object:nil];
}
@end

@interface SampleAppDelegate : NSObject <UIApplicationDelegate>
@end

@implementation SampleAppDelegate
- (void) showString: (NSString *) aString
{
    UITextView *tv = (UITextView *)[[[UIApplication sharedApplication]
        keyWindow] viewWithTag:TEXTVIEWTAG];
    tv.text = aString;
}

// Ruft das Geräte-Token ab
- (void)application:(UIApplication *)application
    didRegisterForRemoteNotificationsWithDeviceToken:
        (NSData *)deviceToken
{
    NSUInteger rntypes = [[UIApplication sharedApplication]
        enabledRemoteNotificationTypes];
    NSString *results = [NSString stringWithFormat:
        @"Badge: %@, Alert:%@, Sound: %@",
        (rntypes & UIRemoteNotificationTypeBadge) ? @"Yes" : @"No",
        (rntypes & UIRemoteNotificationTypeAlert) ? @"Yes" : @"No",
        (rntypes & UIRemoteNotificationTypeSound) ? @"Yes" : @"No"];

    NSString *status = [NSString stringWithFormat:
        @"%@\nRegistration succeeded.\n\nDevice Token: %@\n%@",
        pushStatus(), deviceToken, results];
    [self showString:status];
    NSLog(@"deviceToken %@", deviceToken);
}

// Gibt dem Benutzer den Grund für den Fehlschlag einer Registrierung an
- (void)application:(UIApplication *)application
    didFailToRegisterForRemoteNotificationsWithError:
```

```
      (NSError *)error
{
   NSString *status = [NSString stringWithFormat:
      @"%@\nRegistration failed.\n\nError: %@", pushStatus(),
      [error localizedDescription]];
   [self showString:status];
   NSLog(@"Error in registration. Error: %@", error);
}

// Verarbeitet eine Benachrichtigung
- (void)application:(UIApplication *)application
   didReceiveRemoteNotification:(NSDictionary *)userInfo
{
   NSString *status = [NSString stringWithFormat:
      @"Notification received:\n%@",[userInfo description]];
   [self showString:status];
   CFShow([userInfo description]);
}

// Gibt bei Auslösung durch ein Meldungsdialogfeld die Nutzlast der
// Benachrichtigung an
- (void) launchNotification: (NSNotification *) notification
{
   [self performSelector:@selector(showString:)
      withObject:[[notification userInfo] description]
      afterDelay:1.0f];
}

- (void)applicationDidFinishLaunching:(UIApplication *)application {
   UIWindow *window = [[UIWindow alloc]
      initWithFrame:[[UIScreen mainScreen] bounds]];
   UINavigationController *nav = [[UINavigationController alloc]
      initWithRootViewController:[[TestBedController alloc] init]];
   [window addSubview:nav.view];
   [window makeKeyAndVisible];

   // Lauscht auf den Start der Remote-Benachrichtigungen
   [[NSNotificationCenter defaultCenter] addObserver:self
      selector:@selector(launchNotification:)
      name:@"UIApplicationDidFinishLaunchingNotification"
      object:nil];
}
@end
```

▸ *Rezept 16.1: Grundgerüst eines Push-Clients*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 16 und öffnen das Projekt zu diesem Rezept.

16.4 Nutzdaten für Benachrichtigungen erstellen

Für die Zustellung von Push-Benachrichtigungen über APNS sind drei Dinge erforderlich: ein SSL-Zertifikat, ein Geräte-Token und eine von Ihnen gestaltete Nutzlast im JSON-Format mit der Nachricht, die Sie senden möchten. Wie Sie das Zertifikat und das Geräte-Token erstellen, die Sie dem Server übergeben müssen, haben Sie schon gelesen. Um die JSON-Nutzdaten zu erstellen, müssen Sie im Grunde ein kleines, wohldefiniertes Dictionary ins JSON-Format umwandeln.

JSON (*JavaScript Object Notation*) ist ein einfaches Datenaustauschformat auf der Grundlage von Schlüssel-Wert-Paaren. Auf der Website zu JSON (*www.json.org*) finden Sie eine vollständige Darstellung der Syntax für das Format, in dem Sie Strings, Zahlen und Arrays als Werte verwenden können. Die APNS-Nutzlast umfasst 256 Bytes, in denen Sie den kompletten Inhalt Ihrer Benachrichtigung unterbringen müssen.

Benachrichtigungs-Nutzlasten müssen das Dictionary `aps` enthalten, das die Eigenschaften zum Erstellen der an den Benutzer gesendeten Klänge, Badges oder Meldungsdialogfelder angibt. Darüber hinaus können Sie noch eigene Dictionarys mit beliebigen Daten hinzufügen, die Sie an die Anwendung senden möchten, sofern Sie damit die Grenze von 256 Bytes nicht überschreiten. *Abbildung 16.11* zeigt den Aufbau für einfache (nicht lokalisierte) Benachrichtigungen.

```
aps
     badge : number
     sound : sound file name string
     alert : string
     alert
           body : string
           action-loc-key : string
```

▸ *Abbildung 16.11: Das Dictionary* `aps` *kann einen oder mehrere Benachrichtigungstypen (Badge, Sounddatei, Benachrichtigungsfeld) enthalten.*

Das Dictionary `aps` enthält einen oder mehrere der drei möglichen Benachrichtigungstypen (Badge, Sound, Meldungsfeld). Für Badge- und Soundbenachrichtigungen gibt es dabei jeweils ein Argument: im ersten Fall eine Zahl, im zweiten einen String, der auf eine bereits im Anwendungs-Bundle vorhandene Datei verweist. Wenn diese Datei nicht vorhanden ist (oder wenn der Entwickler `default` als Argument übergibt), wird für alle Benachrichtigungen dieses Typs ein Standardklang abgespielt. Ist in dem Dictionary keine Badge-Anforderung enthalten, entfernt das iPhone jegliches vorhandene Badge von dem Anwendungssymbol.

Um ein Benachrichtigungsdialogfeld anzuzeigen, gibt es zwei Möglichkeiten. Erstens können Sie einen String übergeben, der den angezeigten Text festlegt. Dadurch wird automatisch ein Dialogfeld mit den beiden Schaltflächen **Close** und **View** unter dem Text erstellt. Wenn Sie diese Schaltflächen anpassen möchten, übergeben Sie stattdessen ein Dictionary, in dem Sie den Nachrichtentext als `body` und den String für die Aktionsschaltfläche (normalerweise **View**) als `action-loc-key` angeben. Dadurch wird die Beschriftung **View** durch den von Ihnen beigesteuerten Text ersetzt.

Wenn Sie ein Dialogfeld einblenden wollen, das nur die Schaltfläche **OK** aufweist, übergeben Sie `null` als Argument für `action-loc-key`. Wie beim Antippen von **Close** werden bei einem solchen Dialogfeld jedoch keine Daten an die Anwendung übergeben. Die Anwendung muss dann die Aktualisierungen anfordern, wenn der Benutzer sie das nächste Mal öffnet.

16.4.1 Lokalisierte Benachrichtigungsdialogfelder

Für lokalisierte Anwendungen erstellen Sie das Dictionary `aps > alerts` mit zwei zusätzlichen Schlüsseln. Mit `loc-key` übergeben Sie einen Schlüssel, der in der Datei `Localizable.strings` der Anwendung definiert ist. Das iPhone schlägt diesen Schlüssel nach und ersetzt ihn durch den String in der aktuellen Lokalisierung.

Manchmal treten in Lokalisierungsstrings Argumente wie `%@` oder `%n$@` auf. Wenn das in Ihrer Anwendung der Fall ist, können Sie diese Argumente als Stringarrays über `loc-args` übergeben. In der Regel rät Apple von komplizierten Lokalisierungen ab, da sie einen Großteil der verfügbaren 256 Byte verschlingen können.

16.4.2 Das Dictionary ins JSON-Format umwandeln

Nachdem Sie das Dictionary erstellt haben, müssen Sie es in JSON umwandeln. Dieses Format ist einfach, aber genau. Verwenden Sie nach Möglichkeit eine automatisierte Bibliothek, um aus dem Dictionary einen JSON-String zu machen. Es gibt eine Vielzahl von Lösungen für die verschiedensten Programmiersprachen wie JavaScript, Perl usw. Im Folgenden finden Sie einen kurzen Überblick über die Grundlagen von JSON. *Tabelle 16.1* zeigt Beispiele für die angesprochenen Regeln.

Art der Nutzdaten	JSON-Darstellung
Hallo-Nachricht mit zwei Schaltflächen	`{"aps":{"alert":"hello"}}`
Hallo mit zwei Schaltflächen, aber mithilfe von JSON und einem Benachrichtigungs-Dictionary erstellt	`{"aps":{"alert":{"body":"hello"}}}`
Hallo-Nachricht mit einer OK-Schaltfläche	`{"aps":{"alert":{"action-loc-key":null,"body":"hello"}}}`
Hallo-Nachricht mit den Schaltflächen **Close** und **Open**, wobei Letztere die standardmäßige Schaltfläche **View** ersetzt	`{"aps":{"alert":{"action-loc-key":"Open","body":"hello"}}}`

Hallo-Nachricht, die das Anwendungs-Badge 3 hinzufügt	`{"aps":{"badge":3,"alert":{"body":"hello"}}}`
Sound ohne Anzeige eines Dialogfelds	`{"aps":{"sound":"ping2.caf","alert":{}}}`
Sound mit Anzeige eines Badges und eines Dialogfelds mit nicht standardmäßiger Schaltfläche	`{"aps":{"sound":"ping2.caf","badge":2,"alert":{"action-loc-key":"Open","body":"Hello"}}}`
Zusätzliches Array als selbst erstellte Nutzlast	`{"aps":{"alert":{"body":"Hello"}},"key1":"value1","key2":["a","b","c"]}`

▸ *Tabelle 16.1: Beispiele für JSON-Nutzdaten*

- Die gesamte Nutzlast ist ein Dictionary. Dictionarys bestehen aus Schlüssel-Wert-Paaren, die in geschweifte Klammern gesetzt sind, also `{Schlüssel:Wert, Schlüssel:Wert,Schlüssel:Wert,...}`.
- Die einzelnen Schlüssel-Wert-Paare sind durch Kommata getrennt.
- Strings stehen in doppelten Anführungszeichen, Zahlen nicht. Zu den reservierten Wörtern zählen `true`, `false` und `null`. Sie werden nicht in Anführungszeichen gesetzt.
- Arrays bestehen aus einer Liste von Elementen in eckigen Klammern, also `[Element, Element, Element,...]`.
- Die folgenden Zeichen müssen in Strings durch einen umgekehrten Schrägstrich maskiert werden: , `"` `\` `/`.
- Beim Senden von Nachrichten können Sie die Zeichen für Wagenrücklauf (`\r`) und Zeilenwechsel (`\n`) aus der Nutzlast entfernen.
- Leerzeichen sind optional. Sie können Platz sparen, indem Sie auf Leerzeichen zwischen einzelnen Elementen verzichten.
- Das Dictionary `aps` erscheint im Ordner der obersten Ebene. Die einfachste mögliche Nutzlast ist also etwas wie `{aps:{}}`.

16.4.3 Eigene Daten

Solange in der Nutzlast noch Platz ist, können Sie unter Beachtung des begrenzten Byte-Budgets zusätzliche Informationen in der Form von Schlüssel-Wert-Paaren senden. Wie Sie in *Tabelle 16.1* gesehen haben, kann es sich dabei um Arrays und Dictionarys sowie um Strings, Zahlen und Konstanten handeln. Wie diese zusätzlichen Informationen verwendet und gedeutet werden, bestimmen Sie selbst. Das gesamte Nutzdaten-Dictionary wird an Ihre Anwendung gesendet, sodass der Methode `application:didReceiveRemoteNotification:` über das Benutzer-Dictionary alle übergebenen Informationen zur Verfügung stehen.

Ein Dictionary mit Schlüssel-Wert-Paaren aus Ihrer Feder muss *nicht* notwendigerweise ein Benachrichtigungsdialogfeld anzeigen. Allerdings geben Sie durch ein solches Dialogfeld den Benutzern die Möglichkeit, Ihre Anwendung zu öffnen, falls sie noch nicht läuft. Ist die Anwendung bereits gestartet, gehen die Schlüssel-Wert-Paare im Rahmen des Nutzlast-Dictionarys dort ein.

16.4.4 Daten beim Start empfangen

Wenn Ihr Client eine Benachrichtigung empfängt, wird durch Antippen der Aktionsschaltfläche (laut Voreinstellung **View**) die Anwendung gestartet. Danach sendet das iPhone der Anwendungsdelegierung einen optionalen Callback. Die Delegierung ruft das Benachrichtigungs-Dictionary über eine Implementierung der Methode `application:didFinishLaunchingWithOptions:` ab. Leider kann es sein, dass diese Methode nicht korrekt funktioniert. Im Folgenden finden Sie daher neben der normalen Vorgehensweise zum Abrufen von Benachrichtigungsinformationen auch eine Notlösung.

Normalerweise übergibt das iPhone das Benachrichtigungs-Dictionary über den Parameter der Startoptionen an die Delegierungsmethode. Für Remote-Benachrichtigungen ist dies der offizielle Callback zum Abruf von Daten aus dem Start eines Meldungsfeldes. Die Methode `didReceiveRemoteNotificaton:` wird nicht aufgerufen, wenn das iPhone eine Benachrichtigung empfängt, die Anwendung aber nicht läuft.

Die `finishLaunching`-Methode wurde so entworfen, dass sie mit zwei völlig verschiedenen Situationen umgehen kann. Erstens kümmert sie sich um den Start des Benachrichtigungsdialogfeldes, sodass Sie das Nutzdaten-Dictionary abrufen und die gesendeten Daten verwenden können. Zweitens eignet sie sich aber auch für den Anwendungsstart über `openURL:`. Wenn Ihre Anwendung ein URL-Schema veröffentlicht hat und dieses Schema von einem anderen Programm verwendet wird, kann die Anwendungsdelegierung den Start mit dieser Methode handhaben.

In jedem Fall muss die Methode einen booleschen Wert zurückgeben, in der Regel `YES`, wenn die Anforderung verarbeitet werden kann, und anderenfalls `NO`. Beim Start einer Remote-Benachrichtigung wird der Wert zwar in Wirklichkeit ignoriert, aber er muss nichtsdestoweniger zurückgegeben werden.

Zurzeit funktioniert eine Implementierung dieser Methode nicht korrekt. Die Anwendung zeigt die Benutzeroberfläche an, hängt sich aber dann auf. Zum Glück gibt es eine einfache Lösung, die sich nicht auf die Callback-Methode stützt. Dazu müssen Sie auf Startbenachrichtigungen lauschen und das damit gesendete Dictionary `userInfo` abfangen. Diese Lösung hat den Vorteil, dass sie geprüft wurde und sich als zuverlässig erwiesen hat. Verfolgen Sie die Entwicklerforen von Apple (*http://devforums.apple.com*), um zu erfahren, wann dieses Problem gelöst wird.

Als Erstes fügen Sie die Anwendungsdelegierung über das standardmäßige `NSNotificationCenter` in der normalen `applicationDidFinishLaunching`-Methode als Listener hinzu.

```
[[NSNotificationCenter defaultCenter] addObserver:self
   selector:@selector(launchNotification:)
   name:@"UIApplicationDidFinishLaunchingNotification" object:nil];
```

Anschließend implementieren Sie die Methode für den von Ihnen bereitgestellten Selektor. Hier wartet die Anwendung darauf, dass die grafische Benutzeroberfläche den Ladevorgang beendet, und zeigt dann das Dictionary mit den Benutzerinformationen an, in dem die Daten der Remote-Benachrichtigung gespeichert sind.

```
- (void) launchNotification: (NSNotification *) notification
{
   [self performSelector:@selector(showString:) withObject:
   [[notification userInfo] description] afterDelay:1.0f];
}
```

Zwischen dem Benachrichtigungs-Listener und dem Methoden-Callback können Sie die Benutzerdaten aus den Remote-Benachrichtigungen zuverlässig abrufen. Diese Lösung sollte unabhängig davon funktionsfähig bleiben, ob und wann sich Apple um das Problem mit der Methode `didFinishLaunchingWithOptions` kümmert.

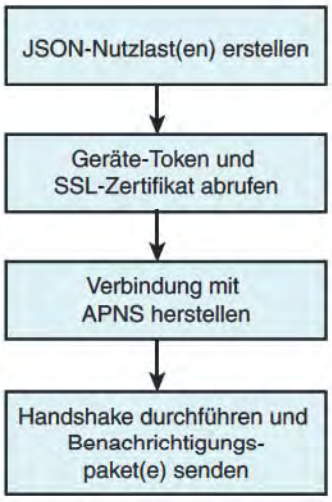
Hinweis

Wenn der Benutzer auf **Close** tippt und die Anwendung später öffnet, wird die Benachrichtigung nicht gesendet. Stattdessen müssen Sie manuell Kontakt mit dem Server aufnehmen, um neue Benutzerinformationen abzurufen. Es ist auch nicht garantiert, dass eine Anwendung eine Benachrichtigung erhält. Selbst wenn der Benutzer nicht auf **Close** tippt, kann die Benachrichtigung einfach verloren gehen. Gestalten Sie Ihre Anwendungen stets so, dass sie zur Aktualisierung des Programms und der Daten nicht einzig und allein auf den Empfang von Push-Benachrichtigungen angewiesen sind.

16.5 Rezept: Benachrichtigungen senden

Der Benachrichtigungsvorgang läuft in mehreren Schritten ab (siehe *Abbildung 16.12*). Als Erstes erstellen Sie die JSON-Nutzdaten, wie Sie im letzten Abschnitt gelesen haben. Danach rufen Sie das SSL-Zertifikat und das Geräte-Token für das iPhone ab, zu dem Sie diese Daten senden möchten. Wie Sie diese beiden Elemente speichern, ist Ihnen überlassen, aber denken Sie daran, dass es sich um sensible Informationen handelt. Öffnen Sie eine sichere Verbindung zum APNS-Server, führen Sie den Handshake aus, senden Sie das Benachrichtigungspaket, und schließen Sie die Verbindung wieder.

Dies ist die elementarste Form der Kommunikation, die voraussetzt, dass Sie nur eine einzige Nutzlast zu senden haben. Sie können jedoch auch eine Sitzung einrichten und mehrere Pakete auf einmal senden, doch dies bleibt Ihnen – ebenso wie das Schreiben von Diensten in anderen Sprachen als Objective-C – als Übungsaufgabe überlassen. In den Apple-Entwicklerforen (*devforums.apple.com*) finden Sie Diskussionen über Push-Provider und einen hervorragenden Ausgangspunkt für die Suche nach Beispielcode für PHP, Perl und andere Sprachen.



▸ *Abbildung 16.12: Die erforderlichen Schritte zum Senden von Remote-Benachrichtigungen*

Beachten Sie, dass APNS auf eine rasche Abfolge von Verbindungen, die wiederholt aufgebaut und wieder geschlossen werden, nicht zu Ihrer Zufriedenheit reagieren mag. Wenn Sie mehrere Benachrichtigungen auf einmal senden müssen, tun Sie das in einer einzigen Sitzung, da APNS Ihre Push-Zustellungen sonst mit einem DNS-Angriff verwechseln könnte.

Rezept 16.2 zeigt, wie Sie eine einzige Nutzlast an APNS senden. Darin können Sie die Schritte erkennen, die zur Umsetzung des vierten und letzten Punktes in *Abbildung 16.12* erforderlich sind. Das Rezept baut auf Code auf, den Stefan Hafeneger entwickelt hat, und nutzt den Beispiel-Quellcode *ioSock* von Apple.

Wie der Server eingerichtet wird, hängt sehr stark von den Sicherheitsanforderungen, den Datenbanken, der Organisation und der Programmiersprache ab. *Rezept 16.2* zeigt die Mindestanforderung zur Implementierung der Serverfunktion. Sie können es als Vorlage verwenden, um Ihren eigenen Server in der Form zu schreiben, die sich für Sie eignet.

16.5.1 Sandbox- und Produktionsumgebung

Für Push-Benachrichtigungen bietet Apple sowohl eine Sandbox-Umgebung zur Entwicklung als auch eine Produktionsumgebung zur Verteilung, wofür Sie jeweils eigene SSL-Zertifikate erwerben müssen. In der Sandbox können Sie Anwendungen entwickeln und testen, bevor Sie sie in den App Store einstellen. Diese Umgebung verfügt nur über eine kleine Anzahl von Servern und ist nicht für großmaßstäbliche Tests gedacht. Das Produktionssystem ist für bereitgestellte und im App Store akzeptierte Anwendungen reserviert.

- Die Sandbox-Server befinden sich unter *gateway.sandbox.push.apple.com*, Port 2195.
- Die Produktionsserver befinden sich unter *gateway.push.apple.com*, Port 2195.

```
// Auf der Grundlage von Code des Entwicklers Stefan Hafeneger
- (BOOL) push: (NSString *) payload
{
   otSocket socket;
   SSLContextRef context;
   SecKeychainRef keychain;
   SecIdentityRef identity;
   SecCertificateRef certificate;
   OSStatus result;

   // Prüft, ob ein Geräte-Token vorhanden ist
   if (!self.deviceTokenID)
   {
      printf("Error: Device Token is nil\n");
      return NO;
   }

   // Prüft, ob ein Zertifikat vorhanden ist
   if (!self.certificateData)
   {
      printf("Error: Certificate Data is nil\n");
      return NO;
   }

   // Richtet eine Verbindung mit dem Server ein
   PeerSpec peer;
   if (self.useSandboxServer)
   {
      printf("Sending message via sandbox\n");
      result = MakeServerConnection("gateway.sandbox.push.apple.com",
         2195, &socket, &peer);
   }
   else
      result = MakeServerConnection("gateway.push.apple.com", 2195,
         &socket, &peer);
   if (result)
   {
      printf("Error creating server connection\n");
      return NO;
   }
   // Erstellt einen neuen SSL-Kontext
   result = SSLNewContext(false, &context);
   if (result)
```

```
{
   printf("Error creating SSL context\n");
   return NO;
}

// Legt Callback-Funktionen für den SSL-Kontext fest
result = SSLSetIOFuncs(context, SocketRead, SocketWrite);
if (result)
{
   printf("Error setting SSL context callback functions\n");
   return NO;
}

// Legt eine SSL-Kontextverbindung fest
result = SSLSetConnection(context, socket);
if (result)
{
   printf("Error setting the SSL context connection\n");
   return NO;
}

// Legt den Serverdomänennamen fest
result = SSLSetPeerDomainName(context,
   "gateway.sandbox.push.apple.com", 30);
if (result)
{
   printf("Error setting the server domain name\n");
   return NO;
}

// Öffnet den Schlüsselbund
result = SecKeychainCopyDefault(&keychain);
if (result)
{
   printf("Error accessing keychain\n");
   return NO;
}

// Erstellt das Zertifikat aus den Daten
CSSM_DATA data;
data.Data = (uint8 *)[self.certificateData bytes];
data.Length = [self.certificateData length];
result = SecCertificateCreateFromData(&data, CSSM_CERT_X_509v3,
   CSSM_CERT_ENCODING_BER, &certificate);
if (result)
```

```
{
   printf("Error creating certificate from data\n");
   return NO;
}

// Erstellt eine Identität
result = SecIdentityCreateWithCertificate(keychain, certificate,
   &identity);
if (result)
{
   printf("Error creating identity from certificate\n");
   return NO;
}

result = SSLSetEnableCertVerify(context, NO);
if (result)
{
   printf("Error disabling cert verify\n");
   return NO;
}

// Richtet das Clientzertifikat ein
CFArrayRef certificates = CFArrayCreate(NULL,
   (const void **)&identity, 1, NULL);
result = SSLSetCertificate(context, certificates);
if (result)
{
   printf("Error setting the client certificate\n");
   CFRelease(certificates);
   return NO;
}

CFRelease(certificates);

// Führt den SSL-Handshake durch
do {result = SSLHandshake(context);}
   while(result == errSSLWouldBlock);

// Wandelt den String in Geräte-Token-Daten um
NSMutableData *deviceToken = [NSMutableData data];
unsigned value;
NSScanner *scanner = [NSScanner
   scannerWithString:self.deviceTokenID];
while(![scanner isAtEnd]) {
   [scanner scanHexInt:&value];
```

```
        value = htonl(value);
        [deviceToken appendBytes:&value length:sizeof(value)];
    }

    // Erstellt C-Eingabevariablen
    char *deviceTokenBinary = (char *)[deviceToken bytes];
    char *payloadBinary = (char *)[payload UTF8String];
    size_t payloadLength = strlen(payloadBinary);

    // Bereitet die Nachricht vor
    uint8_t command = 0;
    char message[293];
    char *pointer = message;
    uint16_t networkTokenLength = htons(32);
    uint16_t networkPayloadLength = htons(payloadLength);

    // Stellt die Nachricht zusammen
    memcpy(pointer, &command, sizeof(uint8_t));
    pointer += sizeof(uint8_t);
    memcpy(pointer, &networkTokenLength, sizeof(uint16_t));
    pointer += sizeof(uint16_t);
    memcpy(pointer, deviceTokenBinary, 32);
    pointer += 32;
    memcpy(pointer, &networkPayloadLength, sizeof(uint16_t));
    pointer += sizeof(uint16_t);
    memcpy(pointer, payloadBinary, payloadLength);
    pointer += payloadLength;

    // Sendet die Nachricht über SSL
    size_t processed = 0;
    result = SSLWrite(context, &message, (pointer - message),
        &processed);
    if (result)
    {
        printf("Error sending message via SSL.\n");
        return NO;
    }
    else
    {
        printf("Message sent.\n");
        return YES;
    }
}
```

► *Rezept 16.2: Nutzdaten auf die APNS-Server übertragen*

Hinweis

Beachten Sie, das es sich beidem eben gezeigten Rezept um eine reine Konsolenanwendung handelt. Um diese auszuführen, starten Sie das Terminal-Programm Ihres Mac (`/Programme/Dienstprogramme/Terminal.app`) und öffnen dort den `build`-Ordner Ihres Projektverzeichnisses. Starten Sie dann die Anwendung mit `pushutil -help`, um z. B. eine Hilfe zum Programm zu erhalten.

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 16 und öffnen das Projekt zu diesem Rezept.

16.6 Rezept: Push-Benachrichtigungen im Einsatz

Nachdem Sie den Client wie in *Rezept 16.1* und Routinen zum Senden von Benachrichtigungen wie die in *Rezept 16.2* eingerichtet haben, können Sie einen Dienst bereitstellen. *Rezept 16.3* erstellt einen Twitter-Client, der wiederholt RSS-Feeds von *search.twitter.com* abfragt und Benachrichtigungen sendet, sobald ein neuer Tweet vorliegt (siehe *Abbildung 16.13*).

Dieser Code baut auf der Push-Routine von *Rezept 16.2* und dem XML-Parser aus *Rezept* 13.13 auf. Dieses Hilfsprogramm ruft die Twitter-Suchdaten als XML-Baum ab und sucht den ersten Knoten des Types »entry«, da Twitter die einzelnen Tweets auf diese Weise speichert.

▸ *Abbildung 16.13: Twitter bildet eine ideale Möglichkeit, um den Abruf von RSS-Feeds zu testen.*

Als Nächstes erstellt der Code einen String aus dem Namen des Tweet-Autors (aus dem Blatt »name«) und dem Inhalt (Blatt »title«) und fügt eine JSON-Version dieses Strings als Benachrichtigungstext zum Dictionary `aps > alert` hinzu. Im Nutzlast-Dictionary `aps` sind ein Sound und ein Meldungsfeld mit einer Schaltfläche festgelegt.

Die Anwendung läuft in einer Schleife mit einem Verzögerungswert, der über ein Kommandozeilenargument festgelegt wird. Alle *n* Sekunden (dieser Wert wird durch das zweite Kommandozeilenargument bestimmt) sucht sie nach einem neuen Tweet und überträgt ihn über APNS, wenn sie fündig wird. *Abbildung 16.13* zeigt dieses Programm, wie es eine Benachrichtigung über einen Tweet auf einem Client-iPhone ausgibt.

```
#define TWEET_FILE [NSHomeDirectory()\
   stringByAppendingPathComponent:@".tweet"]
#define URL_STRING \
   @" http://search.twitter.com/search.atom\
   ?q=+ericasadun+OR+sadun++-sadunalpdag+-alpdag
#define SHOW_TICK NO
#define CAL_FORMAT @"%Y-%m-%dT%H:%M:%SZ"

int main (int argc, const char * argv[]) {

   if (argc < 2)
   {
      printf("Usage: %s delay-in-seconds\n", argv[0]);
      exit(-1);
   }

   NSAutoreleasePool *pool = [[NSAutoreleasePool alloc] init];

   // Ruft Zertifikat und Geräteinformation aus dem aktuellen
   // Verzeichnis ab, das in pushutil festgelegt ist
   char wd[256];
   getwd(wd);
   NSString *cwd = [NSString stringWithCString:wd]
      encoding:NSUTF8StringEncoding];
   NSArray *contents = [[NSFileManager defaultManager]
      directoryContentsAtPath:cwd];

   NSArray *dfiles = [contents pathsMatchingExtensions:
      [NSArray arrayWithObject:@"devices"]];
   if (![dfiles count])
   {
      printf("Error retrieving device token\n");
      exit(-1);
   }
```

```
NSDictionary *dict = [NSDictionary dictionaryWithContentsOfFile:
   [cwd stringByAppendingPathComponent:[dfiles lastObject]]];
if (!dict || ([[dict allKeys] count] < 1))
{
   printf("Error retrieving device token\n");
   exit(-1);
}
[APNSHelper sharedInstance].deviceTokenID = [dict objectForKey:
   [[dict allKeys] objectAtIndex:0]];

NSArray *certs = [contents pathsMatchingExtensions:
   [NSArray arrayWithObject:@"cer"]];
if ([certs count] < 1)
{
   printf("Error finding SSL certificate\n");
   exit(-1);
}
NSString *certPath = [certs lastObject];
NSData *dCert = [NSData dataWithContentsOfFile:certPath];
if (!dCert)
{
   printf("Error retrieving SSL certificate\n");
   exit(-1);
}
[APNSHelper sharedInstance].certificateData = dCert;

// Richtet die Verzögerung ein
int delay = atoi(argv[1]);
printf("Initializing with delay of %d\n", delay);

// Richtet die Dictionarys ein
NSMutableDictionary *mainDict = [NSMutableDictionary dictionary];
NSMutableDictionary *payloadDict =
   [NSMutableDictionary dictionary];
NSMutableDictionary *alertDict = [NSMutableDictionary dictionary];
[mainDict setObject:payloadDict forKey:@"aps"];
[payloadDict setObject:alertDict forKey:@"alert"];
[payloadDict setObject:@"ping1.caf" forKey:@"sound"];
[alertDict setObject:[NSNull null] forKey:@"action-loc-key"];

while (1 > 0)
{
   NSAutoreleasePool *wadingpool =
      [[NSAutoreleasePool alloc] init];
   TreeNode *root = [[XMLParser sharedInstance] parseXMLFromURL:
      [NSURL URLWithString:URL_STRING]];
```

```
TreeNode *found = [root objectForKey:@"entry"];

if (found)
{
   // Ruft den String für den Tweet ab
   NSString *tweetString = [NSString stringWithFormat:
      @"%@-%@", [found leafForKey:@"name"],
      [found leafForKey:@"title"]];

   // Ruft das Veröffentlichungsdatum ab
   NSString *dateString = [found leafForKey:@"published"];
   NSCalendarDate *date = [NSCalendarDate dateWithString:
      dateString calendarFormat:CAL_FORMAT];

   // Ruft die gespeicherten Daten ab
   NSString *prevDateString = [NSString
      stringWithContentsOfFile: TWEET_FILE
      encoding:NSUTF8StringEncoding error:nil];
   NSCalendarDate *pDate = [NSCalendarDate dateWithString:
      prevDateString calendarFormat:CAL_FORMAT];

   // Ruft den Tweet nur ab, wenn seine Daten
   // noch nicht gespeichert sind oder er ein anderes Datum
   // aufweist
   if (!pDate || ![pDate isEqualToDate:date])
   {
      // Aktualisiert die Anzeige mit Angaben zum neuen Tweet
      NSLog(@"\nNew tweet from %@\n \"%@\"\n\n",
         [found leafForKey:@"name"],
         [found leafForKey:@"title"]);

      // Speichert die Veröffentlichungszeit des Tweets
      [dateString writeToFile:TWEET_FILE atomically:YES
         encoding:NSUTF8StringEncoding error:nil];

      // Überträgt den Tweet
      [alertDict setObject:jsonescape(tweetString)
         forKey:@"body"];
      [[APNSHelper sharedInstance] push: [JSONHelper
         jsonWithDict:mainDict]];
   }
}

root = nil;
found = nil;
```

```
        [wadingpool drain];

        [NSThread sleepForTimeInterval:(double) delay];
        if (SHOW_TICK) printf("tick\n");
    }

    [pool drain];
    return 0;
}
```

▸ *Rezept 16.3: Remote-Benachrichtigungen in einem einfachen Twitter-Hilfsprogramm (in der Konsole auszuführen)*

Hinweis

Beachten Sie, dass es sich bei dem eben gezeigten Rezept um eine reine Konsolenanwendung handelt. Um diese auszuführen, starten Sie das Terminal-Programm Ihres Mac (`/Programme/Dienstprogramme/Terminal.app`) und öffnen dort den `build`-Ordner Ihres Projektverzeichnisses. Starten Sie dann die Anwendung mit `pushtweet`, gefolgt von der Anzahl der Sekunden, für deren Dauer der Tweet überwacht werden soll.

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 16 und öffnen das Projekt zu diesem Rezept.

16.7 Der Feedback-Dienst

Apps sind nichts für die Ewigkeit. Die Benutzer können auf ihren iPhones jederzeit Anwendungen hinzufügen, entfernen und ersetzen. Für den APNS ist es sinnlos, Benachrichtigungen auf iPhones zuzustellen, auf denen die entsprechende Anwendung gar nicht mehr vorhanden ist. Als Push-Provider ist es Ihre Aufgabe, inaktive Geräte-Tokens von der Liste der aktiven iPhones zu entfernen. Apple schreibt dazu: »APNS überwacht, wie sorgfältig die Provider den Feedback-Dienst nutzen, um keine weiteren Push-Benachrichtigungen mehr an nicht existierende Anwendungen auf einem Gerät zu senden.« Big Brother beobachtet Sie *wirklich*.

Apple bietet eine einfache Möglichkeit für den Umgang mit inaktiven Geräte-Token. Wenn die Benutzer Apps auf ihren Geräten deinstallieren, funktionieren die Push-Benachrichtigungen nicht mehr. Apple verfolgt diese erfolglosen Zustellversuche nach und stellt Berichte darüber auf dem APNS-Feedback-Server bereit. Der Feedback-Dienst von APNS führt die Geräte auf, die keine Benachrichtigungen mehr empfangen. Als Provider müssen Sie diese Berichte regelmäßig abrufen und die aufgeführten Geräte-Token aussortieren.

Für den Feedback-Server gibt es wie beim Benachrichtigungsserver sowohl eine Sandbox- als auch eine Produktionsadresse, nämlich *feedback.push.apple.com* (Port 2196) und *feedback.sandbox.push.apple.com*. Sie nehmen auf die gleiche Weise über ein SSL-Zertifikat Verbindung mit dem Server auf wie zum Senden von Benachrichtigungen. Nach dem Handshake lesen Sie sogleich die Ergebnisse, denn der Server sendet die Daten sofort, ohne dass eine ausdrückliche Anforderung von Ihrer Seite notwendig wäre.

Die Feedback-Daten umfassen 38 Bytes und setzen sich aus der Zeitangabe (vier Bytes), der Token-Länge (zwei Bytes) und dem Token selbst (32 Bytes) zusammen. Am Zeitstempel können Sie ablesen, wann APNS zum ersten Mal festgestellt hat, dass die Anwendung nicht mehr auf dem Gerät vorhanden ist. Diese Angabe erfolgt in Form einer UNIX-Standardepoche, nämlich der Anzahl der Sekunden seit Mitternacht des 1. Januar 1970. Das Geräte-Token ist im Binärformat gespeichert. Sie müssen es in eine Hexadezimaldarstellung umwandeln, damit es Ihren üblichen Geräte-Tokens entspricht, falls Sie Strings zur Speicherung der Token-Daten verwenden. Zurzeit können Sie die Längenangabe ignorieren. Sie beträgt immer 0 oder 32 Byte, da das Token 32 Byte lang ist.

```
// SSL-Nachricht abrufen
size_t processed = 0;
char buffer[38];
do
{
   // Nächstes Element abrufen
   result = SSLRead(context. buffer. 38. &processed):
   if (result) break;

   // Datum aus den Daten gewinnen
   char *b = buffer;
   NSTimeInterval ti = ((unsigned char)b[0] << 24) +
      ((unsigned char)b[1] << 16) +
      ((unsigned char)b[2] << 8) +
      (unsigned char)b[3];
   NSDate *date = [NSDate dateWithTimeIntervalSince1970:ti];

   // Geräte-ID abrufen
   NSMutableString *deviceID = [NSMutableString string];
   b += 6;
   for (int i = 0; i < 32; i++) [
      deviceID appendFormat:@"%02x", (unsigned char)b[i]];

   // Dictionary zu den Ergebnissen hinzufügen
   [results addObject:
      [NSDictionary dictionaryWithObject:date
      forKey:deviceID]];

} while (processed > 0);
```

Hinweis

Suchen Sie in der Organizer-Konsole von Xcode nach »aps«, um APNS-Fehlermeldungen zu finden.

16.8 Push-Benachrichtigungen beim Entwurf berücksichtigen

Denken Sie beim Entwerfen von Anwendungen mit Push-Benachrichtigungen an das Wachstum des Kundenstamms. Für die normale Datenverarbeitung ist das nicht von Belang, denn das Programm läuft auf einem Gerät und nutzt dessen lokale CPU. Wenn der Entwickler 10.000 weitere Kopien der Anwendung bereitstellt, muss er nur in einen umfangreicheren technischen Support investieren.

Bei Push-Benachrichtigungen kommt es jedoch auf die Größe des Kundenstamms an. Es spielt durchaus eine Rolle, ob Sie 10.000, 100.000 oder 1.000.000 Benutzer haben, denn der Entwickler muss die Dienstschicht bereitstellen, die die Operationen für jede einzelne Einheit ausführt. Je mehr Benutzer bedient werden müsen, umso größer sind die Kosten. Denken Sie daran, dass solche Dienste äußerst zuverlässig sein müssen und dass die Benutzer längere Ausfallzeiten nicht hinnehmen.

Betrachten wir als Beispiel eine Anwendung mit lediglich 10.000 Benutzern. Wenn die Aktualisierungsprüfung alle 15 Minuten stattfindet, wird die Anwendung etwa eine Million Mal am Tag genutzt. Bei Diensten, für die die Aktualität von entscheidender Bedeutung ist, können solche Überprüfungen sogar im Abstand weniger Minuten oder sogar mehrmals pro Minute erfolgen. Mit der Rechenlast steigen auch die Kosten für das Hosting. Cloud-Computing ist zwar eine hervorragende Lösung für solche Bedürfnisse, aber dafür sind hohe Preise für die Entwicklung, Wartung und den täglichen Betrieb zu zahlen.

Noch wichtiger als die Zuverlässigkeit ist die Sicherheit. Viele Dienste erfordern sichere Anmeldeinformationen, die nicht nur auf dem Gerät gespeichert werden, sondern zu dem Dienst hochgeladen werden müssen. Selbst wenn Ihr Dienst keine Authentifizierung dieser Art benötigt, stellt doch schon das Geräte-Token, über das der Dienst Kontakt mit einem bestimmten iPhone aufnimmt, eine sensible Information dar. Sollte dieser Bezeichner in falsche Hände geraten, könnten Spammer damit unerwünschte Nachrichten senden. Jeder Entwickler, der sich auf diesem Gebiet betätigt, muss diese möglichen Gefahren ernst nehmen und extrem sichere Lösungen für die Speicherung und den Schutz dieser Informationen anbieten.

All diese Bedenken zusammengenommen machen deutlich, dass Push-Benachrichtigungen eine ernste Angelegenheit sind. Manche kleinere Entwicklungsfirmen werden für Apps, die auf Benachrichtigungen über neue Informationen angewiesen sind, sogar ganz auf Push-Nachrichten verzichten. Neben der Infrastruktur und den Sicherheitsbedenken kann allein schon die Menge an Arbeit, die dazu erforderlich ist, um solche Dienste korrekt anzubieten, das Budget solcher Entwickler übersteigen. Drittanbieter wie *Key Lime Tie* (*keylimetie.com*) und *Urban Airship* (*urbanairship.com*) bieten eine einsatzbereite Push-Infrastruktur zu erschwinglichen Preisen an. Diese Firmen kümmern sich für Sie um die Bereitstellung der Push-Benachrichtigungen.

Auf der anderen Seite verwenden viele Entwickler Push-Benachrichtigungen für gelegentliche optionale Benachrichtigungen, z. B. um die Benutzer darüber zu informieren, dass im App Store eine neue Version bereitsteht, oder um Tipps zur Nutzung des Produkts zu senden. Wie gut iPhone-Benutzer auf so etwas anspringen, wird sich noch zeigen.

16.9 Zusammenfassung

In diesem Kapitel haben Sie Push-Benachrichtigungen kennengelernt und dabei sowohl gesehen, wie Sie den Client erstellen, als auch die Seite des Providers betrachtet. Sie haben erfahren, welche Arten von Benachrichtigungen Sie senden können und wie Sie die Nutzdaten erstellen, in denen diese Benachrichtigungen auf die Geräte übertragen werden. Darüber hinaus haben Sie gesehen, wie Sie Anwendungen registrieren und die Registrierung aufheben, wie Sie den Benutzern die Möglichkeit geben, den Dienst zuzulassen oder abzulehnen, und wie Sie ein Provider-Programm schreiben, das Twitter-Einträge sendet.

Ein Großteil dessen, was es über die Bereitstellung von Push-Benachrichtigungen zu sagen gibt, würde den Rahmen dieses Buches sprengen. Es ist an Ihnen, einen Server einzurichten und sich um Sicherheit, Bandbreite und Wachstum zu kümmern. Es gibt neben dem hier vorgestellten Objective-C-Beispielcode viele Plattformen und Sprachen, die Sie verwenden können. Doch unabhängig davon finden Sie in den Prinzipien und Rezepten, die in diesem Kapitel besprochen wurden, einen guten Ausgangspunkt. Sie wissen jetzt, welche Probleme es gibt und wie alles funktionieren soll. Es liegt jetzt an Ihnen, die Einzelteile zusammenzusetzen.

- Die großen Vorteile von Push-Benachrichtigungen sind die sofortige Aktualisierung und Darstellung. Wie SMS-Nachrichten lassen sie sich nur schwer übersehen, wenn sie auf dem iPhone eingehen. Wenn Ihre Anwendung jedoch eine solche Unmittelbarkeit nicht braucht, ist es kein Problem, auf Push-Benachrichtigungen zu verzichten.
- Schützen Sie das SSL-Zertifikat und die Geräte-Token! Es ist zwar noch zu früh, um vorherzusagen, wie Apple auf Sicherheitslücken reagieren wird, aber die Erfahrung lehrt, dass diese Reaktion nicht sehr angenehm ausfallen wird.
- Wenn Sie den Benutzern einen Dienst versprochen haben, dann müssen Sie ihn auch kontinuierlich anbieten. Schätzen Sie in Ihrem Geschäftsplan ab, was alles erforderlich ist, um über einen längeren Zeitraum hinweg Benachrichtigungen auszugeben, und wie Sie dies finanzieren wollen. Verbraucher nehmen ausgedehnte Ausfallzeiten nicht hin, weshalb Ihr Dienst äußerst zuverlässig sein muss.
- Bedenken Sie bei der Gestaltung das Wachstum. Auch wenn Ihre Anwendung zu Anfang nicht gleich Zehntausende von Benutzern hat, müssen Sie doch sowohl einen großen als auch einen bescheidenen Erfolg der Anwendung einkalkulieren. Richten Sie ein System ein, das mit dem Benutzerstamm wachsen kann.

17 Core Location und MapKit

Core Location versorgt das iPhone auf Anforderung mit Positionsdaten aus verschiedenen Quellen, während MapKit Möglichkeiten zur interaktiven Nutzung von Kartenmaterial in der Anwendung gibt, wodurch die Benutzer kommentierte Landkarten einsehen und bearbeiten können. Mit Core Location und MapKit können Sie Anwendungen entwickeln, die die Benutzer einsetzen können, um sich mit anderen Personen zu treffen, nach bestimmten Örtlichkeiten zu suchen und standortspezifische Informationen zu liefern. In diesem Kapitel lernen Sie diese Frameworks zur Positionsbestimmung kennen und erfahren, wie Sie sie in Ihre iPhone-Anwendungen einbauen können.

17.1 Wie funktioniert Core Location?

Die Position ist von Bedeutung, und dies ist bei der Gestaltung von Cocoa Touch berücksichtigt worden. *Wo* wir Daten verarbeiten, wird in naher Zukunft fast genauso wichtig sein wie die Frage, *welche* Daten wir verarbeiten und *wie* wir das tun. Das iPhone ist mit seinem Benutzer den ganzen Tag unterwegs und steht sowohl zu Hause als auch auf Reisen zur Verfügung. Dank Core Location können Sie bei der Entwicklung von Anwendungen dieser Mobilität Rechnung tragen.

Core Location geht das Problem der positionsgestützten Programmierung an. Damit können sich Anwendungen in ortsgestützte Web-APIs wie *fireeagle.com*, *outside.in*, *upcoming.org*, *twitter.org* und *flickr.com* einklinken, und es besteht die Möglichkeit, die Benutzer mit geografisch gekennzeichneten Inhalten zu versorgen, sodass sie nach Einrichtungen wie lokalen Restaurants oder nach örtlichen Ereignissen suchen können. Mit der Positionsbestimmung auf Abruf eröffnet sich der mobilen Datenverarbeitung der Zugang zu einer breiten Palette von API-Bibliotheken des Web 2.0.

All diese Merkmale beruhen auf einem wichtigen Aspekt, nämlich der Position. Die Verantwortung dafür, den Benutzern mitzuteilen, wo sie sich befinden, liegt dabei bei Core Location. Das iPhone nutzt verschiedene Möglichkeiten, um den Standort zu bestimmen, und greift dabei auf mehrere Anbieter

zurück, nämlich auf *Skyhook Wireless* (*http://skyhookwireless.com* oder *http://loki.com*), *Google Maps* (*http://maps.google.com/*) und das *GPS-System* (Global Positioning System) des US-Verteidigungsministeriums (*http://tycho.usno.navy.mil/gpsinfo.html*). In den folgenden Abschnitten finden Sie einen Überblick darüber, wie das iPhone den Standort erkennen und anzeigen kann.

17.1.1 GPS-Ortung

Auf den neueren iPhone-Modellen 3G/3GS verfolgt das eingebaute GPS-System Ihre Bewegungen dank einer Reihe von Satelliten in mittlerer Erdumlaufbahn, die dem US-Verteidigungsministerium gehören. Diese Satelliten senden Mikrowellensignale aus, die Ihr iPhone aufnimmt und dazu verwendet, Ihre Position mit hoher Genauigkeit zu berechnen. Wie bei jedem GPS-System erfordert dies eine ungehinderte Verbindung zwischen Ihnen und den Satelliten, weshalb es am besten im Freien ohne Abschattung durch Bäume funktioniert.

GPS-Ortung ist auf der ersten Generation des iPhones und der iPod touch-Serie nicht verfügbar. Diese Geräte müssen auf eine andere Form der Positionsbestimmung zurückgreifen, wie es auch ein iPhone 3G/3GS tut, wenn es kein Satellitensignal empfangen kann.

17.1.2 SkyHook-WiFi-Ortung

In den USA beruht die von Core Location bevorzugte Methode zur Pseudo-GPS-Ortung auf *SkyHook Wireless*. SkyHook ermöglicht eine extrem genaue Positionsbestimmung über WiFi. Wenn ein iPhone die WiFi- und WiMAX-Router kennt, in deren Nähe Sie sich befinden, verwendet es deren MAC-Adressen, um die SkyHook-Datenbanken zu durchsuchen und Ihre Position auf der Grundlage dieser Daten zu bestimmen. Alle iPhone-Modelle, auch der iPod touch, sind WiFi-fähig und können diesen Dienst daher nutzen.

Die WiFi-Datenerfassung funktioniert bei SkyHook wie folgt: SkyHook schickt Mitarbeiter im Auto und zu Fuß aus, die innerhalb des abgedeckten Gebiets, das die meisten Städte in den USA umfasst, die Straßen in Ortsgebieten nach WiFi-Hotspots absuchen. Wenn sie solche *Zugriffspunkte* oder *Access Points* gefunden haben, vermerken sie die Position unter Verwendung einer gewöhnlichen GPS-Ortung und verknüpfen sie mit der WiFi-MAC-Adresse.

Das funktioniert tadellos, solange die WiFi-Router stationär sind, aber überhaupt nicht, wenn jemand seinen WiFi-Router einpackt und mit ihm zum Beispiel nach Kentucky zieht. Das soll jedoch nicht heißen, dass SkyHook-Daten nicht aktualisiert würden. SkyHook bietet eine recht genaue Ortung, und Sie können Ihre Position gewöhnlich auf einige hundert Meter genau bestimmen, auch wenn es immer Menschen geben wird, die mit ihren Routern nach Kentucky oder sonst wohin ziehen. Über das Programm für ehrenamtliche Mitarbeiter von Skyhook können Sie auch direkt Angaben zu Koordinaten und MAC-Adressen einreichen. Nähere Einzelheiten erfahren Sie unter *http://www.skyhookwireless.com/howitworks/submit_ap.php*.

17.1.3 Funkmastenortung

Ein weniger genauer Ansatz zur Ortung beruht auf der Position von Mobilfunkmasten. Hier verwendet das iPhone seine Antenne dazu, die am nächsten liegenden vier oder fünf Mobilfunkmasten zu finden, und berechnet Ihre Position dann anhand der Stärke des vom Turm gesendeten Funksignals. Vermutlich haben Sie eine Ortung über einen Mobilfunkmast schon erlebt: Es handelt sich um die Art Ortung, die Sie ungefähr einen Kilometer entfernt von Ihrem eigentlichen Standort zeigt – sofern Sie nicht gerade direkt neben einem Mobilfunkmast stehen.

iPod Touch-Geräte können die Ortung über Mobilfunkmasten nicht verwenden, da ihnen die GPRS-Funkantenne fehlt, die ein Standardmerkmal aller iPhones ist. Die Funkmastortung dient wegen ihrer geringen Genauigkeit nur als Notlösung, wenn andere Möglichkeiten nicht verfügbar sind.

17.1.4 Internetprovider-Ortung

SkyHook bietet sogar eine vierte Möglichkeit zur Ortung, aber hier handelt es sich um eine, die ich das iPhone nie habe benutzen sehen. Andererseits lebe ich in einem großen Ballungsgebiet, weshalb ich es nie richtig ausprobiert habe. Bei diesem allerletzten Versuch wird der nächste verzeichnete Hauptsitz eines Internetproviders ausfindig gemacht. Es handelt sich um eine absolute Notlösung. Die zurückgegebenen Daten liegen üblicherweise bis zu mehreren Kilometern von Ihrer tatsächlichen Position entfernt – falls Sie nicht gerade zufällig Ihren Internetprovider besuchen.

17.1.5 Die Methoden kombinieren

Das iPhone nähert sich der Position etappenweise. Je nach Grad an Genauigkeit, den Sie verlangen, verwendet es eine Reservemethode. Wenn es Sie mit GPS oder SkyHook WiFi nicht genau orten kann, greift es auf dem iPhone auf die Ortung über Mobilfunkmasten zurück, und wenn das nicht funktioniert, wahrscheinlich auf die Ortung eines Internetproviders über SkyHook. Erst wenn auch das nicht klappt, scheitert die Standortbestimmung.

Die neuesten Versionen des SDK bieten mehrfache asynchrone Erfolgs-Callbacks für jede dieser Methoden. Daher können Sie jederzeit drei bis vier Ergebnisse erhalten. Darüber hinaus laufen diese Methoden weiter, während sich die Position des iPhones ändert. Jeder Callback umfasst ein genaues Maß und gibt an, welches Verfahren verwendet wurde.

Es ist wichtig zu wissen, wie das iPhone vorgeht, denn auf einem iPhone der ersten Generation gehen von zehn Versuchen, Ihre Position zu bestimmen, vielleicht drei oder vier auf WiFi zurück und die übrigen auf die Ortung über Mobilfunkmasten. Auch wenn Sie die gewünschte Genauigkeit auf die höchstmögliche Einstellung setzen, kann es sein, dass Ihnen die bestmögliche Ortung entgeht, falls Sie nicht auf mehrere Callbacks lauschen.

Das kostet vor allem Zeit. Eine Positionsabfrage kann 10 bis 15 Sekunden dauern. Bei der Arbeit mit mehreren Anfragen erfolgen die Bildung eines Mittelwerts und die Wiederholung der besten Ergebnisse am besten im Hintergrund, weit von der Benutzeroberfläche entfernt. Wenn möglich, sollten Sie es vermeiden, Ihren Benutzer darauf warten zu lassen, dass Ihr Programm seine Ortungsanfragen beendet.

Hinweis

Apple verlangt, dass Benutzer beim Start von Core Location alle Ortungsanfragen autorisieren. Liegt die Erlaubnis vor, können Sie Positionsdaten während der gesamten Anwendungssitzung nutzen.

17.2 Rezept: Core Location kurz und bündig

Core Location lässt sich einfach anwenden, wie die folgende einfache Anleitung zeigt. Sie lernen darin Schritt für Schritt, wie Sie ein Programm schreiben, das Positionsdaten abfragt. Diese Vorgehensweise entspricht der üblichen Verwendung. Diese Anleitung und *Rezept 17.1* bilden lediglich Beispiele für die Nutzung der Core Location-Dienste und zeigen Ihnen, wie Sie den Standort eines Benutzers feststellen.

1. Fügen Sie das Core Location-Framework hinzu. Ziehen Sie es in Ihr Xcode-Projekt, und fügen Sie es dem Ordner **Frameworks** in der Spalte **Groups & Files** hinzu. Nehmen Sie die Core Location-Header in Ihren Code auf.
2. Weisen Sie einen Location Manager zu. Legen Sie als seine Delegierung den Hauptansichtscontroller oder die Anwendungsdelegierung fest. Optional können Sie auch den gewünschten Abstandsfilter und die Genauigkeit angeben.

   Der Abstandsfilter gibt die Mindestdistanz in Metern an. Das Gerät muss sich um mindestens diese Strecke bewegen, bevor es eine neue Aktualisierung bekommt. Wenn Sie beispielsweise 5 m angeben, erhalten Sie nur dann neue Ereignisse, wenn das Gerät um diese Entfernung bewegt wurde.

   Die Eigenschaft der Genauigkeit gibt an, wie präzise die Ergebnisse sein müssen, die Sie abfragen. Das heißt nicht, dass der Location Manager irgendeine Garantie über die tatsächliche Genauigkeit gibt. Wenn Sie diese Eigenschaft festlegen, weisen Sie den Manager nur an, Ergebnisse von mindestens dieser Genauigkeit abzurufen (bzw. dies zu versuchen). Ohne Angabe einer Genauigkeit verwendet der Manager die Technologie, die gerade am besten zur Verfügung steht.

   Ist die Genauigkeit für Ihre Anwendung wichtig, informieren Sie den Manager über die Eigenschaft `desiredAccuracy` darüber. Ein hoher Grad an Genauigkeit ist z. B. entscheidend für Anwendungen, die von Fußgängern genutzt werden. Für Benutzer, die in einem Auto fahren, und um festzustellen, in welcher Stadt, in welchem Bundesland oder in welchem Land sich jemand befindet, kann auch eine geringere Genauigkeit ausreichen.
3. Prüfen Sie, ob der Benutzer Core Location aktiviert hat, indem Sie die Eigenschaft `locationSerivcesEnabled` des Location Managers abfragen. Die Benutzer haben die Möglichkeit, Core Location in der Anwendung *Einstellungen* über **Allgemein** ▸ **Ortungsdienste** auszuschalten.

4. Starten Sie die Ortung. Weisen Sie den Location Manager an, die Position zu aktualisieren. Die Delegierungs-Callbacks informieren Sie, wenn die Position bestimmt werden konnte. Dies kann mehrere Sekunden bis zu einer Minute dauern.
5. Verarbeiten Sie die Delegierungs-Callbacks für Ortungsereignisse. Dabei haben Sie es mit zwei Arten von Callbacks zu tun: erfolgreichen Vorgängen, die `CLLocation`-Daten zurückgeben (`locationManager:didUpdateToLocation:fromLocation:`), und Fehlschlägen, die das nicht tun (`locationManager:didFailWithError:`). Fügen Sie diese Delegierungsmethoden zu Ihrem Code hinzu, um Positionsaktualisierungen zu erfassen. *In Rezept 17.1* wird bei erfolgreicher Ortung ein Überblick (`description`) aufgezeichnet, der auch die aktuelle geografische Länge und Breite umfasst.

   Je nach der von Ihnen verlangten Genauigkeit und der verwendeten Ortungsmethode können Sie drei oder vier Callbacks erhalten. Diese Nichtlinearität müssen Sie berücksichtigen.
6. Warten Sie. Die Callbacks treffen asynchron ein, nämlich immer dann, wenn Ortungsdaten verfügbar sind. Die an Ihre Anwendung zurückgegebenen Daten enthalten die Positionsangaben sowie ein Maß für die Genauigkeit.

Testen Sie Core Location-Anwendungen auf einem iPhone-Gerät und nicht im Simulator, denn Letzterer gibt die fest gespeicherten Koordinaten des Apple-Hauptsitzes in Cupertino zurück. Wenn Sie *Rezept 17.1* auf einem Gerät bereitstellen, können Sie die Ergebnisse überprüfen, während Sie mit Ihrem iPhone unterwegs sind.

```
@interface TestBedViewController : UIViewController
   <CLLocationManagerDelegate>
{
   IBOutlet UITextView *textView;
   CLLocationManager *locManager;
}
@property (retain) CLLocationManager *locManager;
@end

@implementation TestBedViewController
@synthesize locManager;

- (void)locationManager:(CLLocationManager *)manager
   didFailWithError:(NSError *)error
{
   // Reagiert auf den (seltenen) Fehlschlag des Location Managers
   NSLog(@"Location manager error: %@", [error description]);
   return;
}

- (void)locationManager:(CLLocationManager *)manager
   didUpdateToLocation:(CLLocation *)newLocation
   fromLocation:(CLLocation *)oldLocation
```

```
{
    // Gibt eine Zusammenfassung der aktuellen Positionsergebnisse aus
    NSLog(@"%@\n", [newLocation description]);
}

- (void) viewDidLoad
{
    // Initialisiert den Location Manager
    self.locManager = [[[CLLocationManager alloc] init] autorelease];
    if (!self.locManager.locationServicesEnabled)
    {
        NSLog(@"User has opted out of location services");
        return;
    }

    self.locManager.delegate = self;
    self.locManager.desiredAccuracy = kCLLocationAccuracyBest;

    // Legt den optionalen Abstandsfilter fest
    self.locManager.distanceFilter = 5.0f; // Angabe in Metern

    // Beginnt mit dem Abruf von Positionsdaten
    [self.locManager startUpdatingLocation];
}
```

▸ *Rezept 17.1: Die geografische Länge und Breite mit Core Location abrufen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 17 und öffnen das Projekt zu diesem Rezept.

17.2.1 Eigenschaften von Ortungsdaten

Jede von dem aktualisierten Ortungs-Callback zurückgegebene `CLLocation`-Instanz weist eine Reihe von Eigenschaften auf, die die Position des Geräts beschreiben. Über die Instanzmethode `description` können die verschiedenen Eigenschaften eines Ortungsobjekts zu einem einzigen Ergebnis in Textform kombiniert werden, wie es in *Rezept 17.1* geschieht. Alternativ können Sie auch die einzelnen Werte Eigenschaft für Eigenschaft abrufen. Es gibt folgende Ortungseigenschaften:

- `altitude` Diese Eigenschaft gibt den Wert der zurzeit gemessenen Höhe als Fließkommazahl in Metern über Meereshöhe zurück. Als Einwohnerin von Denver (der sprichwörtlichen »Mile High City«, die genau eine Meile über dem Meeresspiegel liegt) kann ich Ihnen versichern, dass

die Genauigkeit dieses Wertes bestenfalls grob ist. Verwenden Sie die Ergebnisse aus dieser Eigenschaft mit Vorsicht.

- `coordinate` Die ermittelte geografische Position des Geräts rufen Sie über die Eigenschaft `coordinate` ab. Der Koordinatenwert ist eine Struktur aus den beiden Feldern `latitude` (Breite) und `longitude` (Länge), die beide einen Fließkommawert aufnehmen. Positive Breitenwerte bezeichnen Orte nördlich des Äquators, negative weisen auf eine Position auf der Südhalbkugel hin. Orte mit positiven Längenwerten liegen östlich des Nullmeridians, solche mit negativer Länge westlich davon.
- `course` Der Wert von `course` (0° für Nord, 90° für Ost, 180° für Süd und 270° für West) gibt ungefähr die Richtung an, in die sich das Gerät bewegt. Eine höhere Genauigkeit erzielen Sie jedoch mit `CLHeading`-Instanzen, da diese über den eingebauten Kompass Zugriff auf Magnetfeldmesswerte und die magnetische und die geografische Nordrichtung haben. Hierbei handelt es sich um eine weitere Funktion von Core Location, die weiter hinten in diesem Kapitel noch ausführlicher erklärt wird.
- `horizontalAccuracy` Diese Eigenschaft gibt die Genauigkeit (eigentlich die Unsicherheit oder den Messfehler) der aktuellen Koordinaten an. Stellen Sie sich die zurückgegebenen Koordinaten als den Mittelpunkt eines Kreises mit der horizontalen Genauigkeit als Radius vor, wobei die Position des Geräts irgendwo in diesen Kreis fällt. Je kleiner der Kreis, umso genauer ist die Positionsangabe. Negative Genauigkeitswerte stehen für Messfehler.
- `verticalAccuracy` Diese Eigenschaft ist das Gegenstück der horizontalen Genauigkeit für die Höhenangabe. Sie gibt an, wie weit die wirkliche Höhe von dem angezeigten Wert abweichen kann. Theoretisch sollte sich das Gerät auf der angezeigten Höhe plus/minus dieser Abweichung befinden. In Wirklichkeit aber sind die Höhenmessungen äußerst ungenau, und auch die Angabe der vertikalen Abweichung hat nur wenig mit den tatsächlichen Verhältnissen zu tun.
- `speed` Theoretisch soll dieser Wert die Geschwindigkeit des Geräts in Metern pro Sekunde angeben. In der Praxis sollten Sie diese Eigenschaft eher für Fahrten im Auto als für Spaziergänge benutzen. Die beiden folgenden Rezepte zeigen, wie Sie den groben Wert dieser Eigenschaft nutzen können und wie sich die Geschwindigkeit unabhängig davon ermitteln lässt.
- `timestamp` Diese Eigenschaft gibt an, zu welchem Zeitpunkt die Positionsbestimmung stattgefunden hat. Sie gibt eine `NSDate`-Instanz mit der entsprechenden Zeitangabe zurück.

Hinweis

Kontinuierliche Positionsanfragen verbrauchen viel Energie. Das kann dazu führen, dass der Akku schneller erschöpft wird, wie viele kürzlich im App Store veröffentlichte Anwendungen für Jogger und Radsportler bewiesen haben.

17.3 Rezept: Die Geschwindigkeit messen

Über die von den einzelnen `CLLocation`-Instanzen zurückgegebene Eigenschaft `speed` können Sie die Geschwindigkeit des Geräts verfolgen. *Rezept 17.2* zeigt, wie Sie diese Eigenschaft nutzen. Wenn das Callback des Location Managers die Geräteposition aktualisiert, ruft der Code die Geschwindigkeit ab und zeichnet sie auf. In diesem Code wird die aktuelle Geschwindigkeit in Meilen pro Stunde berechnet, indem der Wert in Meter pro Sekunde mit 2,23693629 multipliziert wird.

Die Methode `viewDidLoad` setzt die gewünschte Genauigkeit auf 10 m, während der in *Rezept 17.1* verwendete Abstandsfilter hier nicht zum Einsatz kommt. Dieses Beispiel eignet sich eher für die Verwendung in einem motorgetriebenen Fahrzeug und nicht für Fußgänger. Wenn Sie gehen, laufen oder Rad fahren, brauchen Sie eine höhere Genauigkeit und eine Vorgehensweise, die ungenaue Messungen ausschließt. Wie das geht, zeigt *Rezept 17.3*.

```
- (void)locationManager:(CLLocationManager *)manager
   didUpdateToLocation:(CLLocation *)newLocation
   fromLocation:(CLLocation *)oldLocation
{
   // Beim Ermitteln einer Geschwindigkeit werden diese Daten in Meilen pro
   // Stunde umgerechnet und aufgezeichnet
   if (newLocation.speed > 0.0f)
   {
      NSString *speedFeedback = [NSString stringWithFormat:
         @"Speed is %0.1f miles per hour",
         2.23693629 * newLocation.speed];
      NSLog(@"%
   }
}

- (void) viewDidLoad
{
   self.locManager = [[[CLLocationManager alloc] init] autorelease];
   if (!self.locManager.locationServicesEnabled)
   {
      NSLog(@"User has opted out of location services");
      return;
   }

   // Richtet die Delegierung und die gewünschte Genauigkeit ein
   self.locManager.delegate = self;
   self.locManager.desiredAccuracy =
      kCLLocationAccuracyNearestTenMeters;

   // Beginnt mit der Erfassung der Ortungsdaten
   [self.locManager startUpdatingLocation];
}
```

▸ *Rezept 17.2: Die Eigenschaft* `speed` *einer* `CLLocation`*-Instanz abrufen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 17 und öffnen das Projekt zu diesem Rezept.

17.4 Rezept: Geschwindigkeit und Abstand berechnen

Für langsame Bewegungen (zumindest langsamer als ein Auto) müssen Sie den Code auf zwei besondere Weisen anpassen. Erstens müssen Sie die gewünschte Genauigkeit auf den bestmöglichen Wert ändern, und zweitens müssen Sie die Eigenschaft `speed` ignorieren und die Geschwindigkeit stattdessen selbst berechnen. In *Rezept 17.3* sind diese beiden Änderungen umgesetzt worden. Dazu wird die zuletzt mit der höchstmöglichen Genauigkeit ermittelte Position nachverfolgt. »Höchstmögliche Genauigkeit« bedeutet in diesem Rezept »innerhalb von 100 m«, also eine wahrscheinliche GPS-Position.

Aus diesen »genauen« Positionen wird durch den Aufruf der `CLLocation`-Methode `getDistanceFrom:` ein Distanzwert berechnet. Die Division dieser Distanz durch die verstrichene Zeit ergibt die Geschwindigkeit. Werte mit geringerer Genauigkeit und solche, bei denen sich das Gerät nicht um mindestens einen Meter bewegt hat, werden von der Methode verworfen.

Für Fußgänger und Radfahrer gibt diese Methode einen genaueren Geschwindigkeitswert an, der allerdings immer noch alles andere als wirklich genau ist. Dies zeigt sich am besten, wenn Sie den Code in der Praxis mit einem 3G oder einem moderneren GPS-fähigen iPhone testen. Falls Sie Core Location-gestützte Anwendungen mit einer größeren Genauigkeit als in diesen Beispielen bereitstellen wollen, müssen Sie die Abtastrate und die Rückmeldung für den tatsächlichen Bedarf optimieren.

```
- (void)locationManager:(CLLocationManager *)manager
   didUpdateToLocation:(CLLocation *)newLocation
   fromLocation:(CLLocation *)oldLocation
{
   if (newLocation.horizontalAccuracy <
      kCLLocationAccuracyHundredMeters) // innerhalb von ca. 100 m
   {
      // Braucht einen Startwert für die Berechnung
      if (self.lastAccurateLocation)
      {
         // Berechnet den zeitlichen und räumlichen Abstand
         NSTimeInterval dTime = [newLocation.timestamp
            timeIntervalSinceDate:
               self.lastAccurateLocation.timestamp];
         float distance = [newLocation
            getDistanceFrom:lastAccurateLocation];
         if (distance < 1.0f) return;

         // Summiert die gesamte Distanz
```

```
            aggregateDistance += distance;

            // Meldet Geschwindigkeit und Distanz
            NSString *reportString = [NSString stringWithFormat:
               @"Speed: %0.1f miles per hour. %0.1f meters.",
               2.23693629 * distance / dTime, aggregateDistance];
            NSLog(@"%
         }

         // Aktualisiert die letzte genaue Position
         self.lastAccurateLocation = newLocation;
      }
   }

   - (void) viewDidLoad
   {
      // Verwendet die höchstmögliche Genauigkeit
      self.locManager = [[[CLLocationManager alloc] init] autorelease];
      if (!self.locManager.locationServicesEnabled)
      {
         NSLog(@"User has opted out of location services");
         return;
      }

      // Initialisiert den Location Manger
      self.locManager.delegate = self;
      self.locManager.desiredAccuracy = kCLLocationAccuracyBest;
      [self.locManager startUpdatingLocation];
      aggregateDistance = 0.0f;
   }
```

► *Rezept 17.3: Geschwindigkeitsinformationen ableiten*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 17 und öffnen das Projekt zu diesem Rezept.

17.5 Rezept: Die Nordrichtung bestimmen

Der eingebaute Location Manager des iPhones kann den berechneten Wert der Eigenschaft `course` zurückgeben, der die aktuelle Bewegungsrichtung angibt, also Nord, Süd, Südost usw., wobei diese Werte die Form einer Fließkommazahl zwischen 0 und 360 annehmen. Dabei steht 0° für Nord, 90° für Ost usw. Dieser Wert wird durch die Nachverfolgung der Positionsänderungen ermittelt. Auf neueren iPhone-Geräten gibt es eine bessere Möglichkeit, die Bewegungsrichtung zu bestimmen. Moderne Modelle verfügen über einen eingebauten Magnetkompass, der die magnetische und die geografische Nordrichtung angibt.

Nicht jedes iPhone kann auf diese Weise die Bewegungsrichtung ermitteln. Ein Kompass wurde zuerst im iPhone 3GS verbaut. Vor dem Abonnieren von Richtungs-Callbacks müssen Sie zunächst prüfen, ob das betreffende Gerät überhaupt die entsprechende Fähigkeit mitbringt. Wenn der Location Manager Richtungsereignisse hervorrufen kann, gibt die Eigenschaft `headingAvailable` den Wert `YES` zurück. Damit können Sie `startUpdatingHeading`-Anforderungen steuern.

```
if (self.locManager.headingAvailable)
   [self.locManager startUpdatingHeading];
```

In Cocoa Touch können Sie Richtungs-Callbacks ebenso filtern wie Distanz-Callbacks. Setzen Sie die Eigenschaft `headingFilter` des Location Managers auf die mindestens erforderliche Winkeländerung, wobei die Angabe in Form einer Fließkommazahl erfolgt. Wenn Sie also keine Richtungsinformationen empfangen möchten, bevor sich das Gerät nicht um mindestens 5° gedreht hat, setzen Sie diese Eigenschaft auf `5.0`. Alle Richtungswerte sind Gradangaben zwischen 0,0 und 360,0. Um einen Richtungswert in Bogenmaß umzuwandeln, dividieren Sie ihn durch 180,0 und multiplizieren ihn mit Pi.

Richtungs-Callbacks geben ein `CLHeading`-Objekt zurück, bei dem Sie die beiden Eigenschaften `magneticHeading` und `trueHeading` für die magnetische und die geografische Nordrichtung abfragen können. Die geografische Nordrichtung zeigt auf den geografischen Nordpol der Erde, die magnetische Nordrichtung auf den Pol des Erdmagnetfelds, dessen Position sich mit der Zeit ändert. Um die geografische Nordrichtung zu berechnen, zieht das iPhone die Abweichung zwischen diesen beiden Richtungen (die *Missweisung, Deviation*) heran.

Auf einem aktivierten iPhone werden die Richtungsinformationen auch dann aktualisiert, wenn der Benutzer in den Einstellungen die Ortungsdienste abgeschaltet hat. Außerdem werden die Benutzer auch nicht um Erlaubnis zur Verwendung dieser Daten gefragt. Kompassinformationen verletzen die Privatsphäre des Benutzers nicht, sodass sie in allen Anwendungen frei zur Verfügung stehen.

Die Eigenschaft `trueHeading` können Sie nur in Verbindung mit der Ortung verwenden, denn das iPhone muss die Position des Geräts kennen, um die Missweisung zur Ermittlung der geografischen Nordrichtung berechnen zu können. Je nachdem, ob sich der Benutzer in Los Angeles, in Perth, in Moskau, in London oder wo auch immer aufhält, hat die Missweisung einen anderen Wert. An manchen Orten sind die Messungen des Magnetkompasses überhaupt nicht zu verwenden. In anomalen Regionen wie Michipicoten Island im Oberen See oder Grants in New Mexico gibt es

Eisenlagerstätten und Lavafahnen, die den Magnetkompass ablenken. Metalle und Magnetfelder, wie sie z. B. von einem Auto, einem Computer oder einem Kühlschrank aufgebaut werden, können das Funktionieren des Kompasses ebenfalls beeinträchtigen. Es gibt verschiedene »Metalldetektor«-Programme im App Store, die diese Deviation nutzen.

Die Eigenschaft `headingAccuracy` gibt den Messfehler an, also den Betrag, um den die tatsächliche Richtung von der angegebenen abweichen kann. Je schmaler dieser Bereich ist, umso genauer ist die Messung. Ein negativer Wert deutet auf einen Fehler beim Messen der Richtung hin.

Mit den Eigenschaften `x`, `y` und `z` von `GLHeading` können Sie Rohwerte für die x-, y- und z-Komponente des Magnetfelds abrufen. Die Werte werden in Mikrotesla gemessen und laut Angabe von Apple in einen Bereich von -128 bis +128 normalisiert. (Angesichts der üblichen Bit-Operationen ist ein Bereich von -128 bis +127 jedoch wahrscheinlicher.) Der Wert entlang einer Achse steht jeweils für die Abweichung vom Verlauf der Magnetfeldlinien.

Rezept 17.4 zieht `CLHeading`-Daten heran, um ein kleines Bild mit einem Pfeil so zu drehen, dass die Spitze stets nach Norden zeigt. Die zugehörige Oberfläche sehen Sie in *Abbildung 17.1*.

▸ *Abbildung 17.1: Der eingebaute Kompass des iPhones und der Code aus Rezept 17.4 sorgen dafür, dass der Pfeil stets nach Norden zeigt.*

```
@implementation TestBedViewController
@synthesize locManager;

// Erfasst Ortungsfehler
- (void)locationManager:(CLLocationManager *)manager
   didFailWithError:(NSError *)error
{
   NSLog(@"Location manager error: %@", [error description]);
```

```
}

// Reagiert auf neue Richtung
- (void)locationManager:(CLLocationManager *)manager
   didUpdateHeading:(CLHeading *)newHeading
{
   // Rechnet die Richtung in Bogenmaß um
   CGFloat heading = -1.0f * M_PI *
      newHeading.magneticHeading / 180.0f;

   // Dreht den Nordpfeil entsprechend
   arrow.transform = CGAffineTransformMakeRotation(heading);
}

// Erlaubt Core Location, bei Bedarf die Gerätekalibrierung anzuzeigen
- (BOOL)locationManagerShouldDisplayHeadingCalibration:
   (CLLocationManager *)manager
{
   return YES;
}

- (void) viewDidLoad
{
   // Initialisiert den Location Manager. Es ist nicht erforderlich, zu
   // prüfen, ob der Benutzer Ortungsdienste zugelassen hat oder nicht.
   self.locManager = [[[CLLocationManager alloc] init] autorelease];
   self.locManager.delegate = self;
   if (self.locManager.headingAvailable)
      [self.locManager startUpdatingHeading];
   else
      arrow.alpha = 0.0f;
}
@end
```

▸ *Rezept 17.4: Die Nordrichtung ermitteln*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 17 und öffnen das Projekt zu diesem Rezept.

17.6 Rezept: Adressen durch Reverse Geocoding ermitteln

Reverse Geocoding bedeutet, die Angabe von Breiten- und Längengraden in für Menschen verständliche Adressen umzuwandeln. MapKit bietet eine Klasse für diesen Zweck, die mithilfe von Google Koordinaten in die Beschreibung von Orten umwandelt. Wenn Sie diese Funktion nutzen, sind Sie an die Geschäftsbedingungen von Google Maps gebunden, die Sie unter *http://code.google.com/apis/maps/iphone/terms.html* nachlesen können.

Eine Anforderung für Reverse Geocoding erfordert wenig mehr, als eine neue Instanz von `MKReverseGeocoder` zuweisen, die Koordinaten und die Delegierung festzulegen und die Instanz anzuweisen, loszulegen. Die Delegierung deklariert das Protokoll `MKReverseGeocoderDelegate` und implementiert die beiden Callbacks (für Erfolg und Fehlschlag) aus *Rezept 17.5*.

Verläuft eine Reverse-Geocoding-Anforderung erfolgreich, liefert der Delegierungs-Callback eine Instanz von `MKPlaceMark`. Dieses Objekt weist ein `addressDictionary` mit Schlüssel-Wert-Paaren auf, die die Adresse beschreiben. *Abbildung 17.2* zeigt den Inhalt dieses Adress-Dictionarys für den Lollipop Lake in Denver.

▶ *Abbildung 17.2: Der Inhalt des Adress-Dictionarys für den Lollipop Lake im Garland Park von Denver in Colorado*

Das `MKPlaceMark`-Objekt bietet dieselben Informationen auch in Form der folgenden einzelnen Eigenschaften außerhalb der Dictionary-Struktur:

- `subThoroughfare` speichert die Hausnummer, z. B. 1600 für 1600 Pennsylvania Avenue (die Adresse des Weißen Hauses).
- `thoroughfare` enthält den Straßennamen, z. B. Pennsylvania Avenue.
- `sublocality` gibt, falls verfügbar, den Stadtteil oder eine Sehenswürdigkeit an, z. B. White House.
- `subAdministrativeArea` ist gewöhnlich der Landkreis, die Gemeinde oder ein anderes Verwaltungsgebiet.

- `locality` enthält die Stadt, z. B. Washington, D.C.
- `administrativeArea` gibt das Bundesland oder den Bundesstaat an, z. B. Maryland oder Virginia.
- `postalcode` ist die Postleitzahl, z. B. 20500.
- `country` speichert den Namen des Landes, z. B. United States.
- `countryCode` ist die Abkürzung des Ländernamens, z. B. US.

Im Adress-Dictionary werden die Namen dieser Eigenschaften mit Großbuchstaben am Anfang geschrieben. So wird aus der Eigenschaft `subThoroughfare` z. B. der Schlüssel `SubThoroughfare`. Dies können Sie in *Abbildung 17.2* erkennen.

Neben diesen Eigenschaften enthält das Adress-Dictionary auch den Eintrag `FormattedAddressLines`, der ein Array vorformatierter Strings für die betreffende Adresse speichert. Diese Strings können Sie zur Anzeige einer Adresse verwenden, z. B. »1600 Pennsylvania Avenue NW«, »Washington, DC 20500«, »USA«.

```
- (void)locationManager:(CLLocationManager *)manager
   didFailWithError:(NSError *)error
{
   NSLog(@"Location manager error: %@", [error description]);
}

- (void)reverseGeocoder:(MKReverseGeocoder *)geocoder
   didFailWithError:(NSError *)error
{
   NSLog(@"Reverse geocoder error: %@", [error description]);
}

- (void)locationManager:(CLLocationManager *)manager
   didUpdateToLocation:(CLLocation *)newLocation
   fromLocation:(CLLocation *)oldLocation
{
   MKReverseGeocoder *geocoder =
      [[MKReverseGeocoder alloc]
         initWithCoordinate:newLocation.coordinate];
   geocoder.delegate = self;
   [geocoder start];
}

- (void)reverseGeocoder:(MKReverseGeocoder *)geocoder
   didFindPlacemark:(MKPlacemark *)placemark
{
   NSLog([placemark.addressDictionary description]);
   if ([geocoder retainCount]) [geocoder release];
}
```

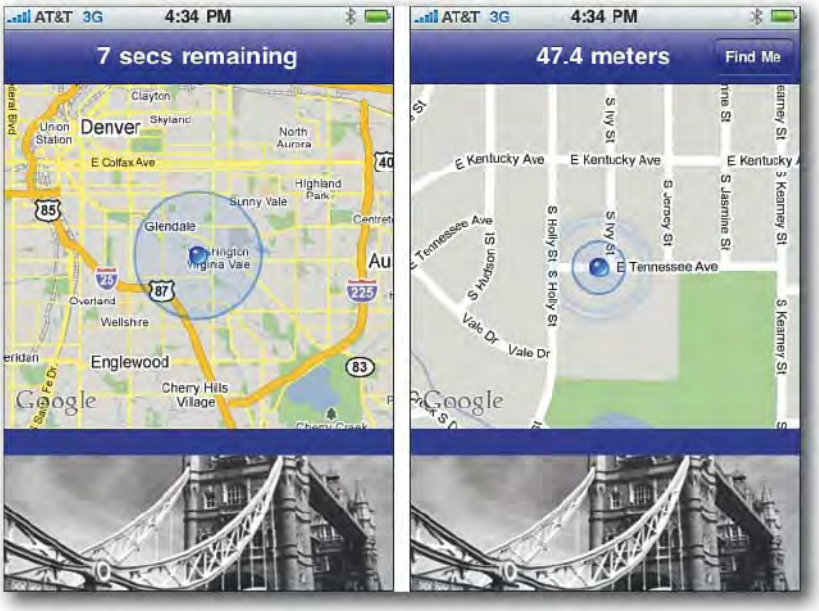
▸ *Rezept 17.5: Adressinformationen aus Koordinaten gewinnen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 17 und öffnen das Projekt zu diesem Rezept.

17.7 Rezept: Den Ort im Kartenbild anzeigen

Die Klasse `MKMapView` zeigt den Benutzern interaktive Karten mit den Koordinaten und dem Maßstab an, die Sie gewählt haben. In Interface Builder können Sie ganz einfach eine Kartenansicht in Ihre grafische Benutzeroberfläche ziehen und über einen `IBOutlet` darauf zugreifen. Im folgenden Codefragment wird der Kartenausschnitt auf die ermittelten Core Location-Koordinaten und einen Bereich von 0,1° geografischer Länge und Breite um diesen Punkt gesetzt. In den USA hat ein solches Gebiet eine Seitenlänge von etwa acht bis zwölf Kilometern und entspricht damit den Ausdehnungen einer kleineren Stadt. *Abbildung 17.3* (links) zeigt einen solchen Bereich auf der Kartenansicht.



▸ *Abbildung 17.3: Ein Bereich mit einer Seitenlänge von einem Zehntelgrad geografischer Länge und Breite deckt ein Gebiet von der Größe einer Kleinstadt ab (etwa acht bis zwölf Kilometer an jeder Seite – Bild links). Wenn Sie dieses Gebiet auf eine Seitenlänge von 0,005 Grad schrumpfen lassen, erhalten Sie eine Straßenansicht. Das Bild zeigt die Stadtteile Garland Park und Virginial Vale von Denver (rechts).*

```
mapView.region = MKCoordinateRegionMake(
    self.bestLocation.coordinate, MKCoordinateSpanMake(0.1f, 0.1f));
```

Die Größe der Region hängt vom Breitengrad ab. Am Äquator entspricht der Abstand zwischen zwei Längengraden ungefähr 111 Kilometern, was sich zu den Polen hin auf 0 verringert. Zwei Breitengrade dagegen liegen stets um 111 Kilometer auseinander.

Um Kartendaten im Stadtteilmaßstab zu sehen, müssen Sie den Koordinatenabstand auf 0,01 zu 0,01 verkleinern. Für Straßenansichten eignet sich ein noch geringerer Abstand von etwa 0,005 Grad in beiden Richtungen. *Abbildung 17.3* (rechts) zeigt den Stadtteil Garland Park in diesem Maßstab.

Sie können jedoch auch die Verwendung von Längen- und Breitengraden vermeiden und die Größe der Gebiete stattdessen durch den Abstand in Metern angeben. Das folgende Codefragment legt als Kartenausschnitt ein Quadrat von 500 m Seitenlänge um den zentralen Koordinatenpunkt fest. Das entspricht etwa 0,005 Grad geografischer Länge und Breite und mithin einer Straßenansicht.

```
mapView.region = MKCoordinateRegionMakeWithDistance(
   self.bestLocation.coordinate. 500.0f. 500.0f):
```

17.7.1 Die bestmögliche Positionsangabe finden

Rezept 17.6 führt auf Anforderung eine timergesteuerte Positionsbestimmung durch. Wenn der Benutzer auf die Schaltfläche **Find Me** tippt, startet der Code einen 10-Sekunden-Timer. Bei der Suche wird angestrebt, die bestmögliche Positionsangabe zu finden. Anhand der horizontalen Genauigkeit für die einzelnen Standorte wird die genaueste Angabe ausgewählt und beibehalten. Nach Ablauf des Timers vergrößert der Ansichtscontroller die Kartenansicht, um die erkannte Position zu zeigen.

In Rezept 17.6 wird die aktuelle Position des Benutzers sowohl während als auch nach der Suche angezeigt. Dazu wird die Eigenschaft `showsUserLocation` auf `YES` gesetzt, wodurch eine pulsierende violette Stecknadel angezeigt wird. Zu Anfang erscheint sie im Mittelpunkt der Kartenansicht am Standort des Geräts, der über Core Location bestimmt wird. In den beiden Screenshots von *Abbildung 17.3* sehen Sie diese Stecknadel jeweils im Mittelpunkt.

Wenn Sie diese Eigenschaft aktivieren, beauftragt die Kartenansicht Core Location damit, die aktuelle Position des Geräts zu ermitteln. Solange die Eigenschaft auf `YES` eingestellt bleibt, verfolgt die Karte den Benutzerstandort nach und aktualisiert ihn regelmäßig. Ein pulsierender Kreis um die Stecknadel zeigt die jeweilige Genauigkeit an. *Rezept 17.7* nutzt diese eingebaute Funktion, um den hier in *Rezept 17.6* verwendeten Ansatz zur Suche nach dem besten Ergebnis zu übergehen.

Ist der Standort bestimmt, kann der Benutzer in *Rezept 17.6* die Karte bearbeiten. Durch die Aktivierung der Eigenschaft `zoomEnabled` wird dem Benutzer erlaubt, den Kartenausschnitt durch Kneif- und Ziehbewegungen zu verändern. Bevor der Code diese Interaktion möglich macht, wartet er auf den vollständigen Abschluss der Suche, damit der Standort in der Mitte liegt, wenn die Steuerung an den Benutzer übergeht.

Ist die Suche abgeschlossen, hört der Code auf, Ortungs-Callbacks anzufordern, indem er `stopUpdatingLocation` aufruft. Gleichzeitig wird der Kartenansicht erlaubt, den Benutzerstandort weiterhin nachzuverfolgen, indem die Eigenschaft `showsUserLocation` auf `YES` gesetzt wird.

Nachdem alle Abonnements für Aktualisierungen aufgehoben sind, setzen die Instanzen des Ansichtscontrollers ihre Delegierung des Location Managers auf `nil`, damit ausstehende Callbacks den Controller nach dem Ablauf des Timers nicht mehr erreichen können. Anderenfalls müssten der Benutzer und die ausstehenden Callbacks um die Steuerung des Bildschirms kämpfen.

```
@implementation TestBedViewController
@synthesize locManager;
@synthesize bestLocation;

- (void)locationManager:(CLLocationManager *)manager
   didFailWithError:(NSError *)error
{
   NSLog(@"Location manager error: %@", [error description]);
}

- (void)locationManager:(CLLocationManager *)manager
   didUpdateToLocation:(CLLocation *)newLocation
   fromLocation:(CLLocation *)oldLocation
{
   // Merkt sich die genaueste bis jetzt gefundene Positionsangabe
   if (!self.bestLocation) self.bestLocation = newLocation;
   else if (newLocation.horizontalAccuracy <
      bestLocation.horizontalAccuracy)
      self.bestLocation = newLocation;

   // Zeigt während der Suche den Standort innerhalb der Stadt an
   mapView.region = MKCoordinateRegionMake(
      self.bestLocation.coordinate,
      MKCoordinateSpanMake(0.1f, 0.1f));

   // Zeigt den Standort des Benutzers an, unterbindet aber die
   // Interaktion
   mapView.showsUserLocation = YES;
   mapView.zoomEnabled = NO;
}

// Sucht n Sekunden lang, um die bestmögliche Positionsangabe zu ermitteln
- (void) tick: (NSTimer *) timer
{
   if (++timespent == MAX_TIME)
   {
      // Deaktiviert den Timer
      [timer invalidate];

      // Beendet die Ortsbestimmung
```

```
        [self.locManager stopUpdatingLocation];
        self.locManager.delegate = nil;

        // Stellt die Schaltfläche Find Me wieder her
        self.navigationItem.rightBarButtonItem =
            BARBUTTON(@"Find Me", @selector(findme));
        if (!self.bestLocation)
        {
            // Position nicht gefunden
            self.title = @"";
            return;
        }

        // Gibt die endgültige Genauigkeit in der Titelzeile an
        self.title = [NSString stringWithFormat:@"%0.1f meters",
            self.bestLocation.horizontalAccuracy];

        // Aktualisiert die Karte im Maßstab einer Straßenansicht und
        // ermöglicht dem Benutzer die Interaktion
        [mapView setRegion:MKCoordinateRegionMake(
            self.bestLocation.coordinate,
            MKCoordinateSpanMake(0.005f, 0.005f)) animated:YES];
        mapView.showsUserLocation = YES;
        mapView.zoomEnabled = YES;
    }
    else
        self.title = [NSString stringWithFormat:@"%d secs remaining",
            MAX_TIME - timespent];
}

// Führt die Ortsbestimmung auf Anforderung des Benutzers durch
- (void) findme
{
    // Deaktiviert die rechte Schaltfläche
    self.navigationItem.rightBarButtonItem = nil;

    // Sucht nach der genauesten Positionsangabe
    timespent = 0;
    self.bestLocation = nil;
    self.locManager.delegate = self;
    [self.locManager startUpdatingLocation];
    [NSTimer scheduledTimerWithTimeInterval:1.0f target:self
        selector:@selector(tick) userInfo:nil repeats:YES];
}
```

```
- (void) viewDidLoad
{
   self.locManager = [[[CLLocationManager alloc] init] autorelease];
   if (!self.locManager.locationServicesEnabled)
   {
      NSLog(@"User has opted out of location services");
      return;
   }
   else
   {
      // Der Benutzer erlaubt im Allgemeinen Aufrufe zur Ortsbestimmung
      self.locManager.desiredAccuracy = kCLLocationAccuracyBest;
      self.navigationItem.rightBarButtonItem =
         BARBUTTON(@"Find Me", @selector(findme));
   }
}
@end
```

► *Rezept 17.6: Den Standort des Benutzers in einer Karte anzeigen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 17 und öffnen das Projekt zu diesem Rezept.

17.8 Rezept: Anmerkungen zum Standort hinzufügen

Rezept 17.6 hat eine Möglichkeit gezeigt, die Standortangabe optisch nachzuverfolgen, während sie immer genauer bestimmt wird. In *Rezept 17.7* wird dieses Prinzip noch eine Stufe weitergeführt, denn hier wird die Position eines Geräts verfolgt, das sich bewegt. Anstatt nacheinander Ortsangaben zu ermitteln und die beste herauszusuchen, wird hier ein weit einfacherer Ansatz verfolgt, der allerdings zu ähnlichen Ergebnissen führt. Die Verantwortung für die Ermittlung des Benutzerstandorts wird der Kartenansicht und der Eigenschaft `userLocation` übertragen.

Wie ich in der Erläuterung von *Rezept 17.6* gesagt habe, wird bei der Aktivierung der Eigenschaft `showsUserLocation` automatisch Core Location beauftragt, den Gerätestandort zu verfolgen. *Rezept 17.7* nutzt diese Möglichkeit, um den Standort jede Sekunde abzufragen. Der Code aktualisiert die Kartenansicht, sodass sie diesen Standort anzeigt, wobei sich die aktuelle Position im Mittelpunkt befindet. Darüber hinaus wird der Stecknadel eine Anmerkung hinzugefügt, sodass sie die aktuellen Koordinaten anzeigt.

Anmerkungen sind Popup-Ansichten, die mit Orten auf der Karte verbunden sind. Sie weisen einen Titel und einen Untertitel auf, die Sie nach Belieben festlegen können. *Abbildung 17.4* hinter dem Rezept zeigt eine Karte mit einer Anmerkungsansicht.

17.8 Rezept: Anmerkungen zum Standort hinzufügen

Die Klasse `MKUserLocation` bietet direkten Zugriff auf die Stecknadel und die zugehörige Anmerkung. Sie enthält die beiden les- und schreibbaren Eigenschaften `titel` und `subtitle`, die Sie nach Ihren Wünschen festlegen können. In *Rezept 17.7* lautet der Titel »Location Coordinates«, während als Untertitel ein String mit dem Längen- und dem Breitengrad verwendet wird.

Die Bearbeitung von Anmerkungen wird durch die Klasse `MKUserLocation` stark vereinfacht, wobei Sie aber auf Anmerkungen zum aktuellen Benutzerstandort in der Kartenansicht beschränkt sind. Der allgemeinere Einsatz von Anmerkungen ist komplizierter und wird im anschließenden *Rezept 17.8* vorgestellt.

```
@implementation TestBedViewController
@synthesize locManager;

// Sucht n Sekunden lang nach der genauesten möglichen Positionsangabe
- (void) tick: (NSTimer *) timer
{
   if (mapView.userLocation)
      [mapView setRegion:
         MKCoordinateRegionMake(
            mapView.userLocation.location.coordinate,
            MKCoordinateSpanMake(0.005f, 0.005f)) animated:NO];
   mapView.userLocation.title = @"Location Coordinates";
   mapView.userLocation.subtitle = [NSString stringWithFormat:
      @"%f, %f",
      mapView.userLocation.location.coordinate.latitude,
      mapView.userLocation.location.coordinate.longitude];
}

// Führt die Ortsbestimmung auf Anforderung des Benutzers durch
- (void) findme
{
   self.navigationItem.rightBarButtonItem = nil;
   [self.locManager startUpdatingLocation];
   [NSTimer scheduledTimerWithTimeInterval:1.0f target:self
      selector:@selector(tick) userInfo:nil repeats:YES];
}

- (void) viewDidLoad
{
   self.locManager = [[[CLLocationManager alloc] init] autorelease];
   if (!self.locManager.locationServicesEnabled)
   {
      NSLog(@"User has opted out of location services");
      return;
   }
   else
```

```
    {
        // Der Benutzer erlaubt im Allgemeinen Aufrufe zur Ortsbestimmung
        self.locManager.desiredAccuracy = kCLLocationAccuracyBest;
        self.navigationItem.rightBarButtonItem =
            BARBUTTON(@"Find Me", @selector(findme));
        mapView.showsUserLocation = YES;
        mapView.zoomEnabled = NO;
    }
}
@end
```

▸ *Rezept 17.7: Den Gerätestandort in der Kartenansicht nachverfolgen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 17 und öffnen das Projekt zu diesem Rezept.

▸ *Abbildung 17.4: Diese Kartenansicht mit Anmerkungen wurde mithilfe von Daten aus MapKit und der Website outside.in erstellt.*

17.9 Rezept: Kartenanmerkungen erstellen

Cocoa Touch enthält keine Klasse für Kartenanmerkungen, was befremdend ist, da Anmerkungen eine so wichtige Rolle in den meisten Anwendungen mit Karten spielen. Stattdessen definiert Cocoa Touch das Protokoll `MKAnnotation`. Daher müssen Sie eine eigene Klasse entwerfen, die diesem

Protokoll folgt. Dazu ist es erforderlich, dass Ihre Klasse über die Eigenschaft `coordinate` und die Instanzmethoden `title` und `subtitle` verfügt. *Listing 17.1* zeigt, wie Sie dies tun. Es erstellt die einfache Klasse `MapAnnotation`, die die vom Protokoll geforderten Elemente für die Koordinaten, den Titel und den Untertitel enthält.

```
@interface MapAnnotation : NSObject <MKAnnotation>
{
   CLLocationCoordinate2D coordinate;
   NSString *title;
   NSString *subtitle;
}
@property (nonatomic, readonly) CLLocationCoordinate2D coordinate;
@property (nonatomic, retain) NSString *title;
@property (nonatomic, retain) NSString *subtitle;
@end
@implementation MapAnnotation
@synthesize coordinate;
@synthesize title;
@synthesize subtitle;

// Initialisierung mit Koordinaten
- (id) initWithCoordinate: (CLLocationCoordinate2D) aCoordinate
{
   if (self = [super init]) coordinate = aCoordinate;
   return self;
}

-(void) dealloc
{
   self.title = nil;
   self.subtitle = nil;
   [super dealloc];
}
@end
```

▸ *Listing 17.1: Ein Objekt für Kartenanmerkungen erstellen*

17.9.1 Anmerkungen erstellen, hinzufügen und entfernen

Um Anmerkungen verwenden zu können, müssen Sie sie erstellen und in die Kartenansicht aufnehmen. Dabei ist es möglich, die Anmerkungen einzeln hinzuzufügen:

```
anAnnotation = [[[MapAnnotation alloc]
   initWithCoordinate:coord] autorelease];
[mapView addAnnotation:anAnnotation];
```

Alternativ können Sie auch ein Array aus Anmerkungen erstellen, um sie alle auf einmal hinzuzufügen:

```
[annotations addObject:annotation];
[mapView addAnnotations:annotations];
```

Wenn Sie Anmerkungen aus einer Karte entfernen wollen, können Sie mit `removeAnnotation:` eine einzelne Anmerkung und mit `removeAnnotations:` sämtliche Elemente in einem Array löschen.

Soll die Kartenansicht in den Zustand ohne Anmerkungen zurückkehren, müssen Sie alle vorhandenen Anmerkungen entfernen. Der folgende Code ruft das Array der vorhandenen Anmerkungen mithilfe der Eigenschaft `annotations` ab und entfernt es dann aus der Karte:

```
[mapView removeAnnotations:mapView.annotations];
```

17.9.2 Anmerkungsansichten

Anmerkungsobjekte sind keine Ansichten. Die in *Listing 17.1* gezeigte Klasse `MapAnnotation` erstellt keine Bildschirmelemente, sondern ist eine abstrakte Klasse, die eine Anmerkung beschreibt. Aus dieser Beschreibung eine Ansicht auf dem Bildschirm zu machen, ist Sache der Kartenansicht. Solche Anmerkungsansichten gehören zur Klasse `MKAnnotationView`. Die Ansicht einer vorhandenen Anmerkung rufen Sie ab, indem Sie die Karte abfragen. Geben Sie dazu die Anmerkung an, und fragen Sie nach der zugehörigen Ansicht:

```
annotationView = [mapView viewForAnnotation:annotation];
```

Fast alle Anmerkungsansichten, mit denen Sie es zu tun bekommen, gehören zur Unterklasse `MKPinAnnotationView` von `MKAnnotationView`. Es handelt sich dabei um die Stecknadeln, die Sie auf Karten verteilen können. Wenn der Benutzer darauf tippt, erscheinen die Anmerkungen in einer Sprechblasenansicht. *Abbildung 17.4* zeigt eine Kartenansicht mit zehn Anmerkungen, von denen eine angetippt wurde. Die zugehörige Sprechblase macht Angaben über den Firmensitz von 5280, dem Stadtmagazin von Denver, mit einem URL zu weiteren Informationen und einer Schaltfläche, die zu diesem URL führt.

17.9.3 Anmerkungsansichten anpassen

Nachdem Sie Anmerkungen mit `addAnnotation:` oder `addAnnotations:` hinzugefügt haben, beginnt die Kartenansicht damit, die zugehörigen Anmerkungsansichten zu erstellen. Nach Abschluss dieses Vorgangs empfängt die Delegierung der Kartenansicht einen Callback. Die Delegierung, die das Protokoll `MKMapViewDelegate` deklarieren muss, wird durch `mapView:didAddAnnotationViews:` informiert, sobald die Ansichten erstellt und der Karte hinzugefügt sind. Dieser Callback gibt Ihrer Anwendung die Gelegenheit, die Anmerkungsansichten anzupassen.

Als zweiter Parameter dieses Callbacks wird ein Array aus Anmerkungsansichten übergeben. Um Merkmale wie das Bild der Ansicht (`image`) oder die angezeigten Schaltflächen zu ändern, können Sie die Elemente dieses Arrays durchgehen. *Listing 17.2* zeigt, wie Sie die einzelnen Anmerkungsansichten auf der Grundlage der jeweiligen Anmerkungen zur Verwendung vorbereiten.

```
- (void)mapView:(MKMapView *)mapView
   didAddAnnotationViews:(NSArray *)views
{
   for (MKPinAnnotationView *mkaview in views)
   {
      if ([mkaview.annotation.title
         isEqualToString:@"Current Location"])
      {
         // Die Anmerkung zur aktuellen Position ist violett und hat
         // keine Schaltfläche
         mkaview.pinColor = MKPinAnnotationColorPurple;
         mkaview.rightCalloutAccessoryView = nil;
         continue;
      }

      // Alle anderen Anmerkungen sind rot und haben eine Schaltfläche
      mkaview.pinColor = MKPinAnnotationColorRed;
      UIButton *button = [UIButton buttonWithType:
         UIButtonTypeDetailDisclosure];
      mkaview.rightCalloutAccessoryView = button;
   }
}
```

▶ *Listing 17.2: Anmerkungsansichten zur Verwendung vorbereiten*

In diesem Beispiel wird aufgrund des Anmerkungstitels entschieden, welche Farbe die Ansicht haben soll und ob sie eine Schaltfläche enthält. Übrigens sind Sie nicht auf das vorgefertigte Anmerkungsprotokoll beschränkt, dessen wenige Anforderungen die in *Listing 17.1* definierte Klasse definiert. Stattdessen können Sie eine eigene Anmerkungsklasse mit allen Instanzvariablen und -methoden schreiben, die Sie brauchen, um genauer zu bestimmen, wie Sie die Anmerkungen zur Vorbereitung der Anmerkungsansichten abfragen.

Jede Anmerkungsansicht bietet über ihre Eigenschaft `annotation` direkten Zugriff auf die zugehörige Anmerkung. Nutzen Sie die Anmerkungsdaten, um genau die Ansicht zu erstellen, die Sie haben wollen. Die folgenden Eigenschaften von Anmerkungsansichten können Sie in Ihren eigenen MapKit-Anwendungen anpassen.

Jede `MKPinAnnotationView` hat eine Farbe, die Sie über die Eigenschaft `pinColor` festlegen. MapKit bietet drei Farben zur Auswahl: Rot (`MKPinAnnotationColorRed`), Grün (`MKPinAnnotationColorGreen`) und Violett (`MKPinAnnotationColorPurple`). Nach den Richtlinien von Apple für Benutzerschnittstellen zeigen rote Stecknadeln Zielorte an, also Plätze, die der Benutzer genauer ansehen oder zu denen er sich bewegen möchte. Grüne Stecknadeln dagegen bezeichnen Ausgangspunkte, an denen der Benutzer seine Reise beginnt. Violette Stecknadeln setzen die Benutzer selbst. Wenn Sie Ihren Benutzern die Möglichkeit geben, selbst Daten zu einer Karte hinzuzufügen, verwenden Sie die violetten Stecknadeln, um anzuzeigen, dass der Benutzer sie gesetzt hat. Wie Sie in den vorherigen

Rezepten gesehen haben, zeigt eine von der Kartenansicht definierte violette Stecknadel die aktuelle Position des Benutzers an.

Jede Anmerkungsansicht hat rechts und links von der Sprechblase je einen Slot, den Sie über die Eigenschaften `rightCalloutAccessoryView` und `leftCalloutAccessoryView` mit Schaltflächen oder anderen eigenen Unteransichten füllen können. *Abbildung 17.4* zeigt eine Sprachblase mit einer Schaltfläche auf der rechten Seite, die weitere Einzelheiten aufdeckt. Diese Schaltfläche wurde in *Listing 17.2* erstellt. Allerdings sind Sie nicht auf Schaltflächen beschränkt, sondern können je nach Bedarf auch Bildansichten und andere normale Cocoa Touch-Ansichten hinzufügen.

Die Eigenschaft `canShowCallout` steuert, ob beim Antippen einer Schaltfläche eine Sprechblasenansicht gezeigt werden soll. Diese Eigenschaft ist laut Voreinstellung aktiviert, kann von Ihnen aber auf `NO` gesetzt werden, wenn Sie nicht wollen, dass die Benutzer durch Antippen Sprechblasen öffnen.

Normalerweise erscheinen die Sprechblasen unmittelbar über der zugehörigen Stecknadel, doch Sie können sie auch verschieben, indem Sie in der Eigenschaft `calloutOffset` einen neuen `GCPoint`-Wert angeben. Außerdem können Sie die Position der Anmerkungsansicht selbst ändern, indem Sie ihre Eigenschaft `centerOffset` bearbeiten. Für Stecknadelanmerkungen ist die Grafik festgelegt, doch können Sie eigene Grafiken hinzufügen, indem Sie der Eigenschaft `image` der Ansicht ein `UIImage` hinzufügen. Passen Sie sowohl die Grafik als auch die Mittelpunktsverschiebung an, um die Karte genau so zu gestalten, wie Sie sie sich vorstellen.

17.9.4 Auf das Antippen von Anmerkungsschaltflächen reagieren

MapKit vereinfacht die Verwaltung der Berührungen von Schaltflächen. Wenn Sie die `AccessoryView`-Eigenschaft einer Sprechblase auf ein Steuerelement setzen, übernimmt MapKit den Callback dafür. Sie müssen kein Target-Action-Paar hinzufügen, da MapKit dies erledigt. Ihre Aufgabe besteht nur darin, den Delegierungs-Callback `mapView:annotationView:calloutAccessoryControlTapped:` zu implementieren, wie Sie es in *Rezept 17.8* sehen.

Rezept 17.8 nutzt den Webdienst *outside.in* (*http://outside.in*), um bedeutsame Örtlichkeiten in der Nähe der gegebenen Koordinaten ausfindig zu machen, wobei die Koordinaten aus den Benutzeraktionen auf der Kartenansicht ermittelt werden. Immer dann, wenn der Benutzer den Kartenausschnitt ändert, empfängt die Delegierung der Kartenansicht einen `mapView:regionDidChangeAnimated:`-Callback. Dieser Callback ermittelt die Koordinaten des Kartenmittelpunkts über die Eigenschaft `centerCoordinate`, reicht sie bei *outside.in* ein und ruft eine XML-Liste bemerkenswerter Orte ab.

Das Rezept geht diese einzelnen Orte nacheinander durch und fügt jeweils eine Anmerkung dafür hinzu. Die XML-Daten liefern den Titel für jeden Ort und als Untertitel einen URL von *outside.in*, der im Callback für das Steuerelement verwendet wird. Wenn der Benutzer auf diese Schaltfläche tippt, öffnet die Callback-Methode den URL im Untertitel. Dadurch ist die Sprechblasenansicht unmittelbar mit einer Webseite verknüpft, auf der weitere Einzelheiten zu dem betreffenden Ort zu finden sind.

```
@implementation TestBedViewController
@synthesize locManager;
@synthesize current;

- (void)locationManager:(CLLocationManager *)manager
   didFailWithError:(NSError *)error
{
   NSLog(@"Location manager error: %@", [error description]);
}

// Aktualisiert die Karte, wenn der Benutzer sie bearbeitet
- (void)mapView:(MKMapView *)aMapView
   regionDidChangeAnimated:(BOOL)animated
{
   // Ruft Anmerkungen ab
   MapAnnotation *annotation;
   NSMutableArray *annotations = [NSMutableArray array];
   self.title = @"Searching...";

   // Fügt eine Anmerkung zur derzeitigen Position hinzu
   if (self.current)
   {
      annotation = [[[MapAnnotation alloc]
         initWithCoordinate:self.current.coordinate] autorelease];
      annotation.title = CURRENT_STRING;
      [annotations addObject:annotation];
   }

   // Räumt die Karte auf
   [mapView removeAnnotations:mapView.annotations];

   // Ruft alle neuen Orte von outside.in ab
   [self performSelector:@selector(setTitle)
      withObject:@"Contacting Outside.in..." afterDelay:0.1f];
   NSString *urlstring = [NSString stringWithFormat:
      @"http://api.outside.in/radar.xml?lat=%f&lng=%f",
      mapView.centerCoordinate.latitude,
      mapView.centerCoordinate.longitude];
   NSData *data = [NSData dataWithContentsOfURL:
      [NSURL URLWithString:urlstring]];
   printf("Received %d bytes of data from outside.in\n", data.length);

   // Prüft, ob gültige Daten vorhanden sind
   NSString *xml = [[[NSString alloc] initWithData:data
      encoding:NSUTF8StringEncoding] autorelease];
   if ([xml rangeOfString:@"places"].location == NSNotFound)
```

```
    {
        // Räumt auf und gibt die Steuerung zurück
        [mapView addAnnotations:annotations];
        return;
    }

    // Analysiert die Daten und ermittelt die Informationen über den Ort
    TreeNode *root = [[XMLParser sharedInstance]
        parseXMLFromData:data];

    // Fügt eine Anmerkung zu jedem Ort hinzu, die die Koordinaten, den
    // Namen und einen URL enthält
    for (TreeNode *node in [root objectsForKey:@"place"])
    {
        // Extrahiert die Koordinaten
        NSArray *coords = [[node leafForKey:@"georsspoint"]
            componentsSeparatedByString:@" "];
        if (coords.count < 2) continue;
        CLLocationCoordinate2D coord;
        coord.latitude = [[coords objectAtIndex:0] floatValue];
        coord.longitude = [[coords objectAtIndex:1] floatValue];

        // Erstellt die Anmerkung
        annotation = [[[MapAnnotation alloc]
            initWithCoordinate:coord] autorelease];
        annotation.title = [node leafForKey:@"name"];
        annotation.subtitle = [node leafForKey:@"url"];

        // Fügt die Anmerkung hinzu
        [annotations addObject:annotation];
    }

    // Zerstört den Stamm
    [root teardown];

    // Fügt die Anmerkungen hinzu
    [mapView addAnnotations:annotations];
}

- (void)locationManager:(CLLocationManager *)manager
    didUpdateToLocation:(CLLocation *)newLocation
    fromLocation:(CLLocation *)oldLocation
{
    // Deaktiviert zeitweilig die weitere Ortsbestimmung
    self.locManager.delegate = nil;
```

```
    [self.locManager stopUpdatingLocation];

    // Legt die aktuelle Positon fest
    self.current = newLocation;

    // Legt den Kartenausschnitt aufgrund der aktuellen Position fest und
    // erlaubt die Benutzerinteraktion
    mapView.region = MKCoordinateRegionMake(newLocation.coordinate,
        MKCoordinateSpanMake(0.02f, 0.02f));
    mapView.zoomEnabled = YES;

    // Stellt die Schaltfläche Find Me wieder her
    self.navigationItem.rightBarButtonItem =
        BARBUTTON(@"Find Me", @selector(findme));
}

// Führt eine Ortsbestimmung aufgrund einer Anforderung des Benutzers
// durch
- (void) findme
{
    // Deaktiviert die rechte Schaltfläche
    self.navigationItem.rightBarButtonItem = nil;
    self.title = @"Searching for location...";

    // Sucht nach dem Standort
    self.locManager.delegate = self;
    [self.locManager startUpdatingLocation];
}

- (void)mapView:(MKMapView *)mapView
    annotationView:(MKAnnotationView *)view
    calloutAccessoryControlTapped:(UIControl *)control
{
    MapAnnotation *annotation = view.annotation;
    [[UIApplication sharedApplication] openURL:
        [NSURL URLWithString:annotation.subtitle]];
}

- (void)mapView:(MKMapView *)mapView
    didAddAnnotationViews:(NSArray *)views
{
    // Initialisiert die einzelnen Ansichten
    for (MKPinAnnotationView *mkaview in views)
    {
        // Die Ansicht für die aktuelle Position bekommt keine Schaltflä-
che
```

```
        if ([mkaview.annotation.title
           isEqualToString:CURRENT_STRING])
        {
           mkaview.pinColor = MKPinAnnotationColorPurple;
           mkaview.rightCalloutAccessoryView = nil;
           continue;
        }

        // Alle anderen Anmerkungsansichten sind rot und haben eine
        // Schaltfläche
        mkaview.pinColor = MKPinAnnotationColorRed;
        UIButton *button = [UIButton buttonWithType:
           UIButtonTypeDetailDisclosure];
        mkaview.rightCalloutAccessoryView = button;
    }
}

- (void) viewDidLoad
{
    self.locManager = [[[CLLocationManager alloc] init] autorelease];
    if (!self.locManager.locationServicesEnabled)
    {
        NSLog(@"User has opted out of location services");
        return;
    }
    else // Der Benutzer erlaubt Ortungsaufrufe in den Einstellungen
    {
        self.locManager.desiredAccuracy = kCLLocationAccuracyBest;
        self.navigationItem.rightBarButtonItem =
           BARBUTTON(@"Find Me", @selector(findme));
        mapView.delegate = self;
    }
}
@end
```

► *Rezept 17.8: Eine interaktive Karte mit Anmerkungen erstellen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 17 und öffnen das Projekt zu diesem Rezept.

17.10 Ein letzter Punkt: Geocoding

Geocoding bedeutet, eine normale Straßenadresse in eine Angabe der geografischen Länge und Breite umzuwandeln, also beispielsweise aus »1600 Pennsylvania Avenue NW, Washington D.C.« die Werte 38,879971 und -76,982887 zu machen, die Koordinaten des Weißen Hauses.

Dies ist die Umkehrung des Reverse Geocoding, bei dem Sie von den Koordinaten ausgehen und daraus eine für Menschen verständliche Adresse machen.

Leider hat Apple in MapKit keine Vorsorge für das normale Geocoding getroffen. Zum Glück gibt es jedoch externe Anbieter dafür, z. B. Yahoo, MapQuest und Virtual Earth. Die Geocoding-API von Yahoo (*http://developeryahoo.com/maps/rest/V1/geocode.html*) ermöglicht Geocoding auf Anforderung über eine einfache REST-API.

Wenn Sie sie nutzen wollen, müssen Sie sich registrieren, um einen Entwickler-Schlüssel für die API zu bekommen. Ohne einen solchen Schlüssel können Ihre Anforderungen nicht funktionieren. Dabei unterliegen Sie den Geschäftsbedingungen von Yahoo. Es gibt eine Quotengrenze für Anforderungen, und außerdem behält sich Yahoo das Recht vor, in Zukunft Gebühren für diesen Dienst zu erheben. Für jede Anwendung, die Sie erstellen, brauchen Sie einen eigenen Schlüssel. Die Anforderung eines solchen Schlüssels ist einfach und erfordert nicht mehr, als die Anwendung zu beschreiben, einen Link zu einer Website anzugeben und sich mit den Yahoo-Anmeldeinformationen zu registrieren.

Um die API zu verwenden, übermitteln Sie den Anwendungsbezeichner und die Straße und die Stadt des Ortes, den Sie lokalisieren möchten. Je mehr Informationen Sie angeben, umso genauer wird das Geocoding-Ergebnis sein. Das folgende Codefragment erstellt einen REST-URL für das Big Chicken in Marietta (Georgia). Der Yahoo-Dienst antwortet mit einem kurzen XML-Ergebnis, das die Länge und Breite des Ortes enthält.

```
NSMutableString *urlstring = [NSMutableString string];
[urlstring appendFormat:
   @"http://local.yahooapis.com/MapsService/V1/geocode?appid=%@",
   API_KEY];
[urlstring appendFormat:
   @"&street=12+Cobb+Parkway&city=Marietta&zip=30062"]
```

Rezept 17.9 nutzt die Yahoo-API, um verschiedene Orte in den USA zu finden. Die Koordinaten in geografischer Länge und Breite werden dann dazu verwendet, um Anmerkungen zu erstellen, was zu der Anzeige aus *Abbildung 17.5* führt. Hierfür werden Anmerkungsobjekte verwendet, die gegenüber dem aus *Rezept 17.8* erweitert wurden. Diese Anmerkungen enthalten einen URL-String für die Einblendschaltfläche und einen Bild-URL-String für das Foto links davon.

In diesem Rezept wurden die Adressen, Websites und Bild-URLs hartkodiert. Dies können Sie jedoch sehr leicht zu einem allgemeineren API-Dienst erweitern, der Informationen über lokale Örtlichkeiten gibt.

```
- (void) findme
{
   NSString *whichLocation = [LOCATIONS objectAtIndex:whichItem];

   // Bestimmt die Koordinaten durch Geocoding
   [self performSelector:@selector(setTitle)
      withObject:whichLocation afterDelay:0.1f];

   // Erstellt den REST-URL
   NSMutableString *urlstring = [NSMutableString string];
   [urlstring appendFormat:
      @"http://local.yahooapis.com/MapsService/V1/geocode?appid=%@",
      API_KEY];

   NSString *locationURLString;
   NSString *picstring;
   // Alle Bilder werden mit freundlicher Genehmigung von Wikipedia
   // (http://en.wikipedia.org) genutzt und sind entweder gemeinfrei
   // oder unterliegen einer Creative Commons-Lizenz
  switch (whichItem)
   {
      case 0:
         // Weißes Haus
         [urlstring appendFormat:
            @"&street=Pennsylvania+Avenue&city=@Washington+DC"];
         locationURLString =
            @"http://en.wikipedia.org/wiki/White_house";
         picstring = @"http://upload.wikimedia.org/\
            wikipedia/commons/a/af/WhiteHouseSouthFacade.JPG";
         break;
      case 1:
         // Big Chicken in Marietta
         [urlstring appendFormat:
            @"&street=12+Cobb+Parkway&city=Marietta&zip=30062"];
         locationURLString =
            @"http://en.wikipedia.org/wiki/Big_Chicken";
         picstring = @"http://upload.wikimedia.org/wikipedia/\
            commons/e/ed/Thebigchicken.jpg";
         break;
      case 2:
         // Zoo von Los Angeles
         [urlstring appendFormat:
            @"&street=5333+Zoo+Drive&city=Los+Angeles&zip=90027"];
         locationURLString =
```

```
            @"http://en.wikipedia.org/wiki/LA_Zoo";
         picstring = @"http://upload.wikimedia.org/\
            wikipedia/en/c/c9/LAzoo.jpg";
         break;
      case 3:
         // Big Hot Dog
         [urlstring appendFormat:
            @"&street=10+Old+Stagecoach+Road&city=Baily&state=CO"];
         locationURLString =
            @"http://en.wikipedia.org/wiki/\
            Coney_Island_Hot_Dog_Stand";
         picstring = @"http://upload.wikimedia.org/wikipedia/\
            commons/e/ea/Coney_Island_2007.JPG";
         break;
      case 4:
         // Randy's Donuts
         [urlstring appendFormat:
            @"&street=4805+West+Manchester+Avenue\
            &city=Inglewood&zip=90301"];
         locationURLString =
            @"http://en.wikipedia.org/wiki/Randy%27s_Donuts";
         picstring = @"http://upload.wikimedia.org/\
            wikipedia/commons/1/1d/2008-0914-RandysDonuts.jpg";
      default:
         break;
   }

   // Ruft die Geocoding-Ergebnisse ab
   NSData *data = [NSData dataWithContentsOfURL:
      [NSURL URLWithString:urlstring]];
   printf("Received %d bytes of data from Yahoo\n", data.length);

   // Ruft die Koordinaten ab
   TreeNode *root = [[XMLParser sharedInstance]
      parseXMLFromData:data];
   CLLocationCoordinate2D coord;
   coord.latitude = [[root leafForKey:@"Latitude"] floatValue];
   coord.longitude = [[root leafForKey:@"Longitude"] floatValue];

   // Richtet die Kartenansicht ein
   mapView.region =
      MKCoordinateRegionMakeWithDistance(coord, 10000, 10000);
   mapView.zoomEnabled = YES;

   // Erstellt die Anmerkung, falls sie sich nicht im Dictionary befindet.
```

```
    // Diese Implementierung von Anmerkungen wurde gegenüber der aus
    // Rezept 17.8 um ein Bild und einen URL erweitert.
    if (![annotationDict objectForKey:whichLocation])
    {
        MapAnnotation *annotation = [[[MapAnnotation alloc]
            initWithCoordinate:coord] autorelease];
        annotation.title = whichLocation;
        annotation.urlstring = locationURLString;
        annotation.picstring = picstring;
        annotation.subtitle = [NSString stringWithFormat:
            @"%f, %f", coord.latitude, coord.longitude];
        [annotationDict setObject:annotation forKey:whichLocation];

        // Lädt erneut die Anmerkungen einschließlich der neuen
        [mapView removeAnnotations:mapView.annotations];
        [mapView addAnnotations:[annotationDict allValues]];
    }

    whichItem = (whichItem + 1) % [LOCATIONS count];
    self.navigationItem.rightBarButtonItem =
        BARBUTTON(whichLocation, @selector(findme));
}
```

▶ *Rezept 17.9: Geocoding von Adressen für die weitere Verwendung in MapKit*

▶ *Abbildung 17.5: Durch Geocoding einer Straßenadresse kann MapKit an der richtigen Stelle der Karte eine Anmerkung anzeigen.*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 17 und öffnen das Projekt zu diesem Rezept.

17.11 Zusammenfassung

Core Location und MapKit arbeiten Hand in Hand und bieten Möglichkeiten, um die Position des Geräts zu bestimmen und Informationen über den Ort in einer praktischen Kartendarstellung anzuzeigen. In diesem Kapitel haben Sie gelernt, wie Sie mit Core Location die geografische Länge und Breite des aktuellen Standorts in Echtzeit ermitteln und diese Koordinaten mit Reverse Geocoding in Adressen umwandeln können. Sie haben erfahren, wie Sie Geschwindigkeit und Richtung sowohl mit den eingebauten Eigenschaften als auch durch Berechnung herausfinden, wie Sie eine Karte einrichten, den dargestellten Ausschnitt anpassen, den Standort anzeigen und eigene Anmerkungen hinzufügen. Bevor Sie zum nächsten Kapitel übergehen, sollten Sie sich mit den folgenden abschließenden Gedanken vertraut machen:

- Bevor Sie eine Entscheidung treffen, wie Sie Ortungsfunktionen in Ihrer Anwendung umsetzen, müssen Sie klären, wer Ihr Zielpublikum ist und wie es Ihre Anwendung nutzen wird. Manche Funktionen von Core Location eignen sich eher für Autofahrer, andere mehr für Fußgänger und Radfahrer.
- Testen, testen, testen, testen, testen, testen! Core Location-Anwendungen müssen erschöpfend getestet und optimiert werden, und zwar sowohl in Xcode als auch im praktischen Einsatz, damit sie im App Store erfolgreich sein können.
- Haben Sie schon einmal gehört, dass jemand sagt: »Habe ich dich nicht gestern auf 104,28393 Grad westlicher Länge gesehen?« Adressen haben für die meisten Menschen mehr Aussagekraft als Koordinaten. Wenden Sie Reverse Geocoding an, um verständliche Informationen anzuzeigen.
- Postleitzahlen sind besonders API-geeignet. Auch wenn Sie in Ihrer Anwendung keine Kartendarstellung vorsehen, können Sie Postleitzahlen in eine herkömmliche Benutzeroberfläche aufnehmen. Mit einer in Koordinaten umgewandelten Postleitzahl können Sie Informationen wie Adressen und Telefonnummern von Geschäften, Parks und Sehenswürdigkeiten in der näheren Umgebung abrufen.
- Sorgfältig entworfene Anmerkungsansichten ergänzen Kartenansichten um sinnvolle Interaktionsmöglichkeiten. Scheuen Sie nicht davor zurück, Schaltflächen, Bilder und andere Elemente zu verwenden, die den Nutzen einer Karte erweitern.

18 Verbindung mit dem Adressbuch

Neben den gewöhnlichen Steuerelementen für Benutzeroberflächen und den Medienkomponenten, die Sie auf jedem Rechner finden, verfügt das iPhone SDK über eine Reihe von gezielten Entwicklerlösungen speziell für das iPhone und den iPod touch. Zu den nützlichsten davon zählt der Zugang zum Adressbuch, über den Sie im Programm Zugriff auf die Kontaktdatenbank haben und sie bearbeiten können. In diesem Kapitel wird das Adressbuch vorgestellt und gezeigt, wie Sie dessen Frameworks in Ihren Anwendungen nutzen können. Sie erfahren hier, wie Sie auf Informationen über einzelne Kontakte zugreifen, wie Sie diese Kontaktinformationen bearbeiten und aktualisieren und wie Sie einen gewünschten Kontakt mithilfe von Prädikaten finden. In diesem Kapitel geht es um die GUI-Klassen, die die interaktive Auswahl, Anzeige und Bearbeitung von Kontakten ermöglichen. Wenn Sie dieses Kapitel durchgearbeitet haben, kennen Sie das Adressbuch in- und auswendig.

18.1 Rezept: Mit dem Adressbuch arbeiten

Das iPhone SDK hat nicht nur ein, sondern zwei Adressbuch-Frameworks, nämlich `AddressBook.framework` und `AddressBookUI.framework`. Wie die Bezeichnungen andeuten, belegen sie zwei verschiedene Nischen im iPhone SDK. `AddressBook` enthält systemnahe C-basierte Strukturen und Routinen zum Zugriff auf Kontaktdaten aus den SQLite-Datenbanken auf dem iPhone. `AddressBookUI` umfasst Objective-C-basierte `UIViewController`-Browserobjekte hoher Ebene, die Benutzern angezeigt werden können. Beide Frameworks sind klein. Sie enthalten nur wenige Klassen und Datentypen.

Auf dem iPhone liegen Kontaktdaten im Bibliotheksordner des Benutzers. Auf dem Macintosh-Simulator können Sie auf diese Daten unter `~/Library/Application Support/iPhone Simulator/User/Library` bzw. `~/Library/Application Support/iPhone Simulator/3.x/Library` seit SDK 3.0 unbeschränkt zugreifen. Die beiden Dateien `AddressBook/AddressBook.sqlitedb` und `AddressBook/AddressBookImages.sqlitedb` verwenden das gewöhnliche SQLite3, um Kontaktdaten und optionale Kontaktbilder zu speichern. Auf dem iPhone liegen diese Dateien in `/var/mobile/Library/AddressBook`, also außerhalb der Sandbox der Anwendung. Um die Kontaktinformationen abzufragen oder zu bearbeiten, können Sie also nicht direkt auf diese Dateien zugreifen, sondern müssen die beiden Adressbuch-Frameworks verwenden.

18.1.1 AddressBookUI

Das Framework `AddressBookUI` enthält einige vorgefertigte Ansichtscontroller, die mit der auf dem Gerät gespeicherten Kontaktdatenbank zusammenarbeiten. Dazu gehören die gewöhnliche Kontaktauswahl, die Anzeige eines einzelnen Kontakts und die Bearbeitung von Kontakten. Sie bestimmen eine Delegierung und legen diese Controller dann auf den Navigationsstack oder wählen eine Modalansicht, wie es in den Rezepten in diesem Kapitel gezeigt wird.

Wie die Bildauswahl und die Kameracontroller, die Sie in Kapitel 7, *Mit Bildern arbeiten*, und Kapitel 15, *Audio, Video und MediaKit*, kennengelernt haben, sind die `AddressBookUI`-Controller nicht besonders flexibel. Apple will, dass Entwickler sie wie bereitgestellt verwenden und nur geringe oder gar keine Anpassungen vornehmen. Darüber hinaus erfordern diese Controller von Ihnen die Fähigkeit, bis zu einem gewissen Grad systemnah zu programmieren. Wie Sie sehen werden, greifen diese Klassen auf umständliche Weise auf das zugrunde liegende Adressbuch zu.

18.1.2 AddressBook

Im C-basierten Framework `AddressBook` bildet der Typ `ABPerson` die grundlegende Struktur für Kontakte. Dieser Datensatz speichert alle Informationen zu jedem Kontakt, z. B. Name, E-Mail-Adresse, Telefonnummern usw. Jeder Datensatz entspricht einem vollständigen Adressbuchkontakt. Mit dem folgenden Code können Sie das Adressbuch nach der Anzahl der zurzeit in seiner Datenbank gespeicherten Objekte abfragen. Trotz ihres Namens erstellt die hier gezeigte Funktion `ABAddressBookCreate()` kein neues Adressbuch, sondern einen Verweis auf das Systemadressbuch.

```
+ (int) contactsCount
{
    ABAddressBookRef addressBook = ABAddressBookCreate();
    return ABAddressBookGetPersonCount(addressBook);
}
```

Einzelne Datensätze lesen Sie mit der Funktion `ABAddressBookCopyArrayOfAllPeople()`. Die folgende Methode ruft die Datensätze als Arrray ab und fügt sie dann einzeln zu einem `ABContact`-Objekt hinzu, wobei `ABContact` ein eigens für dieses Buch entwickelter Objective-C-Wrapper ist.

Solche Wrapper ermöglichen eine einfache Kombination von Adressbuchaufrufen in C und normaler Entwicklung und Speicherverwaltung in Cocoa Touch. Den vollständigen Quellcode für diese und einige andere Wrapperklassen finden Sie im Begleitcode zu diesem Kapitel.

```
+ (NSArray *) contacts
{
   ABAddressBookRef addressBook = ABAddressBookCreate();
   NSArray *thePeople = (NSArray *)
      ABAddressBookCopyArrayOfAllPeople(addressBook);
   NSMutableArray *array = [NSMutableArray
      arrayWithCapacity:thePeople.count];
   for (id person in thePeople)
      [array addObject:[ABContact
         contactWithRecord:(ABRecordRef)person]];
   [thePeople release];
   return array;
}
```

Die Klasse `ABContact` kapselt einen internen `ABRecordRef`-Verweis, den CF-Typ für die einzelnen Kontaktdatensätze. Darüber hinaus werden in diesem Wrapper nur einige Eigenschaften und Methoden erstellt, mit denen Sie die Unterdatensätze des `ABRecordRef` festlegen und auf sie zugreifen können.

```
@interface ABContact : NSObject
{
   ABRecordRef record;
}
@end
```

Nahezu alle `ABRecordRef`-Funktionen tragen das Präfix `ABPerson`, das der auf dem Macintosh, aber nicht auf dem iPhone verfügbaren Klasse `ABPerson` entspricht. Die Funktionsaufrufe drehen sich also um `ABPerson`, doch die durch diese Aufrufe bearbeiteten Daten sind tatsächlich Instanzen von `ABRecordRef`. Der Grund dafür wird deutlicher, wenn Sie bedenken, dass im Framework `AddressBook` die Struktur `ABRecordRef` sowohl für Personen (einzelne Kontakte, sowohl privat als auch geschäftlich) als auch für Gruppen (Sammlungen mehrerer Kontakte, z. B. Arbeitskollegen oder persönliche Freunde) verwendet wird. Das SDK bietet sowohl `ABGroup`- als auch `ABPerson`-Funktionen. Das Thema Gruppen wird weiter hinten in diesem Abschnitt angesprochen.

18.1.3 ABRecord-Strings abrufen und festlegen

Jeder `ABRecord` speichert eine Reihe von einfachen Stringwerten, die u. a. für den Namen, den Titel, die Stellung und die Organisation einer Person stehen. Diese einzelnen Angaben können Sie abrufen, indem Sie die entsprechenden Feldwerte aus dem Datensatz kopieren. In der folgenden Methode wird die Eigenschaftenkonstante `ABPropertyID` verwendet, um das angeforderte Feld im Datensatz zu bezeichnen. Die Methode kopiert den Wert des Feldes, wandelt ihn in einen String um und gibt diesen Inhalt zurück.

```
- (NSString *) getRecordString:(ABPropertyID) anID
{
   return [(NSString *) ABRecordCopyValue(record, anID) autorelease];
}

// Verwendungsbeispiele
- (NSString *) firstname
   {return [self getRecordString:kABPersonFirstNameProperty];}
- (NSString *) lastname
   {return [self getRecordString:kABPersonLastNameProperty];}
```

Auf diese Weise können Sie die folgenden 13 Stringfelder abrufen. Die Bezeichner sind als konstante Integer in der Headerdatei `ABPerson.h` definiert und identifizieren die Felder in einem `ABRecordRef`, die einen String für die jeweilige Eigenschaft enthalten.

- `kABPersonFirstNameProperty`
- `kABPersonLastNameProperty`
- `kABPersonMiddleNameProperty`
- `kABPersonPrefixProperty`
- `kABPersonSuffixProperty`
- `kABPersonNicknameProperty`
- `kABPersonFirstNamePhoneticProperty`
- `kABPersonLastNamePhoneticProperty`
- `kABPersonMiddleNamePhoneticProperty`
- `kABPersonOrganizationProperty`
- `kABPersonJobTitleProperty`
- `kABPersonDepartmentProperty`
- `kABPersonNoteProperty`

Stringförmige Eigenschaften lassen sich genauso einfach festlegen wie abrufen. Wandeln Sie den gewünschten String in einen `CFStringRef` um, und speichern Sie die Daten mit `ABRecordSetValue()` wieder im Datensatz. Beachten Sie, dass Sie mit diesen Aufrufen nicht das Adressbuch aktualisieren, sondern nur die Daten innerhalb des Datensatzes ändern. Wenn Sie Kontaktinformationen speichern möchten, müssen Sie sie in das Adressbuch schreiben. Eine Möglichkeit dazu wird weiter hinten in diesem Abschnitt vorgestellt.

```
- (BOOL) setString: (NSString *) aString
   forProperty:(ABPropertyID) anID
{
   CFErrorRef error;
   BOOL success = ABRecordSetValue(record, anID,
      (CFStringRef) aString, &error);
```

```
    if (!success) NSLog(@"Error %@",
        [(NSError *)error localizedDescription]);
    return success;
}

// Verwendungsbeispiele
- (void) setFirstname: (NSString *) aString
    {[self setString: aString forProperty:
        kABPersonFirstNameProperty];}

- (void) setLastname: (NSString *) aString
    {[self setString: aString forProperty: kABPersonLastNameProperty];}
```

18.1.4 Einfache Datumseigenschaften

Neben den Stringeigenschaften werden im Adressbuch auch drei wichtige Datumsangaben gespeichert, nämlich die Daten, an denen der Datensatz erstellt und zuletzt geändert wurde, sowie optional ein Geburtsdatum. Dazu dienen die folgenden Eigenschaftenkonstanten:

- `kABPersonCreationDateProperty`
- `kABPersonModificationDateProperty`
- `kABPersonBirthdayProperty`

Der Zugriff auf diese Elemente erfolgt genauso wie auf die Strings, allerdings müssen Sie eine Umwandlung in und aus `NSDate`- statt `NSString`-Instanzen vornehmen. Theoretisch können Sie die ersten beiden Eigenschaften zwar ändern, doch am besten überlassen Sie es dem Adressbuch, sich darum zu kümmern.

```
// Gibt ein Datumsfeld aus einem Datensatz zurück
- (NSDate *) getRecordDate:(ABPropertyID) anID
{
    return [(NSDate *) ABRecordCopyValue(record, anID) autorelease];
}

// Ruft den Geburtstag des Kontakts ab
- (NSDate *) birthday
    {return [self getRecordDate:kABPersonBirthdayProperty];}

// Richtet ein Datumsfeld in einem Datensatz ein
- (BOOL) setDate: (NSDate *) aDate forProperty:(ABPropertyID) anID
{
    CFErrorRef error;
    BOOL success = ABRecordSetValue(record, anID,
        (CFDateRef) aDate, &error);
    if (!success) NSLog(@"Error %@",
```

```
        [(NSError *)error localizedDescription]);
    return success;
}

// Legt den Geburtstag des Kontakts fest
- (void) setBirthday: (NSDate *) aDate
    {[self setDate: aDate forProperty: kABPersonBirthdayProperty];}
```

18.1.5 Mehrwertige Datensatzeigenschaften abrufen und festlegen

Da jede Person mehrere E-Mail-Adressen, Telefonnummern usw. haben kann (nur nicht mehrere Geburtstage), verwendet `ABPerson` zu ihrer Speicherung eine *mehrwertige* Struktur, bei der es sich im Grunde genommen um ein Array handelt. Die einzelnen Arrays können Sie über ihren Eigenschaftsbezeichner aus dem Datensatz abrufen. Anstelle eines Strings gibt der Datensatz einen `CFArrayRef` zurück.

Es gibt die folgenden Bezeichner für mehrwertige Eigenschaften:

- `kABPersonEmailProperty`
- `kABPersonPhoneProperty`
- `kABPersonURLProperty`
- `kABPersonDateProperty`
- `kABPersonAddressProperty`
- `kABPersonInstantMessageProperty`

Die ersten drei dieser Elemente (E-Mail-Adresse, Telefonnummer und URL) enthalten *Multistrings*, also Arrays aus Strings, und weisen den Typ `kABMultiStringPropertyType` auf. Die Typen für mehrwertige Inhalte spielen eine wichtige Rolle bei der Speicherung von Daten im Datensatz, denn sie werden herangezogen, um Arbeitsspeicher zuzuweisen und die Größe der einzelnen Felder im Datensatz zu bestimmen.

Das nächste Element, die Datumseigenschaft, nimmt ein Array aus Datumsangaben auf und hat den Typ `kABMultiDateTimePropertyType`. Sowohl die Eigenschaften für die Adresse als auch für Instant Messaging bestehen aus Arrays von Dictionarys und weisen den Typ `kABMultiDictionaryPropertyType` auf.

Hinweis

»Verwandte Personen« bilden eine weitere Multistring-Eigenschaft, die aber zurzeit nicht in der Anwendung *Kontakte* auf dem iPhone genutzt wird. Die Konstante `kABPersonRelatedNames` hält Namen und Beziehungen fest. Beispielsweise kann Mary Ball Washington mit `kABPersonMotherLabel` im Kontakt von George Washington gespeichert werden. In `ABPerson.h` finden Sie eine vollständige Liste von Beziehungskonstanten.

Ein Wertarray für eine dieser Eigenschaften abzurufen ist ganz unkompliziert. Kopieren Sie einfach die Eigenschaft aus dem Datensatz (mit `ABRecordCopyValue()`), und zerlegen Sie sie in ihr Komponentenarray. Das Adressbuch bietet eine Funktion, mit der Sie das Array aus der Eigenschaft in einen standardmäßigen `CFArrayRef` kopieren können.

```
- (NSArray *) arrayForProperty: (ABPropertyID) anID
{
   CFTypeRef theProperty = ABRecordCopyValue(record, anID);
   NSArray *items =
      (NSArray *)ABMultiValueCopyArrayOfAllValues(theProperty);
   CFRelease(theProperty);
   return [items autorelease];
}
```

Es mag zwar so aussehen, als hätten Sie mit diesen beiden Aufrufen alle Informationen abgerufen, doch stimmt dies leider nicht. Bei mehrwertigen Einträgen ist es mit dem Abruf des Wertes allein nicht getan, denn jedes in einem mehrwertigen Array gespeicherte Element hat neben einem Wert auch eine Bezeichnung (Label). *Abbildung 18.1* zeigt eine Kontaktseite aus einem Adressbuch. Gruppierte Elemente weisen eine Bezeichnung auf, um die Rollen der verschiedenen E-Mail-Adressen, Telefonnummern usw. zu unterscheiden. Dieser Kontakt weist drei E-Mail-Adressen und drei Telefonnummern auf, die jeweils mit einem Label gekennzeichnet sind, um deutlich zu machen, wozu sie da sind.

Um alle Informationen abzurufen, die in einer mehrwertigen Eigenschaft gespeichert sind, müssen Sie neben den Werten auch die Bezeichnungen kopieren. Die folgende Methode ruft die einzelnen Labels nach ihrem Index ab und fügt sie einem Label-Array hinzu. Label und Werte zusammen bilden eine vollständige mehrwertige Sammlung.

```
- (NSArray *) labelsForProperty: (ABPropertyID) anID
{
   CFTypeRef theProperty = ABRecordCopyValue(record, anID);
   NSMutableArray *labels = [NSMutableArray array];
   for (int i = 0; i < ABMultiValueGetCount(theProperty); i++)
   {
      NSString *label =
         (NSString *)ABMultiValueCopyLabelAtIndex(theProperty, i);
      [labels addObject:label];
      [label release];
   }
   CFRelease(theProperty);
   return labels;
}
```

▶ *Abbildung 18.1: Mehrwertige Elemente weisen neben dem eigentlichen Wert auch eine Bezeichnung (Label) auf (hier z. B.* **MAIN**, **GOOGLE** *und* **MOBILE** *für die Telefonnummern und* **HOME**, **WORK** *und* **GOOGLE** *für die E-Mail-Adressen).*

Zum Speichern von Einträgen in mehrwertigen Objekten gehen Sie einfach umgekehrt vor. Wandeln Sie die Cocoa Touch-Objekte in eine Form um, mit der der Datensatz umgehen kann. Die folgende Methode erwartet ein Array aus Dictionarys, wobei jedes Dictionary über die beiden Schlüssel `value` und `label` verfügen muss. Die unter diesen Schlüsseln abgelegten Objekte sind der Wert und das Label aus der ursprünglichen mehrwertigen Eigenschaft. Der Code durchläuft das Dictionary-Array und fügt die einzelnen Werte und Label dem beschreibbaren mehrwertigen Objekt zu.

```
- (ABMutableMultiValueRef) createMultiValueFromArray:
   (NSArray *) anArray withType: (ABPropertyType) aType
{
   ABMutableMultiValueRef multi = ABMultiValueCreateMutable(aType);
   for (NSDictionary *dict in anArray)
      ABMultiValueAddValueAndLabel(multi,
         (CFTypeRef) [dict objectForKey:@"value"],
         (CFTypeRef) [dict objectForKey:@"label"], NULL);
   return multi;
}
```

Diese Methode erstellt ein mehrwertiges Objekt mit dem `ABPropetyType`, der als Parameter angegeben wurde. Hier kommen die verschiedenen Arten von Multistring-Typen ins Spiel.

Beispielsweise können Sie eine Eigenschaft für E-Mail-Adressen mit Strings und Labels füllen, indem Sie den Typ `kABMultiStringPropertyType` verwenden. Diese Methode ruft diejenige auf, die ein mehrwertiges Objekt erstellt. und gibt dabei sowohl das Wert/Label-Dictionary als auch den gewünschten Typ an. Ist das mehrwertige Element erstellt, wird es an eine andere Methode übergeben, durch die es gesetzt wird.

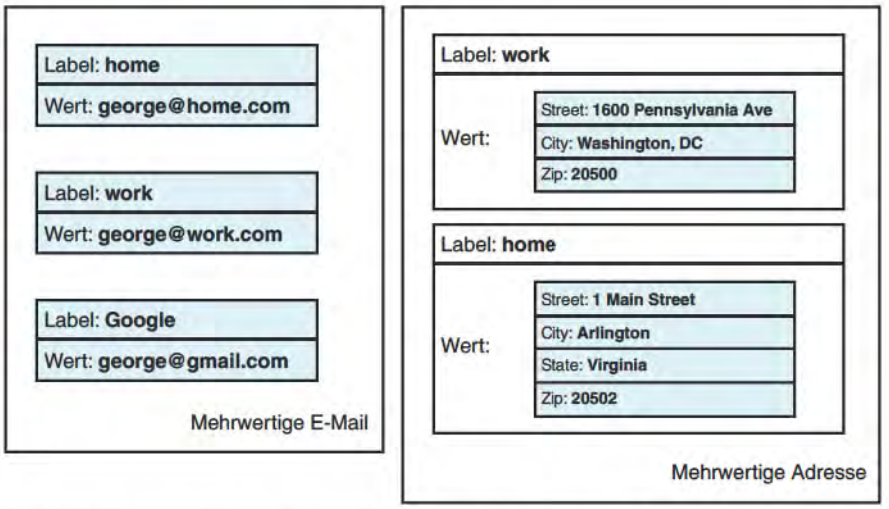
```
- (void) setEmailDictionaries: (NSArray *) dictionaries
{
   ABMutableMultiValueRef multi = [self
      createMultiValueFromArray:dictionaries
      withType:kABMultiStringPropertyType];
   [self setMulti:multi forProperty:kABPersonEmailProperty];
   CFRelease(multi);
}
```

Die Zuweisung eines mehrwertigen Objekts zu einem Datensatz ist einfach und erfolgt über den normalen Aufruf `ABRecordSetValue()`. Die folgende Methode weist ein mehrwertiges Objekt zu einer Eigenschaft in einem Datensatz zu. Im Grunde gibt es keinen Unterschied zwischen diesem Aufruf und demjenigen, der eine einzelne Datums- oder Stringeigenschaft festlegt. Die gesamte Arbeit erschöpft sich darin, das mehrwertige Element erst einmal zu erstellen.

```
- (BOOL) setMulti: (ABMutableMultiValueRef) multi
   forProperty: (ABPropertyID) anID
{
   CFErrorRef error;
   BOOL success = ABRecordSetValue(record, anID, multi, &error);
   if (!success)
      NSLog(@"Error %@",
         [(NSError *)error localizedDescription]);
   return success;
}
```

18.1.6 Eigenschaften für Adressen und Instant Messaging

Die Eigenschaften für Adressen und für Instant Messaging (SMS) speichern Dictionarys anstelle von Strings oder Datumsangaben. Dadurch ist beim Erstellen des mehrwertigen Arrays ein weiterer Schritt erforderlich, denn Sie müssen einen Satz von Dictionarys füllen und dann zusammen mit den Labels zu einem Array hinzufügen. *Abbildung 18.2* zeigt diese zusätzliche Schicht. Wie Sie sehen, bestehen mehrwertige E-Mail-Einträge aus einem Array von Label-Wert-Paaren, bei denen jeder Wert ein einzelner String ist. Für jede Adresse dagegen gibt es ein eigenes Dictionary innerhalb des Wertelements.



▶ *Abbildung 18.2: Im Gegensatz zu mehrwertigen E-Mail-Einträgen, die einen einzelnen String für jeden Wert aufweisen, enthalten mehrwertige Adresseinträge ein gesamtes Adress-Dictionary für jeden Wert.*

Das folgende Beispiel zeigt die erforderlichen Schritte, um ein mehrwertiges Element mit zwei Adressen zu erstellen. Der Code legt die Dictionarys an und fügt sie zusammen mit ihren Labels einem grundlegenden Array hinzu. Das von diesem Code erstellte Array entspricht dem mehrwertigen Adressobjekt auf der rechten Seite von *Abbildung 18.2*.

```
// Erstellt das Array zur Speicherung der Wert-Label-Dictionarys für die
// Adresse
NSMutableArray *addresses = [NSMutableArray array];

// Erstellt die Adresse für das Weiße Haus und fügt sie dem Array hinzu
NSDictionary *wh_addy = [ABContact
   addressWithStreet:@"1600 Pennsylvania Avenue"
   withCity:@"Washington, DC" withState:nil
   withZip:@"20500" withCountry:nil withCode:nil];
   [addresses addObject:[ABContact dictionaryWithValue:wh_addy
   andLabel:kABWorkLabel]];

// Erstellt die Privatadresse (home) und fügt sie dem Array hinzu
NSDictionary *home_addy = [ABContact
   addressWithStreet:@"1 Main Street" withCity:@"Arlington"
   withState:@"Virginia" withZip:@"20502"
   withCountry:nil withCode:nil];
[addresses addObject:[ABContact dictionaryWithValue:home_addy
   andLabel:kABHomeLabel]];
```

Dieser Code nutzt Hilfsmethoden, um die Adress- und die Wert/Label-Dictionarys für das mehrwertige Array zu erstellen. Die folgenden Methoden erstellen die Wert/Label- sowie die Adress- und SMS-Dictionarys, wobei die Schlüssel für Letztere mithilfe von Adressbuch-Schlüsselkonstanten vordefiniert sind.

```
// Erstellt ein Wert/Label-Dictionary
+ (NSDictionary *) dictionaryWithValue: (id) value
   andLabel: (CFStringRef) label
{
   NSMutableDictionary *dict = [NSMutableDictionary dictionary];
   if (value) [dict setObject:value forKey:@"value"];
   if (label) [dict setObject:(NSString *)label forKey:@"label"];
   return dict;
}

// Erstellt ein Adress-Dictionary
+ (NSDictionary *) addressWithStreet: (NSString *) street
   withCity: (NSString *) city
   withState:(NSString *) state withZip: (NSString *) zip
   withCountry: (NSString *) country withCode: (NSString *) code
{
   NSMutableDictionary *md = [NSMutableDictionary dictionary];
   if (street) [md setObject:street
      forKey:(NSString *) kABPersonAddressStreetKey];
   if (city) [md setObject:city
      forKey:(NSString *) kABPersonAddressCityKey];
   if (state) [md setObject:state
      forKey:(NSString *) kABPersonAddressStateKey];
   if (zip) [md setObject:zip
      forKey:(NSString *) kABPersonAddressZIPKey];
   if (country) [md setObject:country
      forKey:(NSString *) kABPersonAddressCountryKey];
   if (code) [md setObject:code
      forKey:(NSString *) kABPersonAddressCountryCodeKey];
   return md;
}

// Erstellt ein SMS-Dictionary
+ (NSDictionary *) smsWithService: (CFStringRef)
   service andUser: (NSString *) userName
{
   NSMutableDictionary *sms = [NSMutableDictionary dictionary];
   if (service) [sms setObject:(NSString *) service
      forKey:(NSString *) kABPersonInstantMessageServiceKey];
   if (userName) [sms setObject:userName
      forKey:(NSString *) kABPersonInstantMessageUsernameKey];
   return sms;
}
```

18.1.7 Mit Adressbuchbildern arbeiten

Jedem Datensatz im Adressbuch kann optional ein Bild beigefügt werden. Sie können Bilddaten in und aus jedem Datensatz kopieren, wobei die Funktion `ABPersonhasImageData()` angibt, ob in einem gegebenen Datensatz solche Daten vorhanden sind. Prüfen Sie damit, ob sich Bilddaten abrufen lassen.

Bilddaten werden als `CFData` gespeichert, es besteht aber eine Verknüpfung mit `NSData`, die mit keinerlei Kosten verbunden ist. Da die Klasse `UIImage` die Umwandlung von Bildern in Daten und umgekehrt ermöglicht, müssen Sie die Daten einfach nur nach Bedarf konvertieren. Mit der Funktion `UIImagePNGRepresentation()` wandeln Sie eine `UIImage`-Instanz in eine `NSData`-Darstellung um, und mit `imageWithData:` erstellen Sie aus `NSData` ein neues Bild.

```
// Gibt ein Bild aus dem Datensatz eines Kontakts zurück
- (UIImage *) image
{
   if (!ABPersonHasImageData(record)) return nil;
   CFDataRef imageData = ABPersonCopyImageData(record);
   UIImage *image = [UIImage imageWithData:(NSData *) imageData];
   CFRelease(imageData);
   return image;
}

// Legt das Bild des Datensatzes fest
- (void) setImage: (UIImage *) image
{
   CFErrorRef error;
   BOOL success;

   if (image == nil) // Entfernt das Bild
   {
      if (!ABPersonHasImageData(record)) return;
      success = ABPersonRemoveImageData(record, &error);
      if (!success) NSLog(@"Error %@",
         [(NSError *)error localizedDescription]);
      return;
   }

   NSData *data = UIImagePNGRepresentation(image);
   success = ABPersonSetImageData(record, (CFDataRef)data, &error);
   if (!success) NSLog(@"Error %@",
      [(NSError *)error localizedDescription]);
}
```

18.1.8 Datensätze erstellen, hinzufügen und löschen

Die Funktion `ABPersonCreate()` gibt eine neue Instanz von `ABRecordRef` zurück. Dieser Datensatz befindet sich außerhalb des Adressbuchs und bildet eine eigenständige Datenstruktur. Alle Methoden, die Sie bis jetzt in diesem Kapitel gesehen haben, bearbeiten einzelne Datensätze, aber keine von ihnen speichert einen Datensatz im Adressbuch. Beachten Sie das, wenn Sie sich die folgende Hilfsmethode ansehen, die einen neu initialisierten Kontakt zurückgibt:

```
+ (id) contact
{
    ABRecordRef person = ABPersonCreate();
    id contact = [ABContact contactWithRecord:person];
    CFRelease(person);
    return contact;
}
```

Um neue Informationen in das Adressbuch zu schreiben, sind zwei Schritte erforderlich. Erst müssen Sie den Datensatz hinzufügen und dann das Adressbuch speichern. Der zweite Schritt wird häufig von Entwicklern vergessen, die noch nicht lange für das iPhone programmieren, was so wirkt, als ob das Adressbuch keine Änderungen annimmt. Die folgende Methode ergänzt das Adressbuch um einen neuen Kontakt, indem es diesen Kontakt zunächst hinzufügt und dann die Änderung speichert.

```
+ (BOOL) addContact: (ABContact *) aContact withError: (NSError **) error
{
    ABAddressBookRef addressBook = ABAddressBookCreate();
    if (!ABAddressBookAddRecord(addressBook, aContact.record,
        (CFErrorRef *) error))
          return NO;
    return ABAddressBookSave(addressBook, (CFErrorRef *) error);
}
```

Mit neuen Kontaktinformationen können Sie einen Kontakt, der bereits im Adressbuch vorhanden ist, nicht überschreiben. Wenn Sie einen neuen Datensatz für »George Washington« erstellen und versuchen, ihn in einem Adressbuch zu speichern, in dem es bereits einen solchen Datensatz gibt, werden Sie keinen Erfolg haben. Das liegt daran, dass der neue Datensatz einen anderen Bezeichner hat als der ursprüngliche. Den eindeutigen Bezeichner eines Datensatzes fragen Sie wie folgt ab:

```
- (ABRecordID) recordID {return ABRecordGetRecordID(record);}
```

Kontaktinformationen können Sie nur aktualisieren, indem Sie den vorhandenen Datensatz lesen, ändern und speichern (wie es in *Rezept 18.7* geschieht) oder den alten Eintrag löschen und eine neue Version hinzufügen.

Zum Entfernen eines Datensatzes müssen Sie wie beim Hinzufügen das Adressbuch speichern. Der Datensatz ist danach immer noch als Objekt vorhanden, aber nicht mehr in der Datenbank des Adressbuchs gespeichert. Um einen Löschvorgang durchzuführen, gehen Sie folgendermaßen vor:

```
- (BOOL) removeSelfFromAddressBook: (NSError **) error
{
   ABAddressBookRef addressBook = ABAddressBookCreate();
   if (!ABAddressBookRemoveRecord(addressBook, self.record,
      (CFErrorRef*) error))
        return NO;
   return ABAddressBookSave(addressBook,  (CFErrorRef *) error);
```

18.1.9 Kontakte suchen

Das Standard-Framework für das Adressbuch ermöglicht eine Präfixsuche in den Datensätzen. Die folgende Funktion gibt ein Array aus Datensätzen zurück, bei denen der vollständige Name (gewöhnlich der Vorname mit angehängtem Nachnamen, was sich aber in Ländern mit umgekehrtem Namensmuster anpassen lässt) mit dem angegebenen String übereinstimmt.

```
NSArray *array = (NSArray *)ABAddressBookCopyPeopleWithName(
   addressBook, CFSTR("Eri"));
```

Hinweis

Die Adressbuchroutinen sind mithilfe der C-Bibliotheken von Core Foundation geschrieben. Viele Klassen gibt es sowohl in *Cocoa Touch Foundation* als auch in *Core Foundation*. Beispielsweise entspricht ein `NSArray*`-Zeiger einem `CFArrayRef` von Core Foundation. Diese Klassen sind »kostenfrei verknüpft«, d. h., sie weisen die gleiche Struktur und den gleichen Funktionsumfang auf und können ohne jegliche Nachteile ineinander umgewandelt werden. In dem oben gezeigten Codefragment wird der von der Core Foundation-Funktion `ABAddressBookCopyPeopleWithName()` zurückgegebene Arrayverweis in einen `NSArray`-Pointer umgewandelt, um auf einfachere Weise einen Objective-C-Wrapper anwenden zu können.

Die Suche wird jedoch weit einfacher, wenn Sie einen Satz von Eigenschaften wie diejenigen aus einer selbst geschriebenen `ABContact`-Klasse mit `NSPredicate`-Instanzen kombinieren. Im folgenden Code werden der Vorname, zweite Vorname, Nachname und Spitzname eines Kontakts mit einem String verglichen. Das Prädikat untersucht die Eigenschaften, um zu bestimmen, ob Kontakte als Treffer gewertet oder verworfen werden. Bei diesem Vergleich werden Groß- und Kleinschreibung sowie diakritische Zeichen nicht unterschieden (`[cd]`) und alle Stellen innerhalb des Strings betrachtet (`contains`), nicht nur der Anfang (`begins with`).

```
+ (NSArray *) contactsMatchingName: (NSString *) fname
{
   NSPredicate *pred;
   NSArray *contacts = [ABContactsHelper contacts];
   pred = [NSPredicate predicateWithFormat:
```

```
        @"firstname contains[cd] %@ OR lastname contains[cd] %@ OR\
        nickname contains[cd] %@ OR middlename contains[cd] %@",
        fname, fname, fname, fname];
    return [contacts filteredArrayUsingPredicate:pred];
}
```

Hinweis

Die Anleitung von Apple zur Programmierung mit Prädikaten (*Predicate Programming Guide*) bietet eine umfassende Einführung in die Grundlagen von Prädikaten. Sie lernen darin, wie Sie Prädikate erstellen und in Ihren Anwendungen einsetzen.

18.1.10 Mit Gruppen arbeiten

Mit Gruppen können Sie Kontakte in zusammengehörige Sätze wie »Arbeit«, »privat« und andere sinnvolle Unterteilungen gliedern. Eine Gruppe ist nichts anderes als ein weiterer `ABRecord`, der allerdings einige besondere Eigenschaften aufweist. In Gruppen werden keine Namen, Adressen und Telefonnummern gespeichert, sondern Verweise auf andere Kontaktdatensätze.

Sind Sie mit Gruppen auf dem iPhone vertraut? Nein? Das liegt daran, dass die Apple-Anwendung *Kontakte* keine Möglichkeit bietet, um Gruppen zu erstellen. Auf dem iPhone können Sie Kontaktgruppen nur über das SDK oder durch Einspielen eines Adressbuchs vom Macintosh hinzufügen.

Die Anzahl der Gruppen im aktuellen Adressbuch ermitteln Sie, indem Sie die Gruppendatensätze abrufen, wie die folgende Methode zeigt. Es gibt keine direkte Möglichkeit, um die Anzahl der Gruppen so wie die Anzahl der Personenkontakte zu bestimmen. Die im Folgenden angegebenen Methoden gehören zur Klasse `ABGroup`, die einen Objective-C-Wrapper für Adressbuchgruppen bildet (so wie `ABContact` für Adressbuchkontakte).

```
+ (int) numberOfGroups
{
    ABAddressBookRef addressBook = ABAddressBookCreate();
    NSArray *groups =
        (NSArray *)ABAddressBookCopyArrayOfAllGroups(addressBook);
    int ncount = groups.count;
    [groups release];
    return ncount;
}
```

Gruppen erstellen Sie mit der Funktion `AbGroupCreate()`, die auf die gleiche Weise wie `ABPersonCreate()` einen `AbRecordRef` zurückgibt. Der Unterschied liegt im Datensatztyp, der für Gruppen `kABGroupType` lautet statt `kABPersonType`.

```
+ (id) group
{
    ABRecordRef grouprec = ABGroupCreate();
```

```
    id group = [ABGroup groupWithRecord:grouprec];
    CFRelease(grouprec);
    return group;
}
```

Gruppenmitglieder können Sie hinzufügen und entfernen, indem Sie `ABGroupAddMember()` bzw. `ABGroupRemoveMember()` aufrufen. Dies wirkt sich nur auf die Datensätze aus. Die Änderungen werden erst mit der Speicherung des Adressbuchs übernommen.

```
- (BOOL) addMember: (ABContact *) contact
    withError: (NSError **) error
{
    return ABGroupAddMember(self.record, contact.record,
    (CFErrorRef *) error);
}

- (BOOL) removeMember: (ABContact *) contact
    withError: (NSError **) error
{
    return ABGroupRemoveMember(self.record, contact.record,
    (CFErrorRef *) error);
}
```

Jedes Mitglied einer Gruppe ist eine Person – oder ein Kontakt, wie es in der Terminologie von `ABContact` heißt. Die folgende Methode durchsucht die Gruppenmitglieder und gibt ein Array aus `AbContact`-Instanzen zurück, die jeweils mit dem `ABRecordRef` für ein Gruppenmitglied initialisiert sind.

```
- (NSArray *) members
{
    NSArray *contacts =
        (NSArray *)ABGroupCopyArrayOfAllMembers(self.record);
    NSMutableArray *array =
        [NSMutableArray arrayWithCapacity:contacts.count];
    for (id contact in contacts)
        [array addObject:
            [ABContact contactWithRecord:(ABRecordRef)contact]];
    [contacts release];
    return array;
}
```

Jede Gruppe hat einen Namen, und dabei handelt es sich um die wichtigste Gruppeneigenschaft, die Sie festlegen und abrufen können. Der Bezeichner lautet `kABGroupNameProperty`, aber ansonsten funktioniert diese Eigenschaft wie jede andere Kontakteigenschaft.

```
- (NSString *) getRecordString:(ABPropertyID) anID
{
   return [(NSString *) ABRecordCopyValue(record, anID) autorelease];
}

- (NSString *) name
{
   NSString *string = [self getRecordString:kABGroupNameProperty];
   return [string autorelease];
}

- (void) setName: (NSString *) aString
{
   CFErrorRef error;
   BOOL success = ABRecordSetValue(record,
      kABGroupNameProperty, (CFStringRef) aString, &error);
   if (!success)
      NSLog(@"Error %@", [(NSError *)error localizedDescription]);
}
```

18.1.11 ABContact, ABGroup und ABContactsHelper

Der Beispielcode zu diesem Abschnitt enthält drei Wrapper-Klassen. Die darin verwendeten Techniken wurden in den Codefragmenten hervorgehoben, die Sie im Verlaufe dieser Erläuterungen gesehen haben. Aufgrund der Länge und der Redundanz dieser Klassen wird das komplette Rezeptlisting (normalerweise *Rezept 18.1*) in diesem Abschnitt nicht abgedruckt. Das »Rezept« dieses Abschnitts setzt sich aus den bereits vorgestellten Codebeispielen zusammen. Im Begleitcode sind diese einzelnen Techniken zu einer Reihe von Adressbuch-Wrappern zusammengefasst. Daher finden Sie *Rezept 18.1* im Beispielcode zu diesem Kapitel.

Die Klasse `ABContact` beruht in gewisser Weise auf der nur auf dem Mac verfügbaren Klasse `ABPerson`, bietet aber eine besser für Cocoa Touch geeignete Schnittstellte aus Objective-C-2.0-Eigenschaften als die C-ähnlichen Eigenschaftsabfragen aus der Apple-Klasse `ABPerson`. Alle kontaktspezifischen Methoden, die Sie in diesem Abschnitt gesehen haben, sind von dieser Klasse abgeleitet.

Eine zweite Klasse namens `ABGroup` kapselt alle Gruppenfunktionen für `AbRecordRef`-Instanzen. Sie bietet Objective-C-Zugriff zum Erstellen und Verwalten von Gruppen. Diese Klasse setzen Sie ein, um neue Gruppen anzulegen und Mitglieder hinzuzufügen oder zu entfernen.

Die letzte Klasse, `ABContactsHelper`, enthält adressbuchspezifische Methoden. Damit können Sie das Adressbuch durchsuchen, Datensatzarrays abrufen usw. Ich habe nur wenige grundlegende Suchmöglichkeiten nach Namen und Telefonnummern hinzugefügt, doch können Sie diese Klasse, die Sie unter *http://github.com/erica* finden, auch um anspruchsvollere Abfragen erweitern.

18.2 Rezept: Das Adressbuch durchsuchen

Suchvorgänge mit Prädikaten sind sowohl schnell als auch effektiv. *Rezept 18.2* zeigt solche Abfragen. Der Code zeigt eine Suchtabelle (wie die aus *Rezept 11.16*) mit einer scrollbaren Kontakttabelle an. Diese Tabelle wird bei Abfragen des Benutzers in der Suchleiste aktualisiert.

Da Suchtabellen zwei Datenquellen haben, werden in diesem Rezept auch zwei Arrays verwendet. Das Kontaktarray speichert die gesamte Kontaktliste des Adressbuchs, während das zweite, gefilterte Array jedes Mal neu aufgebaut wird, wenn der Benutzer die Suchleiste aktualisiert. Dazu wird die Methode `contactsMatchingName:` verwendet, die Sie im vorhergehenden Abschnitt kennengelernt haben.

Wenn der Benutzer auf eine Zeile tippt, wird eine Instanz von `ABPersonViewController` angezeigt. Diese Klasse bietet eine Ansicht, die Einzelheiten zu einem gegebenen Datensatz anzeigt (ähnlich wie *Abbildung 18.1*). Um diesen Ansichtscontroller zu verwenden, müssen Sie ihn zuweisen und initialisieren und seine Eigenschaft `displayedPerson` festlegen. Wie zu erwarten ist, wird in dieser Eigenschaft ein `AbRecordRef` abgelegt.

Die Delegierung von Personenansichtscontrollern ist eingeschränkt. Wenn Sie die Eigenschaft `personViewDelegate` festlegen, können Sie die Methode `personViewController:shouldPerformDefaultActionForPerson:` abonnieren. Diese Methode wird ausgelöst, wenn die Benutzer bestimmte Elemente in der Ansicht auswählen, z. B. Telefonnummern, E-Mail-Adressen, URLs und Adressen. Bei dem Wert `YES` wird die Standardaktion durchgeführt (Wählen der Nummer, E-Mail usw.), bei `NO` geschieht nichts. In *Rezept 18.2* wird dieser Callback eingesetzt, um den Wert des ausgewählten Elements in der Debug-Konsole anzuzeigen. Es ist zwar auch eine Interaktion mit anderen Anzeigeelementen wie dem Kontakthinweis und dem Klingelton möglich, doch wird dabei kein Callback hervorgerufen.

Um das Rezept so zu erweitern, dass eine Bearbeitung möglich ist, setzen Sie die Eigenschaft `allowsEditing` des Personenansichtscontrollers auf `YES`. Dadurch erscheint oben rechts in der Anzeige eine Bearbeitungsschaltfläche. Wenn der Benutzer darauf tippt, werden in der Personenansicht die Bearbeitungsfunktionen aktiviert, die auch in der Anwendung *Kontakte* zur Verfügung stehen.

```
- (NSInteger)tableView:(UITableView *)aTableView
   numberOfRowsInSection:(NSInteger)section
{
   // Normale Tabelle
   if (aTableView == self.tableView)
      return self.contacts.count;

   // Suchtabelle
   self.filteredArray = [ABContactsHelper
      contactsMatchingName:self.searchBar.text];
   return self.filteredArray.count;
}
```

```
- (UITableViewCell *)tableView:(UITableView *)aTableView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Nimmt eine Zelle aus der Warteschlange oder erstellt eine neue
   UITableViewCellStyle style = UITableViewCellStyleSubtitle;
   UITableViewCell *cell = [aTableView
      dequeueReusableCellWithIdentifier:@"BaseCell"];
   if (!cell) cell = [[[UITableViewCell alloc] initWithStyle:style
      reuseIdentifier:@"BaseCell"] autorelease];

   // Ruft die Kontaktinformation ab und legt den Zellentext fest
   NSArray *collection = (aTableView == self.tableView) ?
      self.contacts : self.filteredArray;
   ABContact *contact = [collection objectAtIndex:indexPath.row];
   cell.textLabel.text = contact.contactName;
   cell.detailTextLabel.text = contact.phonenumbers;
   return cell;
}

- (BOOL)personViewController:
   (ABPersonViewController *)personViewController
   shouldPerformDefaultActionForPerson:(ABRecordRef)person
   property:(ABPropertyID)property
   identifier:(ABMultiValueIdentifier)identifierForValue
{
   // Zeigt das ausgewählte Element
   if ([ABContact propertyIsMultivalue:property])
   {
      NSArray *array = [ABContact arrayForProperty:property
         inRecord:person];
      CFShow([array objectAtIndex:identifierForValue]);
   }
   else
   {
      id object = [ABContact objectForProperty:property
         inRecord:person];
      CFShow([object description]);
   }
   return YES;
}

- (void)tableView:(UITableView *)aTableView
   didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Zeigt bei Auswahl einer Zeile die Personenansicht an
```

```
    ABPersonViewController *pvc =
       [[[ABPersonViewController alloc] init] autorelease];
    NSArray *collection = (aTableView == self.tableView) ?
       self.contacts : self.filteredArray;
    ABContact *contact = [collection objectAtIndex:indexPath.row];
    pvc.displayedPerson = contact.record;
    pvc.personViewDelegate = self;
    // pvc.allowsEditing = YES; // Optionale Bearbeitung
    [[self navigationController] pushViewController:pvc animated:YES];
}
```

▸ *Rezept 18.2: Kontakte durch einen Suchvorgang auswählen und anzeigen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 18 und öffnen das Projekt zu diesem Rezept.

18.3 Rezept: Zugriff auf Bilddaten

Rezept 18.3 erweitert *Rezept 18.2* um Vorschaubilder für die Kontakte in den einzelnen Tabellenzellen. Dazu wird ein neues Bild von 45 × 45 Pixel erstellt. Wenn Bilddaten zur Verfügung stehen, wird dieses Bild in dieser Vorschaufläche dargestellt. Anderenfalls bleibt die Vorschau leer. Das Bild wird beim Zeichnen zur `imageView` der Zelle zugewiesen. Diese Eigenschaft wurde erst im SDK der Version 3.0 eingeführt. Für die Bereitstellung auf Firmware früherer Versionen müssen Sie auf die Eigenschaft `image` der Zelle zurückgreifen, die jetzt als veraltet und unerwünscht gilt.

▸ *Abbildung 18.3: Bilddaten zu Adressbuchkontakten lassen sich auf einfache Weise abrufen und anzeigen.*

Abbildung 18.3 zeigt die Benutzeroberfläche aus diesem Rezept. In diesem Screenshot läuft gerade eine Suche (nach Übereinstimmungen mit dem Buchstaben »e«). Alle Einträge, die dieser Suchanfrage genügen, werden mit ihrem Vorschaubild angezeigt.

Beachten Sie, auf welche einfache Weise die Vorschaubilder in diesem Rezept erstellt und verwendet werden. Es sind nur wenige Codezeilen erforderlich, um einen neuen Bildkontext anzulegen, darin zu zeichnen (für Kontakte mit Bildern) und ihn als `UIImage`-Instanz zu speichern.

```
- (UITableViewCell *)tableView:(UITableView *)aTableView
   cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Nimmt eine Zelle aus der Warteschlange oder erstellt eine neue
   UITableViewCellStyle style = UITableViewCellStyleSubtitle;
   UITableViewCell *cell = [aTableView
      dequeueReusableCellWithIdentifier:@"BaseCell"];
   if (!cell) cell = [[[UITableViewCell alloc] initWithStyle:style
      reuseIdentifier:@"BaseCell"] autorelease];

   // Ruft den Kontakt ab
   NSArray *collection = (aTableView == self.tableView) ?
      self.contacts : self.filteredArray;
   ABContact *contact = [collection objectAtIndex:indexPath.row];
   cell.textLabel.text = contact.contactName;
   cell.detailTextLabel.text = contact.phonenumbers;

   // Zeichnet das Bild in die Vorschaufläche
   UIGraphicsBeginImageContext(CGSizeMake(45.0f, 45.0f));
   if (contact.image)
      [contact.image drawInRect:
         CGRectMake(0.0f, 0.0f, 45.0f, 45.0f)];
   UIImage *img = UIGraphicsGetImageFromCurrentImageContext();
   UIGraphicsEndImageContext();

   // Legt das Bild für die Zelle fest
   cell.imageView.image = img;
   return cell;
}
```

► *Rezept 18.3: Adressbuchbilder in Tabellenzellen anzeigen*

Den Rezeptcode finden

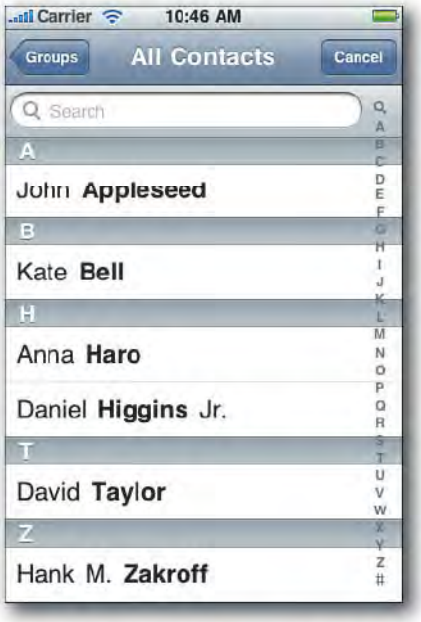
Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 18 und öffnen das Projekt zu diesem Rezept.

18.4 Rezept: Personen auswählen

Das Framework `AddressBookUI` enthält einen praktischen Controller für die Personenauswahl. Das Durchsuchen der gesamten Kontaktliste ist genauso einfach zu bewerkstelligen wie die Anzeige eines einzelnen Kontaktbildschirms. Mit der Klasse `ABPeoplePickerNavigationController` stellen Sie einen interaktiven Browser dar, wie er in *Abbildung 18.4* gezeigt wird.

Sie müssen den Kontroller zuweisen und anzeigen, bevor Sie ihn modal darstellen. Legen Sie die Eigenschaft `peoplePickerDelegate` fest, um Benutzeraktionen in der Ansicht zu erfassen.

```
- (void) action: (UIBarButtonItem *) bbi
{
   ABPeoplePickerNavigationController *ppnc =
      [[ABPeoplePickerNavigationController alloc] init];
   ppnc.peoplePickerDelegate = self;
   [self presentModalViewController:ppnc animated:YES];
}
```



▸ *Abbildung 18.4: Das Steuerelement zur Personenauswahl auf dem iPhone ermöglicht es dem Benutzer, die Kontaktdatenbank zu durchsuchen und eine Person oder Organisation auszuwählen.*

Wenn Sie das Protokoll `ABPeoplePickerNavigationControllerDelegate` deklarieren, muss Ihre Klasse die folgenden drei Methoden implementieren, die auf die Berührung eines Kontakts, einer Kontakteigenschaft auf der Schaltfläche **Cancel** reagieren:

- `peoplePickerNavigationController:shouldContinueAfterSelectingPerson:` Wenn Benutzer einen Kontakt antippen, haben Sie zwei Möglichkeiten. Sie können die betreffende Person als abschließende Wahl anerkennen und den modalen Ansichtscontroller entfernen (wie es in *Rezept 18.4* geschieht), oder Sie können zur individuellen An-

sicht wechseln. Um nur die Person auszuwählen, muss diese Methode NO zurückgeben, und um mit dem individuellen Bildschirm fortzufahren, ist YES erforderlich. Das zweite Argument enthält die ausgewählte Person für den Fall, dass Sie den Vorgang nach Auswahl eines `ABPerson`-Datensatzes abbrechen wollen.

- `peoplePickerNavigationController:shouldContinueAfterSelectingPerson:property:identifier:` Diese Methode wird nicht aufgerufen, bis der Benutzer zum Anzeigebildschirm eines individuellen Kontakts vorgedrungen ist. Dann liegt es bei Ihnen, ob Sie die Steuerung wieder an das Programm übertragen wollen (geben Sie NO zurück) oder fortfahren (YES) wollen. *Rezept 18.2* zeigt, wie Sie feststellen können, welche Eigenschaft angetippt worden ist und wie Sie deren Wert abrufen können. Diese Methode sollte zwar optional sein, ist es zurzeit aber nicht.
- `peoplePickerNavigationControllerDidCancel:` Wenn ein Benutzer CANCEL angetippt hat, brauchen Sie eine Gelegenheit, um die Modalansicht zu entfernen. Diese Methode erfasst das Abbruch-Ereignis, sodass Sie es verwenden können, um den Vorgang auszuführen.

Rezept 18.4 zeigt das einfachste mögliche Beispiel für eine Personenauswahl. Der Code stellt den Picker dar und wartet darauf, dass der Benutzer einen Kontakt auswählt. Wenn dies geschieht, wird der Picker entfernt und der Titel des Ansichtscontrollers (in der Navigationsleiste) geändert, um den vollständigen Namen der ausgewählten Person anzuzeigen. Wenn der Haupt-Callback `NO` zurückgibt, heißt das, dass der Eigenschafts-Callback niemals aufgerufen wird. Sie müssen ihn jedoch trotzdem in Ihren Code aufnehmen, da alle drei Methoden erforderlich sind.

```
// Reagiert auf die Auswahl eines Kontakts durch den Benutzer
- (BOOL)peoplePickerNavigationController:
   (ABPeoplePickerNavigationController *)peoplePicker
   shouldContinueAfterSelectingPerson:(ABRecordRef)person
{
   self.title = [[ABContact contactWithRecord:person] compositeName];
   [self dismissModalViewControllerAnimated:YES];
   [peoplePicker release];
   return NO; // Wird nicht weiter fortgeführt
}

// Eine erforderliche Methode, die bei der reinen Personenauswahl jedoch
// niemals aufgerufen wird
- (BOOL)peoplePickerNavigationController:
   (ABPeoplePickerNavigationController *)peoplePicker
   shouldContinueAfterSelectingPerson:(ABRecordRef)person
   property:(ABPropertyID)property
   identifier:(ABMultiValueIdentifier)identifier
{
   [self dismissModalViewControllerAnimated:YES];
   [peoplePicker release];
   return NO;
```

```
}

// Handhabt einen Abbruch durch den Benutzer
- (void)peoplePickerNavigationControllerDidCancel:
    (ABPeoplePickerNavigationController *)peoplePicker
{
    [self dismissModalViewControllerAnimated:YES];
    [peoplePicker release];
}

// Stellt den Picker dar
- (void) action: (UIBarButtonItem *) bbi
{
    ABPeoplePickerNavigationController *ppnc =
        [[ABPeoplePickerNavigationController alloc] init];
    ppnc.peoplePickerDelegate = self;
    [self presentModalViewController:ppnc animated:YES];
}
```

▸ *Rezept 18.4: Personen auswählen*

18.5 Rezept: Die Eigenschaften in der Kontaktauswahl einschränken

Wenn die Benutzer eine bestimmte Eigenschaft auswählen sollen, z. B. eine E-Mail-Adresse, sollten Sie ihnen nicht auch noch den Straßennamen und die Faxnummer anzeigen. Schränken Sie die im Picker angezeigten Eigenschaften auf diejenigen ein, aus denen die Besucher auswählen sollen. Der Picker in *Abbildung 18.5* ist auf die Auswahl von E-Mail-Adressen beschränkt.

▸ *Abbildung 18.5: Mit der Eigenschaft* `displayedProperties` *eines Pickers können Sie wählen, welche Eigenschaften die Benutzer sehen sollen, in diesem Fall nur die E-Mail-Adressen.*

Damit dies geschieht, müssen Sie ein Array von Eigenschaftstypen an den Controller übergeben, um die anzuzeigenden Eigenschaften festzulegen. Richten Sie dazu die Eigenschaft `displayedProperties` des Pickers ein. *Rezept 18.5* bietet zwei Auswahloptionen, eine für E-Mail-Adressen, eine für Telefonnummern. In diesen Beispielen wird im Eigenschaftsarray jeweils nur eine einzige Eigenschaft angegeben, doch Sie können eine beliebige Anzahl verschiedener Eigenschaften anzeigen lassen.

```
// Sorgen Sie dafür, dass die Benutzer Zugriff auf den Bildschirm mit den
// Einzelangaben haben
- (BOOL)peoplePickerNavigationController:
   (ABPeoplePickerNavigationController *)peoplePicker
   shouldContinueAfterSelectingPerson:(ABRecordRef)person
{
   return YES;
}

// Zeigt die ausgewählte Eigenschaft an
- (BOOL)peoplePickerNavigationController:
   (ABPeoplePickerNavigationController *)peoplePicker
   shouldContinueAfterSelectingPerson:(ABRecordRef)person
   property:(ABPropertyID)property
   identifier:(ABMultiValueIdentifier)identifier
{
   // Die Benutzer können jetzt nur mit E-Mail-Adressen oder Telefonnummern
   // arbeiten
   [self dismissModalViewControllerAnimated:YES];
   NSArray *array = [ABContact arrayForProperty:property
      inRecord:person];
   self.title = (NSString *)[array objectAtIndex:identifier];
   [peoplePicker release];
   return NO;
}

// Handhabt Abbruchvorgänge durch den Benutzer
- (void)peoplePickerNavigationControllerDidCancel:
   (ABPeoplePickerNavigationController *)peoplePicker
{
   [self dismissModalViewControllerAnimated:YES];
   [peoplePicker release];
}

// Wählt eine E-Mail-Adresse aus
- (void) email
{
   ABPeoplePickerNavigationController *ppnc =
      [[ABPeoplePickerNavigationController alloc] init];
```

```
    ppnc.peoplePickerDelegate = self;
    [ppnc setDisplayedProperties:[NSArray
        arrayWithObject:NUMBER(kABPersonEmailProperty)]];
    [self presentModalViewController:ppnc animated:YES];
}

// Wählt eine Telefonnummer aus
- (void) phone: (UIBarButtonItem *) bbi
{
    ABPeoplePickerNavigationController *ppnc =
        [[ABPeoplePickerNavigationController alloc] init];
    ppnc.peoplePickerDelegate = self;
    [ppnc setDisplayedProperties:[NSArray
        arrayWithObject:NUMBER(kABPersonPhoneProperty)]];
    [self presentModalViewController:ppnc animated:YES];
}
```

▸ *Rezept 18.5: Die anzuzeigenden Eigenschaften auswählen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 18 und öffnen das Projekt zu diesem Rezept.

18.6 Rezept: Neue Kontakte hinzufügen

Mit der Klasse `ABNewPersonViewController` ermöglichen Sie es den Benutzern, neue Kontakte zu erstellen. Dieser Ansichtscontroller zeigt einen Bearbeitungsbildschirm an, der das interaktive Anlegen neuer Adressbucheinträge vereinfacht. Nachdem Sie den Ansichtscontroller zugewiesen und initialisiert haben, beginnen Sie damit, einen neuen Kontakt zu erstellen und der Eigenschaft `displayedPerson` zuzuweisen. Wenn Sie möchten, können Sie den Kontaktdatensatz vorab mit Vorgabeeigenschaften füllen. In *Rezept 18.7* wird dieser automatische Ausfüllvorgang genutzt, um bereits vorhandene Kontakte ändern zu können.

Weisen Sie als Nächstes die `newPersonViewDelegate` zu und deklarieren Sie das Protokoll `ABNewPersonViewControllerDelegate`. Die Delegierungen empfangen als einzigen Callback `newPersonViewController:didCompleteWithNewPerson:`, der sowohl bei Auswahl- als auch bei Abbruchereignissen gesendet wird. Prüfen Sie den Parameter `person`, um zu bestimmen, welcher Fall vorliegt (siehe *Rezept 18.6*).

Wenn der Benutzer nach der Bearbeitung des neuen Kontakts auf **Done** tippt, speichern Sie die Kontaktdaten im Adressbuch. Falls Sie dies manuell erledigen und nicht die Klasse `ABContactsHelper` verwenden, müssen Sie darauf achten, dass Sie sowohl den Datensatz hinzufügen als auch das Adressbuch speichern.

Wenn ein Kontakt mit denselben Angaben bereits vorhanden ist, müssen Sie diese besondere Situation auf irgendeine Weise handhaben. In diesem Rezept wird der vorhandene Kontakt entfernt und durch den neuen ersetzt, aber Sie können genauso gut eine Benachrichtigung ausgeben und den Benutzer fragen, wie weiter vorzugehen ist. Der Benutzer kann wählen, ob der ursprüngliche oder der neue Datensatz beibehalten werden soll. Wenn Sie möchten, können Sie die beiden Datensätze auch auf irgendeine Weise zusammenführen.

```
- (void)newPersonViewController:
   (ABNewPersonViewController *)newPersonViewController
   didCompleteWithNewPerson:(ABRecordRef)person
{
   if (person)
   {
      ABContact *contact = [ABContact contactWithRecord:person];
      self.title = [NSString stringWithFormat:
         @"Added %@", contact.compositeName];
      if (![ABContactsHelper addContact:contact withError:nil])
      {
         // Möglicherweise ist ein solcher Datensatz bereits vorhanden.
         // Hier wird der alte entfernt und durch den neuen ersetzt.
         [contact removeSelfFromAddressBook:nil];
         [ABContactsHelper addContact:contact withError:nil];
      }
   }
   else
      self.title = @"Cancelled";

   [self.navigationController popViewControllerAnimated:YES];
}

- (void) add
{
   // Erstellt einen neuen Ansichtscontroller
   ABNewPersonViewController *npvc =
      [[ABNewPersonViewController alloc] init];

   // Erstellt einen neuen Kontakt
   ABContact *contact = [ABContact contact];
   npvc.displayedPerson = contact.record;

   // Richtet die Delegierung ein
   npvc.newPersonViewDelegate = self;

   [self.navigationController pushViewController:npvc animated:YES];
}
```

▸ *Rezept 18.6: Einen neuen Personenansichtscontroller verwenden*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 18 und öffnen das Projekt zu diesem Rezept.

18.7 Rezept: Vorhandene Kontakte bearbeiten

Rezept 18.7 nutzt die Fähigkeit der Klasse `ABNewPersonViewController`, ein Formular vorab auszufüllen, um vorhandene Kontakte zu ändern. Die Methode `modify` zeigt als Erstes einen Controller zur Personenauswahl an. Wenn der Benutzer einen Kontakt auswählt, füllt das Programm mit den zugehörigen Kontaktinformationen einen neuen Personenansichtscontroller, der dann auf den Navigationsstack geschoben wird.

Wie in *Rezept 18.6* unterscheidet der Code zwischen Abbruch und Fertigstellung, allerdings wird nicht versucht, einen vorhandenen Kontakt zu ersetzen. Stattdessen wird der Kontakt einfach wieder hinzugefügt. Der Datensatzbezeichner wird durch die Personenauswahl festgelegt, und da die beiden Bezeichner identisch sind, ist es möglich, die vorhandenen Daten mit den Änderungen zu überschreiben.

```
- (void)newPersonViewController:
   (ABNewPersonViewController *)newPersonViewController
   didCompleteWithNewPerson:(ABRecordRef)person
{
   if (person)
   {
      // Speichert den bearbeiteten Kontakt
      ABContact *contact = [ABContact contactWithRecord:person];
      self.title = [NSString stringWithFormat:
         @"Updated %@", contact.compositeName];
      [ABContactsHelper addContact:contact withError:nil];
   }
   else
      self.title = @"Cancelled";

   [self.navigationController popViewControllerAnimated:YES];
}


- (BOOL)peoplePickerNavigationController:
   (ABPeoplePickerNavigationController *)peoplePicker
   shouldContinueAfterSelectingPerson:(ABRecordRef)person
```

```
{
    [self dismissModalViewControllerAnimated:YES];
    [peoplePicker release];
    ABContact *contact = [ABContact contactWithRecord:person];

    if (doModify)
    {
        // Handhabt die Bearbeitungsanforderung, indem der neue
        // Personenansichtscontroller vorab ausgefüllt wird
        ABNewPersonViewController *npvc =
            [[ABNewPersonViewController alloc] init];
        npvc.displayedPerson = contact.record;
        npvc.newPersonViewDelegate = self;
        [self.navigationController pushViewController:npvc
            animated:YES];
    }
    return NO;
}

- (BOOL)peoplePickerNavigationController:
    (ABPeoplePickerNavigationController *)peoplePicker
    shouldContinueAfterSelectingPerson:(ABRecordRef)person
    property:(ABPropertyID)property
    identifier:(ABMultiValueIdentifier)identifier
{
    // Erforderliche Methode, die bei reiner Personenauswahl aber nie
    // aufgerufen wird
    [self dismissModalViewControllerAnimated:YES];
    [peoplePicker release];
    return NO;
}

- (void)peoplePickerNavigationControllerDidCancel:
    (ABPeoplePickerNavigationController *)peoplePicker
{
    // Handhabt einen Abbruchvorgang durch Entfernen des Controllers
    [self dismissModalViewControllerAnimated:YES];
    [peoplePicker release];
}


- (void) modify
```

```
{
    doModify = YES;
    // Ruft einen neuen Picker-Controller auf
    ABPeoplePickerNavigationController *ppnc =
        [[ABPeoplePickerNavigationController alloc] init];
    ppnc.peoplePickerDelegate = self;
    [self presentModalViewController:ppnc animated:YES];
}
```

► *Rezept 18.7: Einen Adressbuchkontakt auswählen und bearbeiten*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 18 und öffnen das Projekt zu diesem Rezept.

18.8 Rezept: ABUnknownPersonViewController

Was geschieht, wenn Sie einige Informationen wie eine E-Mail-Adresse oder eine Telefonnummer haben, aber keinen Kontakt dazu? Mit `ABUnknownPersonViewController` können Sie solche fragmentarischen Informationen zu einem neuen oder einem vorhandenen Kontakt hinzufügen. Diese Klasse dient dazu, bekannte Eigenschaften mit Kontakten für unbekannte Personen zu verknüpfen. Sie funktioniert wie folgt.Initialisieren Sie den Ansichtscontroller, und erstellen Sie anschließend einen Datensatz, den Sie vorab ausfüllen. *Rezept 18.8* definiert einen Datensatz, der nur aus einer einzigen E-Mail-Adresse besteht. Allerdings können Sie auch noch weitere Elemente hinzufügen, die dann alle in dem Ansichtscontroller angezeigt werden. *Abbildung 18.6* (links) zeigt den Controller aus diesem Rezept mit der allein stehenden E-Mail-Adresse. Den vorab ausgefüllten Datensatz weisen Sie der Eigenschaft `displayedPerson` zu.

Die Schaltflächen **Create New Contact** (neuen Kontakt erstellen) und **Add to Existing Contact** (zu vorhandenem Kontakt hinzufügen) werden durch die Eigenschaft `allowsAddingToAddressBook` gesteuert. Tippt der Benutzer auf die Schaltfläche für einen neuen Kontakt, so erscheint ein Formular (*Abbildung 18.6*, Mitte), das mit den Eigenschaften aus dem Datensatz ausgefüllt ist. Wenn Sie diese Eigenschaft deaktivieren, erscheinen die beiden Schaltflächen nicht (*Abbildung 18.6*, rechts). Setzen Sie `allowsAction` auf `YES`, so kann der Benutzer auf die Elemente tippen, um Kontakt mit der E-Mail-Adresse, Telefonnummer usw. aufzunehmen. Dies ist eine praktische Möglichkeit, um eine Liste interaktiver Kontaktinformationen und URLs in einem vordefinierten Ansichtscontroller anzuzeigen.

▶ *Abbildung 18.6: Mit dem Controller für unbekannte Personen können Sie Eigenschaften anzeigen und zu einem neuen oder vorhandenen Kontakt hinzufügen (links). Wenn der Benutzer auf* **CREATE NEW CONTACT** *tippt, wird das Formular für den neuen Kontakt mit den von Ihnen festgelegten Eigenschaften vorab ausgefüllt (Mitte). Wenn Sie die Adressbuchfunktionen deaktivieren, aber Aktionen aktivieren, erstellen Sie eine Seite, in der der Benutzer durch Antippen Zugriff auf Telefonnummern, E-Mail-Adressen, URLs usw. hat (rechts).*

Die Eigenschaften `alternateName` und `message` stellen den Text für das Namens- und das Organisationsfeld in *Abbildung 18.6* (links) bereit. Sie können diese Felder über den Datensatz zwar mit Daten füllen, doch diese alternativen Angaben werden nicht in den Kontakt übernommen. Es handelt sich um eine gute Möglichkeit, die Benutzer auf etwas aufmerksam zu machen, ohne dass dies irgendwelche Nebenwirkungen hätte.

Legen Sie die Eigenschaft `unknownPersonViewDelegate` fest, und deklarieren Sie das Protokoll `ABUnknownPersonViewControllerDelegate`, um die Methode `unknownPersonViewController:didResolveToPerson:` empfangen zu können. Diese Methode wird aufgerufen, wenn der Benutzer auf **DONE** tippt, und ermöglicht es Ihnen, den Datensatz abzurufen, in dem die Daten gespeichert wurden. Sie bietet auch die Gelegenheit, den Ansichtscontroller zu entfernen und freizugeben, da Sie zu diesem Zeitpunkt wissen, dass der Benutzer das Dialogfeld nicht mehr braucht.

```
- (void)unknownPersonViewController:
   (ABUnknownPersonViewController *)unknownPersonView
   didResolveToPerson:(ABRecordRef)person
{
   [self.navigationController popViewControllerAnimated:YES];
   [unknownPersonView release];
}

- (void) action: (UIBarButtonItem *) bbi
{
   // Erstellt den Datensatz und füllt ihn vorab aus
   ABContact *contact = [ABContact contact];
   NSArray *emails = [NSArray arrayWithObject:
      [ABContact dictionaryWithValue:@"feedback@ericasadun.com"
```

```
        andLabel:kABWorkLabel]];
    contact.emailDictionaries = emails;

    // Erstellt den Controller
    ABUnknownPersonViewController *upvc =
        [[ABUnknownPersonViewController alloc] init];
    upvc.unknownPersonViewDelegate = self;

    // Aktionen führen Aufrufe durch, senden Text, E-Mails usw. Setzen Sie
    // diese Eigenschaft auf NO, um dies zu unterbinden.
    upvc.allowsActions = NO;

    // YES bedeutet, dass diese Eigenschaften zu einem neuen oder
    // vorhandenen Kontakt hinzugefügt werden können
    upvc.allowsAddingToAddressBook = YES;

    // Standardwert, der anstellte des Namens angezeigt wird
    upvc.alternateName = @"Unknown Person";

    // Optionaler Text, der unterhalb des alternativen Namens angezeigt
    // wird
    upvc.message = @"What do you want to do?";

    upvc.displayedPerson = contact.record;

    [self.navigationController pushViewController:upvc animated:YES];
}
```

► *Rezept 18.8: Einzelne Eigenschaften zu Kontakten hinzufügen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 18 und öffnen das Projekt zu diesem Rezept.

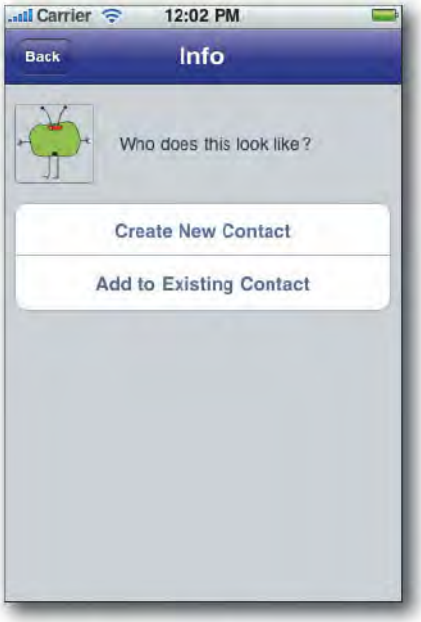
18.9 Ein letzter Punkt: Zufallsbilder hinzufügen

Das Projekt *Monster ID* ist eine Sammlung von Bildern, die Körperteile zeigen, die zu Zufallsbildern zusammengesetzt werden können. Es wurde von Andreas Gohr entwickelt und geht auf einen Webpost von Don Park und die »kombinierten Tierchen« (»combinatoric critters«) zurück. Aus vorhandenen Zeichnungen von Armen, Beinen, Körperrümpfen usw. ergibt sich das zusammengesetzte Bild eines vollständigen Lebewesens.

18.9 Ein letzter Punkt: Zufallsbilder hinzufügen

Die Methode `randomImage` aus *Rezept 18.9* erstellt aus solchen Bestandteilen das Bild eines kleinen Monsters. Die fertige Grafik kann direkt zu einem Adressbuchkontakt hinzugefügt oder im Controller für unbekannte Personen als Ausgangspunkt für einen neuen Kontakt verwendet werden. In jedem Fall können Sie eine `ABContact`-Instanz verwenden und ihr über die Eigenschaft `image` ein Bild zuweisen. Wenn Sie nicht den Controller benutzen, dürfen Sie nicht vergessen, das Adressbuch zu speichern, nachdem Sie das Bild des Datensatzes geändert haben.

Rezept 18.9 verfolgt den Ansatz mit dem Controller für unbekannte Personen. Wenn der Benutzer auf eine Aktionsschaltfläche tippt, wird ein Monster ID-Bild angefordert und in dem Controller angezeigt, wie Sie in *Abbildung 18.7* sehen. Das Bild erscheint auf der Seite **Info**. Mit den üblichen **Create**- und **Add**-Schaltflächen kann der Benutzer das generierte Bild zu einem neuen oder vorhandenen Kontakt hinzufügen. Allerdings können die Benutzer auch auf **Back** tippen, um die Seite zu verlassen, ohne das Bild irgendwo hinzufügen. Jedes Mal, wenn die Seite erscheint, wird ein neues Monster erschaffen.



▶ *Abbildung 18.7: Im Controller für unbekannte Personen können Sie nicht nur E-Mail-Adressen, Telefonnummern und andere Textdaten zu Kontakten hinzufügen, sondern auch Bilder.*

```
// Grafiken von http://www.splitbrain.org/go/monsterid
- (UIImage *) randomImage
{
    // Erstellt ein Zufallsbild aus Monster ID-Grafiken

    CGRect rect = CGRectMake(0.0f, 0.0f, 120.0f, 120.0f);
    UIGraphicsBeginImageContext(CGSizeMake(120.0f, 120.0f));
    UIImage *part;
    part = [UIImage imageNamed:IMAGEFILE(@"oldarms_%d.png", 5)];
    [part drawInRect:rect];
```

```
    part = [UIImage imageNamed:IMAGEFILE(@"oldlegs_%d.png", 5)];
    [part drawInRect:rect];
    part = [UIImage imageNamed:IMAGEFILE(@"oldbody_%d.png", 15)];
    [part drawInRect:rect];
    part = [UIImage imageNamed:IMAGEFILE(@"oldmouth_%d.png", 10)];
    [part drawInRect:rect];
    part = [UIImage imageNamed:IMAGEFILE(@"oldeyes_%d.png", 15)];
    [part drawInRect:rect];
    part = [UIImage imageNamed:IMAGEFILE(@"oldhair_%d.png", 5)];
    [part drawInRect:rect];

    UIImage *image = UIGraphicsGetImageFromCurrentImageContext();
    UIGraphicsEndImageContext();

    return image;
}

- (void) action: (UIBarButtonItem *) bbi
{
    // Erstellt einen leeren Kontakt mit einem Zufallsbild
    ABContact *contact = [ABContact contact];
    contact.image = [self randomImage];

    // Zeigt den Controller für unbekannte Personen an
    ABUnknownPersonViewController *upvc =
        [[ABUnknownPersonViewController alloc] init];
    upvc.unknownPersonViewDelegate = self;
    upvc.allowsActions = NO;
    upvc.allowsAddingToAddressBook = YES;
    upvc.message = @"Who does this look like?";
    upvc.displayedPerson = contact.record;

    [self.navigationController pushViewController:upvc animated:YES];
}

- (void) viewDidLoad
{
    srandom(time(0)); // Zufallsgenerator
    self.navigationController.navigationBar.tintColor =
        COOKBOOK_PURPLE_COLOR;
    self.navigationItem.rightBarButtonItem =
        BARBUTTON(@"Action", @selector(action:));
}
```

► *Rezept 18.9: Zufällige Grafiken im Controller für unbekannte Personen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 18 und öffnen das Projekt zu diesem Rezept.

18.10 Zusammenfassung

In diesem Kapitel wurden die wichtigsten Adressbuchfunktionen der Frameworks `AddressBook` und `AddressBookUI` vorgestellt. Beachten Sie noch folgende abschließende Gedanken über diese Frameworks:

- Die systemnahen Adressbuchfunktionen sind zwar nützlich, der Umgang damit kann aber sehr mühselig sein. Die verschiedenen Hilfsklassen in diesem Kapitel können Ihnen die Arbeit erleichtern.
- Bilder in einem Adressbuch können Sie genauso wie alle anderen Felder abrufen und bearbeiten. Geben Sie einfach Bilddaten statt Strings, Datumsangaben oder mehrwertiger Arrays an. Scheuen Sie auch nicht davor zurück, in Kontaktanwendungen Bilddaten einzusetzen.
- Die Ansichtscontroller des Frameworks `AddressBookUI` arbeiten nahtlos mit den zugrunde liegenden `AddressBook`-Routinen zusammen. Für die üblichen Adressbuchaufgaben besteht keine Notwendigkeit, eigene Benutzeroberflächen zu gestalten.
- Der Controller für unbekannte Personen bietet eine hervorragende Möglichkeit, um einzelne Informationen (z. B. die E-Mail-Adresse eines Unternehmens, eine wichtige Website usw.) in einem Kontakt zu speichern und gleichzeitig den Benutzern die Entscheidung zu überlassen, wo diese Informationen abgelegt werden sollen (und ob dies überhaupt geschehen soll).

19 Core Data

Beginnend mit dem SDK in der Version 3.0 hat Apple auf dem iPhone endlich auch *Core Data* zur Verfügung gestellt, ein Framework für dauerhafte Datenspeicherung. Damit können Sie aus Ihrer Anwendung heraus verwaltete Datenspeicher abfragen und aktualisieren. Mit Core Data gewinnen Sie eine Objektschnittstelle auf der Grundlage von Cocoa Touch, die die relationale Datenverwaltung für die iPhone-Entwicklung mit Objective-C statt mit SQL-Abfragen möglich macht. Dieses Kapitel bildet eine Einführung in Core Data. Sie erhalten hier nur wenige Rezepte, um Ihnen einen Eindruck von dieser Technologie zu vermitteln und einen Ausgangspunkt für eigene Studien zu geben. In diesem Kapitel sehen Sie Core Data im Einsatz. Sie erhalten einen Überblick darüber und arbeiten einige typische Anwendungsmöglichkeiten durch. Eine vollständige Erläuterung von Core Data kann dieses Kapitel nicht leisten (es gibt eigene Bücher zu diesem Thema), aber es bietet einen Ausgangspunkt für alle, die diese Technologie in ihre iPhone-Anwendungen aufnehmen möchten.

19.1 Einführung in Core Data

Core Data vereinfacht die Erstellung und Verwaltung von verwalteten Objekten durch Ihre Anwendung. Vor dem SDK 3.0 erfolgten die Datenverwaltung und der SQL-Zugriff ausschließlich über sehr systemnahe Bibliotheken. Das war weder elegant noch einfach anzuwenden. Jetzt ist Core Data in die Framework-Familie von Cocoa Touch aufgenommen worden und ermöglicht eine einfachere Datenverwaltung auf dem iPhone. Core Data bietet eine flexible Infrastruktur für die Objektverwaltung, Werkzeuge für die Arbeit mit dauerhaften Datenspeichern und Lösungen für den gesamten Lebenszyklus eines Objekts.

Im MVC-Programmierprinzip (Model-View-Controller) gehört Cora Data zum Modell. Die anwendungsspezifischen Daten werden außerhalb der grafischen Benutzeroberfläche definiert und gesteuert. Daher sind die Anwendungsdelegierung, die Instanzen des Ansichtscontrollers und die maßgeschneiderten Modellklassen die Orte, an denen die Funktionen von Core Data Gestalt annehmen. Wen Sie zum Besitzer der Objekte machen, hängt davon ab, wie Sie die Daten verwenden.

In der Regel besitzt die Anwendungsdelegierung gewöhnlich alle gemeinsam genutzten Datenbanken, die in der Anwendung eingesetzt werden. In einfacheren Anwendungen kann ein einziger Ansichtscontroller ausreichen, der sich auch um den Datenzugriff kümmert. Entscheidend ist, dass der Besitzer den gesamten Datenzugriff steuert – das Lesen, Schreiben und Aktualisieren. Jeder Teil der Anwendung, der auf Core Data zurückgreift, muss sich mit diesem Besitzer koordinieren.

Das Datenmodell der Anwendung ist von der grafischen Oberfläche getrennt, doch die Daten existieren nicht im luftleeren Raum. Das SDK 3.0 arbeitet nahtlos mit `UITableView`-Instanzen zusammen. Die Controllerklasse für abgerufene Ergebnisse in Cocoa Touch wurde für die Verwendung mit Tabellen entworfen und bietet praktische Eigenschaften und Methoden für die Aufnahme der Daten in Tabellen. Diese Kombination sehen Sie in den Rezepten dieses Kapitels.

19.1.1 Modelldateien erstellen und bearbeiten

Modelldateien definieren, wie Core Data-Objekte strukturiert sind. Jedes Projekt, das mit dem Framework Core Data verknüpft ist, enthält mindestens eine Modelldatei. Diese `.xcdatamodel`-Dateien definieren die Objekte sowie deren Attribute und Beziehungen.

Jedes Objekt kann eine Reihe sogenannter Attribute als Eigenschaften aufweisen. Mögliche Attributtypen sind Strings, Datumsangaben, Zahlen und Daten. Jedes Objekt kann außerdem über Beziehungen verfügen, also über Verknüpfungen zu anderen Objekten. Dabei kann es sich um einzelne (1:1-Beziehung) oder mehrfache (1:n-Beziehung) Verbindungen handeln. Diese Beziehungen verlaufen in einer Richtung, können aber auch umgekehrt werden, um eine inverse Beziehung anzuzeigen.

Das Modell definieren Sie in Xcode, indem Sie eine neue Datenmodelldatei gestalten. Dazu wählen Sie **File ▸ New File ▸ iPhone OS ▸ Resource ▸ Data Model ▸ Next**. Geben Sie einen Namen für die neue Datei ein, und klicken Sie auf **Next** und dann auf **Finish**. Xcode fügt die neue Modelldatei zu Ihrem Projekt hinzu.

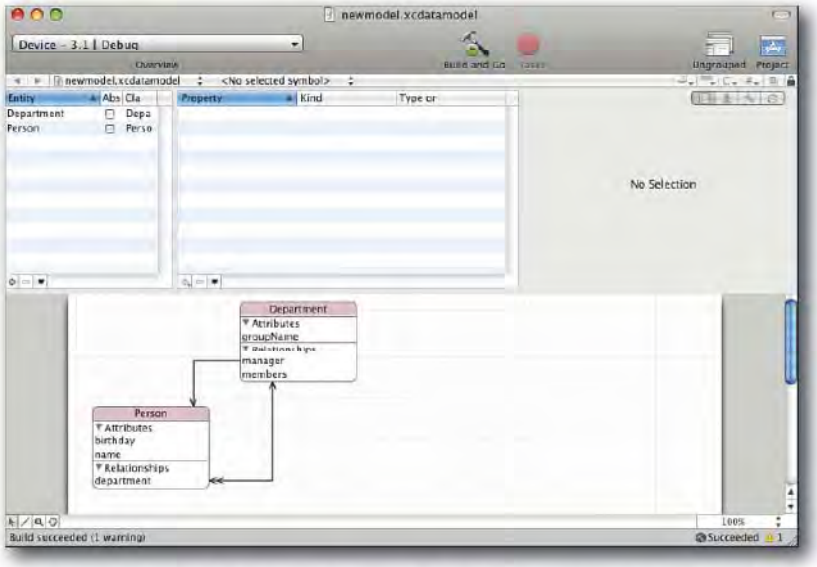
Doppelklicken Sie auf die `.xcdatamodel`-Datei, um sie in einem Editorfenster zu öffnen (siehe *Abbildung 19.1*). In der Liste oben links fügen Sie neue Objektentitäten hinzu (im Grunde genommen »Klassen«), und oben in der Mitte definieren Sie Attribute und Beziehungen (letzten Endes sind das »Instanzvariablen«). Oben rechts finden Sie ein kontextabhängiges Informationsfeld. Darunter erscheint ein Objektdiagramm, das die von Ihnen definierten Entitäten in einem Raster grafisch darstellt.

Um dem Modell eine neue Entität hinzuzufügen (eine Klassendefinition als Grundgerüst für ein Objekt), klicken Sie unten links in der Liste **Entity** auf die Schaltfläche **+**. Standardmäßig werden alle neuen Entitäten zur Laufzeit als Instanzen der Klasse `NSManagedObject` instanziiert. Um dem neuen Objekt einen Namen zu geben, z. B. `Person`, ändern Sie den vorgeschlagenen Namen `Entity`.

Wenn eine Entität markiert ist, können Sie ihr Attribute hinzufügen. Wie bei der Definition von Instanzvariablen geben Sie auch einem Attribut einen Namen und einen Typ. Beziehungen sind Zeiger zu anderen Objekten. Sie können einen einzelnen Zeiger für eine 1:1-Beziehung (z. B. ein Leiter für eine Abteilung) oder einen Satz von 1:n-Beziehungen (alle Mitglieder einer Abteilung) einrichten.

Beachten Sie das Informationsfeld oben rechts. Hier können Sie den Namen des Objekts ändern, seinen Typ festlegen, seinen Standardwert angeben usw.

Das Diagramm in der unteren Hälfte des Editors zeigt die von Ihnen definierten Entitäten und die Beziehungen zwischen ihnen durch Pfeile an. In dem Modell aus der Abbildung gehört jede Person zu einer Abteilung. Eine Abteilung hat einen Leiter (1:1-Beziehung) und eine beliebige Anzahl von Mitgliedern (1:n-Beziehung).



▸ *Abbildung 19.1: Im Modell-Editor können Sie Objektdefinitionen für Core Data-Anwendungen erstellen.*

19.1.2 Headerdateien erstellen

Nachdem Sie das Modell gestaltet und gespeichert haben, können Sie Headerdateien für die einzelnen Entitäten erstellen. Diese Dateien sind nicht erforderlich, erlauben Ihnen aber, in Ihren Anwendungen die Punktschreibweise zu nutzen, anstatt `valueForKey:`-Aufrufe einzusetzen, um die Attribute verwalteter Objekte abzurufen.

Markieren Sie eine Entität, und wählen Sie **File ▸ New File ▸ iPhone OS ▸ Cocoa Touch Class ▸ Managed Object Class ▸ Next ▸ Next ▸ Finish**. Xcode erstellt ein Paar aus einer `.h`- und einer `.m`-Datei für die Entität und fügt sie dem Projekt hinzu. Die Headerdatei für die Abteilung aus dem Projekt von *Abbildung 19.1* sieht beispielsweise wie folgt aus:

```
#import <CoreData/CoreData.h>
@interface Department : NSManagedObject
{
}

@property (nonatomic, retain) NSString * groupName;
@property (nonatomic, retain) NSManagedObject * manager;
@property (nonatomic, retain) NSSet* members;
@end

@interface Department (CoreDataGeneratedAccessors)
- (void)addMembersObject:(NSManagedObject *)value;
- (void)removeMembersObject:(NSManagedObject *)value;
- (void)addMembers:(NSSet *)value;
- (void)removeMembers:(NSSet *)value;
@end
```

Sie können erkennen, dass der Gruppenname ein String ist, dass der Abteilungsleiter (`manager`) auf ein anderes verwaltetes Objekt zeigt und dass die 1:n-Beziehungen der Mitglieder als ein Satz definiert sind.

Diese Datei sieht zwar wie ein normaler Klassenheader in Objective-C aus, doch gibt es keine Implementierungen. Darum kümmert sich Core Data mithilfe des Compiler-Schlüsselworts `@dynamic`.

```
@implementation Department

@dynamic groupName;
@dynamic manager;
@dynamic members;

@end
```

Der Hauptgrund dafür, Core Data-Dateien zu erstellen, liegt darin, dass sie dem Projekt neue Verhaltensweisen und flüchtige Attribute hinzufügen können, also Attribute, die nicht im dauerhaften Speicher abgelegt werden. Beispielsweise können Sie das Attribut `fullName` erstellen, das aus den Attributen `firstName` und `lastName` einer Person zusammengesetzt wird. Außerdem hindert Sie nichts daran, eine Klasse für verwaltete Objekte wie jede andere Klasse auch zu nutzen, also alle Arten von Daten zu verwenden und zu bearbeiten. Sie können jegliches Objective-C-Verhalten in eine Instanz verwalteter Objekte aufnehmen, indem Sie die Implementierungsdatei bearbeiten. Fügen Sie Instanz- und Klassenmethoden nach Bedarf hinzu.

19.1.3 Einen Core Data-Kontext erstellen

Nachdem Sie das Modell für die verwalteten Objekte erstellt haben, ist es an der Zeit, den Code zu schreiben, der auf eine Datendatei zugreift. Um mit Core Data zu arbeiten, müssen Sie im Programm einen Kontext für verwaltete Objekte erstellen. Dieser Kontext führt alle Zugriffs- und Aktualisierungsfunktionen aus, die erforderlich sind, um das Modell und die Datei aufeinander abzustimmen.

Die folgende Methode initialisiert den Kontext für eine Anwendung. Wenn Sie eine vorgefertigte Core Data-Vorlage verwenden, ist diese Arbeit für Sie schon erledigt. Anhand der hier vorgestellten Methode können Sie ablesen, wie Sie diese Aufgabe manuell lösen. Als Erstes werden sämtliche Modelldateien aus dem Anwendungsbundle gelesen und zu einem zentralen Modell zusammengefasst. Anschließend initialisiert die Methode einen Koordinator für den dauerhaften Speicher. Dieser Koordinator bietet einen systemnahen Dateizugriff über das zentrale Modell. Dazu geben Sie einen URL an, der auf die Datei zeigt, in der die Modelldaten gespeichert werden sollen. Abschließend initialisiert die Methode mithilfe des Koordinators den neuen Kontext und speichert ihn als beibehaltene Instanzvariable.

```
- (void) initCoreData
{
   NSError *error;

   // Pfad zur SQLite-Datei
   NSString *path = [NSHomeDirectory()
      stringByAppendingString:@"/Documents/cdintro_00.sqlite"];
   NSURL *url = [NSURL fileURLWithPath:path];

   // Initialisiert das Modell
   NSManagedObjectModel *managedObjectModel =
      [NSManagedObjectModel mergedModelFromBundles:nil];

   // Richtet den Koordinator für den dauerhaften Speicher ein
   NSPersistentStoreCoordinator *persistentStoreCoordinator =
      [[NSPersistentStoreCoordinator alloc]
         initWithManagedObjectModel:managedObjectModel];

   if (![persistentStoreCoordinator
      addPersistentStoreWithType:NSSQLiteStoreType
      configuration:nil URL:url options:nil error:&error])
         NSLog(@"Error %@", [error localizedDescription]);
   else
   {
      // Erstellt den Kontext und weist den Koordinator zu
      self.context =
         [[[NSManagedObjectContext alloc] init] autorelease];
      [self.context
         setPersistentStoreCoordinator:persistentStoreCoordinator];
   }
   [persistentStoreCoordinator release];
}
```

Es ist wichtig, eine Kontextinstanz zu unterhalten, auf die Sie von einem Ansichtscontroller (für einfache Core Data-Anwendungen) oder von der Anwendungsdelegierung (für kompliziertere Anwendungen) aus verweisen können. Der Kontext wird für sämtliche Lese-, Schreib- und Aktualisierungsvorgänge in Ihrer Anwendung eingesetzt.

19.1.4 Neue Objekte hinzufügen

Neue Objekte erstellen Sie, indem Sie Entitäten in den verwalteten Kontext einfügen. Im folgenden Codefragment werden drei neue Elemente erstellt, nämlich eine Abteilung und zwei Personen. Nachdem Sie das verwaltete Objekt eingefügt haben, wodurch eine neue Instanz zurückgegeben wird, legen Sie seine Eigenschaften (die Attribute und Beziehungen) durch Zuweisung fest. Jede Person gehört zu einer Abteilung, jede Abteilung hat eine Reihe von Mitgliedern und einen Leiter. Der Code folgt dem Entwurf aus *Abbildung 19.1*. Für eine Abteilung müssen Sie nicht explizit die Mitglieder angeben, denn wenn Sie das Attribut `department` einer Person festlegen, fügt die inverse Beziehung sie als Mitglied zu der Abteilung hinzu.

```
- (void) addObjects
{
    // Fügt eine neue Abteilung hinzu
    Department *department = (Department *)[NSEntityDescription
        insertNewObjectForEntityForName:@"Department"
        inManagedObjectContext:self.context];
    department.groupName = @"Office of Personnel Management";

    // Fügt eine Person hinzu
    Person *person1 = (Person *)[NSEntityDescription
        insertNewObjectForEntityForName:@"Person"
        inManagedObjectContext:self.context];
    person1.name = @"John Smith";
    person1.birthday = [self dateFromString:@"12-1-1901"];
    person1.department = department;

    // Fügt eine weitere Person hinzu
    Person *person2 = (Person *)[NSEntityDescription
        insertNewObjectForEntityForName:@"Person"
        inManagedObjectContext:self.context];
    person2.name = @"Jane Doe";
    person2.birthday = [self dateFromString:@"4-13-1922"];
    person2.department = department;

    // Richtet die Abteilungsbeziehung ein
    department.manager = person1;
```

```
    // Legt die Daten im dauerhaften Speicher ab
    NSError *error;
    if (![self.context save:&error])
        NSLog(@"Error %@", [error localizedDescription]);
}
```

Änderungen am dauerhaften Speicher erfolgen erst, wenn Sie speichern. Der Speichervorgang bringt die Datenbankdatei auf den Stand des Modells im Arbeitsspeicher. Die einzige Speicheranforderung in diesem Code weist den Kontext an, seinen Status mit dem dauerhaften Speicher zu synchronisieren und alle Änderungen in die Datenbankdatei zu schreiben.

Wenn Sie diesen Code im Simulator ausführen, können Sie die dadurch erstellte `.sqlite`-Datei auf einfache Weise einsehen. Wechseln Sie zum Simulator-Ordner (`~/Library/Application Support/iPhone Simulator/User/Applications`) und dort zu dem Ordner für die Anwendung selbst. In einem Dokumentenordner (dessen Pfad von dem URL abhängt, den Sie zum Erstellen des dauerhaften Speichers verwendet haben) finden Sie eine `.sqlite`-Datei mit der Datenbankdarstellung, die Sie erstellt haben.

Mit dem Kommandozeilenwerkzeug `sqlite3` können Sie den Inhalt dieser Datei untersuchen, indem Sie den Befehl `.dump` verwenden. Im Folgenden sehen Sie die Definitionen für zwei SQL-Tabellen (`department` und `manager`), in denen die Informationen der einzelnen Objekte abgelegt werden, sowie die Einfügebefehle, um die im Code erstellten Instanzen zu speichern.

```
% sqlite3 cdintro_oo.sqlite
SQLite version 3.4.0
Enter ".help" for instructions
sqlite> .dump
BEGIN TRANSACTION;
CREATE TABLE ZDEPARTMENT ( Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT
INTEGER,
ZMANAGER INTEGER, ZGROUPNAME VARCHAR );
INSERT INTO "ZDEPARTMENT" VALUES(1,1,1,1,'Office of Personnel Management');
CREATE TABLE ZPERSON ( Z_PK INTEGER PRIMARY KEY, Z_ENT INTEGER, Z_OPT INTE-
GER,
ZDEPARTMENT INTEGER, ZBIRTHDAY TIMESTAMP, ZNAME VARCHAR );
INSERT INTO "ZPERSON" VALUES(1,2,1,1,-312687720O,'John Smith');
INSERT INTO "ZPERSON" VALUES(2,2,1,1,-248423400O,'Jane Doe');
CREATE TABLE Z_PRIMARYKEY (Z_ENT INTEGER PRIMARY KEY, Z_NAME VARCHAR, Z_SU-
PER
INTEGER, Z_MAX INTEGER);
INSERT INTO "Z_PRIMARYKEY" VALUES(1,'Department',0,1);
INSERT INTO "Z_PRIMARYKEY" VALUES(2,'Person',0,2);
CREATE TABLE Z_METADATA (Z_VERSION INTEGER PRIMARY KEY, Z_UUID VARCHAR(255),
Z_PLIST BLOB);
INSERT INTO "Z_METADATA" VALUES(1,'A4ADDA90-5C26-4E01-8E68-
1C4BB7A910B1',X'62706C6973743030D6010203040506070809 0A0B105F10204E5353746F
```

```
72654D6F64656C56657273696F6E4861736865735665727369 6F6E5F101E4E5353746F7265
4D6F64656C56657273696F6E4964656E746966696572735B4E5353746F7265547970655F-
101D4E535065727369737465 6E63654672616D65776F726B56657273696F6E-
5F10194E5353746F72654D6F64656C56657273696F6E4861736865735F10125F-
4E534175746F56616375756D4C6576656C1003A05653514C6974651 0F1D20C-
0D0E0F5A4465706172746D656E7456506572736F6E4F10203B34C3D1DAC-
08316B2656664A26C9EAC82FE04E4C34FC75A3E0981E678F3909B4F1020DE27A38E
814A2A5F72418573563732F83CBC1CACAADF39FA559420B155E5A9735132000800-
150038005900650085 00A100B600B800B900C000C200C700D200D-
900FC011F0000000000000201000000000000001100000000000000000000000000000121');
CREATE INDEX ZDEPARTMENT_ZMANAGER_INDEX ON ZDEPARTMENT (ZMANAGER);
CREATE INDEX ZPERSON_ZDEPARTMENT_INDEX ON ZPERSON (ZDEPARTMENT);
COMMIT;
sqlite>
```

19.1.5 Die Datenbank abfragen

Um Objekte aus der Datenbank abzurufen, führen Sie eine `fetch`-Anforderung durch, in der Sie die Suchkriterien angeben. Diese Anforderung wird weitergeleitet und zur Initialisierung eines Ergebnisobjekts mit einem Zeiger auf die verwalteten Objekte verwendet, die den Kriterien entsprechen. Der Ergebniscontroller führt den Abruf durch, bevor er dieses Array passender verwalteter Objekte zurückgibt.

Die folgende Methode `fetchObjects` erstellt eine Anforderung und setzt den Entitätstyp dabei auf `Person`, sodass bei der Suche nach `Person`-Objekten im gemeinsam genutzten verwalteten Speicher geforscht wird. Jede Anforderung muss mindestens einen Sortierungsdeskriptor enthalten. In diesem Beispiel wird eine Liste von `Person`-Datensätzen zurückgegeben, die in aufsteigender Reihenfolge nach dem Namensfeld geordnet sind. Der Code zeigt nur die einfachste Form von Anfrage – »Gib alle verwalteten Elemente eines bestimmten Typs zurück!« –, doch Sie können auch weit vielschichtigere Abfragen aufstellen.

```
- (void) fetchObjects
{
    // Erstellt eine einfache Abrufanforderung
    NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
    [fetchRequest setEntity:[NSEntityDescription
        entityForName:@"Person" inManagedObjectContext:self.context]];

    // Fügt einen Sortierungsdeskriptor hinzu. Dies ist erforderlich.
    NSSortDescriptor *sortDescriptor = [[NSSortDescriptor alloc]
        initWithKey:@"name" ascending:YES selector:nil];
    NSArray *descriptors = [NSArray arrayWithObject:sortDescriptor];
    [fetchRequest setSortDescriptors:descriptors];
    [sortDescriptor release];
```

```
    // Initialisiert den Controller für abgerufene Ergebnisse
    NSError *error;
    self.results = [[NSFetchedResultsController alloc]
       initWithFetchRequest:fetchRequest
       managedObjectContext:self.context
       sectionNameKeyPath:nil cacheName:@"Root"];
    self.results.delegate = self;
    if (![[self results] performFetch:&error])
       NSLog(@"Error %@", [error localizedDescription]);

    [self.resutls release];
    [fetchRequest release];
}

- (void) listPeople
{
    [self fetchObjects];
    if (!self.results.fetchedObjects.count)
    {
       NSLog(@"Database has no people at this time");
         return;
    }

    NSLog(@"People:");
    for (Person *person in self.results.fetchedObjects)
       NSLog(@"%@ : %@", person.name, person.department.groupName);
}
```

Die `fetch`-Anforderung wird verwendet, um ein `NSFetchedResultsController`-Objekt zu initialisieren. Diese Klasse verwaltet die von einem Core Data-Abruf zurückgegebenen Daten. Der Ergebniscontroller wird über eine beibehaltene Klasseneigenschaft (`self.results`) verfügbar gemacht. Abrufergebnisse bieten konkreten Zugriff auf Objekte des Datenmodells. Nachdem die Daten abgerufen sind, listet die Methode `listPeople:` die einzelnen Personen nach Name und Abteilung auf. Dazu zieht sie die Eigenschaft `fetchedObjects` des Ergebnisses heran.

19.1.6 Änderungen erkennen

Die abgerufenen Ergebnisse können als Datenquelle für eine Tabelle, zum Ausfüllen eines Formulars für Objekteinstellungen oder für jeglichen anderen Zweck herangezogen werden, der Ihnen einfällt. Unabhängig davon, ob Sie nur ein Objekt abrufen oder viele, bietet Ihnen der Ergebniscontroller direkten Zugriff auf die verwalteten Objekte.

Wie stellen Sie aber sicher, dass die abgerufenen Daten aktuell sind? Nachdem Sie neue Objekte hinzugefügt oder den Datenspeicher auf andere Weise geändert haben, sollten Sie einen neuen Ergebnissatz abrufen. Abonnieren Sie den Callback `controllerDidChangeContent:` des Ergebniscontrollers. Diese Methode benachrichtigt Ihre Klasse, wenn es Änderungen gibt, die sich auf die von Ihnen abgerufenen Objekte auswirken könnten. Um diesen Callback zu abonnieren, deklarieren Sie das Protokoll `NSFetchedResultsControllerDelegate` und weisen die Controllerdelegierung wie folgt zu. Nachdem Sie die Ergebnisdelegierung eingerichtet haben, empfangen Sie jedes Mal einen Callback, wenn der Datenspeicher aktualisiert wird.

```
self.results.delegate = self.
```

19.1.7 Objekte entfernen

Objekte zu entfernen kann schwieriger sein, als Sie denken mögen, vor allem dann, wenn diese Objekte nicht nur einfache Eigenschaften, sondern Beziehungen aufweisen. Betrachten Sie den folgenden Code, der die einzelnen Personen in den abgerufenen Ergebnissen durchgeht, sie löscht und anschließend den Kontext speichert. Beim Speichern des Kontextes in einer Datei schlägt diese Methode fehl.

```
- (void) removeObjects
{
   NSError *error = nil;
   for (Person *person in
      self.results.fetchedObjects)
      [self.context deleteObject:person];

   if (![self.context save:&error])
      NSLog(@"Error %@ (%@)", [error localizedDescription]);
   [self fetchObjects];
}
```

Das liegt daran, dass Core Data vor dem Schreiben von Daten die interne Konsistenz sicherstellt und einen Fehler ausgibt, wenn dies nicht möglich ist. Das verwaltete Modell aus *Abbildung 19.1* enthält verschiedene Querverbindungen. Jede Person kann zu einer Abteilung gehören, die ihrerseits eine Liste ihrer Mitglieder und den Namen ihres Leiters speichert. Diese Verweise müssen gelöscht werden, bevor ein Objekt sicher aus dem dauerhaften Speicher entfernt werden kann. Ansonsten könnten Objekte auf gelöschte Elemente zeigen, was zu ungültigen Verweisen führt.

Der folgende Code zeigt eine andere Version der oben gezeigten Methode, bei der eine Speicherung ohne Fehler möglich ist. Hier werden alle Verweise aus dem Abteilungsobjekt entfernt. Die Methode prüft, ob eine Person ein Leiter ist, und entfernt die betreffende Beziehung. Außerdem filtert sie mithilfe eines Prädikats die Person aus der Abteilung heraus, sodass eine aktualisierte Mitgliederliste übrig bleibt. Nachdem diese Beziehungen entfernt wurden, kann der Kontext korrekt gespeichert werden.

```
- (void) removeObjects
{
   NSError *error = nil;
      if (!self.results.fetchedObjects.count)
   {
      NSLog(@"No one to delete");
      return;
   }

   // Entfernt die einzelnen Personen
   for (Person *person in
      self.results.fetchedObjects)
   {
      // Entfernt ggf. die Person als Abteilungsleiter
      if (person.department.manager == person)
         person.department.manager = nil;

      // Entfernt die Person aus ihrer Abteilung
      NSPredicate *pred = [NSPredicate predicateWithFormat:
         @"SELF != %@", person];
      if (person.department.members)
         person.department.members = [person.department.members
            filteredSetUsingPredicate:pred];

      // Löscht das Personenobjekt
      [self.context deleteObject:person];
   }

   // Speichert den Kontext
   if (![self.context save:&error])
      NSLog(@"Error %@ (%@)", [error localizedDescription],[error user-
Info]);
}
```

Sie können Beziehungen nicht nur wie hier manuell löschen, sondern im Datenmodell-Editor auch Core Data-Löschregeln angeben. Damit steuern Sie, wie ein Objekt auf eine versuchte Löschung reagiert. Sie können Löschanforderungen mit `Deny` ablehnen, um vor dem Entfernen eines Objekts sicherzustellen, dass keine Beziehungen mehr vorhanden sind. Mit `Nullify` werden inverse Beziehungen vor dem Löschen eines Objekts zurückgesetzt, mit `Cascade` werden zusammen mit dem Objekt auch alle seine Beziehungen entfernt. Damit können Sie beispielsweise eine ganze Abteilung (einschließlich ihrer Mitglieder) auf einen Schlag löschen. Mit `No Action` sorgen Sie dafür, dass die Objekte, auf die eine Beziehung zeigt, unverändert bleiben, selbst wenn sie auf das zu löschende Objekt zurückzeigen.

In dem Beispielcode zu diesem Kapitel werden die Beziehungen im einführenden Projekt (praktisch Rezept 0 für dieses Kapitel) mit `Nullify` zurückgesetzt. Die Beziehung zwischen Abteilung und Mitglied ist invers. Wenn Sie `Nullify` verwenden (die standardmäßige Löschregel), müssen Sie beim Löschen von Personen die entsprechenden Mitglieder nicht zuvor aus der Abteilungsliste herausnehmen.

Die Abteilungsleiterbeziehung dagegen ist nicht reziprok. Da es keine inverse Beziehung gibt, können Sie einen Abteilungsleiter nicht löschen, ohne einen neuen zu setzen. Die Methode zum Entfernen von Objekten lässt sich mithilfe dieser Löschregeln wie folgt verkürzen:

```
- (void) removeObjects
{
    NSError *error = nil;

    // Entfernt alle Personen (falls sie vorhanden sind)
    [self fetchObjects];
    if (!self.results.fetchedObjects.count)
    {
        NSLog(@"No one to delete");
        return;
    }

    // Entfernt jede einzelne Person
    for (Person *person in
        self.results.fetchedObjects)
    {
        // Entfernt ggf. eine Person als Abteilungsleiter
        if (person.department.manager == person)
            person.department.manager = nil;

        // Löscht das Personenobjekt
        [self.context deleteObject:person];
    }

    // Speichert den Kontext
    if (![self.context save:&error])
        NSLog(@"Error %@ (%@)", [error localizedDescription],[error user-
Info]);
}
```

Xcode gibt Warnungen aus, wenn nicht reziproke Beziehungen erkannt werden. Vermeiden Sie solche Beziehungen, um den Code zu vereinfachen und für eine bessere interne Konsistenz zu sorgen. Wenn Sie unbedingt eine nicht reziproke Beziehung brauchen, müssen Sie sie beim Aufstellen der Löschmethoden berücksichtigen, wie Sie es hier gesehen haben.

19.2 Rezept: Core Data als Datenquelle für Tabellen verwenden

Core Data ist auf dem iPhone eng mit Tabellenansichten verzahnt. Die Klasse `NSFetchedResultsController` enthält Möglichkeiten zur Vereinfachung der Aufnahme von Core Data-Objekten in die Datenquellen von Tabellen. Die folgende Aufstellung zeigt, dass viele Eigenschaften und Methoden der Klasse für aus Tabellen abgerufene Ergebnisse geeignet sind.

- **Zugriff auf den Indexpfad** In der Klasse für abgerufene Ergebnisse können Sie den Objektindexpfad in beiden Richtungen verwenden: Sie können einzelne Elemente aus dem Array der abgerufenen Objekte über den Indexpfad ansprechen, indem Sie `objectAtIndexPath:` aufrufen, es ist aber auch möglich, mit `indexPathForObject:` den Indexpfad für ein abgerufenes Objekt abzufragen. Diese beiden Methoden eignen sich sowohl für in Abschnitte unterteilte als auch für einfache Tabellen (in denen alle Daten in einem einzigen Abschnitt stehen).
- **Pfad zum Abschnittsschlüssel** Die Eigenschaft `sectionNameKeyPath` verknüpft das Attribut eines verknüpften Objekts mit dem Abschnittsnamen, womit Sie bestimmen können, zu welchem Abschnitt das Objekt gehört. Diese Eigenschaft können Sie jederzeit direkt festlegen, aber auch beim Einrichten des Controllers für abgerufene Ergebnisse initialisieren.

  In *Rezept 19.1* wird mit dem Attribut `section` zwischen den Abschnitten unterschieden, allerdings können Sie für diesen Schlüsselpfad auch jeden anderen Attributnamen verwenden. In diesem Beispiel weist das Attribut ein verwaltetes Objekt anhand des ersten Zeichens in seinem Namen zu einem Abschnitt zu. Um eine einfache Tabelle ohne Abschnitte zu erstellen, setzen Sie den Schlüsselpfad auf `nil`.
- **Abschnittsgruppen** Die Untergruppen von Abschnitten rufen Sie mit der Eigenschaft `sections` ab. Sie gibt ein Array aus Abschnitten zurück, die jeweils die verwalteten Objekte mit dem betreffenden Buchstaben als Abschnittsattribut enthalten.

  Jeder zurückgegebene Abschnitt implementiert das Protokoll `NSFetchedResultsSectionInfo`, sodass er seine Objekte, die Anzahl der Objekte, den Namen und einen Indextitel melden kann (den Titel des Schnellzugriffsindex, der optional oberhalb oder rechts von der Tabelle angezeigt wird).
- **Indextitel** Die Eigenschaft `sectionIndexTitles` erstellt eine Liste von Titeln der Abschnitte mit abgerufenen Daten. In *Rezept 19.1* besteht dieses Array aus einbuchstabigen Titeln. In der Standardimplementierung werden die Werte der einzelnen Abschnittsschlüssel verwendet, um eine Liste aller bekannten Abschnitte zurückzugeben.

  Zwei weitere Methoden, `sectionIndexTitleForSectionName:` und `sectionForSectionIndexTitle:atIndex:`, bieten Möglichkeiten zum Nachschlagen von Abschnittstiteln. Die erste gibt einen Titel für einen Abschnittsnamen zurück, die zweite sucht einen Abschnitt anhand seines Titels. Diese Methoden können Sie überschreiben, um andere Abschnittstitel zu verwenden als die im Schlüssel für den Abschnittsnamen gespeicherten Bezeichnungen.

Wie diese Eigenschaften und Methoden zeigen, sind Instanzen abgerufener Ergebnisse für Tabellen geeignet und auch sofort einsatzbereit. In *Rezept 19.1* werden diese Merkmale verwendet, um die indizierte Tabelle für Farbnamen nachzubauen, die bereits in *Kapitel 11, Tabellenansichten erstellen und verwalten*, vorgestellt wurde. Der Code in diesem Rezept entnimmt die Daten anhand von Indexpfaden aus den abgerufenen Ergebnissen. Dies können Sie in der Methode erkennen, die eine Zelle für eine gegebene Zeile erstellt, und in der Methode, die die Navigationsleiste in der Farbe der ausgewählten Zeile tönt.

Alle Methoden zum Erstellen und Verwalten von Abschnitten sind klein. Die integrierten Zugriffsmöglichkeiten von Core Daten reduzieren diese Methoden auf jeweils ein oder zwei Zeilen, da die gesamte Arbeit zum Erstellen der Abschnitte und für den Zugriff vollständig an Core Data übertragen wird. Der Aufruf, der jeweils die Anforderung für den Datenabruf initialisiert, nennt die Datenattribute, die in den Abschnitten verwendet werden sollen. Anschließend übernimmt Core Data und erledigt den Rest.

```
self.fetchedResultsController = [[NSFetchedResultsController alloc]
    initWithFetchRequest:fetchRequest
    managedObjectContext:self.context
    sectionNameKeyPath:@"section" cacheName:@"Root"];
```

Durch Caching wird der Aufwand für die Darstellung von Daten mit Abschnitten und Indizes verringert. Mehrfache Abrufanforderungen werden ignoriert, wenn die Daten sich nicht geändert haben, was die Kosten für solche Anforderungen im Lebenszyklus der Anwendung verringert. Der Name für den Cache ist völlig willkürlich. Geben Sie entweder `nil` an, um die Zwischenspeicherung zu unterbinden, oder einen Namen in Form eines `NSString`. In dem oben gezeigten Codefragment heißt der Cache `Root`, aber Sie können jeden beliebigen anderen String dafür verwenden.

```
- (UITableViewCell *)tableView:(UITableView *)tableView
    cellForRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Ruft eine Zelle ab oder erstellt sie
    UITableViewCell *cell =
        [tableView dequeueReusableCellWithIdentifier:@"basic cell"];
    if (!cell) cell = [[[UITableViewCell alloc]
        initWithStyle:UITableViewCellStyleDefault
        reuseIdentifier:@"basic cell"] autorelease];

    // Entnimmt ein Objekt aus den abgerufenen Ergebnissen
    NSManagedObject *managedObject =
        [self.fetchedResultsController objectAtIndexPath:indexPath];
    cell.textLabel.text = [managedObject valueForKey:@"name"];
    UIColor *color =
        [self getColor:[managedObject valueForKey:@"color"]];
    cell.textLabel.textColor =
        ([[managedObject valueForKey:@"color"] hasPrefix:@"FFFFFF"]) ?
```

```
            [UIColor blackColor] : color;

    return cell;
}

- (void)tableView:(UITableView *)tableView
    didSelectRowAtIndexPath:(NSIndexPath *)indexPath
{
    // Färbt die Navigationsleiste entsprechend der ausgewählten Zelle
    NSManagedObject *managedObject =
        [self.fetchedResultsController objectAtIndexPath:indexPath];
    UIColor *color =
        [self getColor:[managedObject valueForKey:@"color"]];
    self.navigationController.navigationBar.tintColor = color;
    self.searchBar.tintColor = color;
}

#pragma mark Sections
- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    // Gibt die Abschnittsanzahl aus den Ergebnissen zurück
    return [[self.fetchedResultsController sections] count];
}

- (NSInteger)tableView:(UITableView *)tableView
    numberOfRowsInSection:(NSInteger)section
{
    // Gibt die Anzahl der Zeilen für jeden Abschnitt zurück
    return [[[self.fetchedResultsController sections]
        objectAtIndex:section] numberOfObjects];
}

- (NSArray *)sectionIndexTitlesForTableView:(UITableView *)aTableView
{
    // Gibt das Array der Abschnittsindextitel zurück
    NSArray *searchArray = [NSArray arrayWithObject:UITableViewIndexSear
ch];
    return [searchArray
        arrayByAddingObjectsFromArray:
          self.fetchedResultsController.sectionIndexTitles];
}

- (NSString *)tableView:(UITableView *)aTableView
    titleForHeaderInSection:(NSInteger)section
```

```
{
    // Gibt den Titel eines gegebenen Abschnitts zurück
    NSArray *titles = [self.fetchedResultsController
        sectionIndexTitles];
    if (titles.count <= section) return @"Error";
    return [titles objectAtIndex:section];
}

- (NSInteger)tableView:(UITableView *)tableView
    sectionForSectionIndexTitle:(NSString *)title
    atIndex:(NSInteger)index
{
    // Ruft die Titel für den mit dem Indextitel verknüpften Abschnitt ab
    return [self.fetchedResultsController.sectionIndexTitles
        indexOfObject:title];
}
```

▸ *Rezept 19.1: Eine unterteilte Tabelle mit Core Data aufbauen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 19 und öffnen das Projekt zu diesem Rezept.

19.3 Rezept: Core Data in Suchtabellen

Core Data-Speicher sind für die wirkungsvolle Verwendung von `NSPredicate` entworfen. Mit Prädikaten können Sie Abrufanforderungen erstellen, bei denen nur diejenigen verwalteten Objekte ausgewählt werden, die den Prädikatsregeln genügen. Durch das Prädikat werden die Ergebnisse auf passende Objekte eingeschränkt.

In *Rezept 19.2* wird die Suchtabelle aus *Rezept 11.16* für die Verwendung von Core Data angepasst. Hier wird eine Suchleiste verwendet, um Daten aus dem dauerhaften Speicher auszuwählen und die Ergebnisse im Such-Sheet der Tabelle anzuzeigen. *Abbildung 19.2* zeigt den Suchvorgang.

Wenn sich der Text in der Suchleiste oben in der Tabelle ändert, empfängt die Delegierung der Suchleiste einen `searchBar:textDidChange:`-Callback, woraufhin die Methode einen neuen Datenabruf durchführt. *Rezept 19.2* zeigt diese Abrufmethode, die ein einschränkendes Prädikat aufbaut.

▸ *Abbildung 19.2: Um die Suchtabelle mit Core Data-Daten zu versorgen, müssen die abgerufenen Ergebnisse jedes Mal aktualisiert werden, wenn sich der Text im Suchfeld ändert.*

Die Methode `performFetch` in dem Rezept erstellt aufgrund des Textes in der Suchleiste ein einfaches Prädikat. Sie setzt die Eigenschaft `predicate` der Anforderung, um die möglichen Übereinstimmungen auf Namen mit einem bestimmten Text einzuschränken, wobei nicht zwischen Groß- und Kleinschreibung unterschieden wird. Wegen `contains` wird nach Text an beliebiger Stelle im String gesucht, während das `[cd]` dahinter festlegt, dass Groß- und Kleinschreibung sowie diakritische Zeichen bei dem Vergleich nicht herangezogen werden. Diakritische Zeichen sind zusätzliche Symbole an einem Buchstaben, z. B. die Pünktchen eines Umlauts oder die Tilde, die im Spanischen auf dem Buchstaben n stehen kann.

Für anspruchsvollere Abfragen müssen Sie ein zusammengesetztes Prädikat angeben. Darin können Sie einfache Prädikate mit den üblichen logischen Operatoren wie AND, OR und NOT kombinieren. Dazu können Sie mit der Klasse `NSCompoundPredicate` ein zusammengesetztes Prädikat aus einer Reihe von Einzelprädikaten erstellen oder die Operatoren AND, OR und NOT direkt in den `NSPredicate`-Text einfügen, wie es in Kapitel 18, »*Verbindung mit dem Adressbuch*«, getan wurde.

Keine der Methoden aus *Rezept 19.1* muss für die Verwendung mit der Methode `performFetch` aus *Rezept 19.2* verändert werden. Alle Zell- und Abschnittsmethoden sind an das `results`-Objekt und dessen Methoden gebunden, was die Implementierung auch beim Hinzufügen von Suchfunktionen vereinfacht.

```
- (void) performFetch
{
   // Initialisiert eine Abrufanforderung
   NSFetchRequest *fetchRequest = [[NSFetchRequest alloc] init];
   NSEntityDescription *entity = [NSEntityDescription
      entityForName:@"Crayon" inManagedObjectContext:self.context];
```

```
    [fetchRequest setEntity:entity];

    // Sortiert die Farbelemente in aufsteigender Reihenfolge
    NSSortDescriptor *sortDescriptor =
       [[NSSortDescriptor alloc] initWithKey:@"name"
       ascending:YES selector:nil];
    NSArray *descriptors = [NSArray arrayWithObject:sortDescriptor];
    [fetchRequest setSortDescriptors:descriptors];

    // Ruft die Abfrage ab
    NSString *query = self.searchBar.text;
    if (query && query.length) fetchRequest.predicate =
       [NSPredicate predicateWithFormat:@"name contains[cd] %@",
       query];

    // Initialisiert den Controller für abgerufene Ergebnissse
    NSError *error;
    self.fetchedResultsController =
       [[NSFetchedResultsController alloc]
          initWithFetchRequest:fetchRequest
          managedObjectContext:self.context
          sectionNameKeyPath:@"section" cacheName:@"Root"];
    self.fetchedResultsController.delegate = self;
    [self.fetchedResultsController release];
    if (![[self fetchedResultsController] performFetch:&error])
       NSLog(@"Error %@", [error localizedDescription]);

    [fetchRequest release];
    [sortDescriptor release];
}
```

► *Rezept 19.2: Prädikate für Abrufergebnisse*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 19 und öffnen das Projekt zu diesem Rezept.

19.4 Rezept: Datenbearbeitung in Core Data-Tabellen

Rezept 19.3 zeigt, wie Sie eine elementare Tabellenbearbeitung auch mit Core Data möglich machen. Der Code basiert auf den grundlegenden Bearbeitungsmöglichkeiten von *Rezept 11.12*, allerdings müssen Sie für die Core Data-Version deutliche Änderungen vornehmen:

- **Das Hinzufügen und Löschen von Elementen findet nur in der Datenquelle statt.** Methoden, die einen Bearbeitungsvorgang bestätigen (die also z. B. eine Löschung durchführen) oder neue Zellen hinzufügen, greifen nicht direkt auf die Tabellenansicht zu. Im ursprünglichen Rezept haben alle Methoden nach dem Hinzufügen oder Entfernen die Daten in der Tabellenansicht neu geladen. In *Rezept 19.3* werden die Daten im verwalteten Kontext gespeichert, `reloadData` wird aber *nicht* aufgerufen.
- **Datenaktualisierungen lösen einen erneuten Ladevorgang der Tabelle aus.** Der `reloadData`-Aufruf wird ausgelöst, wenn die Delegierung für abgerufene Ergebnisse einen `controllerDidChangeContent:`-Callback empfängt. Diese Methode wird gesendet, wenn das Objekt der abgerufenen Ergebnisse erkennt, dass die gespeicherten Daten aktualisiert wurden. Dies geschieht, nachdem Datenänderungen im Kontext für verwaltete Objekte gespeichert wurden.
- **Eine Umsortierung der Tabelle ist nicht möglich.** Das Ergebnis der Methode `tableView:canMoveRowAtIndexPath:` ist in *Rezept 19.3* im Code als `NO` festgelegt. Bei abgerufenen Ergebnissen als Datenquellen können die Benutzer die Daten nicht umsortieren, was sich in dieser Methode widerspiegelt.

Diese Änderungen zusammen ermöglichen Hinzufügungen und Löschungen sowie Inhaltsänderungen in der Tabelle. Bearbeitungen des Inhalts werden in diesem Rezept nicht angesprochen, erfordern aber einen ähnlichen Ansatz mit einer Aktualisierung der abgerufenen Ergebnisse, wenn die Benutzer Attribute verändern, die in den Sortierungsdeskriptoren verwendet werden.

Der Code zum Hinzufügen und Löschen folgt dem Ansatz, der zu Anfang dieses Kapitels ausführlich vorgestellt wurde. Objekte werden über eine neue Entitätsbeschreibung hinzugefügt. Anschließend werden ihre Attribute festgelegt und der Kontext gespeichert. Auch beim Löschen von Objekten wird der Kontext gespeichert. Solche Änderungen lösen die entsprechenden Callbacks für die Delegierung der abgerufenen Ergebnisse aus.

Wie dieses Rezept zeigt, vereinfacht die Core Data-Integration das Zusammenspiel von Datenmodell und Benutzeroberfläche. Zum großen Teil liegt das an der vorausschauenden Gestaltung der Klassen durch Apple, wodurch die Verantwortung für die verwalteten Objekte auf diese Klassen fällt. *Rezept 19.3* führt diese Gestaltung und die aus der Verwendung von Core Data resultierende Sparsamkeit des Codes vor.

```
-(void)enterEditMode
{
   // Beginnt die Bearbeitung
   [self.tableView deselectRowAtIndexPath:
      [self.tableView indexPathForSelectedRow] animated:YES];
   [self.tableView setEditing:YES animated:YES];
   [self setBarButtonItems];
}

-(void)leaveEditMode
{
   // Beendet die Bearbeitung
   [self.tableView setEditing:NO animated:YES];
   [self setBarButtonItems];
}

- (BOOL)tableView:(UITableView *)tableView
   canMoveRowAtIndexPath:(NSIndexPath *)indexPath
{
   return NO; // Umsortierung nicht zulässig
}

- (void)tableView:(UITableView *)tableView
   commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
   forRowAtIndexPath:(NSIndexPath *)indexPath
{
   // Löschanforderung
   if (editingStyle == UITableViewCellEditingStyleDelete)
   {
      NSError *error = nil;
      [self.context deleteObject:[fetchedResultsController
         objectAtIndexPath:indexPath]];
      if (![self.context save:&error])
         NSLog(@"Error %@", [error localizedDescription]);
   }

   // Aktualisiert die Schaltflächen nach dem Löschvorgang
   [self setBarButtonItems];

   // Aktualisiert Abschnitte
   [self performFetch];
}

- (void) add
{
```

```
    // Fordert einen String als Aktionselement an
    NSString *todoAction =
        [ModalAlert ask:@"What Item?" withTextPrompt:@"To Do Item"];
    if (!todoAction || todoAction.length == 0) return;

    // Erstellt ein neues Element und legt dessen Aktionsfeld fest
    ToDoItem *item = (ToDoItem *)[NSEntityDescription
        insertNewObjectForEntityForName:@"ToDoItem"
        inManagedObjectContext:self.context];
    item.action = todoAction;
    item.sectionName =
        [[todoAction substringToIndex:1] uppercaseString];

    // Speichert das neue Element
    NSError *error;
    if (![self.context save:&error])
        NSLog(@"Error %@", [error localizedDescription]);

    // Aktualisiert die Schaltflächen nach einer Hinzufügung
    [self setBarButtonItems];

    // Aktualisiert die Abschnitte
    [self performFetch];
}

- (void)controllerDidChangeContent:
    (NSFetchedResultsController *)controller
{
    // Aktualisiert die Tabelle, wenn sich die Inhalte geändert haben
    [self.tableView reloadData];
}
```

► *Rezept 19.3: Bearbeitungsmöglichkeiten für Tabellen an Core Data anpassen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 19 und öffnen das Projekt zu diesem Rezept.

19.5 Rezept: Widerrufs- und Wiederherstellungsmöglichkeiten in Core Data-Tabellen hinzufügen

Core Data vereinfacht die Möglichkeiten für das Widerrufen und Wiederherstellen in Tabellen in einem erstaunlichen Umfang. Solche Operationen werden ohne großen Programmieraufwand automatisch bereitgestellt. Folgende Schritte sind dazu erforderlich:

1. Fügen Sie einen Undo-Manager zum Kontext für verwaltete Objekte hinzu. Nachdem Sie einen solchen Kontext eingerichtet haben (gewöhnlich in der Anwendungsdelegierung), setzen Sie den Undo-Manager auf eine neu zugewiesene Instanz.

```
self.context.undoManager =
   [[[NSUndoManager alloc] init] autorelease];
```

2. Weisen Sie den Undo-Manager im Ansichtscontroller zu.

```
self.context = [(TestBedAppDelegate *)
   [[UIApplication sharedApplication] delegate] context];
self.undoManager = self.context.undoManager;
```

3. Bieten Sie optional eine schüttelgesteuerte Bearbeitung an. Soll Ihre Anwendung auf Schüttelbewegungen dadurch reagieren, dass sie ein Menü zum Widerrufen und Wiederherstellen von Befehlen anzeigt, müssen Sie in die Anwendungsdelegierung die folgende Codezeile aufnehmen:

```
application.applicationSupportsShakeToEdit = YES;
```

4. Sorgen Sie dafür, dass der Ansichtscontroller der First Responder ist, wenn er auf dem Bildschirm erscheint. Dazu stellen Sie die folgenden Methoden bereit. Der Ansichtscontroller gibt seinen Status als First Responder auf, wenn er vom Bildschirm verschwindet.

```
- (BOOL)canBecomeFirstResponder {
   return YES;
}

- (void)viewDidAppear:(BOOL)animated {
   [super viewDidAppear:animated];
   [self becomeFirstResponder];
}

- (void)viewWillDisappear:(BOOL)animated {
   [super viewWillDisappear:animated];
   [self resignFirstResponder];
}
```

Diese Schritte sind alles, was Sie brauchen, um Widerrufsmöglichkeiten in einer Tabelle bereitzustellen. *Rezept 19.4* nimmt diese Möglichkeiten in die Lösch- und Hinzufügemethoden der Tabelle auf. Dazu wird der Core Data-Zugriff in eine Gruppierung für den Widerruf gestellt. Die Aufrufe `beginUndoGrouping`

und `endUndoGrouping` erfolgen jeweils, bevor und nachdem der Kontext aktualisiert wird und seine Änderungen gespeichert werden. Ein Aktionsname beschreibt dabei, welche Operation gerade vonstatten gegangen ist.

Mehr als diese drei Aufrufe (Beginn, Widerruf und Festlegen des Aktionsnamens) ist nicht erforderlich, damit Core Data seine Operationen rückgängig machen kann. Mit diesem minimalen Aufwand gewinnt Ihre Anwendung ein vollständiges Undo-Verwaltungssystem dank Core Data. Beachten Sie, dass die Undo/Redo-Daten wie bei der manuellen Bereitstellung von Widerrufsmöglichkeiten nach dem Beenden der Anwendung nicht fortbestehen.

```
- (void)tableView:(UITableView *)tableView
   commitEditingStyle:(UITableViewCellEditingStyle)editingStyle
   forRowAtIndexPath:(NSIndexPath *)indexPath
{
   [self.context.undoManager beginUndoGrouping];

   // Löschanforderung
   if (editingStyle == UITableViewCellEditingStyleDelete)
   {
      NSError *error = nil;
      [self.context deleteObject:
         [fetchedResultsController objectAtIndexPath:indexPath]];
      if (![self.context save:&error])
         NSLog(@"Error %@", [error localizedDescription]);
   }

   [self.context.undoManager endUndoGrouping];
   [self.context.undoManager setActionName:@"Delete"];

   // Aktualisiert die Schaltflächen nach einem Löschvorgang
   [self setBarButtonItems];

   // Aktualisiert die Abschnitte
   [self performFetch];
}

- (void) add
{
   // Fordert einen Stirng als Aktionselement an
   NSString *todoAction = [ModalAlert ask:@"What Item?"
      withTextPrompt:@"To Do Item"];
   if (!todoAction || todoAction.length == 0) return;

   [self.context.undoManager beginUndoGrouping];
```

```
    // Erstellt ein neues Element und legt dessen Aktionsfeld fest
    ToDoItem *item = (ToDoItem *)[NSEntityDescription
       insertNewObjectForEntityForName:@"ToDoItem"
       inManagedObjectContext:self.context];
    item.action = todoAction;

    // Die Indizierung erfolgt nach dem ersten Zeichen des Aktionsnamens
    item.sectionName =
       [[todoAction substringToIndex:1] uppercaseString];

    // Speichert das neue Element
    NSError *error;
    if (![self.context save:&error])
       NSLog(@"Error %@", [error localizedDescription]);

    [self.context.undoManager endUndoGrouping];
    [self.context.undoManager setActionName:@"Add"];

    // Aktualisiert die Schaltflächen nach einer Hinzufügung
    [self setBarButtonItems];

    // Aktualisiert die Abschnitte
    [self performFetch];
}
```

► *Rezept 19.4: Die Zellenverwaltung um Widerrufs- und Wiederherstellungsmöglichkeiten erweitern*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 19 und öffnen das Projekt zu diesem Rezept.

19.6 Zusammenfassung

Dieses Kapitel hat Ihnen nur einen kleinen Vorgeschmack auf die Möglichkeiten gegeben, die mit Core Data zur Verfügung stehen. In den Rezepten konnten Sie sehen, wie Sie einfache Core Data-Anwendungen gestalten und umsetzen und wie die Core Data-Merkmale auf das Modell der verwalteten Daten angewandt werden. Sie haben gelesen, wie Sie ein Modell definieren, wie Sie Abrufanforderungen implementieren und wie Sie Objekte hinzufügen, löschen, ändern und speichern. Darüber hinaus haben Sie Prädikate und Widerrufsmöglichkeiten kennengelernt und gesehen, wie Sie Core Data mit Tabellenansichten kombinieren. Nachdem Sie dieses Kapitel durchgearbeitet haben, sollten Sie die folgenden abschließenden Gedanken daraus mitnehmen:

- Bei nicht reziproken Beziehungen gibt Xcode eine Compilerwarnung aus. Solche Beziehungen machen weitere Arbeitsschritte erforderlich und hindern Sie daran, einfache Löschregeln wie `Nullify` zu nutzen. Vermeiden Sie nach Möglichkeit solche Beziehungen.
- Wenn Sie Daten aus einem Speicher vor Version 3.0 in eine neue SQLite-Datenbank übertragen, müssen Sie in den Benutzereinstellungen irgendeine Form von Flag setzen. Prüfen Sie, ob Sie bereits eine Datenaktualisierung vorgenommen haben oder nicht. Benutzerdaten sollten Sie bei der Aktualisierung einer Anwendung migrieren, aber nicht mehr später.
- Prädikate gehören zu meinen Lieblingsfunktionen des SDK 3.0. Nehmen Sie sich die Zeit, zu lernen, wie Sie sie aufstellen, und reservieren Sie sie nicht für Core Data, sondern verwenden Sie sie für alle möglichen Arten von Objekten, z. B. auch für Arrays und Mengen.
- Die Möglichkeiten von Core Data gehen weit über die grundlegenden Rezepte hinaus, die Sie in diesem Kapitel gesehen haben. Eine ausführliche Erläuterung zu Core Data finden Sie in dem bei Addison-Wesley erschienenen Buch *Core Data for iPhone* von Tim Isted.

20 StoreKit: Anwendungsinterner Produktverkauf

StoreKit ist neu im SDK 3.0 und bietet Möglichkeiten für anwendungsinterne Zusatzkäufe. Endbenutzer, die die Anwendung im App Store erworben und installiert haben, können dank StoreKit ihre iTunes-Anmeldeinformationen angeben, um Zusatzfunktionen, Abonnements oder Verbrauchsartikel zu kaufen. Dieses Kapitel gibt eine Einführung in StoreKit und zeigt, wie Sie die StoreKit-API verwenden, um Kaufoptionen für Ihre Benutzer bereitzustellen. Hier lesen, wie Sie mit der Nutzung von StoreKit beginnen, wie Sie in iTunes Connect neue Produkte einrichten und wie Sie die Beschreibungen lokalisieren. Außerdem lernen Sie, wie Sie Testbenutzer erstellen und wie Sie die verschiedenen Hürden bei der Entwicklung und Bereitstellung nehmen. Sie erfahren, wie Sie Kaufanforderungen von den Benutzern einholen und wie Sie diese Anforderungen zur Bezahlung an den App Store übergeben. Nachdem Sie dieses Kapitel durchgearbeitet haben, kennen Sie StoreKit von der Produkterstellung bis zum Verkauf.

20.1 Erste Schritte mit StoreKit

Wenn Sie mit Ihrer Anwendung ein Geschäftsmodell verfolgen, das sich nicht in »einmal kaufen, unbegrenzt verwenden« erschöpft, sollten Sie StoreKit verwenden. Damit haben Sie als Entwickler die Möglichkeit, Zusatzprodukte direkt aus Ihrer Anwendung heraus zu verkaufen, und können iTunes-Zahlungen als zusätzliche Quelle für Einkünfte nutzen. Es gibt viele Gründe, um StoreKit zu nutzen. Beispielsweise können Sie mit diesem neuen Framework Abonnements anbieten, bei Spielen zusätzliche Levels bereitstellen oder Sonderfunktionen entsperren.

Das heißt jedoch nicht, dass die Benutzer neuen Code herunterladen. Alle StoreKit-Anwendungen werden mit sämtlichen Funktionen ausgeliefert, allerdings können Sie den Benutzern über StoreKit Zugriff auf Teile der Anwendung gewähren, die zuvor gesperrt waren. Die Benutzer können auch

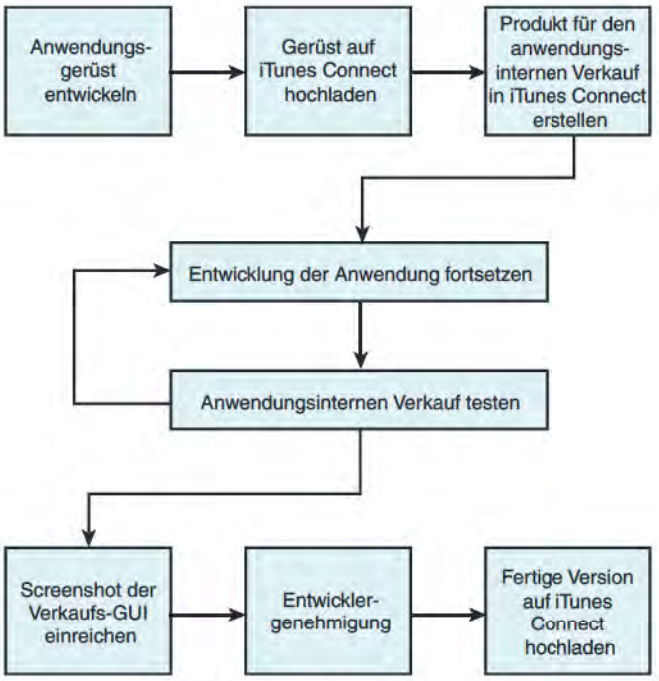
neue Daten herunterladen oder entsperren oder den Zugriff auf abonnierbare Webfeeds autorisieren. StoreKit bietet eine Möglichkeit, um die Benutzer für den Zugriff auf solche Funktionen zahlen zu lassen, die sie dann sofort nach dem Erwerb nutzen können.

Es ist zurzeit jedoch nicht möglich, über diese anwendungsinterne Verkaufsmöglichkeit physische Waren anzubieten (z. B. T-Shirts) oder eine Hilfswährung (z. B. als Kreditspeicher für eine Website). Auch echte Glücksspiele sind verboten. Alle Waren, die Sie über den anwendungsinternen Verkauf veräußern, müssen digital in Ihre Anwendung ausgeliefert werden.

Bei der Verwendung von StoreKit geben Sie an, welche Artikel Sie verkaufen möchten und wie teuer sie sein sollen. StoreKit und iTunes kümmern sich dann um die Einzelheiten und sorgen für die Infrastruktur, die die Einkaufsmöglichkeit über verschiedene API-Aufrufe und Delegierungscallbacks in Ihrer Anwendung anzeigt.

Leider stellt uns StoreKit vor ein Paradoxon, denn Sie können den anwendungsinternen Verkauf erst dann vollständig entwickeln und testen, wenn Sie Ihre Anwendung bereits bei iTunes eingereicht haben – aber gleichzeitig können Sie eine Anwendung erst dann richtig in iTunes einreichen, wenn Sie wissen, dass Sie mit der Entwicklung fertig sind. Was also sollen Sie tun? Wie läuft die Entwicklung für StoreKit ab?

Es gibt zum Glück eine Lösung, die Sie in *Abbildung 20.1* sehen. Um das StoreKit-Paradoxon zu umgehen, laden Sie ein zwar funktionierendes, aber noch nicht vollständiges Anwendungsgerüst auf iTunes Connect hoch. Dabei ist Ihnen jedoch klar, dass Sie diese Binärdatei irgendwann verwerfen und durch eine fertige ersetzen werden.



▸ *Abbildung 20.1: Entwicklung für StoreKit*

Sie müssen das Gerüst hochladen, da Sie eine Anwendung brauchen, für die bereits ein Genehmigungsverfahren läuft, bevor Sie mit der Entwicklung von StoreKit-Anwendungen und -Produkten beginnen können. Ohne eine solche »aktive« Anwendung können Sie keine neuen anwendungsinternen Produkte in iTunes Connect erstellen und diese Produkte auch nicht in der Sandbox-Version von StoreKit testen. Für die Zwecke von StoreKit brauchen Sie eine Anwendung, die entweder überprüft oder schon im App Store genehmigt ist.

Hinweis

Wenn Sie das Anwendungsgerüst zum Test einreichen, müssen Sie das Verfügbarkeitsdatum auf dem Titel **Pricing** von iTunes Connect vorstellen, damit Ihre noch nicht fertige Anwendung nicht versehentlich im App Store zum Verkauf angeboten wird, bevor Sie dazu bereit sind. Sobald Sie die Anwendung tatsächlich anbieten können, setzen Sie das Datum zurück.

Bis Oktober 2009 konnten Anwendungen mit StoreKit nicht kostenlos sein, sondern mussten mindestens mit einem Preis der Stufe 1 (0,79 €) ausgezeichnet sein. Inzwischen ist der anwendungsinterne Verkauf in StoreKit und iTunes nicht mehr auf kostenpflichtige Produkte beschränkt.

Nachdem Sie Ihre Anwendung eingereicht und mindestens ein Element für den anwendungsinternen Verkauf eingerichtet haben, können Sie das Programm und die Zusatzprodukte vollständig entwickeln und testen. In der Sandbox-Version von StoreKit können Sie neue Artikel mit Testbenutzer-Accounts »kaufen«, ohne dass dafür eine echte Kreditkarte belastet wird. In dieser Sandbox können Sie die Funktionen Ihrer Anwendung vor, während und nach dem Verkaufsvorgang testen.

Wenn Sie die Entwicklung abgeschlossen haben und eine endgültige Version im App Store einreichen können, beenden Sie die StoreKit-Entwicklung in iTunes Connect. Dazu müssen Sie einen Screenshot mit der grafischen Benutzeroberfläche für den anwendungsinternen Verkauf hochladen, ausdrücklich jedes Verkaufselement genehmigen, das Gerüst verwerfen und eine vollständige Version der Anwendung hochladen.

In den folgenden Abschnitten werden die einzelnen Schritte dieses Vorgangs ausführlicher vorgestellt, damit Sie StoreKit in Ihren Anwendungen nutzen können.

20.2 Test-Accounts erstellen

Test-Accounts spielen eine wichtige Rolle bei der StoreKit-Entwicklung. Bevor Sie damit beginnen, eine neue StoreKit-fähige Anwendung zu erstellen, sollten Sie mindestens einen Test-Account einrichten. Damit können Sie sich in iTunes anmelden und die Bezahlung Ihrer Anwendungen testen, ohne dass dabei wirklich Geld fließt.

Neue Benutzer fügen Sie auf folgende Weise hinzu. Melden Sie sich bei iTunes Connect an, und wählen Sie **Manage Users ▸ In App Purchase Tests User**. Klicken Sie auf **Add New User**. iTunes Connect zeigt daraufhin das Formular aus *Abbildung 20.2* an. Beachten Sie beim Ausfüllen folgende Punkte:

- Die E-Mail-Adressen müssen jeweils eindeutig, brauchen aber nicht echt zu sein. Solange sich die Adressen nicht mit irgendwelchen anderen im System überschneiden, ist alles in Ordnung. Allerdings haben schon andere Entwickler die einfach einzugebenden Adressen `abc.com`, `abcd.com` usw. mit Beschlag belegt.
- Die Namen und Geburtsdaten müssen nicht echt sein. Ich verwende ein alphabetisches Namenssystem, sodass meine Testbenutzer »a Sadun«, »b Sadun«, »c Sadun« usw. heißen. Alle sind am 1. Januar geboren.
- Die Passwörter müssen mindestens sechs Zeichen lang sein. Wenn Sie die Kennwörter wiederholt eingeben möchten, setzen Sie sie vollständig aus Kleinbuchstaben zusammen. Bei Großbuchstaben im Passwort müssen Sie dauernd die Schaltfläche für Großschreibung auf dem iPhone benutzen und bei Zahlen zwischen verschiedenen Tastaturen wechseln. Ein einziges, leicht zu merkendes Wegwerf-Passwort kann für alle Test-Accounts verwendet werden.
- In diesem Zusammenhang sind die Felder für die geheime Frage und die Antwort zwar sinnlos, können aber trotzdem nicht leer gelassen werden. Sie können nicht in beiden Feldern denselben String eingeben, und der Feldinhalt muss mindestens sechs Zeichen umfassen. Um das Erstellen der Accounts zu vereinfachen, können Sie auf einfache Frage/Antwort-Paare wie `aaaaaa` und `bbbbbb` zurückgreifen.
- Die Auswahl eines iTunes-Stores ist erforderlich, denn damit wird die Region für den Test festgelegt. Wenn Sie mehrere Sprachen für mehrere Stores verwenden möchten, müssen Sie in jeder betroffenen Region einen Test-Account einrichten.

► *Abbildung 20.2: Um in iTunes neue Benutzer hinzuzufügen, füllen Sie dieses Formular aus.*

- Benutzer-Accounts können Sie im laufenden Betrieb löschen und neu erstellen. Wenn Ihnen die Benutzer ausgehen, die noch keine Artikel gekauft haben, können Sie einfach nach Bedarf neue Benutzer anlegen.

> Sie dürfen sich in der Anwendung *Einstellungen* nicht an dem Test-Account anmelden, denn wenn Sie das versuchen, zwingt das iPhone Sie, der normalen Benutzervereinbarung zuzustimmen, und versucht eine gültige Kreditkartennummer von Ihnen in Erfahrung zu bringen. Über *Einstellungen* können Sie sich von einem Account abmelden, aber versuchen Sie nicht, sich damit anzumelden.

20.3 Neue anwendungsinterne Produkte erstellen

Jedes anwendungsinterne Produkt muss in iTunes Connect registriert sein. Melden Sie sich dazu an, und wechseln Sie zu **Manage Your In App Purchase**. Klicken Sie auf **Create New,** und wählen Sie eine Anwendung aus der angezeigten Liste. Sie zeigt alle Anwendungen an, die sich bereits im App Store oder noch im Genehmigungsverfahren befinden. Wählen Sie eine Anwendung aus, indem Sie auf ihr Symbol klicken.

Nachdem Sie eine Anwendung ausgewählt haben, fordert iTunes Connect Sie dazu auf, ein neues Produkt für den anwendungsinternen Verkauf zu erstellen, wie Sie in *Abbildung 20.3* sehen. Diese Abbildung zeigt die beiden obersten Abschnitte des entsprechenden Bildschirms (Preisgestaltung und Angaben zur Anzeige). Ein dritter Abschnitt zur Überprüfung befindet sich darunter und ist über einen Bildlauf zu erreichen.

▶ *Abbildung 20.3: Um in iTunes Conenct neue Elemente für den Verkauf zu erstellen, füllen Sie dieses Formular aus.*

20.3.1 Den Abschnitt zur Preisgestaltung ausfüllen

Im Abschnitt zur Preisgestaltung geben Sie an, wie das Produkt heißt und wie viel es kostet. Sie müssen einen Referenznamen und eine Produkt-ID angeben. Der Referenzname ist willkürlich und wird für die Suchergebnisse in iTunes Connect und im Abschnitt **Top In-App Purchase** für die Anwendung im App Store verwendet. Geben Sie also einen aussagekräftigen Namen an (z. B. »Freischaltung Level 3«), an dem Sie selbst und andere ablesen können, um was es sich bei dem Artikel handelt und wie er in Ihrer Anwendung eingesetzt wird.

Die Produkt-ID ist ein eindeutiger Bezeichner wie der Anwendungsbezeichner. In der Regel verwende ich dazu meinen Anwendungsbezeichner mit angehängtem Produktnamen, also z. B. `com.sadun.scanner.optionalDisclosure`. Diesen Bezeichner brauchen Sie, um den Store zu durchsuchen und Einzelheiten über das Produkt abzurufen. Für die Produkt-ID gelten dieselben Regeln wie für Anwendungsbezeichner. Sie können einen Bezeichner nicht mehrfach verwenden und auch nicht aus dem App Store entfernen. Wenn er einmal registriert ist, bleibt er registriert.

Als Nächstes wählen Sie einen Produkttyp aus, wobei die drei folgenden Möglichkeiten zur Verfügung stehen. Nachdem Sie den Typ ausgewählt und das neue Produktelement gespeichert haben, können Sie keine Änderungen mehr daran vornehmen. Der Typ ist unwiderruflich an die Produkt-ID gebunden. Wenn Sie einen Fehler machen, müssen Sie ein neues Element mit einer neuen Produkt-ID erstellen.

- **Non-consumable** Benutzer kaufen ein solches Produkt nur einmal und können es anschließend kostenlos so oft erneut herunterladen, wie sie möchten. Diese Option verwenden Sie für Zusatzfunktionen, die die Benutzer entsperren können, z. B. für Extralevels in Spielen.
- **Subscription** Benutzer verwenden solche Elemente immer wieder während des Lebenszyklus der Anwendung. Zwar können sie überprüfen, ob sie diese Elemente schon in ihrem Account erworben haben, sie können sie aber nicht ohne Bezahlung erneut herunterladen. Solche Abonnements verwenden Sie, um gebührenpflichtigen Zugriff auf Daten für einen bestimmten Zeitraum zu geben, z. B. für Zeitungsartikel oder Suchvorgänge in medizinischen Datenbanken.
- **Consumable** Verbrauchsartikel ähneln Abonnements darin, dass jeder Download bezahlt werden muss, werden aber auf andere Weise verwendet. Es handelt sich hierbei um Elemente, die mehrmals gekauft werden können, z. B. zusätzliche Trefferpunkte oder erweiterte CPU-Zeit auf einem Hauptserver. Verbrauchsartikel können aufgebraucht werden, ohne dass dafür wie bei Abonnements ein fester Zeitraum vorgegeben ist.

Lassen Sie das letzte Steuerelement im Abschnitt **Pricing** (**Cleared for Sale**) aktiviert. Dieses Markierungsfeld sorgt dafür, dass Ihre Anwendungen (sowohl in Entwicklung als auch in Bereitstellung) aus dem Programm heraus Zugriff auf das Produkt haben.

Hinweis

Sie können die Preisstufe und das Markierungsfeld **Cleared for Sale** während des Genehmigungsverfahrens jederzeit ändern. Sobald die Produkte genehmigt sind, müssen Sie Änderungen daran zur Überprüfung einreichen. Es ist nicht möglich, den Bezeichner zu ändern, einen vorhandenen Bezeichner zu verwenden oder den Typ des Produkts zu ändern, nachdem Sie es erstellt haben.

20.3.2 Produktangaben hinzufügen

Jedes Produkt muss sich Ihrer Anwendung gegenüber selbst beschreiben können. Dazu muss es seinen Preis angeben, den Sie im Abschnitt **Pricing** festlegen, und sowohl einen Anzeigenamen als auch eine Beschreibung (eine Beschreibung, die dem Benutzer sagt, was das Produkt erledigt). Die beiden letzten Elemente können in verschiedene Sprachen übersetzt werden, worunter zurzeit die folgenden fallen:

- Englisch (auch australisches, kanadisches und britisches Englisch)
- Niederländisch
- Französisch (auch kanadisches Französisch)
- Deutsch
- Italienisch
- Japanisch
- Spanisch (auch mexikanisches Spanisch)
- vereinfachtes Chinesisch

Sie müssen Daten für mindestens eine dieser Sprachen bereitstellen, können dies aber auf eine beliebige Anzahl der Sprachen ausdehnen. Es ist nicht möglich, ein neues Produktelement zu erstellen, ohne mindestens ein Paar aus Name und Beschreibung anzugeben. Wenn Sie nur den deutschsprachigen Markt im Auge haben, sollte ein einziger deutscher Eintrag für Sie genügen.

Wollen Sie Ihre Anwendung allerdings weltweit verkaufen, sollten sich die vorhandenen Sprachversionen in der Beschreibung widerspiegeln. Liegen z. B. Ihre Anwendung und Ihr iTunes-Marketingmaterial auch in einer japanischen Version vor, sollen Sie auch eine ins Japanische übersetzte Beschreibung des anwendungsinternen Produkts erstellen. Wenn Sie das nicht tun, können Sie das Produkt zwar immer noch für den anwendungsinternen Verkauf verwenden, allerdings wird dabei nicht die entsprechende lokalisierte Sprachversion angezeigt.

Hinweis

Lassen Sie den Text immer von Muttersprachlern übersetzen, bearbeiten und korrigieren.

Beachten Sie beim Eingeben der Daten die folgenden Punkte. Ihre Anwendung ist der Verbraucher für diese Informationen. Der Text, den Sie in iTunes Connect eingeben, wird für die Verkaufsoberfläche herangezogen, die die Benutzer in Ihrer Anwendung sehen. Die Lokalisierung wählt der Benutzer über seine Spracheinstellungen aus. Wenn Sie ein einfaches Benachrichtigungs-Sheet mit der Auswahl zwischen Kaufen und Abbrechen verwenden möchten, sollten Sie Ihre Formulierungen kurz und knapp halten und nicht geschwätzig werden. Beachten Sie dies auch, wenn Sie eine anspruchsvollere Ansicht anbieten.

Unabhängig davon, wie Sie die Oberfläche für den Verkaufsvorgang gestalten, müssen in Ihrem Text sowohl der Kaufvorgang als auch die Verwendung des Produkts zum Ausdruck kommen – z. B. »Wenn Sie diese Option kaufen, wird der Detailbildschirm der Anwendung entsperrt. Darauf erhalten Sie noch mehr Daten über den ermittelten MDNS-Dienst.« Eine kürzere Beschreibung wie »Zusätzlicher Detailbildschirm« oder »Detailansicht entsperren« sagt dem Benutzer nicht, wie der Kaufvorgang abläuft und was für ein Ergebnis er erwarten darf.

Hinweis

Während des Genehmigungsverfahrens in iTunes Connect können Sie Angaben zum Produkt jederzeit ändern. Sobald das Produkt genehmigt ist, müssen Sie Änderungen wieder zur Begutachtung einreichen.

20.3.3 Einen Screenshot der Verkaufsoberfläche einreichen

Unten in der Produktseite erscheint der Abschnitt **For Review**, den Sie aber erst verwenden, wenn Sie mit der Entwicklung und dem Debugging Ihrer Anwendung fertig sind. Dann laden Sie in das vorgesehene Feld einen Screenshot, der den anwendungsinternen Verkaufsvorgang in Aktion zeigt und die von Ihnen für diesen Zweck erstellte Benutzeroberfläche darstellt.

Abbildung 20.4 zeigt, wie ein solcher Screenshot auszusehen hat. Die Bilder müssen einen Größe von 320 × 480, 480 × 320, 320 × 460 oder 480 × 300 Pixel aufweisen. (Die beiden letzten Größen gelten für Screenshots, in denen die 20 Pixel breite Statusleiste entfernt worden ist.) Der Screenshot zeigt, wie Sie die Verkaufsfunktion gestaltet haben.

20.3.4 Entwicklergenehmigung

Nachdem Sie die Tests in der Sandbox erledigt haben und zu dem Schluss gekommen sind, dass die Anwendung und der anwendungsinterne Verkauf für die Begutachtung durch Apple bereit sind, müssen Sie die Anwendung persönlich genehmigen. Wechseln Sie zu **iTunes Connect ▸ Manage In-App Purchases**, und wählen Sie ein Produktelement aus. Klicken Sie dann auf die grüne Schaltfläche **Approve**.

▸ *Abbildung 20.4: Wenn Sie dazu bereit sind, das Produkt begutachten zu lassen, müssen Sie bei Apple einen Screenshot einreichen, der die Oberfläche für den anwendungsinternen Verkauf zeigt.*

Wie *Abbildung 20.5* zeigt, werden Sie dazu aufgefordert, anzugeben, wie Sie das Produkt einreichen möchten. Wenn Sie **Submit With Binary** aktivieren, reichen Sie das Produktelement mit dem nächsten Upload der Binärdatei ein, bei **Submit Now** liefern Sie es mit einer bereits genehmigten Anwendung (Version 3.x oder höher) zur Begutachtung aus. Die erste Option ist für Anwendungen da, für die gerade erst ein anwendungsinternes Produktelement hinzugefügt wurde, während die zweite dazu dient, neue Produkte zu einer vorhandenen, getesteten Anwendung hinzuzufügen.

▸ *Abbildung 20.5: Wählen Sie, wie Sie das Produkt bei Apple einreichen möchten.*

20.4 Die Anwendung einreichen

Nachdem Sie Ihre Anwendung bestätigt haben, können Sie sie in die Warteschlange zur Begutachtung einreihen. Wenn Sie die erste Option gewählt haben, müssen Sie als Nächstes eine neue Kopie der Binärdatei einreichen, da ansonsten die Anwendung und das Produkt für den anwendungsinternen Verkauf nicht zusammen begutachtet werden.

▸ *Abbildung 20.6: Wählen Sie das Zusatzprodukt aus, das zusammen mit der Anwendung begutachtet werden soll, wenn Sie die neue Binärdatei als Ersatz für die von Ihnen selbst verworfene einreichen.*

Um die neue Binärdatei übergeben zu können, müssen Sie die bisherige Version verwerfen. Wenn Sie das tun, können Sie Ihr Programm aber nicht mehr im Sandbox-Server für den anwendungsinternen Verkauf testen, denn um diesen Dienst zu nutzen, brauchen Sie eine Anwendung, die noch begutachtet wird oder bereits akzeptiert ist. Laden Sie daher die vollständige Version hoch.

Beim erneuten Hochladen der Binärdatei fordert iTunes Connect Sie dazu auf, die Produkte für den anwendungsinternen Verkauf festzulegen, wie Sie in *Abbildung 20.6* sehen. Aktivieren Sie die Produkte, die Sie verwenden möchten, und speichern Sie die Änderungen. Die Zusatzprodukte und die Anwendung werden zusammen begutachtet, um das »Henne-oder-Ei-Problem« zu lösen.

20.5 Die grafische Benutzeroberfläche gestalten

Im Framework StoreKit von Apple gibt es keine vorgefertigte grafische Oberfläche, mit denen Sie die Benutzer zum Kauf von Zusatzprodukten auffordern können, sodass Sie eine eigene wie die aus *Abbildung 20.4* erstellen müssen. Um die lokalisierten Preisangaben und Beschreibungen aus dem App Store abzurufen, erstellen Sie Instanzen von `SKProductsRequest`. Diese Klasse fragt den Store anhand von Ihnen angegebener Bezeichner nach Informationen über die zugehörigen Artikel, wobei die Bezeichner in iTunes Connect für Produktelemente zum anwendungsinternen Verkauf registriert sein müssen.

Weisen Sie eine neue Instanz für Produktabfragen zu, und initialisieren Sie sie mit den Bezeichnern. Dabei können Sie sowohl Bezeichner für Produkte verwenden, die bereits eingerichtet sind, als auch für solche, die Sie in Zukunft hinzufügen möchten. Da Bezeichner lediglich Strings sind, können Sie eine Schleife programmieren, die Bezeichner nach einem Namensschema zusammenbaut (z. B. `com.sadun.app.item1`, `com.sadun.app.item2` usw.), um für zukünftiges Wachstum gewappnet zu sein. Das folgende Codefragment sucht nach einem einzelnen Element:

```
// Erstellt die Produktabfrage und startet sie
SKProductsRequest *preq = [[SKProductsRequest alloc]
   initWithProductIdentifiers:[NSSet setWithObject:PRODUCT_ID]];
preq.delegate = self;
[preq start];
```

Bei der Verwendung einer Produktabfrage muss Ihre Delegierung das Protokoll `SKProductsRequestDelegate` deklarieren und implementieren, das aus drei einfachen Callbacks besteht. *Listing 20.1* zeigt diese Callback-Methoden für eine einfache Anwendung. Wenn eine Antwort eingeht, sucht dieser Code nach einem Produkt (falls ein solches mithilfe des vorstehenden Codefragments abgefragt wurde) und gibt die lokalisierte Preisangabe und die Beschreibung zurück.

Anschließend erstellt der Code eine einfache Benachrichtigung mit der Beschreibung als Text sowie zwei Schaltflächen (eine mit dem Preis und eine mit **No Thanks**). Diese Benachrichtigung dient als einfache Benutzeroberfläche für den Verkauf.

Hinweis

StoreKit funktioniert nicht, wenn Sie nicht auf irgendeine Weise mit dem Internet verbunden sind. In Kapitel 14, *Gerätefähigkeiten*, finden Sie Rezepte, um den Netzwerkzugriff zu prüfen.

```
- (void)request:(SKRequest *)request
   didFailWithError:(NSError *)error
{
   [self doLog:
      @"Error: Could not contact App Store properly, %@",
      [error localizedDescription]];
}

- (void)requestDidFinish:(SKRequest *)request
{
   // Gibt die Abfrage frei
   [request release];
   [self doLog:@"Request finished."];
}

- (void)productsRequest:(SKProductsRequest *)request
   didReceiveResponse:(SKProductsResponse *)response
{
   // Findet ein Produkt
   SKProduct *product = [[response products] lastObject];
   if (!product)
   {
      [self doLog:@"Error Could not find matching products"];
      return;
   }

   // Ruft den lokalisierten Preis ab
   NSNumberFormatter *numberFormatter =
      [[NSNumberFormatter alloc] init];
   [numberFormatter
      setFormatterBehavior:NSNumberFormatterBehavior10_4];
   [numberFormatter setNumberStyle:NSNumberFormatterCurrencyStyle];
   [numberFormatter setLocale:product.priceLocale];
   NSString *formattedString = [numberFormatter
      stringFromNumber:product.price];
   [numberFormatter release];

   // Zeigt die Informationen an
   [self doLog:product.localizedTitle];
   [self doLog:product.localizedDescription];
   [self doLog:@"Price %@", formattedString];
```

```
    // Erstellt die Benutzeroberfläche
    NSArray *buttons = [NSArray arrayWithObject: formattedString];
    if ([ModalAlert ask:describeString withCancel:@"No Thanks"
       withButtons:buttons])
    {
       // Führt den Verkaufsvorgang durch
    }
}
```

▸ *Listing 20.1: Callback-Methoden für die Produktabfrage*

20.6 Zusatzprodukte verkaufen

Um Produkte aus Ihrer Anwendung heraus verkaufen zu können, müssen Sie als Erstes einen Transaktionsbeobachter hinzufügen, was Sie am besten in der `finishedLaunching`-Methode der Anwendungsdelegierung erledigen. Verwenden Sie Ihre Hauptmodellklasse als Beobachter, und sorgen Sie dafür, dass diese Klasse das Protokoll `SKPaymentTransactionOberserver` deklariert und implementiert.

```
[[SKPaymentQueue defaultQueue] addTransactionObserver:mainClass];
```

Wenn Sie den Beobachter eingerichtet haben, können Sie unter Verwendung der Oberfläche aus *Listing 20.1* mit dem eigentlichen Verkaufsvorgang beginnen.

```
if ([ModalAlert ask:describeString
   withCancel:@"No Thanks" withButtons:buttons])
{
   // Kauft das Produkt
   SKPayment *payment = [SKPayment
      paymentWithProductIdentifier:PRODUCT_ID];
   [[SKPaymentQueue defaultQueue] addPayment:payment];
}
else
{
   // Stellt die Benutzeroberfläche wieder her, um eine Schaltfläche für den
   // Kauf anzuzeigen, oder führt auf andere Weise zu einem kaufbereiten
   // Zustand zurück
}
```

StoreKit fordert den Benutzer auf, den anwendungsinternen Erwerb zu bestätigen, wie Sie in *Abbidlung 20.7* sehen, und übernimmt dann den Verkaufsvorgang. Möglicherweise müssen sich die Benutzer an einem Account anmelden, bevor sie fortfahren können.

▶ *Abbildung 20.7: Die Benutzer müssen den Kauf bestätigen, nachdem sie Ihre Benutzeroberfläche verlassen haben und in das eigentliche App Store/StoreKit-Verkaufssystem gelangt sind.*

20.6.1 Vor dem Test vom iTunes-Account abmelden

Bevor Sie die in iTunes Connect eingerichteten Test-Accounts verwenden können, müssen Sie sich von Ihrem jetzigen, echten Account abmelden. Starten Sie die Anwendung *Einstellungen*, wählen Sie **Store**, und klicken Sie auf **Abmelden**.

Wie in diesem Kapitel bereits erwähnt wurde, dürfen Sie sich *nicht* wieder mit den Angaben Ihres Test-Accounts anmelden. Beenden Sie einfach das Programm *Einstellungen*, und kehren Sie zu Ihrer Anwendung zurück. Nachdem Sie auf **Buy** geklickt haben, werden Sie dazu aufgefordert, sich bei iTunes anzumelden. Wählen Sie an dieser Stelle **Use Existing Account**, und geben Sie Ihre Account-Informationen an.

Hinweis

StoreKit können Sie nicht im Simulator testen, sondern nur auf einem physischen iPhone, iPod touch oder iPad.

20.6.2 Die Steuerung nach dem Verkaufsvorgang zurückgewinnen

Der Transaktionsbeobachter empfängt Callbacks über den Erfolg oder Fehlschlag des Zahlvorgangs. *Listing 20.2* zeigt ein Grundgerüst für die Reaktion auf abgeschlossene und abgebrochene Zahlungen. Wenn der Benutzer den Kaufvorgang abschließt, ist die Transaktion entweder erfolgreich verlaufen oder fehlgeschlagen. Beim Erfolg führen Sie die Aktion aus, für die der Benutzer bezahlt hat, indem Sie z. B. Daten herunterladen oder Funktionen entsperren.

```
- (void)paymentQueue:(SKPaymentQueue *)queue
   removedTransactions:(NSArray *)transactions
{
}

- (void) completedPurchaseTransaction:
   (SKPaymentTransaction *) transaction
{
   // Führen Sie hier die Aktion durch, die die gekaufte Funktion
   // entsperrt

   // Schließt die Transaktion ab
   [[SKPaymentQueue defaultQueue] finishTransaction: transaction];
   [ModalAlert say:@"Thank you for your purchase."];
}

- (void) handleFailedTransaction: (SKPaymentTransaction *) transaction
{
   if (transaction.error.code != SKErrorPaymentCancelled)
      [ModalAlert say:@"Transaction Error. Please try again later."];
   [[SKPaymentQueue defaultQueue] finishTransaction: transaction];
}

- (void)paymentQueue:(SKPaymentQueue *)queue
   updatedTransactions:(NSArray *)transactions
{
   for (SKPaymentTransaction *transaction in transactions) {
      switch (transaction.transactionState) {
         case SKPaymentTransactionStatePurchased:
         case SKPaymentTransactionStateRestored:
            [self completedPurchaseTransaction:transaction];
            break;
         case SKPaymentTransactionStateFailed:
            [self handleFailedTransaction:transaction];
            break;
         case SKPaymentTransactionStatePurchasing:
            [self repurchase];
            break;
         default: break;
      }
   }
}
```

► *Listing 20.2: Auf Zahlungen reagieren*

20.6.3 Kaufvorgänge registrieren

Es gibt verschiedene Möglichkeiten, um Kaufvorgänge zu registrieren. Sie können eine Synchronisierung mit einem Webserver durchführen, lokale Dateien anlegen, Benutzervoreinstellungen festlegen oder Schlüsselbundeinträge hinzufügen. Welche Lösung Sie wählen, bleibt Ihnen überlassen, Sie dürfen nur nicht zulassen, dass die Information über den Kaufvorgang verloren geht. Wenn ein Benutzer eine freizuschaltende Funktion, ein Abonnement oder Daten kauft, muss Ihre Anwendung die versprochenen Elemente auch bereitstellen.

Am einfachsten ist es, Funktionen mithilfe von Benutzervoreinstellungen freizuschalten. Der folgende Code erstellt eine neue Einstellung, die angibt, dass der Benutzer ein Sondermerkmal erworben hat. Nach Abschluss des Kaufvorgangs aktualisiert der Code die Datenbank der Benutzereinstellungen und blendet die Schaltfläche für den Kauf in der Benutzeroberfläche aus.

```
// Aktualisiert die Benutzervoreinstellungen
[[NSUserDefaults standardUserDefaults] setBool:YES
   forKey:@"Supports Disclosure"];
[[NSUserDefaults standardUserDefaults] synchronize];

// Blendet die Kauf-Schaltfläche aus
self.navigationItem.leftBarButtonItem = nil;
```

Die Anwendung kann dann bei jedem Start nach dieser Voreinstellung suchen.

Normalerweise haben die Benutzer keine Möglichkeit, sich in Ihre Anwendung einzuhacken, um die Voreinstellungen manuell zu ändern. Die Anwendung befindet sich in einer Sandbox (andere Anwendungen haben keinen Zugriff auf ihre Dateien), und die Daten können auch nicht auf einem Macintosh-System bearbeitet werden. In einem geknackten System ist das jedoch möglich, wenn Sie wie hier eine einfache Voreinstellung verwenden. Wenn Sie sich Sorgen über Softwarepiraterie machen, können Sie einen sichereren Ansatz verwenden.

Falls Sie irgendwelche Bedenken haben, sollten Sie irgendeine Art von überprüfbarem Authentifizierungsschlüssel verwenden statt des normalen booleschen Wertes. Alternativ können Sie auch den Systemschlüsselbund einsetzen (siehe Kapitel 13, *Netzwerke*), einen sicheren Datenspeicher, der sich auch von einer geknackten iPhone-Kommandozeile aus nicht so einfach manipulieren lässt.

Die folgende Routine zeigt ein einfaches Beispiel für die Speicherung des Kaufvorgangs im Schlüsselbund:

```
-(void) unlockMaxGameLevels
{
   KeychainItemWrapper *wrapper = [[KeychainItemWrapper alloc]
      initWithIdentifier:@"CustomGameApp" accessGroup:nil];
   [wrapper setObject:@"MaxUnlocked" forKey:(id)kSecValueData];
   [wrapper release];
}
```

Die Verwendung des Schlüsselbundes bietet den zusätzlichen Vorteil, dass die darin gespeicherten Daten auch nach einer Löschung und Neuinstallation der Anwendung noch vorhanden sind.

Wenn Sie einen externen Server verwenden, um Kaufvorgänge zu registrieren und zu authentifizieren, müssen Sie dafür sorgen, dass dessen Einstellungen auch auf dem Gerät widergespiegelt werden. Die Benutzer müssen in der Lage sein, ihre Programme unabhängig vom Netzwerkzugriff zu verwenden. Eine lokale Einstellung (z. B. »Dienst aktiviert bis zum 6. Juni 2011«) sorgt dafür, dass die Anwendungen auch dann laufen kann und korrekt reagiert, wenn der abonnierte Dienst nicht erreicht werden kann.

Verschiedene junge Unternehmen wie *Urban Airship* (*urbanairship.com*) und der Dienst *iLime* von *Key Lime Tie* (*ilime.com*) bieten jetzt die Datenauslieferung für den anwendungsinternen Verkauf an. Dazu stellen sie Server bereit, auf die Sie Inhalte aus Ihren Anwendungen auslagern können, sorgen für die Übermittlung an die Kunden und helfen Ihnen, diesen Inhalt nach Bedarf auf dem neuesten Stand zu halten.

20.6.4 Einkäufe gebührenfrei wiederholen

Die gebührenfreie Wiederholung von Einkäufen kann notwendig sein, wenn eine Anwendung auf einem Gerät deinstalliert und dann wieder neu installiert wurde oder wenn die Anwendung auf einem zweiten Gerät eingerichtet wird, das zum selben iTunes-Account gehört. Sind im iTunes-Account eines Kunden mehrere Geräte vermerkt, z. B. bei einer Familie mit fünf iPhones und iPods, wird bei einem Kaufvorgang auf einem dieser Geräte auch allen anderen erlaubt, das Produkt ohne weitere Zahlung herunterzuladen.

StoreKit ermöglicht die kostenlose Wiederholung von Einkäufen, was vor allem für Verbrauchsartikel und Abonnements wichtig ist, bei denen Sie nicht wollen, dass die Kunden ein bereits gültiges Produkt erneut erwerben. Artikel, die nicht verbraucht werden, kann der Benutzer kostenlos unendlich oft erneut kaufen. Für solche Artikel übergeben Sie einfach eine Kaufanforderung. Die App Store-Schnittstelle zeigt ein Fenster an, in dem der Benutzer darüber informiert wird, dass er das Produkt bereits gekauft hat und es kostenlos erneut herunterladen kann.

Um Einkäufe über einen iTunes-Account zu wiederholen, rufen Sie `restoreCompletedTransactions` auf. Dies wirkt sich wie eine weitere Zahlung aus und hat dieselben Callbacks zur Folge. Um einen wiederholten Einkauf getrennt von einem Ersteinkauf zu erfassen, suchen Sie nach `SKPaymentTransactonStateRestored` als Transaktionsstatus, wie es in *Listing 20.2* geschieht.

```
- (void) repurchase
{
    // Erwirbt ein bereits gekauftes Produkt erneut
    [[SKPaymentQueue defaultQueue] restoreCompletedTransactions];
}
```

Der Grund dafür ist, dass es bei Kaufereignissen nicht nur ein, sondern zwei mögliche erfolgreiche Ergebnisse gibt. Das erste Ereignis ist der abgeschlossene Verkauf, bei dem der Benutzer den Artikel erworben hat und der Zahlungsprozess abgeschlossen ist, das zweite der hier beschriebene wiederholte Kauf. Sorgen Sie dafür, dass der Handler für die Zahlungswarteschlange beide Zustände berücksichtigt.

Es gibt dabei jedoch ein Problem. Stellen Sie sich z. B. einen Verbrauchsartikel wie einen Kredit zum Senden von Faxnachrichten vor. Wenn der Benutzer die Anwendung deinstalliert und dann erneut einrichtet, wird durch einen kostenlosen wiederholten Einkauf der vollständige Kredit wiederhergestellt, auch wenn er in Wirklichkeit schon genutzt worden ist. Anwendungen mit Verbrauchsartikeln müssen mit besonderem Augenmerk auf die Sicherheitsinfrastruktur entworfen werden und brauchen eine serverseitige Buchführung, die die Benutzerkredite und den Verbrauch der Artikel nachverfolgt.

Bieten Sie ruhig Möglichkeiten für eine kostenlose Wiederholung von Einkäufen an, aber stimmen Sie diese Vorgänge mit Ihrer Serverdatenbank ab. Wie Sie in dem folgenden Abschnitt lesen werden, markiert Apple jeden Verkaufsvorgang über eine Bestätigung mit einem eindeutigen Bezeichner. Ein erneut erworbener Artikel behält den ursprünglichen Bezeichner, sodass Sie zwischen neuen und wiederholten Einkäufen unterscheiden können.

20.6.5 Mehrere Artikel kaufen

Benutzer können mehr als eine Kopie von Verbrauchsartikeln und Abonnements kaufen. Um Mehrfachkäufe zu ermöglichen, setzen Sie die Eigenschaft `quantity` für die Bezahlung. Der folgende Code erstellt eine Zahlungsanforderung für drei Kopien eines Produkts, z. B. für drei Monate eines Abonnements, 1000 Siegpunkte für eine Spielfigur o. Ä.

```
SKMutablePayment *payment = [SKMutablePayment
   paymentWithProductIdentifier:PRODUCT_ID];
payment.quantity = 3;
[[SKPaymentQueue defaultQueue] addPayment:payment];
```

20.6.6 Verzögerungen beim Registrieren von Kaufvorgängen

Wenn bei einem Kaufvorgang eine Verbindung zu einem Server erforderlich ist und die Registrierung nicht beendet werden kann, dürfen Sie die Transaktion nicht abschließen! Rufen Sie `finishTransaction:` erst dann auf, wenn alle Einrichtungsarbeiten für Ihren Kunden garantiert erledigt sind.

Wenn der Benutzer die Anwendung beendet, bevor Sie die neu erworbenen Produkte einrichten konnten, ist das jedoch kein Problem, da die Transaktion bis zum nächsten Start des Programms in der Warteschlange verbleibt. Dann erhalten Sie eine weitere Gelegenheit, die Einrichtung abzuschließen.

20.7 Kaufbestätigungen überprüfen

Bei einer erfolgreichen Kauftransaktion gibt es eine Bestätigung, die im `NSData`-Rohformat gesendet wird und einem kodierten JSON-String entspricht. Sie enthält eine Signatur sowie Informationen über den Kauf. Das folgende Beispiel zeigt eine solche Bestätigung aus einem meiner eigenen Verkäufe:

```
{
   "signature" =
"AbtAgJQlIPicxP/g4ubwT/noCER4jE+LuGNfxfy++DsiEUrdOYNcf6GqljT+/qDlLCv-
SZUnWGG7YrACLDFQRREftjNDmkgekbErdP8uI9IANOsH6vkHx5sc/2p9hHRbG6AY/
CDDj11g+esLRe8HYGxCBaHlIMa+o/ZKtHr3Rl+jUMIIDUzCCAjugAwIBAgIIZRSRTdlYBLU-
```

```
wDQYJKoZIhvcNAQEFBQAwfzELMAkGA1UEBhMCVVMxEzARBgNVBAoMCkFwcGxlIEluYy4xJjA
kBgNVBAsMHUFwcGxlIENlcnRpZmljYXRpb24gQXVOaG9yaXR5MTMwMQYDVQQDDCpBcHBsZS-
BpVHVuZXMgU3RvcmUgQ2VydGlmaWNhdGlvbiBBdXRob3JpdHkwHhcNMDkwNjE1MjIwNTU-
2WhcNMTQwNjE0MjIwNTU2WjBkMSMwIQYDVQQDDBpQdXJjaGFzZVJlY2VpcHRDZXJOaWZ-
pY2FOZTEbMBkGA1UECwwSQXBwbGUgaVR1bmVzIFNOb3JlMRMwEQYDVQQKDApBcHBsZSBJbm-
MuMQswCQYDVQQGEwJVUzCBnzANBgkqhkiG9w0BAQEFAAOBjQAwgYkCgYEAytGMXZy3gitJ2J
MKFojSDynC/9yYezyn9HBX+u3/3VcpWE2XhcgGKYqNBA1+AewOzrKO774OsokTu4qymEx1O
ph8UTmsZewBOESMHBEjF7FN6/HccsQUYC3WagrHnT12HG2IhOOAm/ZhpWzjOHS4m813LpI-
yoOOsewMvMNL2hkcCAwEAAaNyMHAwDAYDVR0TAQH/BAIwADAfBgNVHSMEGDAWgBQ2Hejin-
YLSARi1MmsO1OMLkVhDOjAOBgNVHQ8BAf8EBAMCB4AwHQYDVR0OBBYEFKmDg/IZSMU+ElcI
FMzNo36ZXyT1MBAGCiqGSIb3Y2QGBQEEAgUAMAOGCSqGSIb3DQEBBQUAA4IBAQARpJs+O2Y
3gL8gHdASkrfZHFpwINd1VcB5VF5LkVpnFz63zylA/3cGIDG91b/d5NIwZjkVt4Bgvd62o/
mCbzCsWiNfSKTJVFK1D78BDQoSO2oHTuQuz1BR7xzNHxQZ9OzUS6ZX9SC8N3g3A1jEtAyDh
ZNB+CRBBXLwZdnBUeBsT9QLpjvTnekZcGTnUO8zfCjGF3eBJEu9eP6WgexK1xMSp72kEO-
mYbn6yTi3D4YrcYx4Q3n/57VBP2en8qXWeP5oHDsLTGzLRsWdoB3VxJLrF2ivL8JS8zqCOqy
ac452pN6xunRuzyyfpaqzQL12BzFEe44xna2byektSbtquA5LNAAAAAA==";
   "purchase-info" =
"ewoJIml0ZW0taWQiIDOgIjMzMDI5MjgwNiI7Cgkib3JpZ2luYWwtdHJhbnNhY3Rpb24ta-
WQiIDOgIjEwMDAwMDAwMDAwNTIyOTMiOwoJInB1cmNoYXNlLWRhdGUiIDOgIjIwMDktMDkt-
MDQgMTU6MzU6MjYgRXRjLOdNVCI7CgkicHJvZHVjdC1pZCIgPSAiY29tLnNhZHVuLnN-
jYW5uZXIuZGlzY2xvc3VyZTIiOwoJInRyYW5zYWN0aW9uLWlkIiA9ICIxMDAwMDAwMDAwM-
DUyMjkzIjsKCSJxdWFudGl0eSIgPSAiMSI7Cgkib3JpZ2luYWwtcHVyY2hhc2UtZGFOZSIg-
PSAiMjAwOS0wOS0wNCAxNTozNToyNiBFdGMvRO1UIjsKCSJiaWQiIDOgImNvbS5zYWR1bi-
5TY2FubmVyIjsKCSJidnJzIiA9ICIxLjAiOwp9";
   "pod" = "100";
   "signing-status" = "0";
}
```

Apple empfiehlt eindringlich, alle Bestätigungen auf den Servern zu überprüfen, um Hackerangriffe zu verhindern und sicherzustellen, dass die Kunden die angeforderten Artikel auch tatsächlich gekauft haben. Wie das geht, zeigt *Listing 20.3*.

Sie müssen mit `POST` eine Anforderung an einen der beiden Server von Apple senden. Welchen URL Sie verwenden, hängt von der Bereitstellung der Anwendung ab: *buy.itunes.apple.com* ist für Produktionssoftware da, *sandbox.itunes.apple.com* für die Entwicklung.

Der Rumpf der Anforderung enthält ein JSON-Dictionary aus einem Schlüssel (`"receipt-data"`) und einem Wert (einer Base64-kodierten Version der Daten aus der Transaktionsbestätigung). Gewöhnlich verwende ich die Base64-Erweiterung für `NSData` von *CocoaDev* (*http://www.cocoadev.com/index.pl?BaseSixtyFour*), um `NSData`-Objekte in Base64-kodierte Strings umzuwandeln. CocoaDev ist eine hervorragende Quelle für Mac- und iPhone-Entwickler.

Eine gültige Bestätigung gibt ein JSON-Dictionary wie das folgende zurück, das einen Transaktionsbezeichner, eine Produkt-ID für den gekauften Artikel, den Bundle-Bezeichner für die Host-Anwendung und ein Kaufdatum enthält sowie vor allem einen Status zurückgibt.

```
{"receipt":{"item_id":"330292806",
   "original_transaction_id":"1000000000052438", "bvrs":"1.0",
   "product_id":"com.sadun.scanner.disclosure2",
   "purchase_date":"2009-09-04 19:23:15 Etc/GMT", "quantity":"1",
   "bid":"com.sadun.Scanner",
   "original_purchase_date":"2009-09-04 19:23:15 Etc/GMT",
   "transaction_id":"1000000000052438"}, "status":0}
```

Eine gültige Bestätigung hat immer den Status 0. Jeder andere Wert zeigt an, dass die Bestätigung ungültig ist.

Die reine Überprüfung des Statuswerts reicht für die Validierung jedoch oft nicht aus, denn es ist nicht allzu schwer, einen Proxyserver einzurichten, der alle Aufrufe für den Validierungsserver abfängt und auf alle Anfragen den JSON-String `{"status":0}` zurückgibt. Darüber hinaus können auch die Bestätigungsdaten, die zusammen mit der Validierungsanforderung gesendet werden, leicht in die Daten serialisiert werden, die Sie im Abschnitt `"receipt"` des oben gezeigten JSON-Dictionarys gesehen haben. Aus diesem Grund sollten Sie die Überprüfung der Bestätigung vorsichtig und im Rahmen des Kaufvorgangs einsetzen, in dessen Verlauf es weniger wahrscheinlich ist, dass sich Proxyserver in die Kommunikation mit Apple einschalten.

```
// Erstellt eine JSON-Anforderung
NSString *json = [NSString stringWithFormat:
   @"{\"receipt-data\":\"%@\"}",
   [transaction.transactionReceipt base64Encoding]];

// Wählt einen Server zur Überprüfung der Bestätigung aus
NSString *urlsting = SANDBOX ?
   @"https://sandbox.itunes.apple.com/verifyReceipt" :
   @"https://buy.itunes.apple.com/verifyReceipt";

// Erstellt eine URL-Anforderung
NSMutableURLRequest *urlRequest = [NSMutableURLRequest
   requestWithURL:[NSURL URLWithString: urlsting]];
if (!urlRequest) NOTIFY_AND_LEAVE(@"Error creating the URL Request");

// Verwendet POST und legt als Rumpf einen kodierten JSON-String fest
[urlRequest setHTTPMethod: @"POST"];
[urlRequest setHTTPBody:[json dataUsingEncoding:NSUTF8StringEncoding]];

// Übermittelt die Anforderung und ruft die Antwort ab
NSError *error;
NSURLResponse *response;
NSData *result = [NSURLConnection sendSynchronousRequest:urlRequest
returningResponse:&response error:&error];
```

```
// Es sollte ein gültiger JSON-String mit einem Bestätigungs-Dictionary
// empfangen werden
NSString *resultString = [[NSString alloc] initWithData:result
encoding:NSUTF8StringEncoding];
CFShow(resultString);
[resultString release];
```

▸ *Listing 20.3: Eine Bestätigung überprüfen*

20.8 Zusammenfassung

Das Framework StoreKit bietet eine hervorragende Möglichkeit, um mit Ihren Anwendungen Geld zu verdienen. Wie Sie in diesem Kapitel gelesen haben, können Sie eine eigene Verkaufsoberfläche einrichten, um Dienste und Funktionen aus Ihrer Anwendung heraus anzubieten. Beachten Sie die folgenden abschließenden Gedanken:

- Auch wenn der Einrichtungs- und Testvorgang wie ein Henne-oder-Ei-Problem aussehen mag, so ist es doch wie hier gezeigt möglich, StoreKit-Anwendungen mit einem Minimum an Kopfzerbrechen zu entwickeln und bereitzustellen.
- Sie müssen Ihre Binärdatei verwerfen und dann eine neue einrichten, wenn Sie neue Zusatzprodukte hinzufügen. Achten Sie darauf, dass sowohl die Anwendung als auch die Zusatzartikel zur Begutachtung durch Apple bereit sind.
- Schließen Sie die Transaktion nicht ab, bevor die Einrichtung des neu erworbenen Produkts vollständig, endgültig und hundertprozentig abgeschlossen ist, selbst wenn Sie dazu auf einen Neustart der Anwendung warten müssen. Informieren Sie den Benutzer, wenn der Vorgang unerwarteten Verzögerungen unterliegt.
- Mit Ihren Methoden können Sie nur Produktinformationen von Zusatzartikeln abrufen, die bereits für die zurzeit laufende Anwendung registriert sind. Es ist nicht möglich, von einem Programm aus Produkte für ein anderes anzufordern.
- Vergessen Sie nicht, den Beobachter für den Verkauf einzurichten! Über diese Unterlassung sind schon mehr Entwickler gestolpert als über jedes andere StoreKit-Problem.

21 Bedienungshilfen und andere iPhone OS-Dienste

Die Interaktion der Anwendungen mit den Standarddiensten des iPhones erfolgt auf verschiedene Weisen, von denen Sie in diesem Kapitel einige kennenlernen. Anwendungen können ihre Schnittstellen für den Handler von VoiceOver definieren, einer Bedienungshilfe auf dem iPhone, um gesprochene Beschreibungen der Oberflächenelemente bereitzustellen. Als Entwickler können Sie auch Bundles erstellen, die auf die mitgelieferte Anwendung *Einstellungen* zugreifen, sodass die Benutzer über diese Schnittstelle Zugang zu den Voreinstellungen Ihres Programms haben. Es ist auch möglich, dass eine Anwendung ein öffentliches URL-Schema deklariert, sodass andere iPhone-Programme Kontakt damit aufnehmen und Dienste anfordern können, die sie selbst anbieten. In diesem Kapitel wird die Interaktion zwischen Anwendungen und Diensten vorgestellt. Sie lernen, wie Sie solche Funktionen in Ihren Programmen umsetzen und wie Sie im Code, in Interface Builder und in den unterstützenden Dateien solche »Dienstbrücken« anlegen.

21.1 VoiceOver zu Anwendungen hinzufügen

Bedienungshilfen machen das iPhone auch für Benutzer mit Behinderungen geeignet. Das iPhone OS bietet den Benutzern die Möglichkeit, Anzeigen zu vergrößern (zu zoomen), Farben umzukehren usw. Als Entwickler konzentrieren Sie sich bei den Bedienungshilfen vor allem auf VoiceOver, eine Möglichkeit, mit der sehbehinderte Benutzer sich die grafische Oberfläche »anhören« können. VoiceOver übersetzt die visuelle Darstellung eines Programms in eine hörbare Beschreibung.

Verwechseln Sie VoiceOver nicht mit Voice Control! *VoiceOver* ist eine Möglichkeit, um eine Audiobeschreibung der Benutzeroberfläche bereitzustellen, und beruht vor allem auf Gesten, während *Voice Control* die firmeneigene Stimmerkennungstechnologie von Apple für Freisprecheinrichtungen ist.

In diesem Abschnitt erhalten Sie einen kurzen Überblick über VoiceOver. Sie lesen hier, wie Sie Labels und Hinweise für Bedienungshilfen hinzufügen und sowohl im Simulator als auch auf dem iPhone testen. Bedienungshilfen sind ab der dritten Generation der Geräte verfügbar, z. B. auf dem iPhone 3GS oder einem iPod touch der dritten Generation.

21.1.1 Bedienungshilfen in Interface Builder

Im Bereich **Accessibility** des Identitäts-Informationsfeldes in Interface Builder (siehe *Abbildung 21.1*) können Sie zu den `UIKit`-Elementen Ihrer Oberfläche Labels und Hinweise (*Hints*) hinzufügen. Dazu geben Sie Strings in die beiden dafür vorgesehenen Felder ein. Diese beiden Felder und der Text dienen unterschiedlichen Zwecken für die Bedienungshilfen: Labels bezeichnen Ansichten, Hinweise beschreiben sie. Neben diesen Feldern finden Sie ein Markierungsfeld, um Bedienungshilfen im Allgemeinen zu aktivieren (**Accessibility ▸ Enabled**) und eine Reihe von Markierungsfeldern für die einzelnen Aspekte (**Traits**).

▸ *Abbildung 21.1: Im Identitäts-Informationsfeld von Interface Builder können Sie Angaben zu den Bedienungshilfen machen.*

Labels

Ein gutes Label sagt dem Benutzer, worum es sich bei einem bestimmten Element handelt, und das oft in einem einzigen Wort. Beschriften Sie eine barrierefreie Oberfläche ebenso, wie Sie eine Schaltfläche mit Text versehen. Labels wie »Bearbeiten«, »Löschen« und »Hinzufügen« beschreiben, was ein Objekt tut und eignen sich sehr gut als Text für Schaltflächen und für Bedienungshilfen.

Aber bei den Bedienungshilfen geht es nicht nur um Schaltflächen. »Feedback«, »Benutzerfoto« und »Benutzername« können Inhalt und Funktion einer Textansicht, einer Bildansicht oder eines Textlabels beschreiben. Wenn ein Objekt optisch eine Rolle in der Benutzeroberfläche spielt, sollte es das in VoiceOver auch akustisch tun. Beachten Sie bei der Gestaltung von Labels folgende Tipps:

- **Geben Sie im Label nicht den Typ der Ansicht an.** Schreiben Sie also nicht »Lösch-Schaltfläche«, »Feedback-Textansicht« oder »Textfeld Benutzername«, da VoiceOver diese Angaben automatisch hinzufügt, sodass aus der Bezeichnung »Lösch-Schaltfläche« im Identitäts-Informationsfeld bei der VoiceOver-Wiedergabe »Lösch-Schaltfläche-Schaltfläche« wird.

- **Schreiben Sie den ersten Buchstaben des Labels groß, setzen Sie am Ende aber keinen Punkt.** VoiceOver beachtet die Großschreibung, um das Label mit der richtigen Betonung zu sprechen. Wenn Sie einen Punkt setzen, senkt VoiceOver am Ende des Labels den Ton, sodass der Übergang zu dem nachfolgenden Objekttyp nicht gut klingt. »Löschen. Schaltfläche« klingt falsch, »Löschen-Schaltfläche« richtig.
- **Bündeln Sie die Informationen.** Wenn Sie es mit vielschichtigen Ansichten zu tun haben, die als eine einzige Einheit fungieren, fassen Sie alle Informationen zu dieser Ansicht in einem einzigen beschreibenden Label zusammen, das Sie dann mit der Elternansicht verknüpfen. Beispielsweise bündeln Sie in einer Tabellenansicht mit mehreren Unteransichten, die keine eigenen Steuerelemente aufweisen, alle Textinformationen in einem einzigen Label, das die gesamte Tabelle beschreibt.
- **Geben Sie Labels nur auf der direkten Interaktionsebene an.** Wenn die Benutzer in Unteransichten arbeiten, bringen Sie die Label dort an. Elternansichten, deren Kindansichten nicht zugänglich sind, brauchen keine Labels.
- **Übersetzen Sie die Labels.** Durch die Übersetzung der Bedienungshilfe-Strings machen Sie sie für ein größeres Publikum verfügbar.

Hinweise (Hints)

Hinweise teilen dem Benutzer mit, was er von der Betätigung eines Elements erwarten kann. Vor allem beschreiben sie Ergebnisse, die nicht offensichtlich sind. Stellen Sie sich z. B. eine Oberfläche vor, bei der beim Antippen eines Namens wie etwa Hans Mustermann versucht wird, die betreffende Person anzurufen. Der Name selbst gibt keinerlei Information darüber, was bei der Berührung passiert, sodass Sie den Benutzer mit einem Hinweis darauf aufmerksam machen sollten, z. B. durch »Ruft diese Person an« oder noch besser: »Ruft Hans Mustermann an«. Beachten Sie folgende Tipps, um gute Hinweise zu schreiben:

- **Schreiben Sie in Satzform.** Beginnen Sie den Satz mit einem Großbuchstaben, und beenden Sie ihn mit einem Punkt selbst dann, wenn der Hinweis kein Subjekt hat. Durch dieses Format kann VoiceOver den Hinweis mit der richtigen Betonung artikulieren.
- **Formulieren Sie den Satz so, dass er beschreibt, was das Element tut, und nicht, was der Benutzer tut.** »*[Dieses Textlabel]* ruft diese Person an« vermittelt dem Benutzer den richtigen Zusammenhang, »Rufen Sie diese Person an« dagegen nicht.
- **Nennen Sie nicht den Namen oder den Typ des GUI-Elements.** Verweisen Sie in den Hinweisen nicht auf das Element der Benutzeroberfläche, das der Benutzer betätigt, sondern verzichten Sie darauf, den Namen (sein Label, also z. B. »Löschen«) und den Typ (seine Klasse, z. B. »Schaltfläche«) zu nennen. VoiceOver fügt diese Informationen dort hinzu, wo sie für nötig befunden werden, was zu einer redundanten Ausgabe wie »Löschen-Schaltfläche *[Label]* Schaltfläche *[VoiceOver-Beschreibung]* Schaltfläche *[Hinweis]* entfernt das Element vom Bildschirm« führt. Verwenden Sie stattdessen: »Entfernt das Element vom Bildschirm.«

- **Nennen Sie nicht die Aktion.** Beschreiben Sie nicht die Aktion, die der Benutzer ausführen muss. Sagen Sie also nicht »Wegstreichen entfernt dieses Element vom Bildschirm« oder »Antippen ruft diese Person an«. VoiceOver verwendet einen eigenen Satz von Gesten zur Aktivierung von Elementen der Benutzeroberfläche. Verweisen Sie niemals direkt auf Gesten.
- **Geben Sie ausführliche Hinweise.** »Ruft an« beschreibt das Ergebnis nicht so deutlich wie »Ruft diese Person an« oder noch besser »Ruft Hans Mustermann an«. Eine kurze, aber ausführliche Erklärung hilft dem Benutzer mehr als eine, die so sparsam ist, dass der Benutzer die Einzelheiten erraten muss. Formulieren Sie keine Hinweise, die sich der Benutzer wiederholt anhören muss, bevor er weitermachen kann.
- **Übersetzen Sie die Hinweise.** Wie bei Labels ist auch die Übersetzung von Bedienungshilfen-Hinweisen für einen breit gefächerten Kundenstamm wichtig.

Bedienungshilfen aktivieren

Das Markierungsfeld **Enabled** entscheidet, ob eine `UIKit`-Ansicht VoiceOver einsetzt oder nicht. In der Regel sollten Sie diese Funktion aktiviert lassen, sofern es sich bei der Ansicht nicht um einen Container handelt, dessen Unteransichten zugänglich sein müssen. Aktivieren Sie diese Funktion nur auf der direkten Interaktions- oder Präsentationsebene. Eine VoiceOver-Beschreibung von Ansichten, die zur Gliederung anderer Ansichten dienen, ist nicht sinnvoll. Schließen Sie sie von VoiceOver aus.

Die Zellen von Tabellenansichten bieten ein gutes Beispiel für Bedienungshilfen-Container, also für Objekte, die andere Objekte enthalten. Für sie gelten folgende Regeln:

- Eine Tabellenzelle ohne eingebettete Steuerelemente sollte zugänglich sein.
- Eine Tabellenzelle mit eingebetteten Steuerelementen sollte nicht zugänglich sein, sondern die darin enthaltenen Steuerelemente.

Außerhalb von Interface Builder müssen nicht zugängliche Container angeben, wie viele zugängliche Kindansichten sie haben und welche das sind. Weitere Einzelheiten für die Programmierung von Bedienungshilfen-Containern finden Sie in den entsprechenden Apple-Richtlinien (*Accessibility Programming Guide*). Selbst geschriebene Containeransichten müssen das Protokoll `UIAccessibilityContainer` deklarieren und implementieren.

Aspekte (Traits)

Aspekte beschreiben das Verhalten von `UIKit`-Elementen. VoiceOver zieht diese Aspekte zur Erläuterung der Oberfläche heran. Wie *Abbildung 21.1* zeigt, können Sie Ansichten zwölf mögliche Aspekte zuweisen. Wählen Sie diejenigen aus, die sich für die gewählte Ansicht eignen. Diese Auswahl können Sie jederzeit im Programmcode ändern.

Die Bedienungshilfen-Dokumentation von Apple verlangt lediglich, dass Sie eines der vier folgenden Elemente auswählen, die sich gegenseitig ausschließen: **Button**, **Link**, **Static Text** oder **Search Field**. Wenn eine Schaltfläche auch als Link fungiert, können Sie entweder **Button** oder **Link** auswählen, aber nicht beides zugleich. Nehmen Sie den Aspekt, der die Verwendung des Steuerelements am besten beschreibt. Gleichzeitig kann eine Schaltfläche auch ein Bild zeigen oder einen Sound abspielen, wenn der Benutzer darauf tippt. Diese Aspekte können Sie ohne Einschränkung hinzufügen.

21.1.2 Bedienungshilfen im Code einrichten

Jede `UIKit`-Ansicht gehorcht dem Protokoll `UIAccessibility` und weist Eigenschaften auf, mit denen Sie Labels und Hinweise sowie die anderen Bedienungshilfen-Merkmale aus *Abbildung 21.1* festlegen können. Diese Eigenschaften können Sie in Interface Builder oder direkt im Code setzen. *Listing 21.1* richtet die Eigenschaft `accessibilityHint` so ein, dass der Hinweis zu einer Schaltfläche aktualisiert wird, wenn der Benutzer einen Benutzernamen in ein zugehöriges Textfeld eingibt, sodass der Hinweis den neuen Namen widerspiegelt.

```
- (BOOL)textField:(UITextField *)textField
   shouldChangeCharactersInRange:(NSRange)range
   replacementString:(NSString *)string
{
   // Bemerkt eine Änderung des Benutzernamens im Textfeld  und
   // aktualisiert den zugehörigen Hinweis   NSString *username = text-
Field.text;
   if (username && username.length > 1)
      callbutton.accessibilityHint = [NSString
      stringWithFormat:@"Places a call to %@", username];
   else
      callbutton.accessibilityHint =
         @"Places a call to the person named in the text field.";
   return YES;
}
```

▸ *Listing 21.1: Bedienungshilfen-Informationen durch den Programmcode aktualisieren*

21.1.3 Bedienungshilfen im Simulator testen

Das Bedienungshilfen-Informationsfeld im iPhone-Simulator dient dazu, barrierefreie Anwendungen vor der Bereitstellung auf dem iPhone zu testen. Dieses Informationsfeld simuliert die VoiceOver-Interaktion mit der Anwendung und gibt unmittelbare optische Rückmeldungen in einem Schwebefenster (es gibt keine Tonausgabe), ohne dass Sie direkt die Schnittstelle für VoiceOver-Gesten verwenden müssen. Da Sie viele VoiceOver-Gesten im Simulator gar nicht nachahmen können (z. B. dreifaches Wegstreichen und aufeinanderfolgendes Halten und Tippen), konzentriert sich das Informationsfeld vor allem auf die Beschreibung der Oberflächenelemente und weniger auf die Reaktion auf VoiceOver-Gesten.

Diese Funktion aktivieren Sie über **Einstellungen ▸ Allgemein ▸ Bedienungshilfen**. Schalten Sie **Accessibility Inspector** auf *On*. Daraufhin erscheint sofort das in Abbildung 21.2 gezeigte Informationsfeld und listet die aktuellen Einstellungen für das gerade markierte Element auf.

Um das Informationsfeld zu aktivieren und zu deaktivieren, verwenden Sie das eingekreiste X in der oberen linken Ecke. Wenn Sie darauf klicken, schrumpft das Informationsfeld zu einer einzigen, deaktivierten Zeile. Wenn Sie erneut darauf klicken, wird das Informationsfeld im aktiven Modus wiederhergestellt. Lassen Sie das Informationsfeld deaktiviert, bis Sie wirklich ein GUI-Element untersuchen müssen.

Wie VoiceOver stört das Informationsfeld die normalen Gesten und verlangsamt die Arbeit, sodass Sie es nur sparsam einsetzen sollten, also gewöhnlich zum Testen. Sie sollten Ihre Anwendung mit verfügbarem, aber deaktiviertem Informationsfeld starten. Wechseln Sie zu dem Bildschirm, mit dem Sie arbeiten wollen, und aktivieren Sie dann das Informationsfeld.

Die Anwendung aus *Abbildung 21.1* nutzt den Code aus *Listing 21.1*. Der Bedienungshilfen-Hinweis der Schaltfläche wird aktualisiert, wenn sich der Text im Eingabefeld ändert. Wenn Sie das Informationsfeld aktivieren, können Sie den jeweils aktuellen Hinweis sehen, während Sie den Inhalt des Textfeldes ändern, und so überprüfen, ob er tatsächlich dem eingegebenen Text entspricht.

21.1.4 Bedienungshilfen auf dem iPhone testen

Der Test auf dem iPhone ist ein entscheidender Schritt bei der Entwicklung von Bedienungshilfen, denn dort können Sie tatsächlich mit VoiceOver arbeiten statt mit dem textgestützten Informationsfeld. Sie hören, was die Benutzer hören, und können die Benutzeroberfläche mit Ihren Fingern und Ihren Ohren testen statt mit den Augen.

Wie im Simulator können Sie VoiceOver auch auf dem iPhone im laufenden Betrieb ein- und ausschalten. Es ist zwar möglich, VoiceOver in den Einstellungen zu aktivieren und Ihre Anwendung zu testen, während diese Funktion läuft, doch ist es weit einfacher, einen besonderen Umschalter zu verwenden, da Sie dann nicht mit komplizierten VoiceOver-Gesten von Einstellungen zu Ihrem Programm wechseln müssen. Sie schalten VoiceOver ab, starten Ihre Anwendung mit normalen iPhone-Gesten und schalten VoiceOver wieder an, wenn Sie für den Test bereit sind.

▸ *Abbildung 21.2: Das Bedienungshilfen-Informationsfeld des iPhone-Simulators zeigt das zurzeit markierte GUI-Element mit seinem Label, seinem Hinweis und anderen Bedienungshilfen-Eigenschaften an.*

Um den Umschalter nutzen zu können, gehen Sie wie folgt vor:

1. Wechseln Sie im Programm *Einstellungen* zu **Allgemein ▸ Bedienungshilfen**.
2. Suchen Sie die Option **Home-Dreifachklick**. Das dreifache Antippen der Schaltfläche **Home** ist ein Kurzbefehl für Bedienungshilfen, den der Benutzer selbst festlegen kann. Tippen Sie auf **Home-Dreifachklick**, um das Fenster **Startseite** zu öffnen.
3. Wählen Sie **VoiceOver ein/aus**, um diese Funktion für den Dreifachtipp zu definieren. Sobald Sie Ihre Auswahl getroffen haben (rechts neben dieser Option erscheint ein Häkchen), können Sie VoiceOver durch dreifaches Antippen der Schaltfläche **Home** unter dem iPhone-Bildschirm aktivieren und deaktivieren. Eine gesprochene Bestätigung gibt die aktuelle VoiceOver-Einstellung an.

Mit dem VoiceOver-Umschalter können Sie sich die Mühen ersparen, die der Wechsel zu Ihrer Anwendung mit Drei-Finger-Gesten und mehrfachem Antippen von Schaltflächen mit sich bringt. Allerdings sollen Sie durchaus mit VoiceOver-Gesten und -Interaktionsmöglichkeiten vertraut sein. *Tabelle 21.1* gibt einen Überblick über die VoiceOver-Gesten, die Sie zum Testen Ihrer Anwendung kennen sollten.

Aufgabe	VoiceOver-Geste
VoiceOver ein-/ausschalten	Tippen Sie dreifach auf die Schaltfläche **Home**.
Bildschirmvorhang ein-/ausschalten	Tippen Sie dreifach mit drei Fingern auf den Bildschirm.
VoiceOver-Ausgabe ein-/ausschalten	Um die VoiceOver-Sprachausgabe insgesamt auszuschalten (nicht nur für eine einzige Beschreibung), doppeltippen Sie mit drei Fingern auf den Bildschirm. VoiceOver selbst wird dadurch nicht deaktiviert.
Vorlesen des aktuellen Elements abbrechen	Doppeltippen Sie mit zwei Fingern. Doppeltippen Sie erneut, um das Vorlesen wieder aufzunehmen. Wenn VoiceOver nicht aktiviert ist, halten Sie mit dieser Geste auf dem Startbildschirm die Audiowiedergabe an und nehmen sie wieder auf.
Schaltflächen antippen	Methode 1: Tippen Sie, und halten Sie einen Finger auf der Schaltfläche, während Sie mit einem anderen Finger auf den Bildschirm tippen. Methode 2: Tippen Sie, um die Schaltfläche zu markieren, und doppeltippen Sie auf den Bildschirm, um sie zu aktivieren.

Durch eine Textansicht scrollen	Methode 1: Tippen Sie, und halten Sie einen Finger auf der Textansicht, während Sie mit einem anderen Finger tippen, um nach oben oder unten zu scrollen. Methode 2: Tippen Sie, um die Textansicht zu markieren, und doppeltippen Sie auf den Bildschirm, um nach oben oder unten zu scrollen.
Texteinfügemarke verschieben	Ist eine bearbeitbare Textansicht oder ein solches Feld markiert, können Sie die Einfügemarke verschieben, indem Sie mit einem einzigen Finger nach oben oder unten streichen. Je nach Voreinstellung erfolgt die Verschiebung zeichen- oder wortweise.
Menü für Sprachausgabe öffnen	Tippen Sie, und halten Sie einen Finger auf der Textansicht, während Sie mit einem anderen Finger nach oben oder unten streichen, um zwischen zeichenweiser und wortweiser Bewegung sowie dem Bearbeitungsmodus auszuwählen. Bei Letzterem wird die zuletzt festgelegte Bewegungsoption verwendet. (Diese Geste, offiziell »Rotorsteuerung« genannt, soll als Ziehbewegung mit gespreizten Fingern ausgeführt werden. Die hier angegebene Vorgehensweise hat sich bei Tests allerdings als weniger störanfällig erwiesen.)
Text markieren	Verschieben Sie die Einfügemarke, und rufen Sie den Bearbeitungsmodus auf (siehe oben). Tippen Sie auf die Textansicht, und halten Sie einen Finger darauf. Ziehen Sie nach links oder rechts.
Text eingeben	Wechseln Sie in den Texteingabemodus, indem Sie ein Textfeld oder eine Textansicht markieren und dann auf den Bildschirm doppeltippen. Die Tastatur erscheint auf dem Bildschirm. Eingabemethode 1: Tippen Sie mit dem linken Zeigefinger auf eine Schaltfläche der Tastatur, und halten Sie den Finger darauf. Tippen Sie mit dem rechten Zeigefinger auf eine andere Stelle des Bildschirms. Dies ist die beste Möglichkeit, um die Lösch-Schaltfläche wiederholt zu betätigen. Eingabemethode 2: Tippen Sie auf eine Schaltfläche, um sie auszuwählen, und doppeltippen Sie auf den Bildschirm, um das entsprechende Zeichen einzugeben.
Schieberegler bewegen	Markieren Sie den Regler, und streichen Sie dann mit einem Finger nach oben oder unten, um den Wert zu verändern.
Durch eine Liste scrollen	Streichen Sie mit drei Fingern nach oben oder unten.

Durch die Startseite des iPhone-Anwendungsbildschirms (Springboard) blättern	Streichen Sie mit drei Fingern nach links oder rechts.
Ein Element markieren und vorlesen	Tippen Sie auf das Element.
Ausgewähltes Element buchstabieren oder wortweise vorlesen	Streichen Sie mit einem einzigen Finger nach oben oder unten. Dabei werden die Einstellungen aus dem Menü für gesprochenen Text verwendet.
Nächstes oder vorhergehendes Element vorlesen	Streichen Sie mit einem einzigen Finger nach links oder rechts.
Gesamten Bildschirminhalt vorlesen	Streichen Sie zweimal nach oben. Dies funktioniert leider nicht durchgängig. Alternativ können Sie wie folgt vorgehen: Streichen Sie wiederholt zum ersten Element auf dem Bildschirm nach links, und streichen Sie dann mit zwei Fingern nach unten. Durch die Zwei-Finger-Streichbewegung wird der Bildschirminhalt jetzt beginnend mit dem ausgewählten Element vorgelesen.
iPhone entsperren	Markieren Sie den Entsperr-Regler, und doppeltippen Sie auf den Bildschirm.

▸ *Tabelle 21.1: Gebräuchliche VoiceOver-Gesten für Anwendungen ab iPhone OS 3.1*

Beachten Sie vor allem die Funktion Bildschirmvorhang, mit der Sie die iPhone-Anzeige vollständig ausblenden können, um die Verwendung Ihres Programms über eine rein akustische Schnittstelle zu testen. Wenn Sie sich einen Eindruck davon verschaffen möchten, was für eine Herausforderung es bedeutet, eine iPhone-Anwendung zu nutzen, ohne sehen zu können, versuchen Sie, das Taschenrechner-Programm mit aktiviertem Bildschirmvorhang zu bedienen.

21.2 Rezept: Eigene Einstellungs-Bundles hinzufügen

Über die Klasse `NSUserDefaults` verwaltet das iPhone Voreinstellungen für Anwendungen. Damit können Sie Informationen speichern, die Ihr Programm zwischen zwei Sitzungen festhalten muss, beispielsweise den aktuellen Spieler, eine Liste von Spielständen oder die zuletzt verwendete Ansichtskonfiguration. Über Benutzereinstellungen legen Sie programmgesteuert Werte in einer dauerhaften Datenbank ab, die mit Ihrer Anwendung verknüpft ist. Diese Voreinstellungen werden im Ordner `Library` in der Sandbox des Programms abgelegt, und zwar in einer Eigenschaftslistendatei mit dem Anwendungsbezeichner als Namen.

Benutzervoreinstellungen handhaben Sie wie veränderbare Dictionarys. Wie in normalen Dictionarys erfolgt auch hier das Festlegen und Abrufen der Objekte über Schlüssel. Allerdings sind die Einträge für Benutzereinstellungen auf die standardmäßigen Typen für Eigenschaftslisten beschränkt, also auf `NSString`, `NSNumber`, `NSDate`, `NSData`, `NSArray` und `NSDictionary`. Wenn Sie Informationen speichern müssen, die nicht in diese Klassen fallen, können Sie eine andere Datei verwenden (im Bibliotheks- oder Dokumentenordner der Sandbox) oder die Objekte als `NSData` serialisieren und diese Daten dann in den Voreinstellungen ablegen.

Die Methode `synchronize` setzt die Datenbank der Voreinstellungen auf die letzten Änderungen, die im Arbeitsspeicher vorgenommen wurden. Durch diese Synchronisierung sorgen Sie dafür, dass die Einstellungsdaten in der Datei auf dem neuesten Stand sind, was sehr wichtig ist, wenn das Programm aus irgendeinem Grund unterbrochen werden sollte. Das folgende Codefragment zeigt, wie Sie Daten in dem Voreinstellungssystem aus *Rezept 8.5* festlegen, synchronisieren und abrufen.

```
[[NSUserDefaults standardUserDefaults]
   setObject:colors forKey:@"colors"];
[[NSUserDefaults standardUserDefaults]
   setObject:locs forKey:@"locs"];

[[NSUserDefaults standardUserDefaults] synchronize];

NSLog(@"%@", [[NSUserDefaults] objectForKey:@"lastViewTag"]);
```

21.2.1 Das Programm »Einstellungen«

iPhone-Anwendungen können eigene Voreinstellungen zum Programm *Einstellungen* hinzufügen (siehe *Abbildung 21.3*). Damit greifen Sie auf dieselben Benutzereinstellungen zu wie im Code, allerdings bietet das Programm *Einstellungen* eine komfortable Oberfläche für die Benutzer. Alle Änderungen, die die Benutzer auf diesem Bildschirm vornehmen, aktualisieren die standardmäßigen Voreinstellungen.

Die Einstellungen für Drittanbieterprogramme werden hinter den Systemeinstellungen aufgeführt, sehen ansonsten aber genauso aus wie diejenigen, die Apple vorab in das System geladen hat, und verhalten sich auch genauso. Wie der Screenshot in *Abbildung 21.3* zeigt, können Sie in eigenen Voreinstellungen verschiedene Möglichkeiten zur Interaktion wie Textfelder, Schalter und Schieberegler anbieten.

Da durch diese Einstellungen normale `NSUserDefaults`-Einträge erstellt werden, können Sie sie bequem im Code abfragen und bearbeiten. So definiert *Rezept 21.1* beispielsweise das Feld **Name** (siehe *Abbildung 21.3*, rechter Screenshot, erstes Element der oberen Gruppe), dessen Wert im Schlüssel `@"name_preference"` gespeichert wird. Ob der Benutzer einen Wert für diesen Schlüssel eingegeben hat, können Sie in der Anwendung ermitteln:

```
NSLog(@"%@", [[NSUserDefaults] objectForKey:@"name_preference"]);
```

► *Abbildung 21.3: Eigene Einstellungs-Bundles von Drittanbieteranwendungen erscheinen auf dem Bildschirm des Programms „Einstellungen" hinter den mitgelieferten Einstellungen (links). Vom Entwickler definierte Elemente für Voreinstellungen können Textfelder (normale und sichere), Schalter, Schieberegler, Optionsschalter, Gruppentitel und untergeordnete Bereiche enthalten (rechts).*

21.2.2 Vermeiden Sie sensible Informationen

Speichern Sie in Voreinstellungen keine sensiblen Account-Daten wie Benutzernamen und die Auswahl von Optionen! Passwörter werden zwar durch Punkte verschleiert, in der Anwendungs-Sandbox aber im Klartext gespeichert. Wenn Sie mit sensiblen Informationen zu tun haben, sollten Sie stattdessen den Schlüsselbund des iPhones verwenden. Einstellungs-Bundles können zurzeit nicht mit dem Schlüsselbund kombiniert werden. Rezepte für die Nutzung des Schlüsselbunds finden Sie in *Kapitel 13, Netzwerke*.

21.2.3 Das Einstellungsschema

Eine Kopie des Einstellungsschemas befindet sich im Entwicklerordner unter `/Developer/Platforms/iPhoneOS.platform/Developer/Library/Xcode/Plug-ins/iPhoneSettingsPlistStructDefs.xcodeplugin`. Xcode verwendet diese Datei, um die Syntax der Einstellungsliste zu prüfen. In dieser Datei sehen Sie alle Definitionen und alle erforderlichen und optionalen Attribute zur Angabe Ihrer Voreinstellungen. Sollte Apple die Definitionen jemals erweitern oder ändern, können Sie die Änderungen in dieser Datei finden.

21.2.4 Ein Einstellungs-Bundle definieren

Jeder Einstellungsbereich entspricht einer Eigenschaftslistendatei. *Rezept 21.1* zeigt den Quellcode für den Bereich aus *Abbildung 21.3* (rechts), führt die einzelnen Einstellungstypen aus dem SDK vor und gibt eine Beispieldefinition. Mögliche Typen sind Textfelder (Strings), Schieberegler (Fließkommazahlen), Schalter (boolesche Werte) und Auswahlfelder (eine Auswahl aus *n* Möglichkeiten). Außerdem können Sie Elemente gruppieren und mit Kindbereichen verknüpfen.

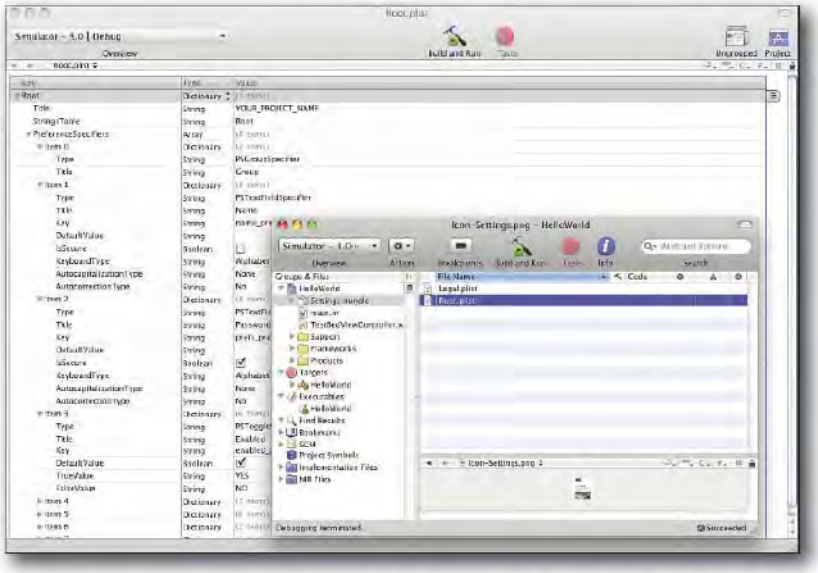
Um neue Einstellungen vorzusehen, erstellen Sie ein Dictionary und fügen es dem Array `PreferencesSpecifiers` hinzu. Jedes Voreinstellungs-Dictionary muss mindestens einen Typ und einen Titel enthalten. Manche Einstellungen, z. B. das Gruppenelement `PSGroupSpecifier`, brauchen auch nicht mehr, um zu funktionieren. Andere, z. B. Textfelder, benötigen dagegen eine ganze Reihe an Eigenschaften. Wie Sie in *Rezept 21.1* sehen, müssen Sie für diese Elemente Groß- und Kleinschreibung und das Autokorrekturverhalten sowie den Tastaturtyp festlegen und angeben, ob der Text durch die Passwort-Sicherheitsfunktion verschleiert werden soll.

Um Ihrem Programm ein neues Einstellungs-Bundle hinzuzufügen, führen Sie die folgenden Schritte durch. Alternativ können Sie das neue Bundle auch anlegen, indem Sie **File ▸ New File ▸ iPhone OS ▸ Resource ▸ Settings Bundle** wählen.

1. Erstellen Sie die einzelnen Eigenschaftenlisten, jeweils eine für jeden Bildschirm. Die primäre Datei muss `Root.plist` heißen.
2. Erstellen Sie einen neuen Ordner, und fügen Sie die Eigenschaftenlisten hinzu.
3. Nennen Sie den Ordner `Settings.bundle`. OS X gibt eine Warnung wegen dieses Namens aus, doch bestätigen Sie ihn einfach. Der Ordner wird in ein Bundle umgewandelt. (Um den Inhalt des neuen Bundles einzusehen, klicken Sie mit der rechten Maustaste oder bei gedrückter Ctrl-Taste darauf und wählen **Paketinhalt zeigen** aus dem Kontextmenü.)
4. Ziehen Sie das Bundle in die Spalte **Groups & Files** Ihres Xcode-Projekts (siehe Abbildung 21.4).
5. Erstellen Sie eine Version der Hauptsymboldatei für Ihre Anwendung (gewöhnlich `icon.png`) in der Größe 29 × 29 Pixel und fügen Sie sie unter dem Namen `Icon-Settings.png` dem Projekt hinzu. Diese Grafik wird im Programm Einstellungen neben dem Anwendungsnamen dazu verwendet, Ihr Bundle zu kennzeichnen, wie Sie in *Abbildung 21.3* (links) sehen. Für das kleine Symbol neben **Hello World** wird diese verkleinerte Grafik verwendet.

Wenn Sie das Programm das nächste Mal ausführen, wird das Einstellungs-Bundle installiert und in der Anwendung *Einstellungen* zur Verfügung gestellt. Sollten in Ihrem Quellcode Syntaxfehler enthalten sein, wird statt der erwarteten Einstellungen ein leerer Bildschirm angezeigt. Dies können Sie am besten vermeiden, indem Sie die Einstellungen Schritt für Schritt entwickeln.

Xcode enthält ein interaktives Fenster mit Syntaxüberprüfung, das aber nur eingeschränkte Möglichkeiten hat. Öffnen Sie die Eigenschaftenliste, und wählen Sie **View ▸ Property List Type ▸ iPhone Settings plist**. Wenn Sie mit systemnäheren Werkzeugen vertraut sind, finden Sie es wahrscheinlich einfacher, die Bearbeitung manuell in *TextEdit* oder im eigenständigen *Property List Editor* zu bearbeiten.



▶ *Abbildung 21.4: Fügen Sie* `Settings.bundle` *zur Liste* **Groups & Files** *Ihres Projekts hinzu. Wenn Sie auf eine Eigenschaftenliste doppelklicken, können Sie sie in Xcode weiterbearbeiten.*

Hinweis

In der Eigenschaft `File` wird in *Rezept 21.1* keine Erweiterung angegeben, da `.plist` angenommen wird.

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE plist PUBLIC "-//Apple//DTD PLIST 1.0//EN"
"http://www.apple.com/DTDs/PropertyList-1.0.dtd">
<plist version="1.0">
<dict>
   <key>Title</key>
   <string>YOUR_PROJECT_NAME</string>
   <key>StringsTable</key>
   <string>Root</string>
   <key>PreferenceSpecifiers</key>
   <array>
      <dict>
         <key>Type</key>
         <string>PSGroupSpecifier</string>
         <key>Title</key>
         <string>Group</string>
      </dict>
      <dict>
```

```
        <key>Type</key>
        <string>PSTextFieldSpecifier</string>
        <key>Title</key>
        <string>Name</string>
        <key>Key</key>
        <string>name_preference</string>
        <key>DefaultValue</key>
        <string></string>
        <key>IsSecure</key>
        <false/>
        <key>KeyboardType</key>
        <string>Alphabet</string>
        <key>AutocapitalizationType</key>
        <string>None</string>
        <key>AutocorrectionType</key>
        <string>No</string>
    </dict>
    <dict>
        <key>Type</key>
        <string>PSTextFieldSpecifier</string>
        <key>Title</key>
        <string>Password</string>
        <key>Key</key>
        <string>prefs_preference</string>
        <key>DefaultValue</key>
        <string></string>
        <key>IsSecure</key>
        <true/>
        <key>KeyboardType</key>
        <string>Alphabet</string>
        <key>AutocapitalizationType</key>
        <string>None</string>
        <key>AutocorrectionType</key>
        <string>No</string>
    </dict>
    <dict>
        <key>Type</key>
        <string>PSToggleSwitchSpecifier</string>
        <key>Title</key>
        <string>Enabled</string>
        <key>Key</key>
        <string>enabled_preference</string>
        <key>DefaultValue</key>
        <true/>
        <key>TrueValue</key>
```

```
        <string>YES</string>
        <key>FalseValue</key>
        <string>NO</string>
    </dict>
    <dict>
        <key>Type</key>
        <string>PSSliderSpecifier</string>
        <key>Key</key>
        <string>slider_preference</string>
        <key>DefaultValue</key>
        <real>0.5</real>
        <key>MinimumValue</key>
        <integer>0</integer>
        <key>MaximumValue</key>
        <integer>1</integer>
        <key>MinimumValueImage</key>
        <string></string>
        <key>MaximumValueImage</key>
        <string></string>
    </dict>
    <dict>
        <key>Type</key>
        <string>PSMultiValueSpecifier</string>
        <key>Key</key>
        <string>multi_preference</string>
        <key>DefaultValue</key>
        <string>One</string>
        <key>Title</key>
        <string>MultiValue</string>
        <key>Titles</key>
        <array>
            <string>one</string>
            <string>two</string>
            <string>three</string>
            <string>four</string>
        </array>
        <key>Values</key>
        <array>
            <string>one</string>
            <string>two</string>
            <string>three</string>
            <string>four</string>
        </array>
    </dict>
    <dict>
```

```
            <key>Type</key>
            <string>PSGroupSpecifier</string>
            <key>Title</key>
            <string>Info</string>
        </dict>
        <dict>
            <key>Type</key>
            <string>PSChildPaneSpecifier</string>
            <key>Title</key>
            <string>Legal</string>
            <key>File</key>
            <string>Legal</string>
        </dict>
    </array>
</dict>
</plist>
```

▸ *Rezept 21.1: Einen eigenen Bereich für Einstellungen anlegen*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 21 und öffnen das Projekt zu diesem Rezept.

21.2.5 Einstellungen und Benutzer

Einstellungs-Bundles bilden für Entwickler zwar eine klar umrissene Möglichkeit, Benutzereinstellungen zu zentralisieren, doch lässt die Erfahrung aus der Praxis Zweifel daran aufkommen, ob man sie wirklich einsetzen sollte. Nur wenige iPhone-Benutzer sind sich darüber im Klaren, dass sie Einstellungen für die Produkte von Drittanbietern auch außerhalb von deren Programmen vornehmen können, und noch weniger Personen verwenden solche Einstellungen regelmäßig. Die meisten wollen für die Aufgaben, die Sie mit einem Programm erfüllen, innerhalb von dessen Grenzen bleiben, und das gilt auch für die Einstellungen.

Daher haben viele (wenn nicht sogar die meisten) App Store-Anbieter die Verwendung von Einstellungs-Bundles aufgegeben und die Einstellungen stattdessen direkt in ihre Programme aufgenommen. In Einstellungsansichten können die Benutzer die Voreinstellungen schnell finden und anpassen. Leider ist es ziemlich arbeitsintensiv und umständlich, solche Ansichten zu erstellen.

Zum Glück gibt es einen Mittelweg zwischen reinen Einstellungs-Bundles und eigenen Ansichten. Das Projekt *Llama Settings* unter *Google Code* (*http://code.google.com/p/llamasettings/*) umfasst eine Reihe von Klassen, die Eigenschaftenlisten lesen (auch in Ihrem Einstellungs-Bundle), sodass

Sie einen Einstellungsbildschirm in Ihrem Programm anzeigen können, ohne dass dadurch ein Übermaß an Zusatzarbeit erforderlich wäre. Das Projekt wurde von Scott Lawrence entwickelt und wird auch von ihm unterhalten.

Die Llama-Settings-Klassen liegen als Open-Source-Code vor und bieten die üblichen Anzeige- und Interaktionselemente wie Gruppentitel, Schieberegler und Schalter. Darüber hinaus ermöglichen sie jedoch auch Farbwähler, URL-Starter usw. Diese Elemente können zwar in der Anwendung *Einstellungen* von Apple nicht eingesetzt werden, doch können Sie sie ohne weiteren Programmieraufwand in Ihrer Anwendung nutzen, wenn Sie standardmäßige Eigenschaftenlisten definieren.

21.2.6 Benutzervoreinstellungen abrufen

Einstellungen können Sie über Bundles, Ansichten innerhalb der Anwendung, Zugriff im Code oder durch eine Mischung dieser Verfahren anfordern und festlegen. Denken Sie bei der Nutzung der Einstellungen aber daran, dass einige Elemente vielleicht noch gar nicht existieren. Wenn ein Benutzer das Einstellungs-Bundle noch nicht geöffnet hat, sind möglicherweise nicht einmal die in den Eigenschaftenlisten dieses Bundles angegebenen Voreinstellungen in Kraft getreten. Bei den meisten Objekten können Sie dies mit `objectForKey:` herausfinden. Gibt es einen Schlüssel gar nicht, so gibt diese Methode `nil` zurück.

Es gibt einen Fall, in dem der Wert `nil` bei der Programmierung eine besondere Bedeutung hat. Eine Voreinstellung, die Sie immer im Code festlegen sollten, ist der Schlüssel für die »letzte Version«, in dem die zuletzt ausgeführte Version der Anwendung festgehalten wird. Bei jedem Start der Anwendung rufen Sie diese Voreinstellung ab.

Lautet die Voreinstellung `nil`, ist die Anwendung frisch installiert worden. Sie wird also zum ersten Mal ausgeführt, weshalb Sie an dieser Stelle wahrscheinlich Dateien vorbereiten und andere Einrichtungsaufgaben vornehmen. Anschließend legen Sie für diesen Schlüssel einen Wert fest, der die zurzeit bereitgestellte Version des Programms angibt. (Vergessen Sie nicht, die Einstellungen anschließend zu synchronisieren!)

Es reicht jedoch nicht aus, nur zu prüfen, ob der Wert `nil` lautet. Durch einen Abruf dieser Einstellung können Sie stets erkennen, ob der Benutzer die Anwendung gerade von einer früheren Version aktualisiert hat. Wenn die Version bei der letzten Ausführung von der jetzigen abweicht, haben Sie die Gelegenheit, Aktualisierungsaufgaben vorzunehmen, damit die Benutzerdaten mit dem jüngsten Release kompatibel bleiben.

21.3 Rezept: URL-gestützte Dienste erstellen

Die mitgelieferten Programme von Apple bieten verschiedene Dienste an, die sich über URL-Aufrufe nutzen lassen: Mit *Safari* können Sie Webseiten öffnen, mit *Karten* haben Sie Zugriff auf Landkarten, und in *Mail* können Sie über einen `mailto:`-URL einen Brief verfassen.

Diese Dienste funktionieren, da das iPhone weiß, mit welcher Anwendung ein *URL-Schema* jeweils verknüpft ist. Bei diesem Schema handelt es sich um den ersten Teil des URLs, der vor dem Doppelpunkt steht, z. B. `http` oder `ftp`. Ein URL, der mit `http:` beginnt, wird in *Mobile Safari* geöffnet,

während `mailto:` stets mit *Mail* verknüpft ist. Wussten Sie aber, dass Sie Ihre eigenen URL-Schemas definieren und in Ihren Anwendungen nutzen können? Allerdings können auf dem iPhone nicht alle Standardschemas eingesetzt werden. Beispielsweise steht `ftp` nicht zur Verfügung.

Wenn Sie eigene Schemas verwenden, können Sie damit Ihre Anwendungen starten, sobald Mobile Safari (oder ein anderes Programm) einen URL des betreffenden Typs öffnet. Ist für Ihre Anwendung z. B. `xyz` registriert, werden alle `xyz://`-Links zur Verarbeitung an Ihre Anwendung weitergereicht, wo sie der optionalen Methode `application:handleOpenURL:` übergeben werden. Die Anwendung wird unabhängig davon gestartet, ob Sie eine Handler-Methode definieren oder nicht. Wenn Sie lediglich wollen, dass das Programm ausgeführt wird, können Sie einen solchen anwendungsübergreifenden Start einfach dadurch erreichen, dass Sie das Schema hinzufügen und den URL öffnen.

Handler erweitern diese Vorgehensweise über den bloßen Start der Anwendung hinaus so, dass irgendetwas mit dem URL geschieht. Beispielsweise kann eine bestimmte Datendatei geöffnet, ein bestimmter Name abgerufen, ein bestimmtes Bild angezeigt oder irgendeine andere in dem Aufruf enthaltene Information verarbeitet werden.

21.3.1 URL-Schemas verwenden

Der schemagestützte Anwendungsstart weist viele Vorteile auf. Betrachten Sie als Beispiel *Twitterific* von *Iconfactory*. Der Entwickler Craig Hockenberry bietet einen Dienst an, über den Benutzer und Drittentwickler seine Anwendung starten und damit einen vorausgefüllten, zur Einsendung bereiten Tweet öffnen können.

Dadurch können Entwickler ihren Anwendungen ohne jeglichen Programmieraufwand Twitter-Unterstützung hinzufügen. Da sich Twitterific um die Speicherung sensibler Angaben wie Benutzername und Passwort kümmert, müssen Sie lediglich für den eigentlichen Text Sorge tragen. Wird Twitterific aufgerufen, geht die Steuerung an dieses Programm über, in dem die Benutzer dann den Tweet fertigstellen können. Anschließend beenden die Benutzer Twitterific und können auf Wunsch zu der ursprünglichen Anwendung zurückkehren.

Solche Dienste erzielen den größten Erfolg, wenn sie irgendeine Form von Leistungssteigerung oder Datennutzung bieten. Bei Twitterific geht es nicht darum, Tweets zu erstellen, denn dazu ist nur wenig Code erforderlich (siehe *Kapitel 13, Netzwerke*). Wenn Sie das aber selbst erledigen wollen, müssen Sie entweder die Verantwortung für die sichere Speicherung der Anmeldeinformationen übernehmen oder die Benutzer dazu zwingen, die Informationen jedes Mal anzugeben. Der Dienst Twitterific ermöglicht es Ihnen, diese Probleme links liegen zu lassen und sich stärker auf das Funktionieren Ihrer Anwendung zu konzentrieren.

21.3.2 Nachteile von Diensten

Es gibt jedoch nicht nur gute Nachrichten, was Drittanbieterdienste angeht, sondern auch einen eindeutigen Nachteil. Wenn Ihre Anwendung auf einen Dienst angewiesen ist, zwingen Sie Ihre Benutzer im Grunde dazu, eine zweite Anwendung herunterzuladen, und dabei muss es sich nicht unbedingt um das Programm handeln, das die Benutzer bevorzugen würden. Stellen Sie sich z. B. einen treuen Be-

nutzer von *Echofon* (früher *TwitterFon*) vor, der Twitterific auf seinem Gerät nicht installiert hat. Wenn Sie nun Twitterific zur Voraussetzung machen, können Sie damit auf Widerstand stoßen.

Alle Funktionen, die auf Drittanbieterdiensten beruhen, müssen optional sein. Nehmen wir beispielsweise an, Echofon würde ein eigenes URL-Schema für Tweets einrichten. Wenn Ihre Anwendung die Tweet-Erstellung mithilfe von Diensten anbietet, müssen Sie sie in einem solchen Fall so umfangreich und flexibel gestalten, dass die Benutzer ihren bevorzugten Client auswählen können.

Ein weiterer Nachteil besteht darin, dass iPhone-Anwendungen nicht mitteilen können, welche Schemas verfügbar sind. Es gibt keine Möglichkeit, das Gesamtangebot an Diensten abzurufen. Apple unterhält kein öffentliches Register, das Sie durchsuchen könnten, um herauszufinden, was es alles gibt. Die Nutzung von Diensten ist letzten Endes Vertrauenssache.

Sie können jedoch herausfinden, ob ein gegebener URL-Dienst verfügbar ist. Wenn die Methode `canOpenURL:` von `UIApplication` den Wert `YES` zurückgibt, haben Sie die Garantie, dass `openULR:` eine andere Anwendung starten und den betreffenden URL öffnen kann. Es ist nicht gewährleistet, dass der URL gültig ist, sondern nur, dass das Schema korrekt für eine vorhandene Anwendung registriert ist.

```
if ([[UIApplication sharedApplication] canOpenURL:aURL])
   [[UIApplication sharedApplication] openURL:aURL];
```

21.3.3 Anwendungsübergreifende Eigenwerbung

Es gibt noch einen weiteren wichtigen geschäftlichen Aspekt beim schemagestützten Anwendungsstart, nämlich die anwendungsübergreifende Eigenwerbung. Durch die Definition von URL-Schemas kann Ihre Anwendung nachsehen, ob weitere Programme aus dem Angebot Ihrer Firma auf dem Gerät verfügbar sind. Wenn die Anwendung selbst den URL nicht handhaben kann (wenn also `canOpenURL:` den Wert `NO` zurückgibt), können Sie Links zum App Store angeben und die Benutzer dazu auffordern, andere Programme Ihrer Firma herunterzuladen.

21.3.4 Schemaregistrierung: Den URL deklarieren

Um einen Dienst zu Ihrer Anwendung hinzuzufügen, sind zwei Schritte erforderlich. Erstens müssen Sie das URL-Schema in `Info.plist` deklarieren, und zweitens müssen Sie einen Handler zu Ihrer Anwendungsdelegierung hinzufügen. Wie dies geschieht, erfahren Sie in diesem und dem folgenden Abschnitt.

Um das URL-Schema zu deklarieren, müssen Sie Angaben für den *Launch Service* des iPhones machen. Fügen Sie dazu den Eintrag `CFBundleURLTypes` zur Datei `Info.plist` hinzu. Dabei handelt es sich um ein Array aus Dictionarys, die beschreiben, welche URL-Typen die Anwendung öffnen und verarbeiten kann. Jedes Dictionary weist zwei Schlüssel auf, nämlich `CFBundleURLName` und ein Array aus `CFBundleURLSchemes`.

Der URL-Name ist eine abstrakte Bezeichnung, für die Sie einen beliebigen String verwenden können. Auf dem Mac wird sie für die sichtbare Beschreibung im Finder herangezogen, doch auf dem iPhone bildet sie lediglich eine Möglichkeit zur Ordnung der Schemas.

Das Schema-Array ist eine Liste von Präfixen, die zu dem abstrakten Namen gehören, wobei Sie ein, aber auch viele Schemas angeben können. In dem folgenden Code wird nur ein einziges deklariert. Dem Namen können Sie ein x voranstellen. Auch wenn das iPhone keiner Normung durch irgendeine Standardisierungsorganisation unterliegt, zeigt das Präfix x an, dass es sich um einen nicht registrierten Namen handelt.

```
<key>CFBundleURLTypes</key>
<array>
   <dict>
      <key>CFBundleURLName</key>
      <string>com.sadun.demonstration</string>
      <key>CFBundleURLSchemes</key>
      <array>
         <string>x-sadun-services</string>
      </array>
   </dict>
</array>
```

Der iPhone-Entwickler Emanuele Vulcano hat auf der Website von *CocoaDev* mit einer informellen Registrierung begonnen (*http://cocoadev.com/index.pl?ChooseYourOwniPhoneURLScheme*). iPhone-Entwickler können ihre Schemas in einer zentralen Liste bekannt machen, die Ihnen zeigt, welche Dienste Sie nutzen können, und Ihnen die Gelegenheit bietet, Ihre eigenen Dienste anzukündigen. Aufgeführt werden die Dienste mit ihren URL-Schemas und einer Beschreibung, wie andere Entwickler sie einsetzen können.

21.3.5 Schemaregistrierung: Die Handler-Methode hinzufügen

Als zweiten Schritt zur Arbeit mit URL-Schemas müssen Sie eine Anwendungs-Delegierungsmethode namens `application:handleOpenURL:` vorsehen:

```
- (BOOL)application:(UIApplication *)application
   handleOpenURL:(NSURL *)url {}
```

Wenn Ihre Anwendungsdelegierung diese Methode implementiert, können Sie damit auf einen `openURL:`-Aufruf aus einem anderen Programm reagieren. Ihre Methode muss einen booleschen Wert zurückgeben: entweder `YES`, wenn der URL erfolgreich verarbeitet werden konnte, oder andernfalls `NO`. Das Grundgerüst dieser Funktion sieht wie folgt aus:

```
- (BOOL)application:(UIApplication *)application
   handleOpenURL:(NSURL *)url
{
   // Ruft den String ab
   if (!url) return NO;
   NSString *URLString = [url absoluteString];

   // Hier steht Ihr eigener Code
   return YES;
}
```

In iPhone-URLs ist der Doppelpunkt zwingend erforderlich, die Schrägstriche dahinter aber nicht. Beispielsweise ist `mailto:foo@bar` ein gültiger URL. Sie müssen nicht `mailto://foo@bar` verwenden.

Zerlegen Sie den String im Aufruf, indem Sie das einleitende URL-Schema bis zum Doppelpunkt entfernen:

```
NSRange colon = [URLString rangeOfString:@":"];
NSString *request = [URLString substringFromIndex:
   (colon.location + 1)];
```

Es ist Aufgabe Ihres Handlers, die weitergeleitete Anforderung zu verarbeiten. Dazu definieren Sie das Protokoll und implementieren die Art und Weise, wie es abgerufen wird. Betrachten Sie dazu das folgende Anforderungsprotokoll als Beispiel:

```
x-sadun-services:command?param1=p1&param2=p2&param3=p3&...
```

Dieses Protokoll setzt voraus, dass hinter dem Fragezeichen ein Befehl steht, auf den ein Satz von Parameterpaaren folgt. Diese Paare enthalten jeweils einen Parameternamen in Klartext und einen URI-kodierten Text als Parameterwert, die durch ein Gleichheitszeichen getrennt sind.

Ihr Code führt die gesamte Analyse der Anforderung aus. Kann die Nutzung von URL-Schemas Dritten eine Angriffsmöglichkeit bieten? Nein. Die tatsächliche Gefahr ist sehr gering, falls es überhaupt eine gibt. Sie können wählen, ob Sie die Anforderung verarbeiten oder ignorieren oder was auch immer damit tun. `NSStrings` bilden für Ihre Anwendung nur eine sehr geringe Gefahr.

21.3.6 Die Steuerung an die aufrufende Anwendung zurückgeben

Bei sorgfältiger Programmierung können Sie es der aufrufenden Anwendung ermöglichen, nach der Verarbeitung einer URL-Anforderung die Steuerung zurückzuerlangen. Je nachdem, wie Sie das Protokoll definieren und implementieren, kann Ihre Anwendung angefordertes Material zusammen mit einer Statusmeldung darüber zurücksenden, ob die Operation erfolgreich verlaufen ist. Das folgende Beispiel zeigt die Einfüge-Anforderung einer Anwendung, die programmübergreifendes Kopieren und Einfügen erlaubt:

```
x-sadun-services:paste?scheme=iping&data=hello+world&\
clipboard=test1&password=foobar&expire=1500
```

Dieser Beispiel-URL zeigt die Anforderung, »Hello World« in die Zwischenablage `test1` aufzunehmen. Dabei handelt es sich um eine sichere Zwischenablage, deren Daten nach 1500 Sekunden (25 Minuten) getilgt werden. Beachten Sie hier vor allem den Parameter `scheme`, der dem Dienst mitteilt, auf welches Programm er reagiert. Nach dem Einfügevorgang öffnet der Dienst einen neuen URL mit den Ergebnissen und verwendet dabei das Schema, um die aufrufende Anwendung anzusprechen.

Auf den meisten iPhone- und iPod touch-Geräten dauert der gesamte Vorgang 4,5 bis 5 Sekunden, was Sie auch selbst ausprobieren können. Der Beispielcode für dieses Rezept enthält zwei Anwendungen, von denen die eine (iPong) ein Server für das Kopieren und Einfügen ist und die andere (iPing) als Testclient dient. Der Client misst, wie lange es vom Senden der Anforderung bis zum Ein-

gehen des Antwort-URLs dauert. Nachdem Sie den Server installiert haben, können Sie mit dem Client alle möglichen Situationen durchspielen, um den Dienst und sein Protokoll zu testen.

Hierbei öffnet sich jedoch eine kleine Sicherheitslücke. Wenn der aufrufende Client »lügt« und einen falschen Schemaparameter angibt, kann er die Steuerung an eine dritte Anwendung übergeben. Gehört das Schema nicht zu einem tatsächlich vorhandenen Programm, hängt sich die Anforderung einfach auf. Auf der anderen Seite ist es ein großer Gewinn, wenn der Dienst zu der ursprünglichen Anwendung zurückkehren kann, denn dann müssen die Benutzer nicht das zweite Programm beenden und das erste erneut starten. Es sollte mehr Anwendungen geben, die diese Möglichkeit bieten, nachdem sie die Verarbeitung einer Anforderung abgeschlossen haben, sei es durch Senden einer E-Mail oder durch Einstellen eines Tweets.

21.3.7 Eigene Schemas implementieren

Wenn eine Anwendung auf dem iPhone installiert wird, weist sie das iPhone OS durch ihre `Info.plist`-Datei an, sie mit den von Ihnen definierten Schemas zu verknüpfen. Sobald das Betriebssystem auf eines dieser Schemas stößt, öffnet es die entsprechende Anwendung, um den URL zu verarbeiten. *Rezept 21.2* zeigt ein Beispiel für das Grundgerüst einer Methode, die sich um das Öffnen des URLs kümmert.

In diesem Rezept können Sie erkennen, wie Sie das URL-Schema abrufen und in seine Parameter zerlegen und wie Sie zu der aufrufenden Anwendung zurückkehren. Der eigentliche Beispielcode für iPong enthält ebenfalls alle diese Bestandteile, verwendet aber einen anspruchsvolleren Handler als die Methode aus *Rezept 21.2*.

Hinweis

Um Videos in YouTube zu öffnen, verwenden Sie den URL-String *http://www.youtube.com/watch?v=VIDEO_IDENTIFIER* oder *http://www.youtube.com/v/VIDEO_IDENTIFIER*. Das iPhone nutzt keine eigenen Schemas für YouTube.

```
- (BOOL)application:(UIApplication *)application
   handleOpenURL:(NSURL *)url
{
   // Ruft den String ab
   if (!url) return YES;
   NSString *URLString = [url absoluteString];

   // Findet die Position des Doppelpunkts
   NSRange colon = [URLString rangeOfString:@":"];
   if (colon.location == NSNotFound) return YES;

   // Entnimmt den Befehl und das Parameter-Dictionary
   NSString *action = [URLString substringFromIndex:
```

```
        (colon.location + 1)];
    NSMutableDictionary *paramDict = [NSMutableDictionary dictionary];
    NSRange r = [action rangeOfString:@"?"];
    if (r.location != NSNotFound)
    {
        NSString *paramString = [action substringFromIndex:
            (r.location + 1)];
        NSArray *parameters = [paramString
            componentsSeparatedByString:@"&"];
        action = [action substringToIndex:r.location];

        for (NSString *eachParam in parameters)
        {
            NSArray *pair = [eachParam
                componentsSeparatedByString:@"="];
            if ([pair count] != 2) continue;
            NSString *key = [[pair objectAtIndex:0] lowercaseString];
            NSString *value = [pair objectAtIndex:1];
            [paramDict setValue:value forKey:key];
        }
    }

    // Hier erfolgt die eigentliche Verarbeitung anhand der Parameter

    // Kehrt mit einem Ergebnis zu iPong zurück
    NSString *scheme = [paramDict objectFoKey:@"scheme"];
    if (!scheme) return YES;

    NSString *urlString = [NSString stringWithFormat:
        @":pasteservice?status=Success", scheme];
    NSURL *outurl = [NSURL urlWithString:urlString];
    if ([application canOpenURL:outurl]
        [application openURL:outurl];

    return YES;
}
```

► *Rezept 21.2: Auf URL-Schemaanforderungen reagieren*

Den Rezeptcode finden

Den in diesem Rezept verwendeten Code erhalten Sie unter *http://github.com/erica/iphone-3.0-cookbook-*. Falls Sie das Diskimage mit dem gesamten Beispielcode zum Buch heruntergeladen haben, wechseln Sie zum Ordner für Kapitel 21 und öffnen das Projekt zu diesem Rezept.

21.4 Zusammenfassung

Eine Anwendung, die Dienste von iPhone OS nutzt, wird in einem größeren Umfeld aktiv. Bedienungshilfen, Einstellungen und URL-Schemas sind Beispiele dafür, wie eine Anwendung über ihren eigenen Funktionsumfang hinausgehen und die Möglichkeiten von iPhone OS nutzen kann. Nehmen Sie aus der Lektüre dieses Kapitels die folgenden Gedanken mit:

- VoiceOver und die anderen Bedienungshilfen des iPhones sind noch sehr, sehr neue Funktionen, sodass Sie erwarten können, dass sie sich mit zunehmender Reife der Plattform noch weiterentwickeln. Gerade diese Funktion wird sich noch ändern, wenn Apple seinen Kundenstamm behinderter iPhone-Benutzer weiter ausbaut.
- Wenn Sie Ihrer Anwendung Labels und Hinweise für Bedienungshilfen hinzufügen, erweitern Sie damit Ihr Publikum ebenso wie durch die Lokalisierung. All dies kostet wenig Mühe, macht sich für Ihre Benutzer aber deutlich bezahlt.
- Die meisten Benutzer greifen in der Anwendung *Einstellungen* nicht auf die Voreinstellungen von Drittanbietern zurück, doch einige tun es. Überlegen Sie, ob Sie den Zugriff auf die Einstellungen sowohl innerhalb als auch außerhalb Ihres Programms ermöglichen sollten.
- Erweitern Sie den Benutzerstamm Ihrer Anwendung, indem Sie hilfreiche Funktionen anderer Programme anbieten. Mit URL-Schemas können Sie eine Nachfrage durch andere Anwendungsentwickler hervorrufen, die letzten Endes zu gesteigerten Verkaufszahlen für Ihr Produkt führt. Bieten Sie anderen Entwicklern über eigene URL-Schemas einen Satz von hervorragenden und leicht zu nutzenden Diensten an.

A Schlüssel in Info.plist

Tabelle A.1 führt viele der Schlüssel in der Datei `Info.plist` auf, die auf dem iPhone zur Verfügung stehen, und beschreibt ihre Verwendung. Als Erstes wird jeweils der Schlüsselname im Code genannt, darunter folgt ggf. der englischsprachige String, der im Info.plist-Editor von Xcode angezeigt wird, wenn Sie im Kontextmenü nicht **Show Raw Keys/Values** ausgewählt haben.

Schlüssel	Typ	Verwendung
`CFBundleDisplayName` **Bundle display name.**	String	Der Anzeigename des Anwendungs-Bundles. Dieser Name kann mit `InfoPlist.strings` lokalisiert werden. Xcode setzt diesen Wert zu Anfang auf den Namen, unter dem Sie das Projekt erstellt haben.
`CFBundleName` **Bundle name.**	String	Der Anzeigename der Anwendung in Kurzform, normalerweise identisch mit `CFBundleDisplayName`.
`CFBundleDevelopmentRegion` **Localization native development region.**	String	Die Heimatregion des Bundle-Autors. Legt den Standardwert für die Lokalisierung fest.
`CFBundleAllowMixed Localizations` **Localized resources can be mixed.**	Boolescher Wert	Erlaubt Frameworks, lokalisierte Ressourcen abzurufen.

`CFBundleExecutable` **Executable file.**	String	(Erforderlicher Schlüssel.) Der Name der ausführbaren Datei im Anwendungs-Bundle. Der Name dieser Eigenschaft wird automatisch zur Eigenschaftenliste hinzufügt. Der Wert beruht zu Anfang auf der Variable `${EXECUTABLE_NAME}` in Ihrem Xcode-Projekt und ist gewöhnlich mit dem Projektnamen identisch.
`CFBundleIconFile` **Icon file.**	String	Der Name der Symboldatei für die Anwendung. Normalerweise ist dies `icon.png`, doch können Sie auch einen anderen Namen verwenden. Xcode fügt keinen Standardwert hinzu. App Store verlangt, dass dieser Wert korrekt festgelegt ist.
`CFBundleIdentifier` **Bundle identifier.**	String	(Erforderlicher Schlüssel.) Der eindeutige Bezeichner für Ihre Anwendung. Informationen darüber, wie Sie solche Bezeichner auswählen und registrieren, finden Sie in Kapitel 1, *Einführung in das iPhone SDK*, und 2, *Ein erstes Projekt erstellen*.
`CFBundleInfoDictionaryVersion` **InfoDictionary version.**	String	(Erforderlicher Schlüssel.) Normalerweise auf 6.0 gesetzt. Dieser Wert gibt die aktuelle Version der Eigenschaftenlisten-Struktur an. Xcode fügt diesen Wert automatisch zu Ihrer `Info.plist`-Datei hinzu. Lassen Sie ihn unverändert.
`CFBundleLocalizations` **Localizations.**	Array	Eine Liste von Strings, die alle unterstützten Lokalisierungen angeben, z.B. `en`, `fr`, `ja` usw. Jedes Element dieses Arrays ist ein String, der den Namen einer Sprache angibt und auch Regionsbezeichner enthalten kann wie in `en-us` und `en-uk`.
`CFBundleVersionString` **Bundle version.**	String	String mit der Versionsnummer der Anwendung. App Store erfordert, dass diese Nummer für jeden Build eindeutig sein muss.

`CFBundleShortVersion`
`➥String`
Bundle version string, short.
	String	Dieser Schlüssel ist zwar auch auf dem iPhone verfügbar, ist aber eigentlich für Mac-Anwendungen gedacht, die Versionsbezeichnungen aus drei Zahlen (Release, Revision und Wartungs-Release) verwenden.

`CFBundleURLTypes` **URL types.**	Array	Ein Array aus Dictionarys, die die von der Anwendung unterstützten URL-Schemas beschreiben. Einzelheiten zum Aufbau dieses Arrays finden Sie in Kapitel 21, *Bedienungshilfen und andere iPhone OS-Dienste*.
`LSRequiresIPhoneOS` **Application requires iPhone environment.**	Boolescher Wert	Zeigt an, ob die Anwendung nur auf dem iPhone ausgeführt werden kann.
`NSMainNibFile` **Main nib file base name.**	String	Gibt die primäre `xib`/`nib`-Datei an, die in der Anwendung genutzt wird.
`UIInterfaceOrientation` **Initial interface orientation.**	String	Gibt an, welche Oberflächenorientierung beim Start der Anwendung verwendet werden soll. Mögliche Werte sind `UIInterfaceOrientationPortrait` (Hochformat, Standardwert), `UInterfaceOrientationPortraitUpsideDown` (Hochformat), `UIInterfaceOrientationLandscapeLeft` (Querformat) und `IInterfaceOrientationLandscapeRight` (Querformat).
`UIPrerenderedIcon` **Icon already includes gloss and bevel effects.**	Boolescher Wert	Legt fest, ob das iPhone das Anwendungssymbol mit einem Schimmer versieht (`NO`) oder es so lässt, wie es ist (`YES`).
`UIRequiresPersistentWiFi` **Application uses Wi-Fi.**	Boolescher Wert	Bei `YES` teilt dieser Schlüssel dem iPhone OS mit, dass die Anwendung eine Wi-Fi-Verbindung benötigt. Gibt es keine solche Verbindung, wird der Benutzer beim Programmstart dazu aufgefordert, Kontakt mit einem Wi-Fi-Netzwerk aufzunehmen. Bei `NO` schließt das iPhone OS alle aktiven Wi-Fi-Verbindungen nach 30 Minuten. Dies geschieht bei der Einstellung `YES` nicht, sodass die Wi-Fi-Verbindung dann dauerhaft verwendet werden kann.
`UIStatusBarHidden` **Status bar is initially hidden.**	Boolescher Wert	Bei `true` wird die Statusleiste beim Start ausgeblendet.

`UIStatusBarStyle` **Status bar style.**	String	Legt unter Verwendung der Standardkonstanten ein Anfangsformat für die Statusleiste fest. Voreingestellt ist eine graue Leiste. Mögliche Werte sind: `UIStatusBarStyleDefault` (grau, Standardwert), `UIStatusBarStyleBlackTranslucent` (schwarz und durchsichtig [Alpha-Wert 0,5]) und `UIStatusBarStyleBlackOpaque` (schwarz und undurchsichtig).
`UIRequiredDeviceCapabilities` **Required device capabilities.**	Array	Gibt an, welche Gerätefähigkeiten vorhanden sein müssen, damit die Anwendung ausgeführt werden kann, z.B. eine Kamera. Im iPhone OS 3.1 sind diese Werte möglich: `wifi`, `accelerometer`, `location-services`, `gps`, `magnetometer`, `microphone`, `opengles-1`, `opengles-2`, `armv6`, `armv7` und `peer-peer`. Anhand dieser Angabe können der App Store und iTunes erkennen, welche Gerätefunktionen ein Programm braucht, um ordnungsgemäß zu funktionieren. Einzelheiten darüber erfahren Sie in Kapitel 14, *Gerätefähigkeiten*.
`UISupportedExternalAccessoryProtocols` **Supported external accessory protocols.**	Array	Definiert die von der Anwendung zur Kommunikation mit Drittherstellerhardware unterstützten Zugriffsprotokolle.
`UIViewEdgeAntialiasing` **Renders with edge antialiasing.**	Boolescher Wert	Gibt an, ob Core Animation-Ebenen Daten, die nicht pixelgenau sind, einem Antialiasing unterziehen. Der Standardwert lautet `NO`. Wenn Sie den Wert auf `YES` ändern, wird die Darstellungsqualität erhöht, wobei aber die Leistung merkbar sinkt.
`UIViewGroupOpacity` **Renders with group opacity.**	Boolescher Wert	Gibt an, ob Core Animation-Unterebenen die Transparenz ihrer übergeordneten Ebene erben. Der Standardwert beträgt `NO`.

▸ *Tabelle A.1: Gebräuchliche Schlüssel in* `Info.plist`

Stichwortverzeichnis

B

C

E

F

M

N

P

V

W

X