App-Programmierung für Einsteiger (iPhone-Ausgabe)

Norbert Usadel

SMART BOOKS

Norbert Usadel

App-Programmierung leicht gemacht

SMART BOOKS

App-Programmierung leicht gemacht

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.

Copyright © 2012 Smart Books Publishing AG, Ruessenstr. 5a, CH-6340 Baar ZG

ISBN:
Buch 978-3-908498-07-0
PDF 978-3-908498-90-2
ePub 978-3-908498-91-9

2. Aufage 2012

Lektorat:	Jeremias Radke
Projektleitung:	Horst-Dieter Radke
Korrektorat:	Dr. Anja Stiller-Reimpell
Layout und Satz:	Susanne Streicher
Covergestaltung:	Johanna Voss, Florstadt
Coverfoto:	ist2_535716 / iStockphoto
Illustrationen:	iStock_000000123028 / iStockphoto
Druck und Bindung:	M. P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

Umwelthinweis:
Dieses Buch wurde auf chlorfrei gebleichtem Papier gedruckt. Die Einschrumpffolie – zum Schutz vor Verschmutzung – ist aus umweltverträglichem und recyclingfähigem PE-Material.

Trotz sorgfältigem Lektorat schleichen sich manchmal Fehler ein. Autoren und Verlag sind Ihnen dankbar für Anregungen und Hinweise!

Smart Books Publishing AG	c/o dpunkt.verlag GmbH Ringstr. 19 b - 69115 Heidelberg - Deutschland
http://www.smartbooks.ch	E-Mail: smartbooks@dpunkt.de

Alle Rechte vorbehalten. Die Verwendung der Texte und Bilder, auch auszugsweise, ist ohne die schriftliche Zustimmung des Verlags urheberrechtswidrig und strafbar. Das gilt insbesondere für die Vervielfältigung, Übersetzung, die Verwendung in Kursunterlagen oder elektronischen Systemen. Der Verlag übernimmt keine Haftung für Folgen, die auf unvollständige oder fehlerhafte Angaben in diesem Buch oder auf die Verwendung der mitgelieferten Software zurückzuführen sind. Nahezu alle in diesem Buch behandelten Hard- und Software-Bezeichnungen sind zugleich eingetragene Warenzeichen oder sollten als solche behandelt werden.

Besuchen Sie uns im Internet!
www.smartbooks.ch
www.smartbooks.de

Übersicht

Inhaltsverzeichnis

Kapitel

1

Einleitung

Einleitung

Über dieses Buch

Das Buch richtet sich an die Leser, die den Schnelleinstig in die Welt der App-Programmierung suchen. Es ist stark praxisorientiert geschrieben. Learning by doing steht im Vordergrund. Durch insgesamt zehn Workshops wird der Einstieg in die Programmierung von Apps erleichtert. Aber Vorsicht! Es besteht Suchtgefahr! Hat man einmal den Bogen raus, kann man nicht mehr loslassen, und nach der Lektüre dieses Buches ist der Weg zu den eigenen Apps frei. Sie erlernen spielerisch den Umgang mit Xcode und dem Interface Builder.

Ich habe den Workshops einen Theorieteil vorangestellt. Dieser ist so knapp wie möglich gehalten. Er reicht aus, sich die nötigen Grundbegriffe für den Start in die App-Programmierung anzueignen. Spezielle Programmiertechniken und Module der Entwicklungsumgebung werden in den einzelnen Workshops erläutert.

Und nun: Viel Spaß beim Learning by doing. Werden Sie noch heute ein iPhone-Entwickler und ein guter X-Coder.

Aufbau des Buches

Die ersten sechs Kapitel des Buches befassen sich mit der Theorie rund um das iPhone-SDK. Sie lernen dort die Grundschritte für den Tanz mit Xcode und dem Interface Builder. Ein Kapitel enthält den Einstieg in die Programmiersprache Objektive C. In diesem Kapitel sollten Sie die Programm-Schnipsel in Xcode und dem Interface Builder nachvollziehen. Dort wird beschrieben, wie Ihnen das gelingt. Der Grundgedanke, der dahintersteckt, ist, dass Sie das Programmieren nicht durch Lesen erlernen, sondern indem Sie sich fortwährend mit der Entwicklungsumgebung an Ihrem Mac auseinandersetzen: Code eingeben, kompilieren und sehen, was geschieht. Und das Ganze wieder von vorne, dabei nicht den Spaß an der Sache verlieren. In einem Kapitel erhalten Sie einen Ausblick auf Xcode 4. Die neue Entwicklungsumgebung vereint Xcode und den Interface Builder in einem Fenster. Sie können problemlos Ihre Projekte, die Sie unter Xcode 3 hier in diesem Buch erstellt haben, in dem neuen Programm testen und ablaufen lassen.

Webseite zum Buch

Falls Sie Fragen und Anregungen haben, besuchen Sie meine Webseite www.appzitty.de. In dieser URL dreht sich alles um die App-Programmierung. Im Bereich www.appzitty/download liegen die einzelnen Xcode-Projekte aus diesem Buch für Sie bereit. Ich bin für Experimente jederzeit offen. Sie können mich aber auch unter der E-Mail-Adresse buch@appzitty.de erreichen. Es würde mich freuen, auf einem dieser Wege Kontakt mit Ihnen aufnehmen zu dürfen.

Danksagung

Ein Buch kann man nur schreiben, wenn im Hintergrund Menschen mitwirken und den Autor aufmuntern. Dank an meine Familie: Alex, David und Gaby. Dank an die Informatik-Beratung Renner. An Herrn Renner konnte ich mich jederzeit mit fachlichen Fragen wenden. Dank auch an Hans-Peter Kusserow, der mir immer mit Tipps und Tricks zur Seite stand, egal, auf welchem Projektstand sich das Buch gerade befand. Dank auch an das Lektorat. Dank gebührt auch meinem Freund Siggi, der mich in zahlreichen Saunagängen immer wieder aufmunterte, dieses Buch weiterzuschreiben.

Kapitel

2

Der Start

»Aller Anfang ist leicht, und die letzten Stufen werden am schwersten und seltensten erstiegen.« Soweit das Zitat von Johan Wolfgang von Goethe aus dem Roman »Wilhelm Meisters Wanderjahre« Und so lautet auch der Leitspruch der nächsten Kapitel. Der Einstieg wird Ihnen so leicht wie möglich gemacht. Sie starten mit Ihrem Mac und diesem Buch und los geht's. Die letzten Schritte sind steiniger, aber mit meiner Hilfe werden Sie zu den Wenigen gehören, die auch sie meistern. Dieses Buch ist so praxisorientiert, wie ein Buch über App-Programmierung es eben sein kann. Lassen Sie es neben dem Mac liegen und feuern Sie Ihr Xcode mit Ihren eigenen Versuchen an. Ausprobieren und nochmals ausprobieren. Die ersten Lernerfolge werden sich schnell einstellen. Folgen Sie den einzelnen Workshops, die Ihnen immer neue Aspekte der App-Programmierung eröffnen. Und nun: Viel Spaß bei Ihren App-Projekten!

Die Hardware

Sie benötigen einen Mac und ein iPhone, um zu starten. Das SDK, also das »Software Development Kit« mit seinen Komponenten, läuft nur auf Apple-Geräten. Es gibt da keine Alternative. Sollten Sie also noch keinen Apple-Computer haben, wäre der günstigste Einstieg ein Mac Mini. Er ist neu für circa 650 Euro erhältlich. Sie können sich auch einen gebrauchten Mac zulegen. Diese sind als Intel-Variante ab etwa 400 Euro zu bekommen. Der Mac muss einen Intel-Prozessor haben. Und Snow Leopard Mac OS X 10.6 oder höher muss installiert sein, sonst läuft das neuste SDK nicht auf ihm.

Produktübersicht einiger Mac-Modelle

Sie benötigen weiterhin ein iPhone 3 GS ab 8 GB Speicher, um alle Features ausprobieren zu können, die von der Entwicklungsumgebung des SDKs angeboten werden. Es gibt zwar in der Entwicklungsumgebung des SDKs einen Simulator. Dieser unterliegt aber gewissen Einschränkungen. So bleibt nur, eine App auf einem richtigen iPhone, iPad oder iPod Touch zu testen.

Das iPhone-SDK

Sie bekommen das iPhone-SDK gratis unter diesem Link: http://developer.apple.com/devcenter/ios/index.action

Das iOS Dev Center

Im iOS Dev Center müssen Sie sich registrieren lassen und das SDK herunterladen. Der Download ist kostenlos. Sie erhalten nun Zugriff auf Tutorials, Referenzen, Tools und Beispiele rund um die Programmierung von Apps. Die Sammlung

ist sehr umfangreich und nicht mit den SDKs vergleichbar, die nur eine Sammlung von Bibliotheken sind.

Den Download beginnen Sie hier mit einem Klick.

Die neuste Version ist das SDK 4.1 mit Xcode 3.2.4. Es läuft nur unter Snow Leopard. Ein Klick genügt, und der Download der 3 GB großen Datei beginnt. Sie müssen circa eine Stunde warten, bis der Download komplett ist. Die entpackte Datei des SDKs mit allen seinen Zusatzprogrammen nimmt etwa 7 GB Speicher auf Ihrer Festpatte ein.

Die Entwicklerlizenz

Wenn Sie Ihre App in den App Store bringen möchten, benötigen Sie eine Entwicklerlizenz. Diese kostet 99 Dollar im Jahr und kann unter dem Link http://developer.apple.com/programs/start/standard/ beantragt werden.

Das Procedere bei der Beantragung einer Entwicklerlizenz

Die Entwicklerlizenz enthält folgende Leistungen:

- Zugriff auf sämtliche Ressourcen wie beispielsweise das aktuelle SDK, Dokumentationen und Programmierbeispiele
- das Erstellen von Entwicklertests und Adhoc-Profilen
- technischen Support
- Zugriff auf alle Entwicklerforen
- Möglichkeit zur Veröffentlichung Ihrer App im App Store

Sie müssen sich nicht sofort registrieren und können später entscheiden, ob es sich für Sie lohnt, eine App in den Store zu bringen. Wollen Sie jedoch eine App veröffentlichen, ist die Registrierung der einzige Weg. Sie müssen Zeit einkalkulieren, um den Status eines DevPro-Entwicklers zu bekommen. Das kann gut 10 Tage dauern. Manchmal reagiert Apple wie eine Behörde. So kann es passieren, dass Apple Sie auffordert, weitere Angaben zu machen und diese dann per Fax an die Registrierungsstelle zu schicken.

Kapitel

3

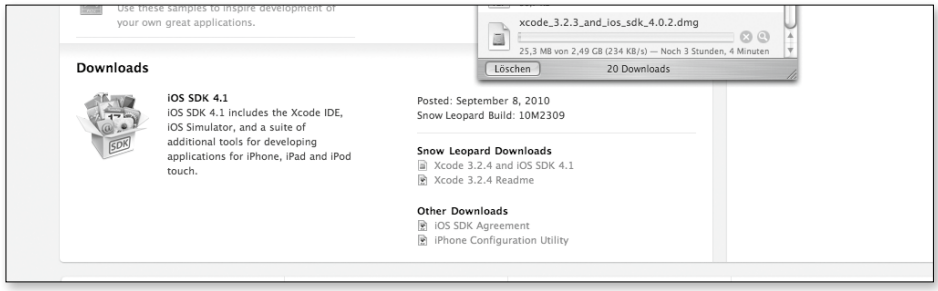
Die Arbeit mit dem iPhone-SDK

In diesem Kapitel machen Sie die ersten Schritte mit dem iPhone-SDK. Sie werden es installieren und sich mit der Arbeitsumgebung vertraut machen. Außerdem lernen Sie die wichtigsten Module kennen. Am Ende des Kapitels erstellen Sie schon Ihre erste kleine App.

iPhone Developerkit (SDK) installieren

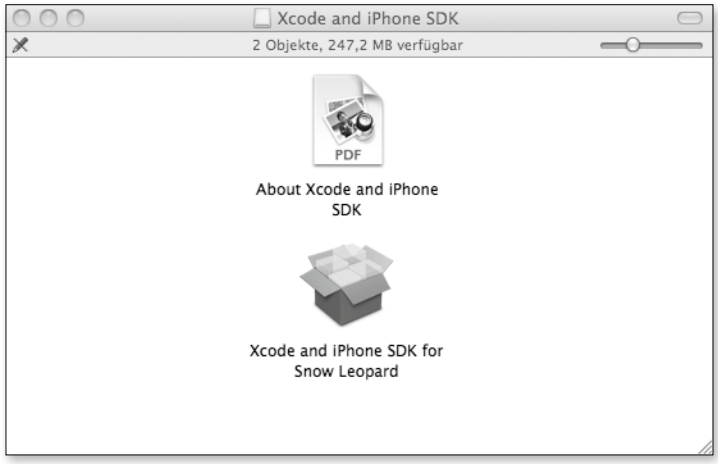
Ihre neue Entwicklungsumgebung, das iPhone-SDK mit Xcode, bekommen Sie auf der Seite http://developer.apple.com/devcenter/ios/index.action.

Folgen Sie den Anweisungen und laden Sie sich das Programm herunter. Das dauert seine Zeit, denn das gesamte Paket ist 2,4 GB groß.



Hier sehen Sie den Ladevorgang des iOS SDK 4.1.

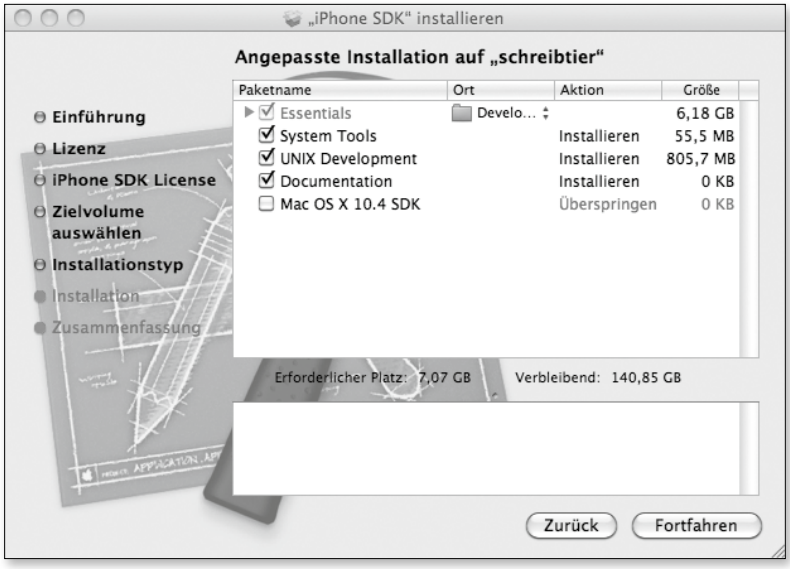
1. Installieren Sie das SDK per Doppelklick, indem Sie auf das Image der Datei klicken.



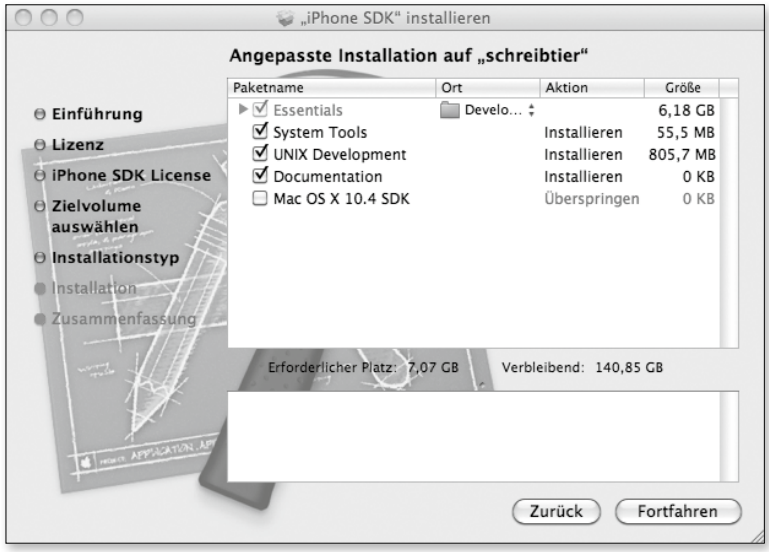
2. Danach erscheint eine Dialogbox. Bestätigen Sie mit der Schaltfläche *Fortfahren*.

3. Es folgen weitere Anweisungen. Sie müssen im nächsten Schritt den Lizenzbedingungen zustimmen.

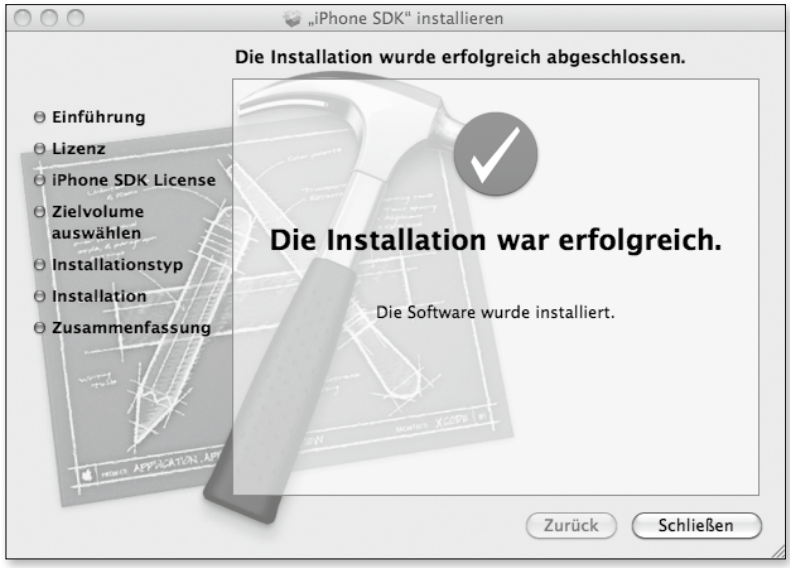
4. Hier wählen Sie den Installationstyp. Lassen Sie die Voreinstellungen so, wie sie sind und bestätigen Sie mit dem Button *Fortfahren*. Danach wird das Programm an dem angegebenen Ort auf der Festplatte installiert.



5. Hier starten Sie den eigentlichen Installationsvorgang. Das SDK wird auf der Festplatte »Schreibtier« installiert. Lassen Sie die Voreinstellungen bestehen und bestätigen Sie den Vorgang mit *Fortfahren*. Sie sehen, dass die Installation 7 GB Festplattenspeicher belegt.



6. War die Installation erfolgreich, erscheint eine runde, grüne Fläche in der Dialogbox. Quittieren Sie den Vorgang mit *Schließen*. Ihr SDK ist nun installiert.



Nach erfolgreicher Installation wurde das SDK im Ordner Developer installiert

Das SDK liegt im Ordner Developer.

Öffnen Sie nun den Ordner *Developer*. Es erscheinen die unten aufgeführten Ordner. Hier befinden sich die einzelnen Komponenten des iPhone-SDKs. Das Herzstück ist das Programm Xcode. Mit diesem Programm verwalten Sie Ihre Projekte und entwickeln Ihre Apps.

Die Komponenten des iPhone-SDKs mit Xcode

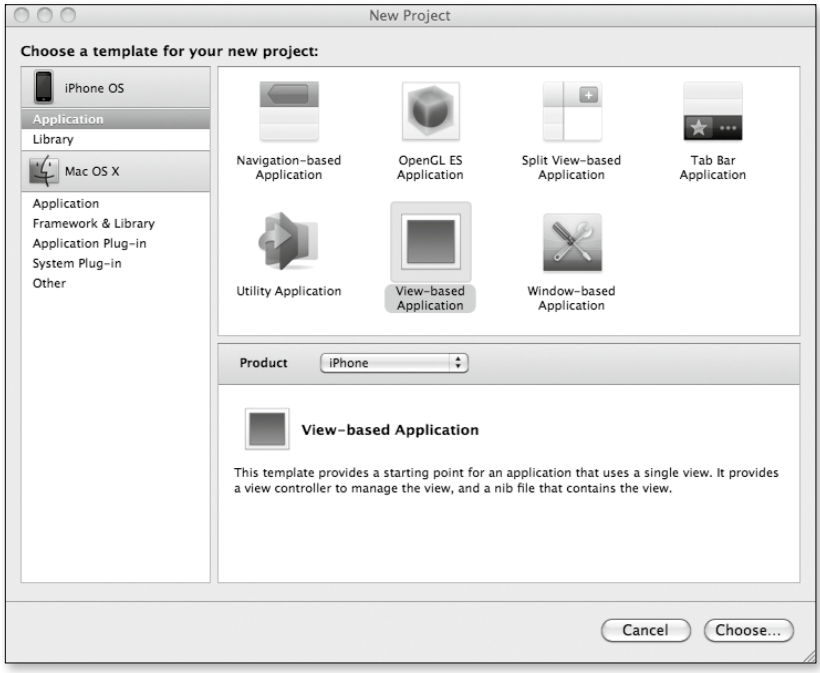
Xcode

Xcode ist die Schaltzentrale der App-Entwicklung. Mit diesem Programm schreiben Sie Code und kompilieren Ihren Quellcode zu einem lauffähigen Programm. Hier findet auch die Fehlersuche, das Debugging, statt. Ebenso wird eine enge Verknüpfung zu anderen Werkzeugen wie Instruments oder dem Interface Builder hergestellt. Die gesamte Projektverwaltung übernimmt Xcode. Die Vorgehensweise in Xcode wird in diesem Kapitel und den folgenden Workshops noch besprochen. Xcode ist also ein zentrales Entwicklertool mit einer integrierten Entwicklerumgebung. Im Englischen heißt das dann »Integrated Development Enviroment« – kurz IDE.

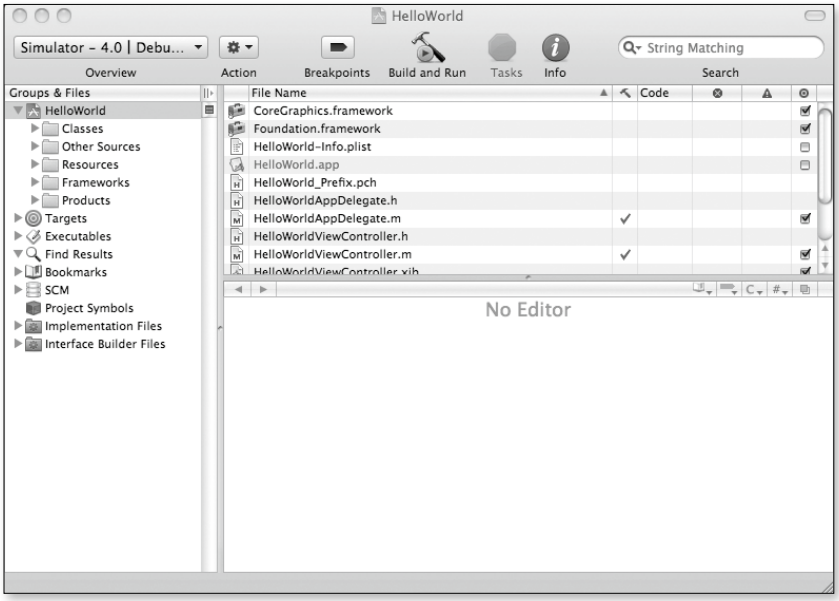
Damit Sie die IDE besser kennenlernen, beginnt nun der erste kleine Workshop:

1. Starten Sie unter dem Pfad /Developer/Applications Xcode. Nachdem Sie das Programm aufgerufen haben, startet diese Eingabemaske.

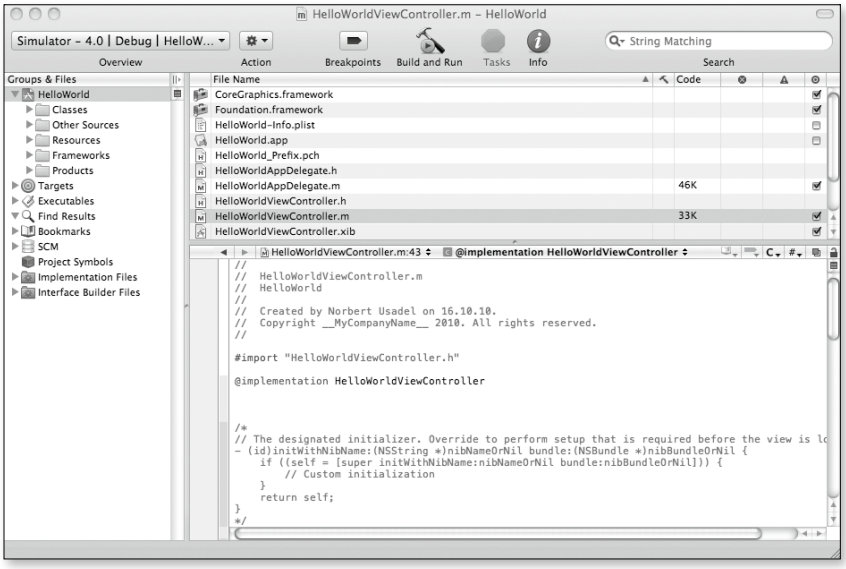
2. Wählen Sie *Create a new Xcode project*. Danach erscheint das Eingabefenster, das Sie unten abgebildet finden. Hier können Sie sich zwischen verschiedenen Projektvorlagen entscheiden. Wählen Sie nun *View-based Application* aus.



3. Vergeben Sie nun einen neuen Projektnamen. Nennen Sie Ihr erstes Projekt »HelloWorld« und bestätigen Sie Ihre Eingabe. Danach erscheint die Arbeitsumgebung von Xcode mit den angelegten Dateien für Ihr Projekt. Es wurden auch zwei Klassen angelegt. Eine davon wird nun bearbeitet.



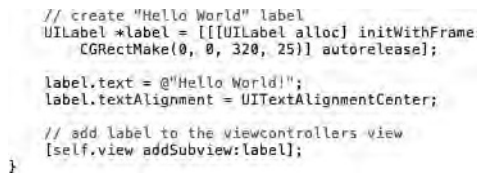
4. Klicken Sie auf die Datei `HelloWorldViewContoller.m`. Sie sehen, dass im Editorfenster der Quelltext angezeigt wird.



5. Geben Sie nun die folgenden Zeilen Code ein:

```
UILabel *label = [[[UILabel alloc] initWithFrame:
    CGRectMake(0, 0, 320, 25)] autorelease];

label.text = @"Hello World!";
label.textAlignment = UITextAlignmentCenter;

// add label to the viewcontrollers view
[self.view addSubview:label];
```

Sie sehen unter Punkt 6, in welchem Programmteil die Codezeilen eingesetzt werden.



```
    // create "Hello World" label
    UILabel *label = [[[UILabel alloc] initWithFrame:
        CGRectMake(0, 0, 320, 25)] autorelease];

    label.text = @"Hello World!";
    label.textAlignment = UITextAlignmentCenter;

    // add label to the viewcontrollers view
    [self.view addSubview:label];
}
```

AUFGEPASST

Sie müssen den Code genau so eingeben, wie er da steht. Mit geschweiften und eckigen Klammern. Die eckigen Klammern erzeugen Sie, indem Sie ⌥ + [5] und ⌥ + [6] auf Ihrer Tastatur drücken. Die geschweiften Klammern liegen fast direkt daneben. Sie werden mit ⌥ +[8] und ⌥ + [9] erzeugt.

6. Der eingegebene Code der Datei sieht dann nach Ihrer Eingabe im Editor so aus:

```
#import "HelloWorldViewController.h"

@implementation HelloWorldViewController


/*
// The designated initializer. Override to perform setup that is
required before the view is loaded.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil {
 if ((self = [super initWithNibName:nibNameOrNil bundle:nib-
 BundleOrNil])) {
  // Custom initialization
 }
```

```
 return self;
}
*/

/*
// Implement loadView to create a view hierarchy programmatical-
ly, without using a nib.
- (void)loadView {
}
*/


// Implement viewDidLoad to do additional setup after loading the
view, typically from a nib.
- (void)viewDidLoad {
 [super viewDidLoad];

 // create "Hello World" label
 UILabel *label = [[[UILabel alloc] initWithFrame:
  CGRectMake(0, 0, 320, 25)] autorelease];

 label.text = @"Hello World!";
 label.textAlignment = UITextAlignmentCenter;

 // add label to the viewcontrollers view
 [self.view addSubview:label];
}


/*
// Override to allow orientations other than the default portrait
orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrien-
tation)interfaceOrientation {
 // Return YES for supported orientations
 return (interfaceOrientation == UIInterfaceOrientationPortrait);
}
*/

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
 [super didReceiveMemoryWarning];
```

```
    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
    // Release any retained subviews of the Main-View.
    // e.g. self.myOutlet = nil;
}


- (void)dealloc {
 [super dealloc];
}

@end
```

Hier können Sie auch überprüfen, wo Sie die paar Zeilen Code einsetzen und ob Sie alles richtig eingegeben haben. Ihr erstes Projekt ist nun fertig. Um zu sehen, ob Ihr Programm auch läuft, starten Sie es in der Xcode-Toolbar. Klicken Sie hierzu auf das Play-Symbol mit dem Hammer *Build and Run*.

Die Xcode-Toolbar mit dem zentralen Symbol Build and Run

Ist der Programmcode fehlerfrei, startet automatisch der iPhone-Simulator, und Sie können dort Ihre erste App betrachten. Wenn Ihre App fehlerfrei läuft und das »Hello World« im Simulator angezeigt wird, können Sie Ihr erstes Projekt abspeichern.

Der iPhone-Simulator in Aktion

Auf Ihrer Festplatte wird das ganze Projekt so abgespeichert:

Die Ordnerstruktur Ihres HelloWorld-Projektes

Sie sehen hier die Struktur Ihres Projektes. Die gleichen Dateien werden auch in Xcode angezeigt. Sie erhalten damit die Übersicht, welche Datei welche Funktion hat.

Dateien mit der Endung pch:

Die Precompiled Header-Datei ist eine vorübersetzte oder vorkompilierte Datei. Bei großen Projekten wird damit die Übersetzungszeit des Compilers Xcode beträchtlich verringert. Sie enthält Programmanweisungen, die in allen anderen Quelldateien im Programm verfügbar sein sollen.

Dateien mit der Endung plist:

Die Property-list-Datei ist eine Datei, die bestimmte Eigenschaften speichert und diese anderen Modulen zur Verfügung stellt. In Xcode hält diese Parameterdatei die Verbindung zwischen dem Betriebssystem und dem Endgerät, um den Start des Endgerätes zur ermöglichen.

Dateien mit der Endung xcodeproj:

Wenn Sie auf diese Datei klicken, starten Sie das Xcode-Projekt, das Sie unter dem Namen der Datei angelegt haben. In diesem Fall würde das HelloWorld-Projekt in Xcode gestartet.

Dateien mit der Endung .xib:

Diese Dateien werden vom Interface Builder gebildet. Der Interface Builder ist ein Bestandteil des iPhone-SDKs. Die Datei enthält Objektbeschreibungen der Benutzerschnittstelle. Im Interface Builder werden Sie lernen, wie man objektorientiert programmiert. 95 Prozent der Arbeit eines App-Programmierers finden in Xcode und dem Interface Builder statt. Beide Programme sind also eng miteinander verzahnt. Im Interface Builder legen Sie die gesamte grafische Benutzeroberfläche Ihrer Projekte fest. Das geschieht dort ohne Quellcodeeingabe, eben objektorientiert. Wie man den Interface Builder nutzt, wird in diesem Kapitel noch weiter beschrieben.

Dateien mit der Endung .m:
Diese Dateien sind reine Objective-C-Dateien. Sie werden in der Regel nicht verändert. Sie werden vom System aufgerufen, um Ihre App zu starten.

Dateien mit der Endung .h:
Das ist eine Header-Datei. Sie hält optionale Methoden bereit, die zu gegebener Zeit aufgerufen werden. Diese Datei befindet sich im Ordner *Classes*.

iPhone-Simulator

Den iPhone-Simulator haben Sie im vorherigen Kapitel schon kennengelernt. Starten Sie Ihr HelloWorld-Projekt neu. In der Xcode-Umgebung benutzen Sie den Play-Button mit dem Hammer *Build und Run*. Ist Ihr Projekt fehlerfrei, wird automatisch der iPhone-Simulator gestartet.

Ihr erstes Projekt im iPhone-Simulator

Hier sehen Sie Ihr Projekt in einem iPhone-4-Simulator. Den Simulator können Sie im Menü *Hardware* unter dem Menüpunkt *Gerät* auf das iPad oder das iPhone 3 umschalten. Die Bedienung erfolgt mit der Maus. Kompliziertere Gesten werden durch das Klicken der Maustaste ausgeführt. Drückt man die ⌥ dabei, werden Gesten mit zwei Fingern simuliert. Wenn Sie Ihre App auf Herz und Nieren testen wollen, brauchen Sie jedoch ein richtiges Endgerät.

Folgende Einschränkungen gibt es bezüglich des Simulators auf Endgeräte zu verzeichnen:

- keine Telefonfunktionen. Es kann nicht telefoniert werden. SMS können nicht versendet werden.

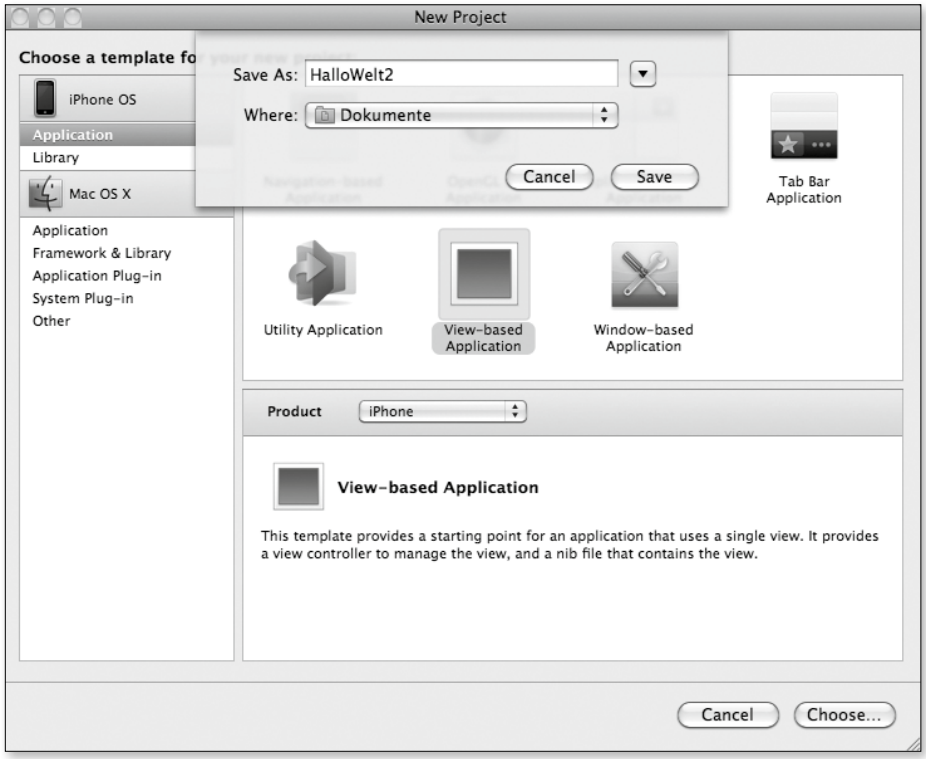
- keine GPS-Funktion. Der Simulator kann nicht geortet werden, da das GPS-Modul fehlt.
- kein Beschleunigungssensor
- keine Kameraunterstützung
- kein Gyroskop-Support
- keine Mehrfingergesten
- kein 3D-Rendering mit OpenGL
- keine Installation von Apps aus dem Store

Interface Builder

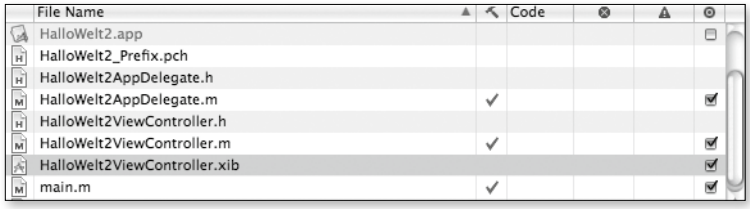
Der Interface Builder dient dazu, grafische Benutzeroberflächen zu schaffen. Dies erspart es Ihnen, sehr viel Code in Xcode einzugeben. Durch einfaches Drag & Drop können Sie im Interface Builder Ihre App grafisch gestalten. Dort werden beispielsweise Controls (Buttons und Textfelder) und Views auf einer Designeroberfläche zusammengestellt und anschließend per Property-Inspectoren weiter konfiguriert und in Xcode zu Quellcode umgewandelt. Property- Inspectoren finden Sie im Interface Builder in den Inspector-Fenstern. Mit ihnen können Sie den einzelnen grafischen Elementen Attribute zuordnen, das können farbliche Akzente, der Inhalt von Buttons oder die Schriftart einer ganzen Oberfläche sein.

Der Interface Builder ist mit Xcode verzahnt. Aus Xcode heraus können Sie den Interface Builder starten, die Oberflächen bearbeiten und zurück zu Xcode springen, um dann Ihr ganzes Projekt zu kompilieren. Sie werden in diesem Kapitel ein neues HelloWorld-Projekt kennenlernen, in dem Sie mit einer Zeile Quelltext auskommen.

1. Starten Sie Xcode mit einem Doppelklick. Wählen Sie wieder *View-based Application*. Benennen Sie das neue Projekt als »HalloWelt2«.

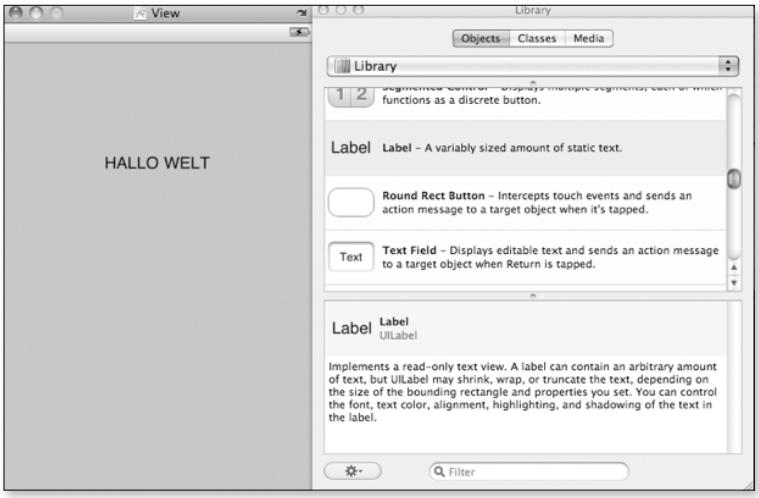


2. Die neue Projektstruktur wurde in Xcode angelegt. Suchen Sie die Datei HalloWelt2viewController.xib und starten Sie diese mit einem Doppelklick. Diese Datei ist mit dem Interface Builder verbunden und öffnet ihn.



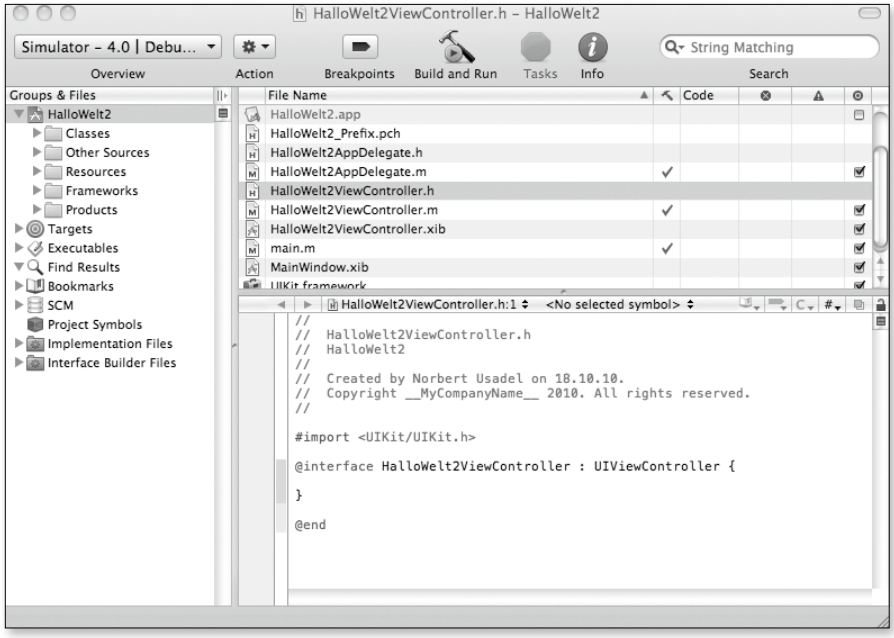
3. Der Interface Builder startet, und Sie sehen vier Fenster: View Attributes, View, Libary und ein Fenster mit dem Namen HalloWelt2ViewController.xib. Sie bemerken daran, dass der Interface Builder mit Xcode verknüpft ist, denn die Datei HalloWelt2ViewController.xib wurde ja in Xcode ursprünglich angelegt. Sie benutzen nun die *Library* und das Fenster *View*.

4. Klicken Sie nun in das Fenster *Library* und suchen Sie den Punkt *Label* aus. In ein Label können Sie einen statischen Text einfügen. Es ist so, als würden Sie ein Etikett beschriften. Greifen Sie sich das Label und ziehen Sie es via Drag & Drop aus das Feld *View*. Das Feld ist noch leer.

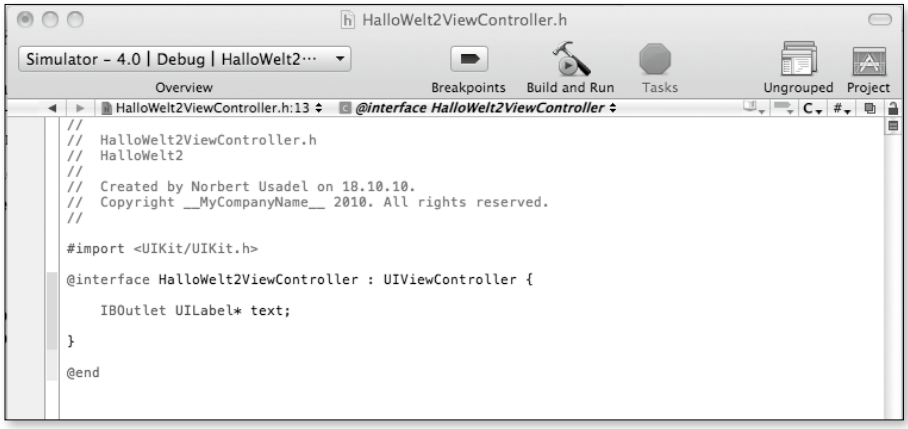


5. Um das Label mit dem entsprechenden Inhalt zu füllen, klicken Sie in das Fenster *Label Attributes*. Dort befindet sich ein Feld, in dem Sie den statischen Text »HALLO WELT« eingeben können. Wenn die Eingabe erfolgreich war, sehen Sie im Fenster *View* das Ergebnis.

6. Nun ist Ihre Arbeit im Interface Builder erledigt. Speichern Sie das Projekt ab und springen Sie danach in Xcode zurück. Öffnen Sie per Doppelklick die Header-Datei *HalloWelt2ViewController.h*. Die Datei wird im Editor geöffnet, und Sie können nun gleich den Code eingeben.



7. Nun geben Sie genau eine Zeile Quellcode ein. Sie definieren dort das Feld, das Sie vorhin in den IB eingebaut haben. Die Zeile lautet IBOutlet UILabel * text; danach speichern Sie die Datei ab. Starten Sie nun das Programm in der Toolbar mit dem *Build and Run*.



8. Wenn Sie alles richtig gemacht haben, startet der Simulator und Sie sehen Ihr zweites »Hallo Welt«.

TIPP

Sie können jetzt jederzeit das Projekt in Xcode starten, zurück in den IB springen und das erstellte Label verändern und auf der Fläche hin und her ziehen. Probieren Sie es einmal aus. Wenn Sie eine Veränderung vorgenommen haben, speichern Sie das Ganze ab, springen in Xcode und kompilieren neu. Das macht Spaß, und Sie verinnerlichen gleichzeitig die Arbeitsabläufe zwischen Xcode und dem Interface Builder. Unter »kompilieren« versteht man, dass Ihr eingegebener Quellcode in eine ausführbare App umgewandelt wird. Die Veränderungen, die Sie im Quellcode vornehmen, werden dann auf dem Simulator umgesetzt, da er die ausführbare App startet.

Die vier Fenster des Interface Builders

Die XIB-Datei des Interface Builders

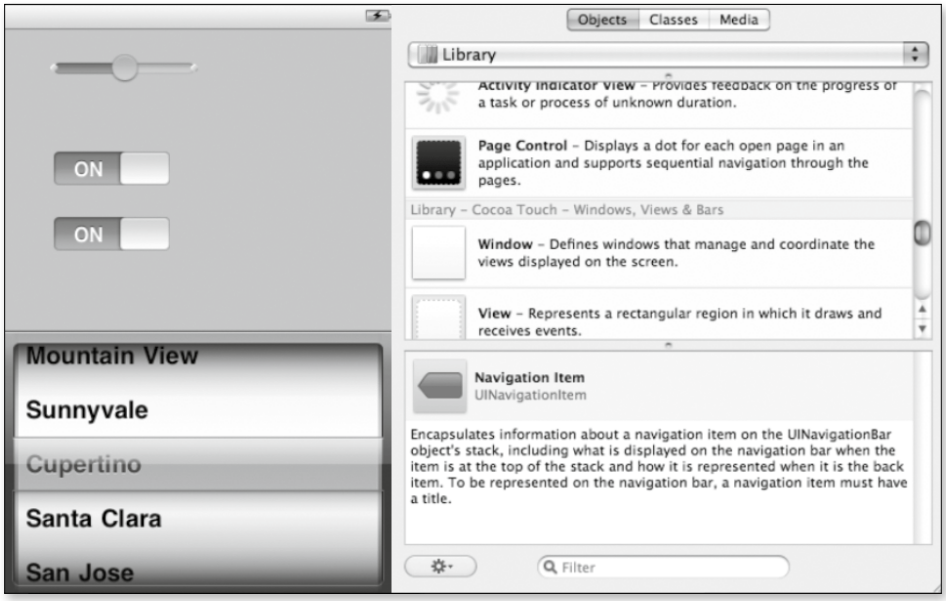
Die XIB Datei:

Die XIB-Datei ist eine Schnittstellendatei, die im Interface Builder drei Elemente beinhaltet.

Files Owner, First Responder und View. Die ersten beiden Dateien sind Platzhalter für Objektverknüpfungen. Files Owner stellt die Verknüpfung zu einer Klasse mit äußeren Objekten her. First Responder steht für das Objekt, welches gerade in der Hauptschleife des Programms abgearbeitet wird. Beide Dateien sind sogenannte Nib-Files. In ihnen lassen sich Objekte, Proxy-Objekte und Connections speichern. Kurzum bezieht sich das auf die Objekte, die in der Bibliothek aufgerufen werden. Grafisch lassen sich diese Objekte durch Beziehungen im IB verknüpfen.

View ist eine Datei, die zum Interface Builder gehört. Sie wird als echtes Objekt definiert. Klickt man auf View, öffnet sich der View, und Sie können dort die grafische Benutzeroberfläche Ihres Projektes gestalten. Im Bild sehen Sie das Fenster *View*,

das mit Objekten aus der Bibliothek gefüllt wurde. Button, Schaltflächen und Picker lassen sich via Drag & Drop auf das Fenster *View* bewegen und sich dort positionieren. Dabei wird von View Hilfe per magnetischer Linien und Gestaltungsvorschläge angeboten.



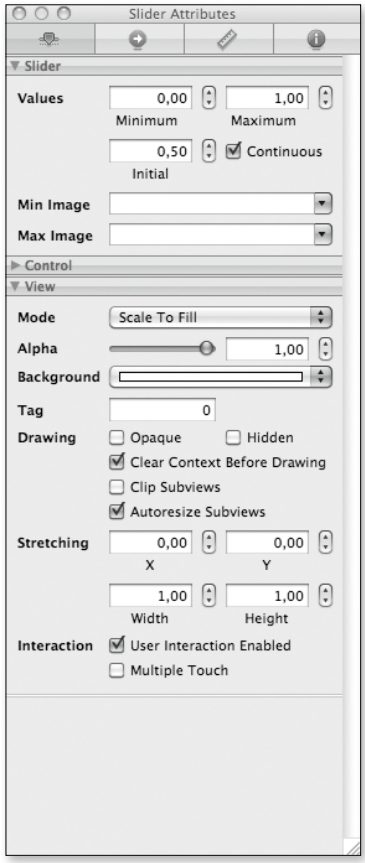
Die Library korrespondiert mit der Gestaltungsfläche View.

Die Bibliothek

Die Bibliothek dient zur Medien- und Objektauswahl. Dort treffen Sie eine Auswahl an Objekten und Klassen. Es werden die grafischen Ressourcen Ihres Projektes angezeigt, und Sie erhalten beispielsweise eine Beschreibung über das ausgewählte Objekt. Die Library ist in Objekte, Klassen und Media untergliedert. Per einfachem Drag & Drop lassen sich diese auf die View-Umgebung ziehen.

Der Inspector

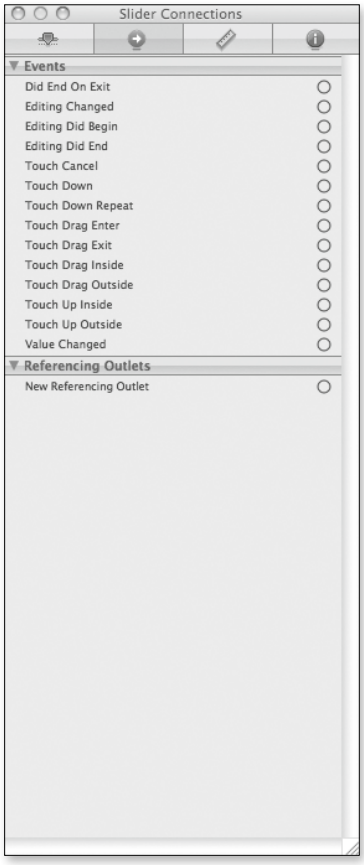
Mit Hilfe des Inspectors lassen sich die Objekte im View-Bereich weiter formatieren. Der Inspector ist in vier Bereiche unterteilt, die durch die obere Symbolleiste im Fenster angesteuert werden können.



Der Attribute-Inspector

Attribute-Inspector

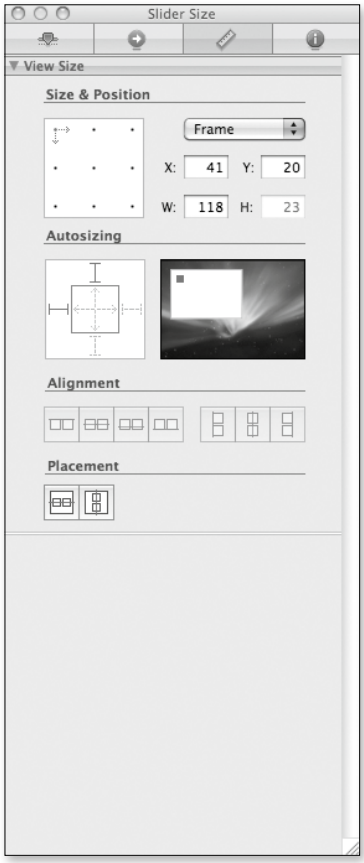
Der Attribute-Inspector steuert die Layout-Attribute eines Objektes. So lassen sich dort Schriften auswählen, oder man kann den Objekten eine bestimmte Farbauswahl zuordnen.



Der Connection-Inspector

Connection-Inspector

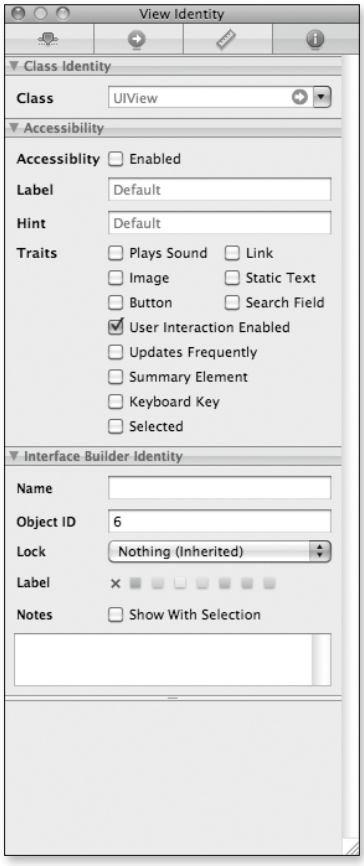
Der Connection-Inspector konfiguriert die Verbindungen der Objekte zueinander und im ganzen Projekt.



Der Size-Inspector

Size-Inspector

Der Size-Inspector steuert die Positionierung der einzelnen Objekte. Es lassen sich dort die Größenverhältnisse anpassen und neu berechnen.



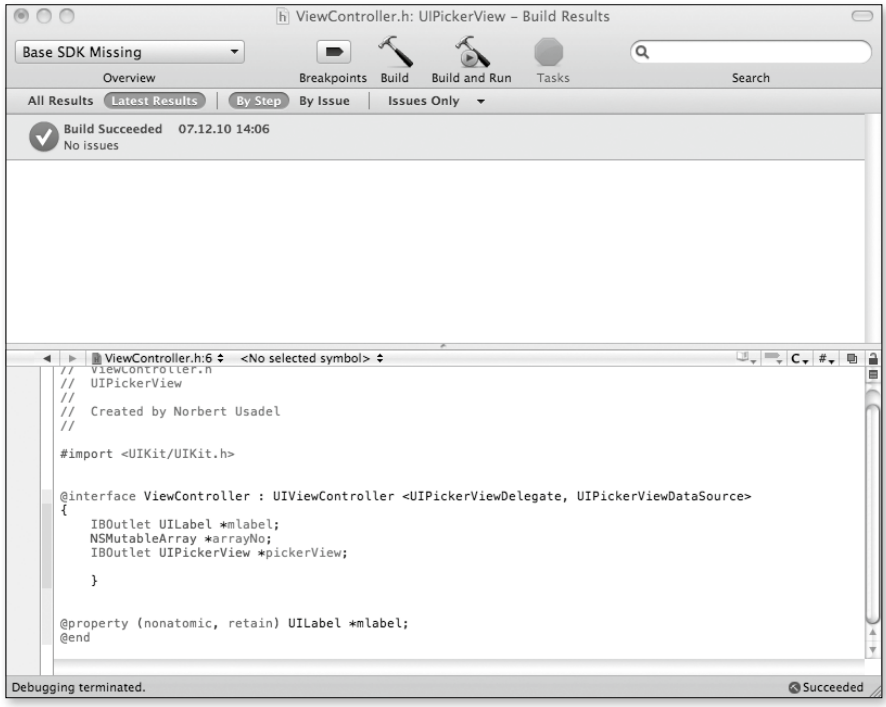
Der Identity-Inspector

Identity-Inspector

Der Identity-Inspector listet Metainformationen und Einstellungen zur Klassenzugehörigkeit eines Objektes auf und stellt sie in einer Ansicht zusammen.

Debugger

Den Debugger dürften Sie schon kennengelernt haben. Er überprüft den Quellcode auf Fehler. Klicken Sie in Xcode auf das Symbol *Build and Run*, so wird automatisch der Debugger gestartet. Ist alles in Ordnung, wird das Programm für den Simulator freigegeben.

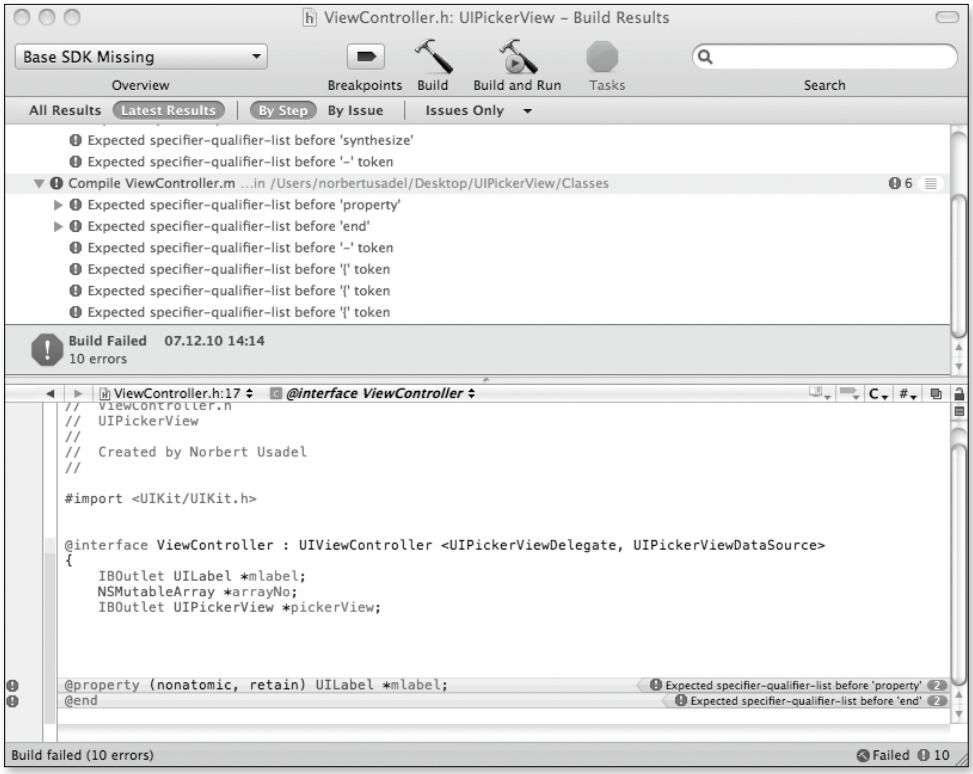


Der Debugger meldet: Build Succeeded. Es ist alles in Ordnung.

Stimmt etwas nicht, setzt der Debugger einen Breakpoint. Dabei werden Kommentare sichtbar, die Auskunft darüber geben, was falsch sein könnte. Es kann sein, dass Sie nur eine geschweifte Klammer vergessen haben. Dies wird aber sofort vom Debugger erkannt, und es folgt ein Hinweis auf die fehlende Klammer. Im folgenden Code habe ich die untere geschweifte Klammer gelöscht.

```
{
 IBOutlet UILabel *mlabel;
   NSMutableArray *arrayNo;
   IBOutlet UIPickerView *pickerView;

   }// Diese Klammer löschen.
```

Das führt zu dem Ergebnis im unteren Fenster. Der Debugger kreidet dann eine Vielzahl von Fehlern an. Setzt man die Klammer wieder an die richtige Stelle, ist alles in Ordnung. Der Quelltext wird kompiliert und zu einer ausführbaren App umgewandelt.



Die Fehleranzeige des Debuggers: Die fehlende Klammer erzeugt im Debuggerfenster eine Anzahl von Fehlern.

Im Verlauf des Buches wird immer wieder auf den Debugger eingegangen.

Instruments

Instruments ist Bestandteil des SDKs und prüft, nachdem Sie Ihre App entwickelt haben, das Programm auf sein Laufzeitverhalten. Es nimmt Festplattenzugriffe, Speicherlecks, den Speicherverbrauch im Hauptspeicher und die CPU-Aktivität unter die Lupe. Die beste App taugt eben nichts, wenn sie langsam ist oder das iPhone zum Absturz zwingt.

Das Hilfsprogramm Instruments

Leider muss sich der App-Entwickler selber darum kümmern, wie sein Programm Hauptspeicher verwaltet. Intruments ist dabei eine wichtige Hilfe, weil es vom Start an jedes einzelne Objekt unter die Lupe nehmen kann. Speicherlecks werden damit aufgedeckt. Das Programm kann von Xcode aus über die Option *Run / Run with Performance Tools / Leaks* aufgerufen werden.

Dashcode

Dashcode ist eigentlich dazu da, Widgets oder Webanwendungen zu entwickeln. Es kann sein, dass Sie für Ihre App im Web genau das gleiche Outfit, wie bei der nativen Programmierung gewählt haben. Sie können somit Teile in das Web stellen, die Ihre App erst im Nachhinein lädt. Oder es kann sein, dass es so aussieht, als ob Sie auf die App im iPhone zugriffen, es aber in Wirklichkeit im Web tun.

Das Programm Dashcode macht den Webauftritt einfach.

Manchmal ist es sinnvoll, nur eine Web-App zu programmieren, weil der Aufwand zu groß ist, eine native App zu entwickeln. Es gibt mitlerweile zahlreiche Web-Apps, die durch den Browser der App aufgerufen werden. Diese Apps brauchen nicht mit Xcode programmiert zu werden. Die Herstellung solcher Apps kann man in Dashcode vornehmen. Der Aufwand ist wesentlich geringer, da viel Programmierarbeit entfällt. Dadurch bekommen diese Web-Apps aber trotzdem das typische Aussehen von iPhone-Apps, die mit Xcode gestaltet wurden. Wenn das der Fall sein sollte, benutzen Sie Dashcode.

Weitere Tools im Überblick

Das SDK hält noch weitere Tools für Sie bereit, es ist also recht umfangreich. Im unteren Fenster haben Sie sie alle im Überblick. Die Tools sind auf Ihrer Festplatte im *Utillities*-Ordner des SDKs installiert. Mit ihnen können Sie Java Applet außerhalb des Browsers betrachten, Icons entwickeln, USB-Schnittstellen überprüfen, Programme komprimieren und Scripte per Python erstellen.

Weitere kleine Utillities des iPhone-SDKs

Kapitel

4

iOS – Das iPhone-Betriebssystem

In diesem Kapitel werden der Aufbau und die Architektur des iOS dargestellt. Es dient dazu, Objektive-C besser zu verstehen und bestimmte Bestandteile des Betriebsystems einordnen zu können.

Das neue iOS 4.3.2

Die einzelnen Bestandteile des iOS 4.3

Die Übersicht über das iPhone-Betriebssystem bekommt man unter diesem Link: http://www.apple.com/de/ios/. Die Aktualisierung erfolgt über iTunes.

Die wichtigsten Funktionen des iOS sind:

1. Multitasking: Mehrere Apps laufen gleichzeitig
2. Ordnerverwaltung für Apps: schafft Ordnung auf dem Desktop
3. AirPrint: druckt direkt auf AirPrint-fähigen Druckern
4. AirPlay: streamt Daten wie Musik oder Photos auf entsprechende Geräte
5. Suchfunktion, um das iPhone/iPad zu finden
6. Game Center: eröffnet die Welt des Spiele-Netzwerks
7. E-Mail-Funktion: ein Postfach für alle Mail-Accounts und die Verwaltung von Nachrichten in Threads

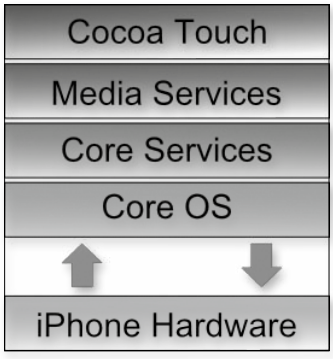
Mit iOS 4.3 führte Apple eine Hotspot-Funktion ein. Diese ermöglicht es, den Internetzugang des iPhone per WLAN anderen Geräten zur Verfügung zu stellen.

Die mitgelieferten Bestandteile des iOS 4.3

Das Schichtenmodel

Als Einsteiger bewegen Sie sich in den ersten zwei bis drei Schichten des hier beschriebenen Modells. Dort lernen Sie, wie man in Cocoa Touch mit Objective-C programmiert.

Das iOS unterscheidet sich im Grundaufbau nur unwesentlich von Mac OS X für Desktop-Computer. Die größten Unterschiede bestehen im User-Interface, bedingt durch die Einführung des Multitouch-Screens bei iPhone, iPod touch und iPad.



Das Vier-Schichten-Modell des iOS

Das iOS ist in vier Schichten aufgebaut:

- Cocoa Touch (NSFoundation, UIKit)
- Media (Quartz, Core Animation, Open GL, Audio, ...)
- Core Services (Core Location, Adress Book, SQLite, CFNetwork, ...)
- Core OS (I/O Threads, Sockets, Bonjour, Key Chain, ...)

Je tiefer man in diesem Modell nach unten stößt, desto weiter ist man in den Eingeweiden des Betriebssystems verfangen. Alle Schichten sind aber für den Entwickler zugänglich. Um mit Komfort programmieren zu können, gibt es Frameworks, die bestimmte Routinen schon in sich tragen. Im weiteren Verlauf erhalten Sie eine Übersicht über die Schichten.

Cocoa Touch

Cocoa ist die wichtigste Schicht für Entwickler. In dieser Schicht wird am meisten programmiert. Mit dieser Schicht haben Sie in den beiden Hallo-Welt-Beispielen schon Bekanntschaft gemacht. Sie haben dort Felder im User-Interface-Framework (UI) definiert.

Es gibt Bereiche, die auch »Frameworks« genannt werden. Der erste Strang in diesem Framework ist dafür zuständig, alles zu regeln, was im Hintergrund einer App abläuft. Das können zum Beispiel die Datenspeicherung und das Dateimanagement sein. Diese Funktionen werden unter dem Namen NSFoundation-Kit zusammengefasst.

Der andere Strang an Klassen ist das AppKit. Das AppKit fasst alles zusammen, was mit dem Benutzerinterface zu tun hat. Sie erinnern sich bestimmt an den Interface Builder, den man von Xcode aufruft. Dort werden die Methoden und Klassen des AppKit aufgerufen.

Das AppKit nennt man auch UIKit, es wurde für das iPhone/iPad neu gebündelt, da die Eingabe der Daten über ein Touchscreen stattfindet. Es können so, speziell für das iPhone, folgende Aspekte berücksichtigt werden: Multi-Touch-Verhalten, Screen-Rotation, View Controller, Events und Gesten.

In der Cocoa-Schicht können Sie aber auch noch andere Frameworks nutzen:

- Apple Push Notification Service

- AdresssBook UI Framework
- In App E-Mail
- Map Kit Framework
- Peer-to-Peer-Support

Media

Die Media-Schicht oder auch der Media Layer verwaltet Grafik-, Audio,- und Videotechnologien. Die Bestandteile sind:

- Quartz, eine Bibliothek für vektorisierbares Zeichnen
- Core Animaton, die zur Umsetzung visueller Effekte dient
- OpenGL ES, dient zur Herstellung schneller 2-D- und 3-D-Grafiken
- AV Foundation: Dieses Framework steht für das Aufnehmen und Abspielen von Audio-Inhalten
- Core Audio: dient ebenfalls zum Bearbeiten von Audiodaten, bietet aber mehr Möglichkeiten als die AV Foundation
- Open AL: dient als Audio Bibliothek für Spiele.
- Video: dient zum Abspielen von Videos in verschieden Formaten

Core Services

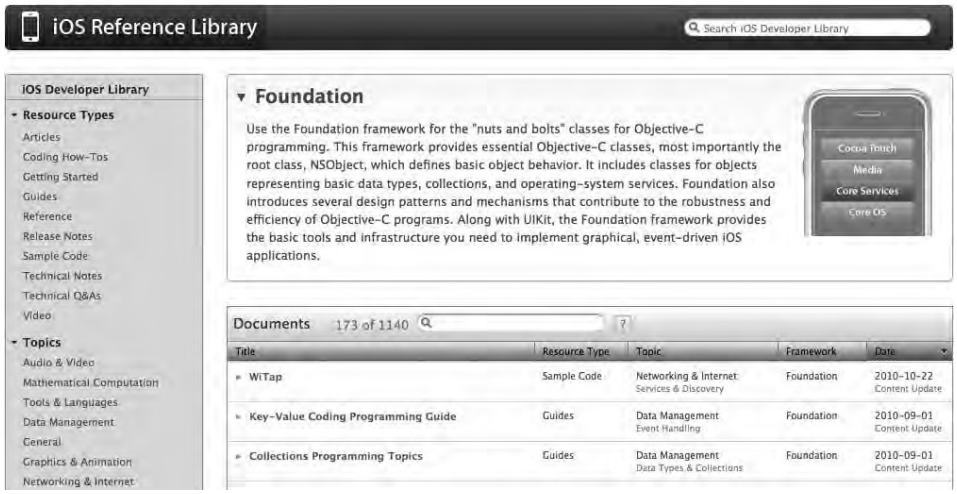
Diese Schicht dient zum Verwalten und Speichern von Daten. Es können hier Datenbanken angelegt werden. Es ist aber auch möglich, mit dem Core Location Framework Positionsbestimmungen per WLAN, Mobilfunk oder GPS durchzuführen und umzusetzen.

Core OS

In der untersten Schicht des iOS-Betriebsystems können Hardwarekomponenten direkt angesprochen werden, es können Netzwerkzugriffe programmiert werden. Sie können aber auch gerne ganz ans Eingemachte gehen und direkt auf den Kernel des Unix-Betriebsystems zugreifen.

iOS Developer Library

Das iOS 4 ist natürlich ein komplexes Thema. Mit all seinen Schichten, Frameworks und Klassen erhalten Sie unter http://developer.apple.com/library/ios/navigation/index.html den kompletten Überblick über wirklich alles, was mit dem mobilen Betriebssystem von Apple zu tun hat.



Hier sehen Sie ein Kapitel der iOS Reference Library.

Die iOS Reference Library ist ganz praktisch, wenn Sie eben mal etwas nachschlagen wollen. Mit ihren über 1200 Dokumenten ist sie recht einzigartig, was die Informationsfülle und die Qualität der Texte anbelangt. Das Ganze gibt es aber leider nur in Englisch. Die vier Schichten des Betriebssystems werden dort erklärt. Durch Querverweise erhalten Sie dort ein umfassendes Bild über die Architektur des Betriebssystems. Sie bekommen dort jede Menge Beispiele und Erklärungen geliefert. Die einzelnen Entwicklertools zum iOS 4 werden ebenso vorgestellt wie Code-Beispiele zu den einzelnen Frameworks. Die Hilfe von Xcode greift auf die Bibliothek zu. So werden Sie, wenn Sie Ihre Projekte realisieren, immer wieder auf die Library zugreifen.

Kapitel

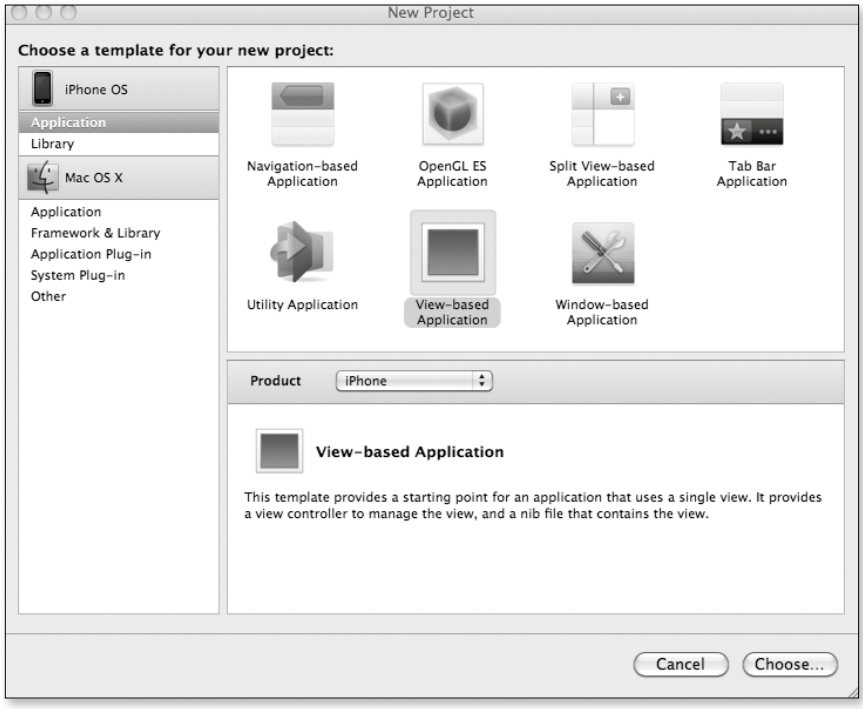
5

Objective-C: der Kurzeinstieg

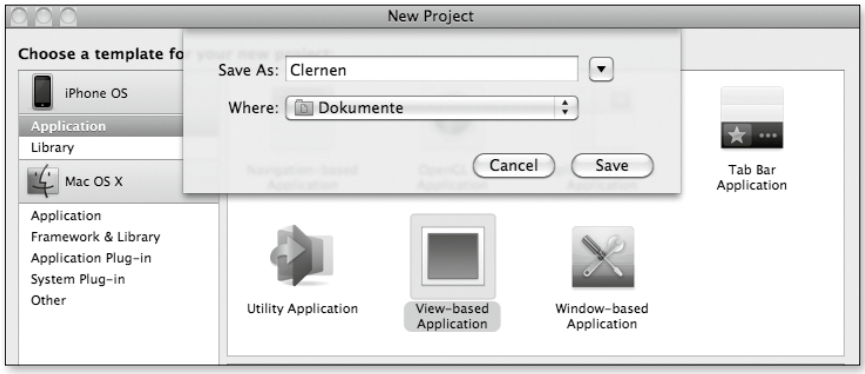
Da dies ja ein praktisch orientiertes Buch ist, werden Sie auch unter Zuhilfenahme von Xcode und dem Debbuger Objective-C lernen. Sie können in Xcode den Quelltext eingeben und im Debugger die Ausgabe, die der Quellcode verursacht, überprüfen.

Objective-C: der Kurzeinstieg

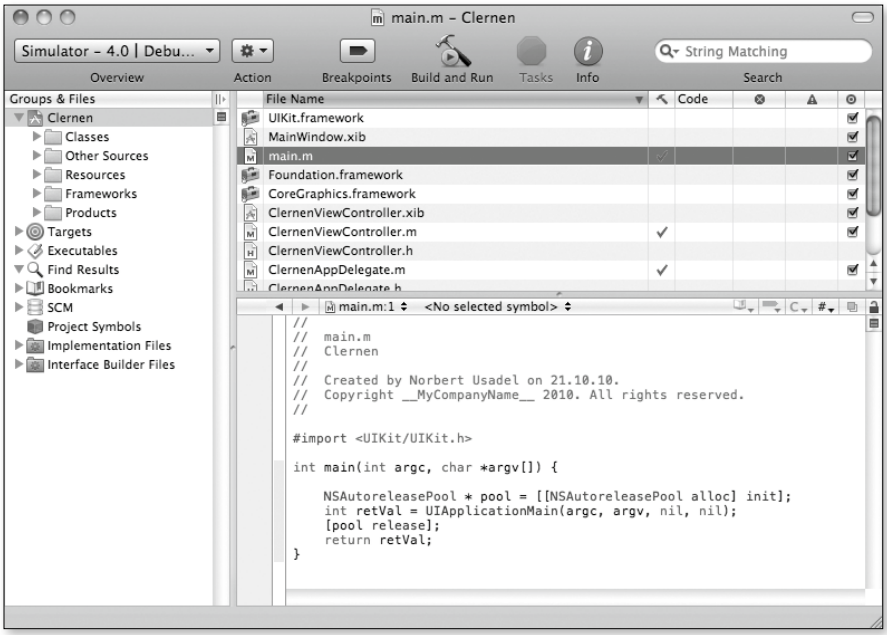
1. Starten Sie wie gewohnt Xcode mit einem Doppelklick und wählen Sie *Create a new Projekt* aus. Im nächsten Fenster wählen Sie *View based Application* aus.



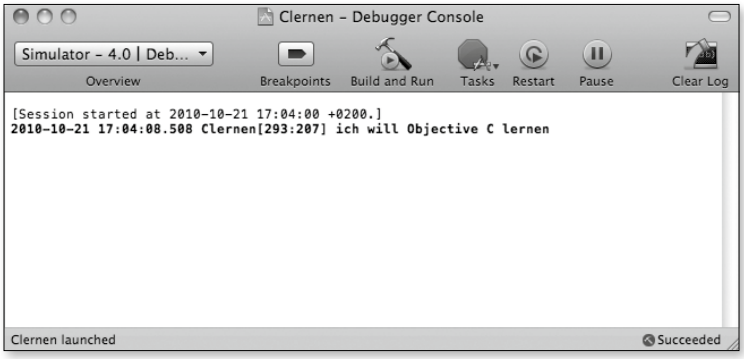
2. In diesem Schritt vergeben Sie einen Namen für Ihr Projekt. Ich habe hier den Namen *Clernen* vergeben. Danach speichern Sie das neue Projekt ab, indem Sie auf den *Save*-Button klicken.



3. Die Arbeitsumgebung von Xcode öffnet sich mit dem neu angelegten Projekt. Die Datei *main.m* wird für die Code-Eingabe benötigt. Wenn Sie diese Datei anwählen, erscheint der Quelltext unten im Editorfenster. In der Datei *main.m* geben Sie in Zukunft die Codebeispiele ein, die in diesem Kapitel angeboten werden.



4. Der nächste Schritt besteht darin, das Consolenfenster des Debuggers zu öffnen. In ihm werden Programmfehler angezeigt. Hier wird es aber auch genutzt, um eine Bildschirmausgabe des Programms zu erzeugen. Um dieses Fenster zu öffnen, wählen Sie im Menü *Run* in Xcode den Menüpunkt *Console* an.



Das Fenster der Debugger-Console

5. Springen Sie nun wieder zu der Datei *main.m* in Xcode zurück. Geben Sie die folgende Zeile Code ein:

   ```
   NSLog (@" ich will Objective-C lernen");
   ```

   Sie platzieren die Zeile Code unter dem Ausdruck:

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

   Achten Sie darauf, dass Sie die Zeile auch genau so schreiben, wie sie da steht.

6. Danach starten Sie in der Toolbar von Xcode den *Build-and-Run*-Prozess. Sie sollten nun im Consolenfenster des Debuggers die Ausgabe: "ich will Objective-C lernen" sehen.

Auf diese Weise können Sie nachvollziehen, welche Codezeilen zu welcher Ausgabe führen.

Nach diesen praktischen Vorarbeiten folgt nun die Theorie.

Variablen

Variablen sind dazu da, Speicher zu reservieren, in dem sich Daten abspeichern lassen. Dabei gibt es verschiedene Arten von Variablen, die Sie unter C anmelden müssen. In machen Programmiersprachen entfällt diese Anmeldung von Variablen und wird vom Compiler dynamisch vergeben, so dass der Programmierer sich gar nicht überlegen muss, welche Variablen er braucht. In C müssen Sie sich allerdings vorher überlegen, welche Variablen Sie für Ihr Programm benötigen. Im Nachfolgenden finden Sie die Übersicht der Datentypen, die Sie einsetzen können:

Übersicht der Variablen

Name	Beschreibung/Wertebereich	Größe
Bool	NO oder YES	1 Byte
char	-128 bis 127 Buchstaben/Sonderzeichen	1 Byte
unsigned char	0 bis 255 Buchstaben/Sonderzeichen	1 Byte
short	- 32768 bis 32767 Ganzzahlen	2 Byte
unsigned short	0 bis 65538 Ganzzahlen	2 Byte
int	-2147483648 bis 2147483647 Ganzzahlen	4 Byte
unsigned int	0 bis 4294967295 Ganzzahlen	4 Byte

long int	-2147483648 bis 2147483647 Ganzzahlen	4 Byte
unsigned long	0 bis 4294967295 Ganzzahlen	4 Byte
long long int	-9223372036854775808 bis 9223372036854775807	8 Byte
unsigned long long	0 bis 18446744073709551615 Ganzzahlen	8 Byte
float	1.2E-38 bis 3.4E + 38 Fließkommazahlen	4 Byte
	Genauigkeit 6 Stellen	
double	2.3E-308 bis 1.7E+308, Genauigkeit 15 Stellen	8 Byte
long double	16 Byte Gleitkommazahl	16 Byte

Variablen anlegen

Eine Variable wird so deklariert:

```
datentype variablenname = wert;
```

Im Code sieht das dann so aus:

```
int zahl = 1848
char buchstabe = ' V '
float gleitkommazahl = 110.78
```

So werden die Variablen definiert und ihnen gleichzeitig ein Wert zugewiesen. Überprüfen Sie das einmal in Xcode und geben Sie, wie oben beschrieben, diese Zeilen Code ein:

```
int main(int argc, char *argv[]){
    int ganzeZahl = 1848;
    float pi=3.1415926;
    NSLog (@"hallo,ganzeZahl=%i pi = %f", ganzeZahl,pi);
```

Hier werden die Variablen ganzeZahl und pi definiert.

Kompilieren Sie das ganze mit *Built and Run* und sehen Sie, was im Debugger-Fenster ausgegeben wird:

```
[Session started at 2010-10-21 17:21:13 +0200.]
2010-10-21 17:21:15.085 Clernen[445:207] hallo,ganzeZahl=1848 pi = 3.141593
```

Die Ausgabe der vorher definierten Variablen im Debugger-Fenster

Es wurden hier erst Variablen deklariert und jeweils mit einem Wert (1848 und 3,14 ...) versehen und anschließend in einer Anweisung zur Ausgabe in das

Debugger-Fenster gebracht. C benutzt da Formatierungszeichen, um genau festzulegen, wie eine Zahl dargestellt und ausgegeben wird. In diesem Fall waren das die Formatierungszeichen `%i` und `%f`. Tragen Sie zwischen dem `%` und dem Buchstaben eine Zahl ein, so bestimmt die Zahl die Anzahl der Stellen, die ausgegeben werden sollen. Dies sieht dann so aus: `%10i` zeigt Ihnen eine rechtsbündig zentrierte Zahl mit zehn Stellen an. `%10.2f` zeigt Ihnen eine zehnstellige Fließkommazahl mit zwei Nachkommastellen an.

Berechnungen

Es lassen sich natürlich mit den deklarierten Variablen auch Berechnungen anstellen. Dazu habe ich ein kleines Beispiel für den Debugger und Xcode vorbereitet. Geben Sie in der *main.m* dieses Code-Beispiel ein. Achten Sie darauf, dass die Syntax auch so wiedergegeben wird, wie sie im unten genannten Beispiel steht.

```
int main(int argc, char *argv[]){
    int a = 100;
    int b = 50;
    int c;
    c = a * b; NSLog (@" c=%i",c);
    c = a / b; NSLog (@" c=%i",c);
```

Die Variable C zeigt Ihnen den Wert der Berechnungen an.

Hier wurden drei Variablen angelegt: a, b und c. a hat den Wert 100 und b den Wert 50. In der darauf folgenden Anweisung wird a mit b multipliziert und als c ausgegeben.

```
[Session started at 2010-10-21 17:36:53 +0200.]
2010-10-21 17:36:54.471 Clernen[576:207] c=5000
2010-10-21 17:36:54.473 Clernen[576:207] c=2
```

Die Ausgabe im Consolen-Fenster. c hat den Wert 2 und 5000.

Sie sehen, das Ganze hat funktioniert. Die Variable c zeigt nun den Wert der Berechnungen an. Nämlich einmal den Wert 5000 und dann den Wert 2. Sie können nun den Wert der Variablen im Quellcode verändern, neu kompilieren und sehen, wie sich die Werte im Debugger-Fenster verändern. Versuchen Sie einmal, mehr Variablen zu definieren und diese in Berechnungen einzubeziehen. Es stehen Ihnen diese Operatoren zur Verfügung: `+, -, *; /.` Setzen Sie in Berechnungen das % als Operator ein, zeigt es Ihnen den Rest einer Division. Das nennt man auch »modulo«.

Verzweigungen und Bedingungen

Verzweigungen und Bedingungen nennt man auch »Kontrollstrukturen«. Sie steuern in Programmen Abläufe und verzweigen in bestimmte Programmteile. Dies geschieht über Vergleichsoperatoren:

Die Vergleichsoperatoren

Operator	Beispiel	Beschreibung
==	a ==b	wahr, wenn a gleich b ist
!=	a !=b	wahr, wenn a ungleich b ist
<	a < b	wahr, wenn a kleiner b ist
>	a > b	wahr, wenn a größer b ist
<=	a <= b	wahr, wenn a kleiner oder gleich b ist
>=	a >=b	wahr, wenn a größer oder gleich b ist

Die if Verzweigung:

```
if (condition) {
    statements // Block 1
}
else {
    Statements // Block 2-
}
```

Die Anweisungen in Block 1 werden nur ausgeführt, wenn die Bedingung condition wahr ist. Ist sie das nicht, wird in Block 2 weitergemacht.

Hier noch ein kleines Beispiel für Xcode und Debugger:

```
int main(int argc, char *argv[]){
    int a = 2;
    int b = 1;
    if (a==b){
        NSLog (@"a ist gleich b");
    }
else if (a<b){
        NSLog (@"a ist kleiner als b");
    }
else {
        NSLog (@"b ist gleich a");
    }
```

Hier wurde eine Verzweigung programmiert.

Je nachdem, wie Sie die Variablen a und b mit Werten versehen, sieht die Ausgabe dann so aus.

```
[Session started at 2010-10-21 18:56:42 +0200.]
2010-10-21 18:56:43.781 Clernen[944:207] a ist kleiner als b

[Session started at 2010-10-21 18:57:49 +0200.]
2010-10-21 18:57:51.076 Clernen[970:207] a ist gleich b

[Session started at 2010-10-21 18:58:40 +0200.]
2010-10-21 18:58:41.528 Clernen[991:207] a ist kleiner als b

[Session started at 2010-10-21 18:59:25 +0200.]
2010-10-21 18:59:26.251 Clernen[1015:207] b ist kleiner als a
```

Das Ergebnis der festgelegten Bedingungen wird angezeigt.

Eine Schachtelung von if-Anweisungen ist erlaubt. Dies ist aber manchmal ziemlich unübersichtlich. In C gibt es die Möglichkeit, eine Mehrfachauswahl zu treffen. Eine Mehrfachauswahl programmiert man immer dann, wenn aufgrund eines Ausdrucks mehrere Alternativen zur Auswahl stehen. Ich habe ein schönes Beispiel für Xcode und den Debugger vorbereitet:

```
int main(int argc, char *argv[]){
    int tagderwoche = 3;
    switch (tagderwoche)
    {
        case 1 : NSLog(@"Montag");
            break;
case 2 : NSLog(@"Dienstag");
            break;
case 3 : NSLog(@"Mittwoch");
            break;
case 4 : NSLog(@"Donnerstag");
            break;
       case 5 : NSLog(@"Freitag");
            break;
case 6 : NSLog(@"Samstag");
            break;
case 7 : NSLog(@"Sonntag");
            break;
}
```

Das Case-Konstrukt ersetzt verschachtelte If-Bedingungen.

In diesem Fall muss die Ausgabe im Debugger »Mittwoch« lauten. Hat die Variable (tagderwoche) den Wert 7, wird der Sonntag ausgegeben.

```
[Session started at 2010-10-21 19:15:14 +0200.]
2010-10-21 19:15:15.469 Clernen[1075:207] Mittwoch
```

Die Ausgabe der Case-Abfrage. Mittwoch hat den Wert 3.

Schleifen

Ein Schleifendurchlauf wird so lange wiederholt, bis die Bedingung erfüllt ist. In Objective-C gibt es drei verschiedene Schleifenarten:

- die for-Schleife
- die while-Schleife
- die do-while-Schleife

Die for-Schleife

Die for-Schleife nennt man auch »Zählschleife«. Sie hat folgende Syntax:

```
for (Initialisierung; Bedingung; Endanweisung)
    Anweisung;
```

Sollen alle Zahlen bis 1000 ausgegeben werden, schreibt man das wie folgt:

```
i = 1
for (i=1; i<=1000; i++)
    NSLog(@"$d", i)
```

Die while-Schleife

Die while-Schleife erlaubt eine bedingte Anweisung. Zuerst wird die Bedingung im Schleifenkopf geprüft. Ist diese wahr, werden die Anweisungen im Schleifenkörper ausgeführt.

Die Syntax sieht so aus:

```
while (Conditions) {
statements
}
```

Die Fibonacci-Zahlen

Die Fibonacci-Zahlen kleiner als 1000 werden im nächsten Beispiel errechnet. Die Fibonacci-Folge ist eine unendliche Folge von Zahlen, bei der sich die aktuelle Zahl aus der Addition der beiden vorhergehenden Zahlen ermittelt. Benannt ist diese

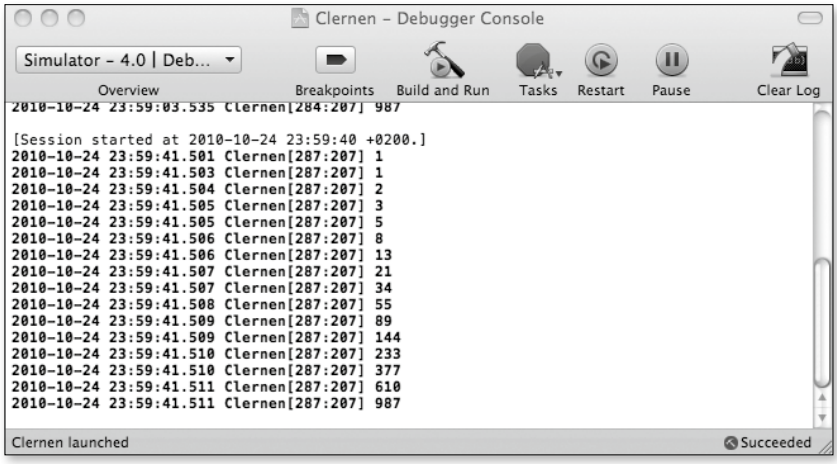
Zahlenfolge nach Leonardo Fibonacci, der im Jahr 1202 damit das Wachstum einer Kaninchenpopulation errechnet hatte.

Für Xcode und den Debugger sieht das dann so aus: Zuerst folgt der Code, dann die Ausgabe im Debugger.

```
int main(int argc, char *argv[]){
    int Zahl = 0;
    int altezahl = 1;
    int neuezahl;
    while (altezahl < 1000)
    {
        NSLog(@"%d ", altezahl);
        neuezahl = altezahl + zahl;
        zahl = altezahl;
        altezahl = neuezahl;
}
```

Die Anweisung errechnet die Fibonacci-Zahlenreihe.

Die Zahlenreihe, die das Programm errechnet, sieht in der Bildschirmausgabe des Debuggers so aus:



Die Fibonacci-Zahlenreihe bis 1000

Do-while-Schleife

Im Gegensatz zur while-Schleife gibt es noch die do-while-Schleife.

Die Syntax:

```
do {
statements
} while (condition);
```

In der do-while-Schleife werden erst die Anweisungen ausgeführt und dann die Bedingungen abgefragt. Die Schleife wird also in jedem Fall ein Mal durchlaufen. Diese Schleife nennt man auch »fußgesteuert«. Die while-Schleife ist dagegen kopfgesteuert, weil die abzufragende Bedingung gleich im Kopf abgefragt wird. In einer fußgesteuerten Schleife werden auf jeden Fall alle Anweisungen ausgeführt, bevor die Bedingung geprüft wird.

Klassen

In diesem Kapitel haben Sie bis jetzt etwas über Kontrollstrukturen erfahren. Ein großes Feld von Objektive-C ist, da es sich ja um eine objektorientierte Sprache handelt, das Erstellen von Klassen. Wenn Sie im Interface Builder arbeiten, sprechen Sie eben diese vordefinierten Klassen an. Eine Klasse ist im Prinzip der Bauplan eines Objekts. Ein Objekt kann beispielsweise ein Rechteck sein oder ein Schaltknopf oder ein Datenpicker etc.

In einer Klasse entwerfen Sie einen Bauplan für ein Objekt. Haben Sie dies einmal getan, kann der Bauplan jederzeit wieder aufgerufen werden. Sie können in Xcode solche Klassen selber entwerfen und abspeichern. Im Prinzip sind auch die ganzen Frameworks, die Ihnen im Interface Builder und Xcode begegnen, Ansammlungen von Klassen, die Objekte beschreiben.

Im Nachfolgenden sehen Sie die Klassendefinition einer Bruchklasse. Hier wurde also ein Objekt erstellt, das in einer Klasse beschrieben werden muss. Wenn Sie eine Klasse aufrufen, wird sie meist in der Header-Datei .h angezeigt.

```
/ Klassenbezeichnung und Ableitung
@interface Bruch : NSObject < Protokoll >
// Instanzvariablen, für jede Instanz angelegt.
{
 NSInteger zaehler;
 NSInteger nenner;
}
// Eigenschaften der Instanzen
@property( assign ) NSInteger zaehler;
@property( assign ) NSInteger nenner;
```

```
// Methoden:
- (void) printLn;
- (float) floatValue;
@end
```

Die strukturierte Programmierung

Das war nun der Kurzeinstieg in die Programmiersprache Objective-C. Sie haben hier die Grundelemente kennengelernt. Falls Sie tiefer in die Programmiersprache einsteigen wollen, helfen Ihnen andere Fachbücher weiter. Zwei Titel möchte ich Ihnen empfehlen: Einmal Cocoa Programming for Mac OS X von Aaron Hillegass. Ein zweites, sehr umfangreiches und informatives Buch ist im Smart Books-Verlag erschienen: Amin Negm-Awad: Objective-C und Cocoa, Band 1: Grundlagen. Als dritte Möglichkeit sei noch auf die iOS-Library verwiesen. Dort finden Sie eine ausführliche Dokumentation auf Englisch unter dem Link http://developer.apple.com/library/ios/navigation/index.html.

Wenn Sie programmieren wollen, sollten Sie Hilfsmittel nutzen, die die logischen Abläufe Ihres Programms grafisch darstellen. Sie können den Ablauf eines Programms ohne Code in einer grafischen Übersicht entwickeln. Diese grafische Struktur erstellt man in einem Struktogramm. Wenn Sie ein Mal den Ablauf Ihres Programms damit dargestellt haben, können Sie Struktur eins zu eins mit Quelltext füllen. Dies gilt nicht nur für Objective-C, sondern für fast alle modernen Programmiersprachen wie Pascal, Java oder C ++.

Es gibt drei grafische Gebilde, mit denen Sie den Ablauf eines Programms komplett darstellen können. Das sind die Sequenz, die Selektion und die Iteration. In den einzelnen Abschnitten finden Sie die Entsprechung zu Objective-C.

Die Sequenz

In der Sequenz werden alle Anweisungen untereinander geschrieben und wie ein Stapel abgearbeitet. Umgesetzt in Objektive C könnte das so aussehen:

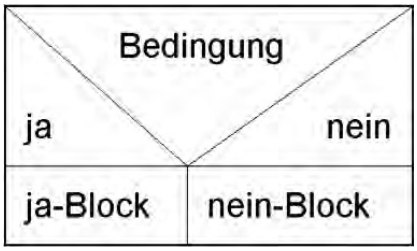
```
c = a * b; NSLog (@" c=%i",c);
c = a / b; NSLog (@" c=%i",c);
```

Die Anweisung

Die Selektion

Die Bedingungen haben Sie in den Beispielen in diesem Kapitel schon kennengelernt. Das sind die If-Abfragen, die den Benutzer zum Beispiel durch ein Programm führen. Die können so aussehen:

```
        NSLog (@"a ist gleich b");
    }
else if (a<b){
        NSLog (@"a ist kleiner als b");
    }
else {
        NSLog (@"b ist gleich a");
    }
```

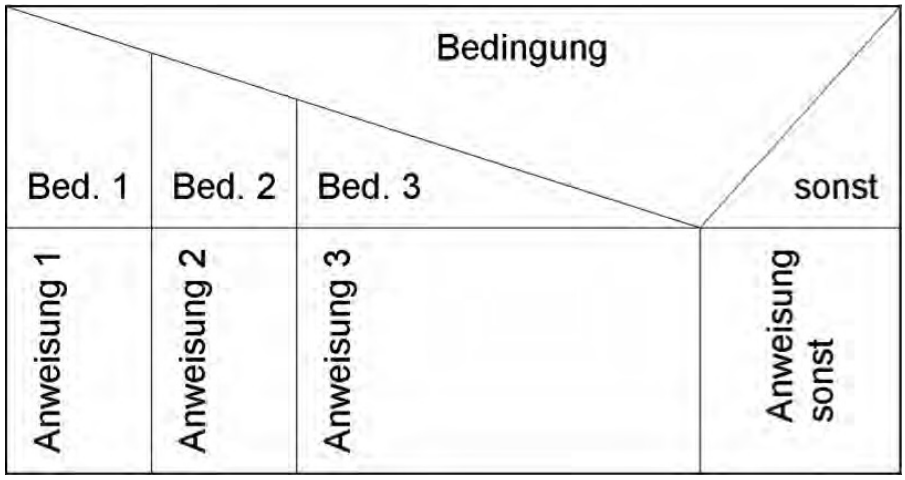


Die If-Abfrage

Das Case-Konstrukt

Auch das Case-Konstrukt haben Sie schon kennengelernt. Es ersetzt ineinander verschachtelte If-Abfragen. Hier ein Programmbeispiel aus dem Straßenverkehr:

```
int ampel = 4
switch (ampel)
    {
        case 1 : NSLog(@"ROT");
            break;
case 2 : NSLog(@"GRUEN");
            break;
case 3 : NSLog(@"GELB");
            break;
case 4 : NSLog(@"DUNKELGELB");
            break;
}
```



Das Case-Konstrukt

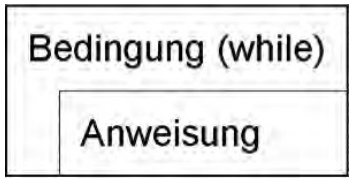
Die Iteration

Hier wird so lange etwas wiederholt, bis ein Kriterium für das Ende erfüllt ist.

Die while-Schleife

Hier ist ein Beispiel für eine kopfgesteuerte Schleife. In Quelltext umgesetzt sieht das in C so aus:

```
{
 int i = 0;
 while (i < 10)
 {
 printf("%i. Wiederholung.\n",i++);
 }
 return 0;
}
```



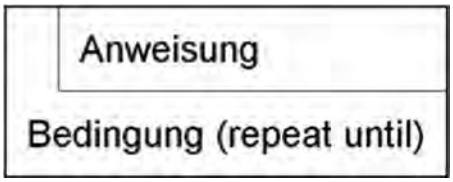
Die while-Schleife

Die do-while-Schleife

In der do-while-Schleife wird erst am Ende der Schleife die Bedingung geprüft. Das Programm durchläuft auf jeden Fall ein Mal die Schleife.

```
int zaehler = 1;
do
{
 zaehler = zaehler + 1;
}
while (zaehler <= 10);
NSLog(@"Der Wert von zaehler ist %d", zaehler);
```

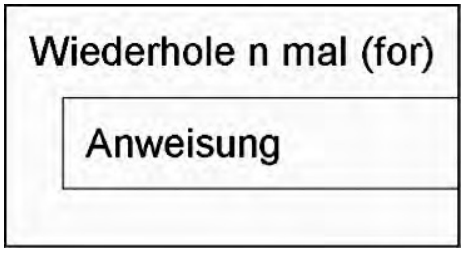
Die Bildschirmausgabe lautet hier "der Wert von Zaehler ist 11"



Die do-while-Schleife

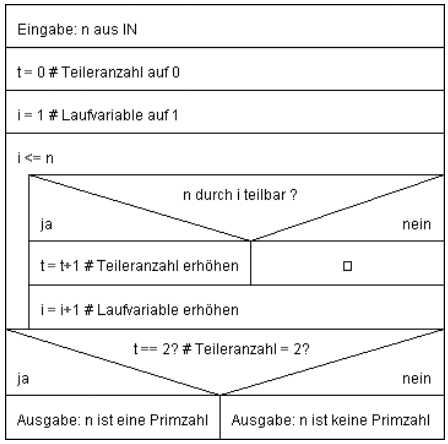
Die Zählschleife
In der folgenden Zählschleife wird ein Array mit den Zahlen eins, zwei, drei und vier gefüllt. In der Zählschleife wird der Inhalt des Arrays ausgelesen und auf dem Bildschirm ausgegeben.

```
int smartArray[4] = {1, 2, 3,4};

for (int i = 0; i < 4; i++) {
printf("%d\n", smartArray[i])
```



Das grafische Symbol für eine Zählschleife

Die einzelnen grafischen Blöcke können Sie beliebig zu einem Programm zusammenstellen. Sie werden von oben nach unten gelesen. Das untere Struktogramm zeigt Ihnen den Algorithmus eines Primzahltesters.

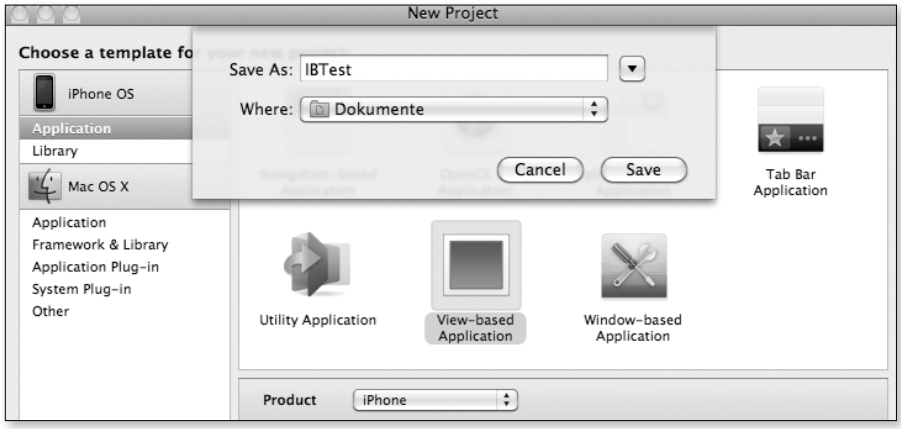


Der Programmablauf für einen Primzahlentester

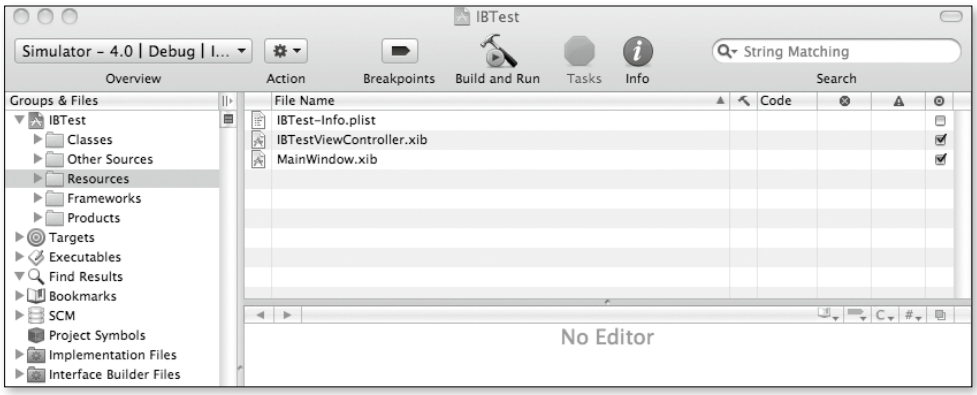
Objective-C objektorientiert

Nachdem im vorherigen Kapitel ja wieder reichlich Theorie zu verarbeiten war, folgt nun wieder ein praktisch orientierter Abschnitt. Es ist der Teil der objektorientierten Programmierung. Sie brauchen dafür den Interface Builder und Xcode. Den Interface Builder haben Sie ja bereits kennengelernt. Dort wird die objektorientierte Arbeit verrichtet. Es findet dort die Gestaltung der Benutzeroberfläche mit den einzelnen Bedienelementen des iPhones per Drag&Drop statt. In Xcode wird der logische Programmablauf durch Objektive C zusammengestellt.

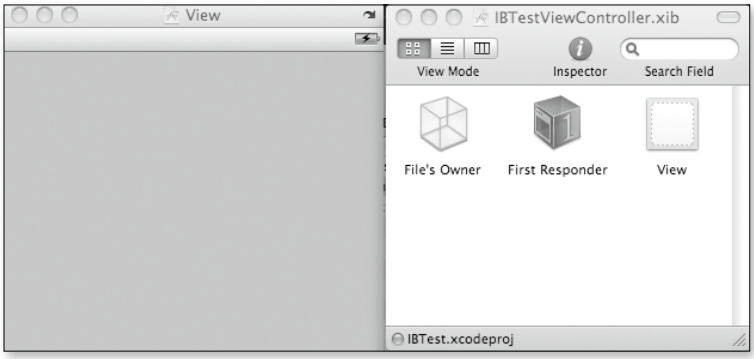
1. Legen Sie in Xcode ein neues Projekt mit dem Namen *IBtest* an. Wählen Sie wieder ein *View-Based-Application*-Projekt.



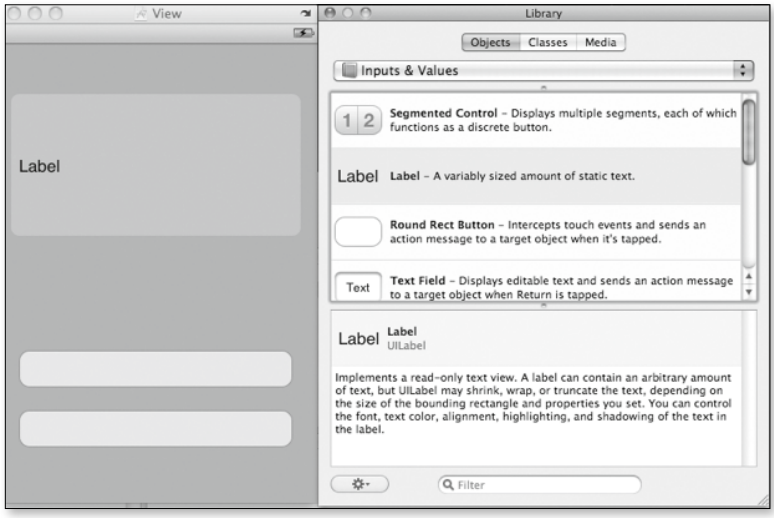
2. Danach wählen Sie im Xcode-Fenster im Ordner *Resources* die Datei *iÍBTest-ViewController.xib* an.



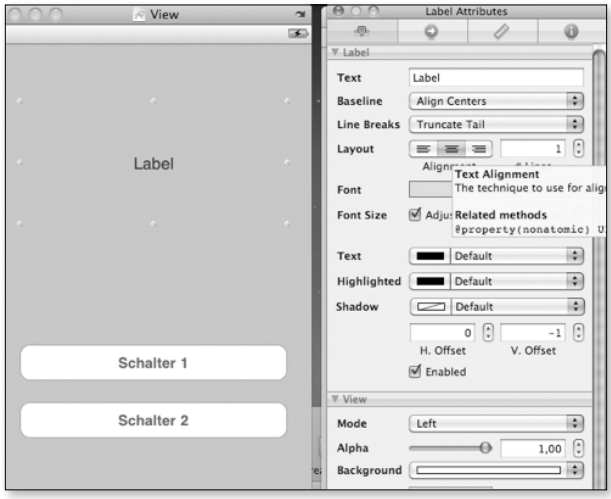
3. Durch Doppelklick auf diese Datei startet automatisch der Interface Builder. Sie sehen nun zwei Fenster. Zum einen sehen Sie das Fenster *View* und das Fenster *iÍBTestViewController.xib*. Über dieses Fenster besteht die Verknüpfung zu Xcode. Im Interface Builder arbeiten Sie objektorientiert. Das heißt, dass Sie die einzelnen Elemente einer App in einer grafischen Benutzeroberfläche zusammenstellen können.



4. Im nächsten Schritt werden für das neue Projekt drei Flächen in der View-Umgebung angelegt. Dazu benötigen Sie die Library des Interface Builders. Die Library enthält sichtbare und unsichtbare Steuerelemente. Sie brauchen für Ihr Projekt aus der Kategorie *Inputs & Values* die Schaltflächen *Label* und *Round Rect Button*. Bewegen Sie einmal die Fläche *Label* durch Klicken und Ziehen auf das Fenster *View*. Bewegen Sie zwei *Round Rect Buttons* ebenfalls auf das Fenster *View* und positionieren Sie die drei Flächen wie in der unteren Abbildung dargestellt.



5. Für den nächsten Schritt wird der Inspector des Interface Builders benötigt. Wenn Sie im View in die Fläche *Label* klicken, zeigt Ihnen der Inspector die dazugehörigen Attribute an. Sie können in ihm wie in einer Textverarbeitung das Feld *Label* formatieren. Wählen Sie im Menüpunkt *Layout* nun die zentrierte Darstellung an. Die Schrift wird von nun an im Label zentriert dargestellt.

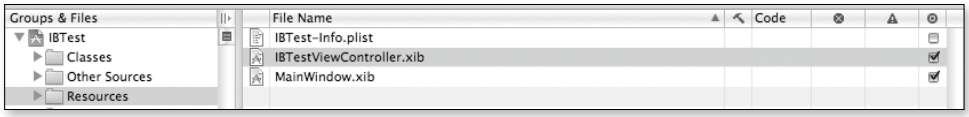


6. Beschriften Sie nun die unteren beiden Buttons mit *Schalter 1* und *Schalter 2*, indem Sie in die Flächen doppelklicken und den Text eintippen. Der *Inspector* wird Ihnen die Beschriftung im Feld *Attributes* anzeigen.

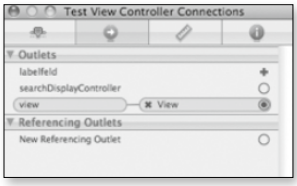
Speichern Sie nun im Interface Builder Ihre Arbeit ab und springen Sie zu Xcode zurück. In Xcode öffnen Sie im Projektfenster unter *Classes* die Headerdatei *IBTestViewController. h.* Dort nehmen Sie die Variable *labelfeld* ein:

```
...interface IBTestViewController : UIViewController {
    IBOutlet UILabel *labelfeld;
}
...
```

7. Sie definieren dort eine Instanzvariable mit dem Namen *labelfeld*. Nach dieser Code-Eingabe speichern Sie die Datei ab. Danach rufen Sie in Xcode wieder die Datei *IBTestViewController.xib* auf und springen dadurch in den Interface Builder zurück.



8. Im Interface Builder werden nun Verknüpfungen zwischen den einzelnen Feldern und Xcode festgelegt. Sie verknüpfen als Nächstes die Variable *labelfeld* mit dem Label im View. Dazu ziehen Sie aus dem Inspectorfenster *Test View Controller Connections* hinter dem Feld für *labelfeld* eine Verbindung zum View. Die Verbindung ist danach aktiv. Die Verbindung der Instanzvariable aus der Headerdatei mit den Steuerelementen ist damit hergestellt. Diese Variable wird Ihnen auch im Connection-Inspector unter *Outlets* angezeigt.

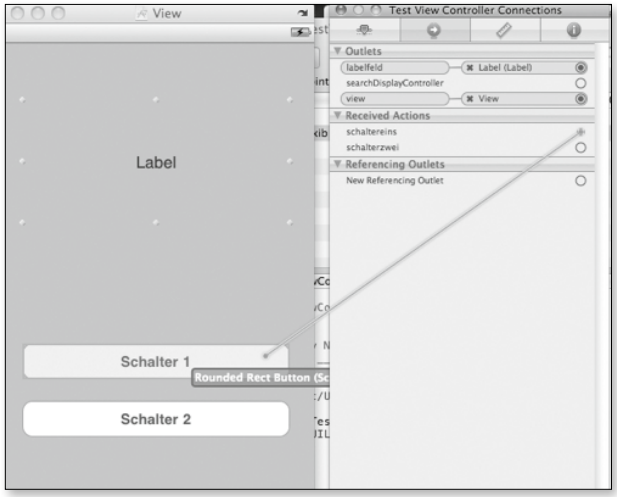


9. Springen Sie nun zurück zu Xcode. Dort müssen noch zwei Methoden in die *Headerdatei* eingetragen werden. Diese dienen dazu, dass die beiden Schaltflächen im View auch hinterher in der App benutzt werden können. Geben Sie diese wie folgt ein:

```
...interface IBTestViewController : UIViewController {
    IBOutlet UILabel *labelfeld;
}
-(IBAction) schaltereins;
-(IBAction) schalterzwei;
...
```

10. Wenn die Eingabe erfolgt ist, speichern Sie die Datei in Xcode ab und springen wieder in den Interface Builder. Sie merken schon jetzt, dass 90 Prozent der App-Programmierung durch Hin-und-her-Springen zwischen Xcode und dem Interface Builder passiert.

    Im Interface Builder erscheinen *schaltereins* und *schalterzwei* nun als *Received Actions* im Connection-Inspector.



11. Beide Felder müssen Sie nun den Schaltern im View zuordnen. Dies geschieht wieder durch drag & drop. Wählen Sie im Inspector hinter dem Feld *schaltereins* den kleinen Kreis an. Halten Sie die Maustaste gedrückt und ziehen Sie den Mauszeiger auf die Fläche *schaltereins* im View. Lassen Sie die Maustaste los. Es erscheint nun ein Auswahlmenü: Dort können Sie definieren, welche Eigenschaften die Schaltflächen haben sollen. Wählen Sie für *schaltereins Touch Down* und für *schalterzwei Touch up Inside*.

12. Die Verbindungen sind nun hergestellt. Das heißt für Sie, dass die Arbeit im Interface Builder abgeschlossen ist. Speichern Sie das Projekt ab und springen Sie zurück zu Xcode. Dort öffnen Sie diesmal die Datei *IBTestViewController.m*. Dort tragen Sie die paar Zeilen Code ein:

```
...Implementation IBTestViewController
-(IBAction) schaltereins {
    [labelfeld setText:@"Hallo lieber Smartbook Leser"];
}
-(IBAction) schalterzwei {
    NSString *caption = @"Hallo Objektprogrammierer";
[labelfeld setText: caption];
}
```

```
@implementation IBTestViewController

-(IBAction) schaltereins {
    [labelfeld setText:@"Hallo lieber Smartbook Leser"];

}

-(IBAction) schalterzwei {
    NSString *caption = @"Hallo Objektprogrammierer";
    [labelfeld setText: caption];

}
```

13. Es werden hier jetzt Aktionen festgelegt, die ausgelöst werden, wenn der Anwender *Schaltfläche eins* oder *Schaltfläche zwei* benutzt. Dem Feld *Labelfeld* werden konstante Zeichenketten zugeordnet. Danach sind Sie mit der Eingabe fertig. Mit *Build and Run* in der Toolbar von Xcode können Sie nun Ihr Projekt kompilieren. Hat der Debugger nichts mehr zu bemängeln, startet der iPhone-Simulator und Sie können dort Ihre App betrachten:

Ihr nächstes Projekt in Aktion

Kapitel

6

Geldverdienen mit Apps

In diesem Kapitel bekommen Sie eine Anleitung dazu, wie Ihre App in den App Store gelangt. Sie erhalten einen Leitfaden über die Gestaltung Ihrer App und Sie bekommen Daten und Fakten an die Hand, um die Marktchance Ihrer App bestimmen zu können.

Daten und Fakten

Inzwischen sind mehr Smartphones als PCs mit dem Internet verbunden. Gegen Ende 2010 nutzten circa eine Milliarde Menschen Smartphones täglich, um damit ins Internet zu gelangen. Im September 2009 gab Apple bekannt, dass 2 Millionen Apps heruntergeladen wurden. Es befinden sich derzeit über 350 000 Apps im App Store. Und dieses Angebot der Apps im Store soll sich in den nächsten Jahren gar verdreifachen. Der Markt boomt. Der App-Markt, egal ob der von Apple oder der anderer Smartphones, ist fast nicht mehr überschaubar. Wenn Sie also mit einer App Geld verdienen wollen, muss sie gut sein. Sie muss am besten in den App-Charts unter den Top Ten landen.

So sollte Ihre App gestaltet sein

Haben Sie schon mal eine App gesehen, bei der eine Bedienungsanleitung mitgeliefert wird? Die Frage stellt sich erst gar nicht. Sie finden hier im Folgenden eine Zusammenfassung, wie Sie Ihre App gestalten sollten. Ich habe sieben Punkte zusammengestellt. Bedenken Sie, dass 80 Prozent aller Apps nur ein Mal gestartet werden, um dann auf dem Gerät zu verstauben oder sofort wieder gelöscht zu werden.

1. Die App sollte intuitiv und einfach bedienbar sein.
2. Der Aufbau sollte logisch sein.
3. Die App sollte Spaß machen.
4. Die App sollte klar gestaltet sein.
5. Der Nutzen der App muss für den Benutzer sofort erkennbar sein.
6. Die benutzten Grafiken sollten eindrucksvoll sein.
7. Prüfen Sie den Aufwand, den Ihr Projekt verursacht.

Apples Richtlinien

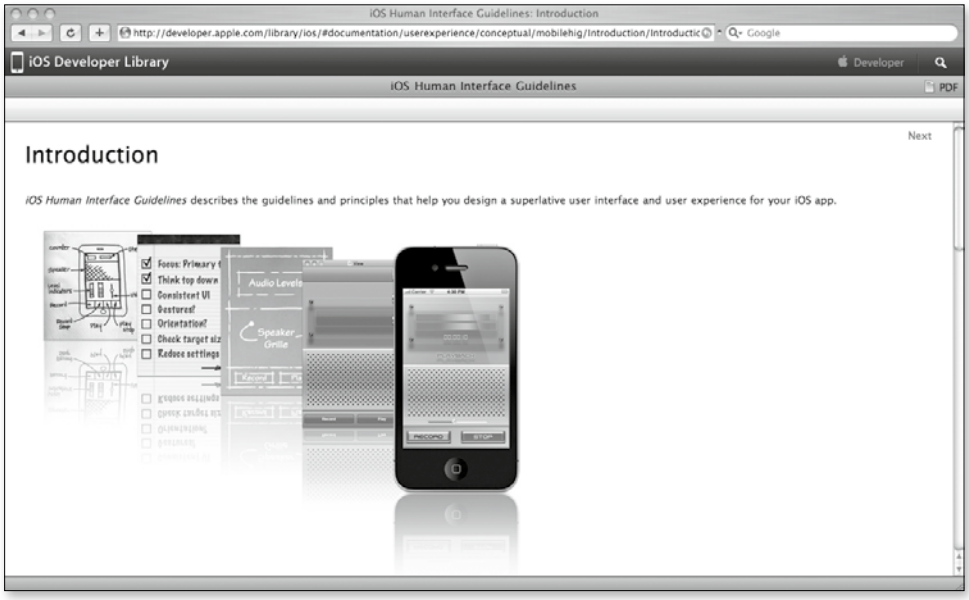
In diesem Abschnitt erfahren Sie mehr über die Apple-Richtlinien, an die Sie sich halten müssen, wenn Sie erfolgreich eine App gestalten möchten. Es gibt die »iPhone Human Interface Guidelines«, Richtlienen zum Erscheinungsbild. Dann

sind da noch die »Mac App Store Review Guidelines«, die Richtlinien zum Inhalt und zur Programmierumgebung.

iPhone Human Interface Guidelines

Bevor Sie mit der Entwicklung Ihrer App beginnen, bedenken Sie bitte, dass es seitens Apple Richtlinien für die Gestaltung einer App gibt. Diese Richtlinien heißen »iPhone Human Interface Guidelines«. Sie finden sie in der iOS-Library unter diesem Link: http://developer.apple.com/library/ios/#documentation/UserExperience/Conceptual/MobileHIG/Introduction/Introduction.html.

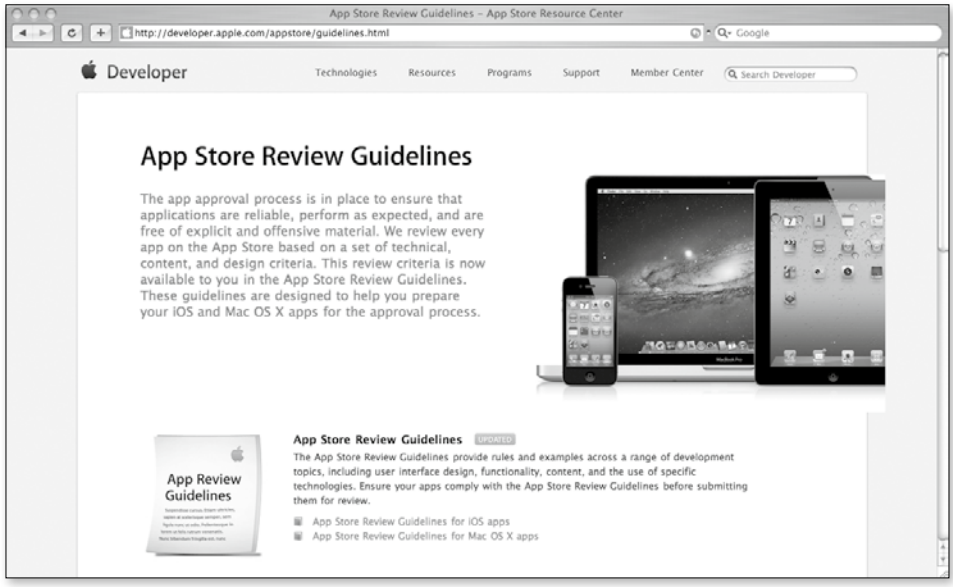
Es ist wichtig, dass Sie sich mit den Inhalten vertraut machen. Verstößt Ihre App gegen eine der Vorgaben, wird sie abgelehnt.



Die iOS Human Interface Guidelines

App Store Review Guidelines

Sie haben nun gelernt, wie Ihre App aussehen sollte. Wenn Sie die App in den App Store bringen wollen, müssen Sie aber noch an den Apple-Richtlinien für Apps vorbei. Apple hat einen ganzen Katalog aufgestellt, wie eine App auszusehen hat. Wenn Sie sich als Entwickler registriert haben, können Sie diesen Katalog unter folgendem Link einsehen. http://developer.apple.com/appstore/mac/resources/approval/guidelines.html



Die App Store Review Guidelines

Es handelt sich hierbei um die »Mac App Store Review Guidelines«. Das ist ein sieben Seiten umfassendes Dokument, welches festlegt, wie eine App aufgebaut sein sollte.

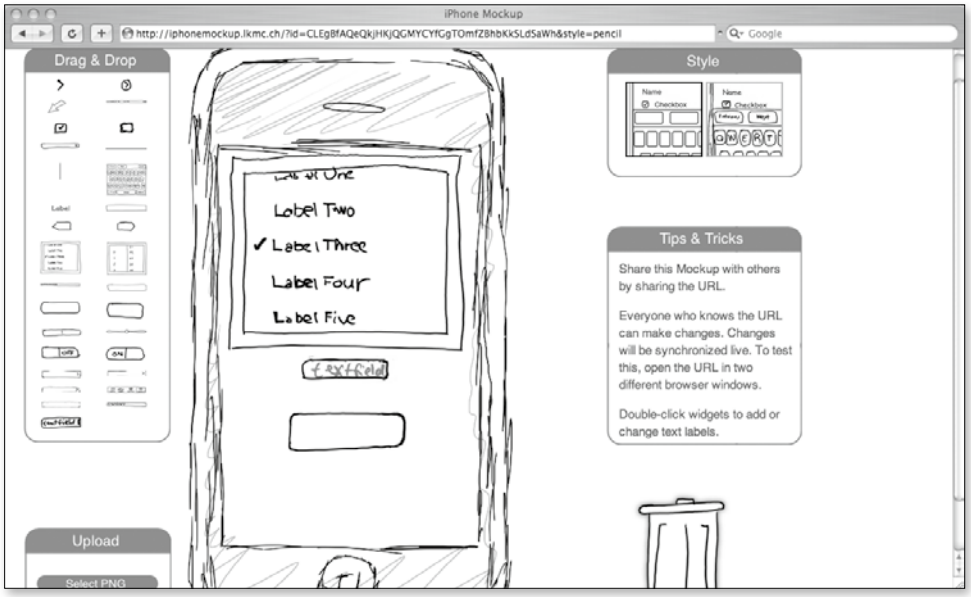
Hier sind die wichtigsten Punkte zusammengefasst:

- keine Apps, die abstürzen
- Apps müssen ihrer Beschreibung entsprechen und dürfen sich nicht anders verhalten.
- Es dürfen keine Beta-Versionen eingestellt werden.
- keine pornografischen, rassistischen oder Gewalt verherrlichenden Inhalte
- keine Spam-Apps. Das sind viele gleichartige Apps mit gleichem Inhalt.
- keine Apps, die zum Konsum von Drogen aufrufen
- Die Apps dürfen im Prinzip nur in Xcode programmiert sein.
- Apps dürfen nicht mit Java programmiert werden.

Die Planung Ihrer App

Bevor Sie Ihre Anwendung in den App Store bringen, steht die Planung der App an. Hierzu fertigen Sie in groben Zügen das Outfit Ihrer einzelnen Screens an. Mittlerweile gibt es für diese Mock-Ups viele Hilfsmittel im Netz. Natürlich können Sie auch einen Bleistift und ein leeres Blatt Papier nehmen, um Ihre Planung zu fixieren. Es gibt Stencil-Kits, eine Schablone, mit der Sie die Screens der App gestalten können. Die Schablone und andere Hilfsmittel können Sie auf der Seite http://www.uistencils.com/products/iphone-stencil-kit bestellen. Sie beinhaltet alle Elemente, die Sie zur Gestaltung Ihrer App-Bildschirme auf Papier benötigen.

Unter dem Link http://iphonemockup.lkmc.ch/ können Sie den Bildschirm Ihrer App online gestalten.



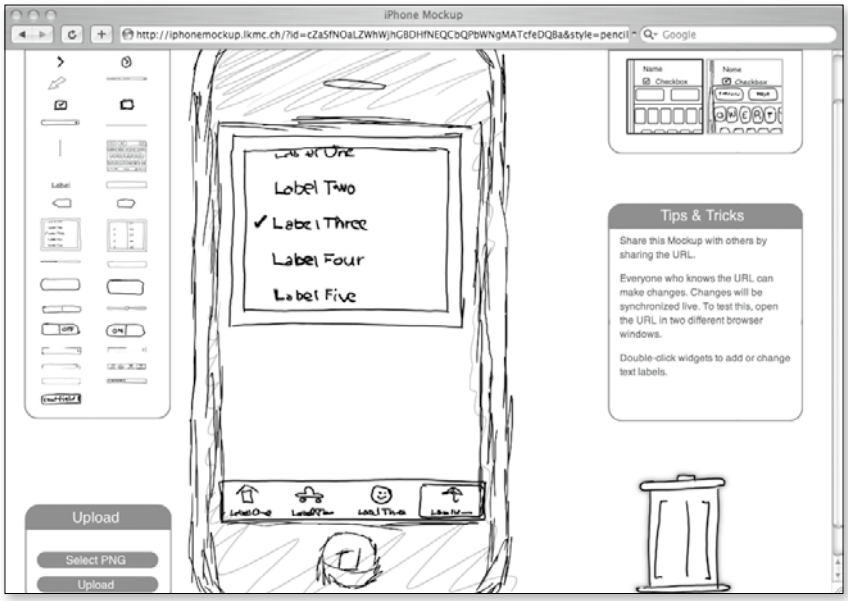
Hier können Sie den Bildschirm Ihrer App online gestalten.

Sie sollten bei der Planung Ihrer App die Seiten zunächst grob planen. Gehen Sie nicht so sehr ins Detail. Legen Sie aber genau die Navigation fest. Es ist wichtig, dass alle Seiten miteinander verbunden sind und dass der Benutzer eine Steuerung vorfindet, die ihn wieder an den Anfang der App führt.

Es gibt drei Arten der Benutzerführung: die Tab-Bar, das Single-Main-View und das Table-View, die im Folgenden vorgestellt werden. Die Workshops in diesem Buch zeigen Ihnen aber auch, wie sie konkret programmiert werden.

Die Tab-Bar

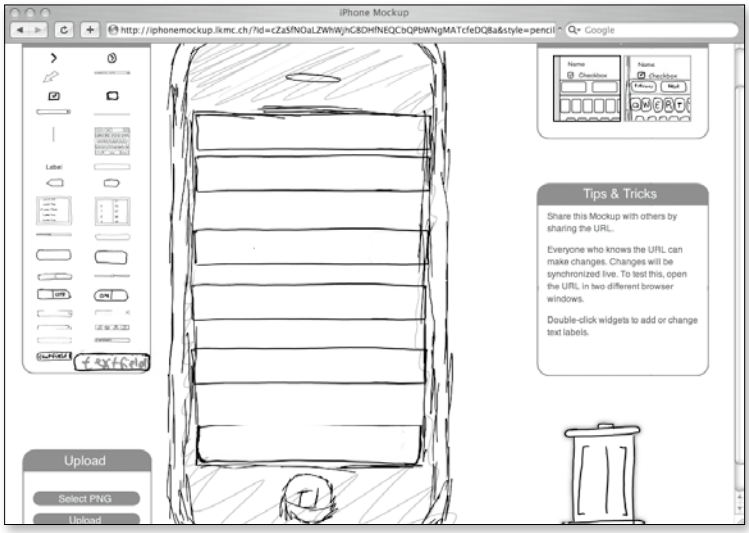
In der Tab-Bar finden Sie am unteren Rand des Bildschirms eine Navigationsleiste, die bis zu fünf Elemente beinhalten kann. Die App wird über diese Navigationsleiste gesteuert. Der Entwickler braucht keine Steuerung auf den einzelnen Seiten zu entwickeln.



Navigation der App mit einer Tab-Bar

Single Main-View

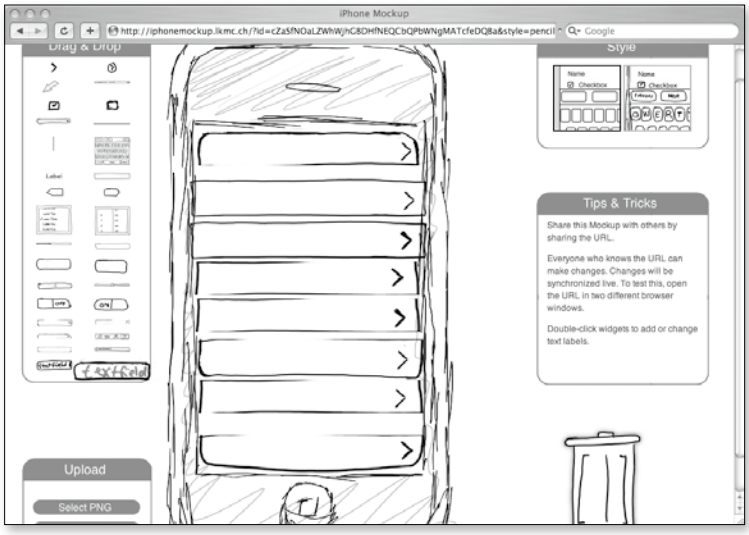
Hier findet die Navigation über Blöcke auf dem Bildschirm statt. Diese Navigationsart wird meistens genommen, wenn ähnliche Informationen angezeigt werden sollen. Der Entwickler muss sich hier aber um die Steuerung selber kümmern. Er muss gewährleisten, dass man von den angewählten Blöcken auch wieder zurück an den Ausgangspunkt kommt.



Die Steuerung der App über Blöcke im Single Main-View

Table-View

Wenn Sie viele Informationen anzuzeigen haben, wählen Sie den Table-View. Die Benutzerführung wird hier durch eine scrollbare, bildschirmfüllende Liste veranlasst. Hierbei müssen Sie wieder für die Steuerung sorgen. Bedenken Sie, dass der Benutzer immer wieder zu der ersten Liste mit den Schlagwörtern zurückkehren muss, um den nächsten Menüpunkt aufzurufen.



Die App-Steuerung mit einem Table-View

Nun haben Sie die gebräuchlichen Methoden der Benutzerführung kennengelernt. Sie können die drei Methoden auch miteinander kombinieren.

Werbung auf dem iPhone

Neuerdings ist es möglich, in Apps Werbung einzubauen. Klickt der Anwender in der App auf die Werbung, erhalten Sie dafür eine Vergütung. Google bietet da eine entsprechende Lösung: AdSense.

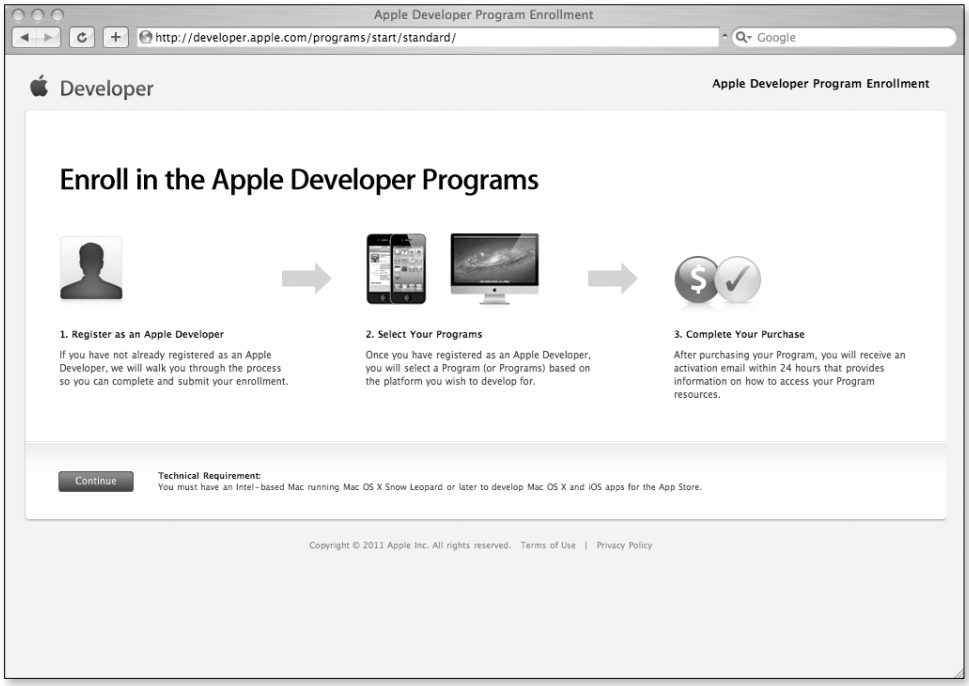
Apple bietet einen neuen Dienst an: iAds. iAds blendet Werbung innerhalb von Apps ein. Apple setzt dabei auf HTML 5 und Java, so dass die Werbung besonders interaktiv betrieben werden kann. Die Werbung wird in der App geöffnet. In diesem Fall springt kein Browser an, um die App zu verlassen. Beendet man die Werbung, wird wieder zum Ausgangspunkt des Programms verzweigt. Weitere Informationen bekommen Sie unter http://advertising.apple.com/.

Werbung auf dem iPhone gibt es seit Mitte 2010 mit iAd

Wer seine Apple-Apps mit Werbung bestücken möchte, bekommt von Apple das Programm iAd-Producer zur Verfügung gestellt. Unter der URL http://developer.apple.com/iad/iadproducer/ können Entwickler sich das Programm herunterladen. Man muss dazu Mitglied im iOS-Developer-Programm sein.

Der Weg in den App Store

Wenn Sie Ihre App in den App Store bringen möchten, benötigen Sie eine Entwicklerlizenz. Die kostet 99 US-Dollar im Jahr und kann unter dem Link http://developer.apple.com/programs/start/standard/ beantragt werden.



Die Registrierung als Apple Developer

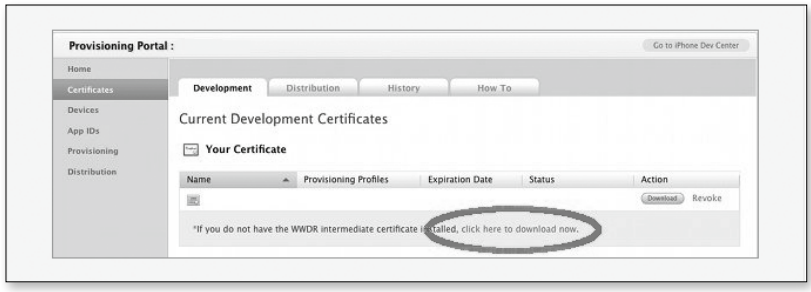
Sie enthält folgende Leistungen:

- Zugriff auf sämtliche Ressourcen wie z. B. das aktuelle SDK, Dokumentationen und Programmierbeispiele
- das Erstellen von Entwicklertests und Adhoc-Profilen
- technischen Support
- Zugriff auf alle Entwicklerforen
- die Möglichkeit zur Veröffentlichung Ihrer App im App Store.

Das ist der erste Schritt. Er ist zwingend erforderlich. Sie müssen für das ganze Prozedere der Anmeldung Zeit einkalkulieren. Es kann bis zu zehn Tage dauern, bis Sie Zugang zu dem Entwicklungsprogramm erhalten.

Danach stehen Sie vor dem iPhone-Provisioning-Portal. Dort können Sie als Mitglied des Developer-Programms die nötigen Schritte für die Distribution Ihrer App

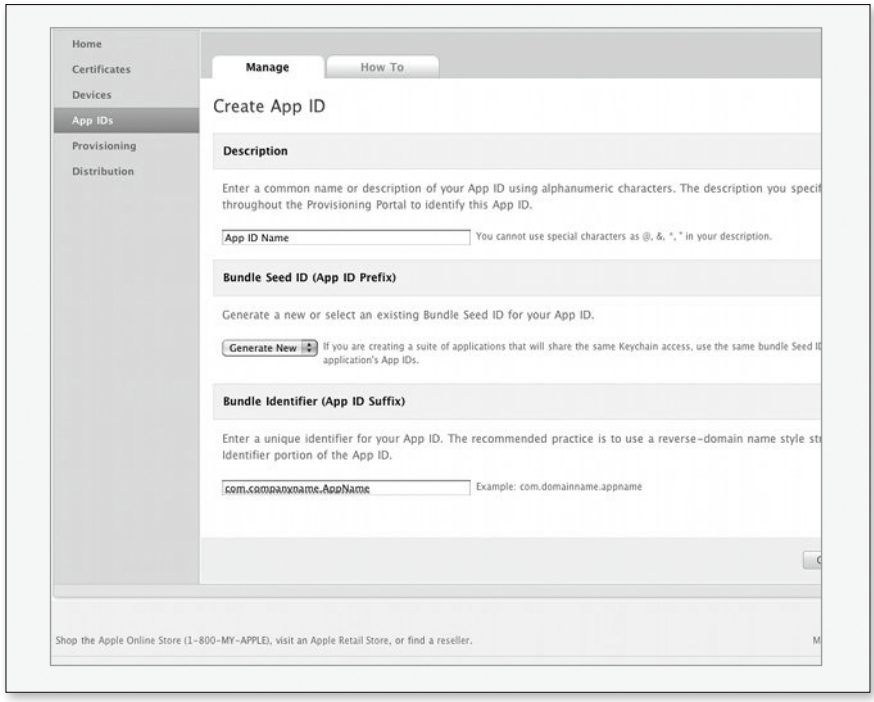
vornehmen. Dieses Portal leitet Sie in einem weiteren Schritt an iTunes Connect, die eigentliche Verkaufsplattform, weiter.



Das Provisioning Portal mit dem Reiter Development

Diese Schritte müssen Sie im iPhone-Provisioning-Portal abarbeiten:

Sie bekommen dort Zertifikate, mit denen Sie Ihre Anwendungen signieren können. Die Zertifikate sind asymmetrisch verschlüsselt. Anhand der Schlüssel kann Apple Ihre App dann eindeutig Ihrem Entwicklerstatus zuordnen.



Erstellung einer App-ID

Im Bereich *Devices* können Sie bis zu 100 Geräte zu Betatests anmelden. Die Geräte werden dabei über ihre Geräte-ID identifiziert. Wenn Sie nicht wissen, wie die Geräte-ID Ihres Gerätes lautet, schließen Sie Ihr Gerät an den Mac an und starten iTunes. Die Geräte-ID wird dort angezeigt. Dies geschieht aber nur, wenn Sie sich für den Adhoc-Vertrieb Ihrer App entschieden haben. Ihre App bekommt dort eine App-ID. Die dient dazu, Ihre Anwendung zu bezeichnen und einer Produktfamilie zuzuordnen.

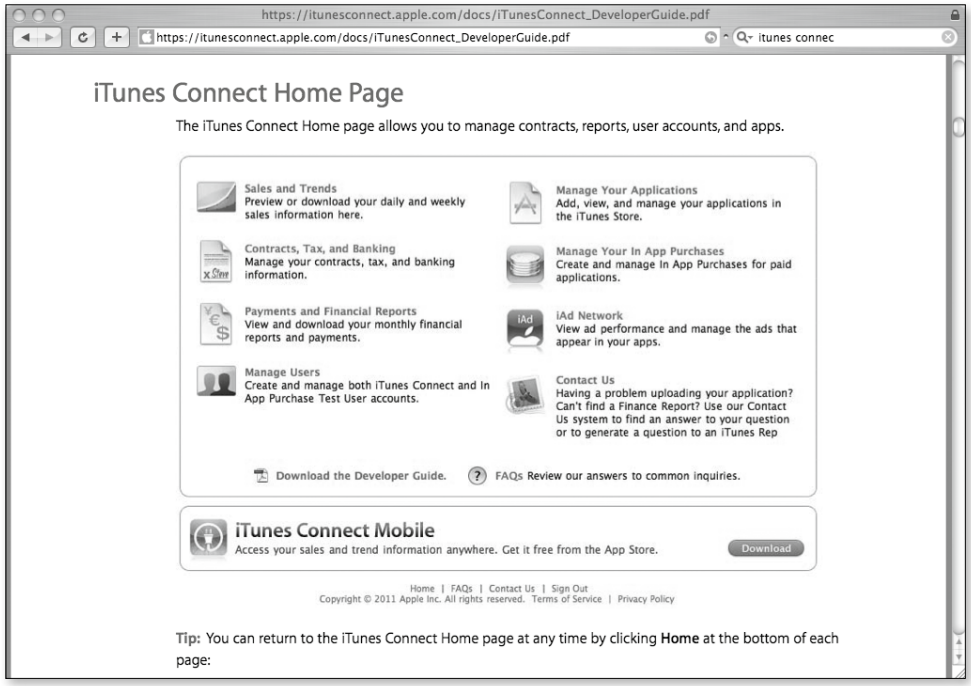
Hier bereiten Sie den Adhoc-Vertrieb vor. 100 iPhones können hier verwaltet werden.

Der nächste Punkt, der abzuhaken ist, ist das Provisioning-Profile. An dieser Stelle wird Ihre Anwendung, nachdem sie fehlerfrei ist, digital signiert. Hier werden die angesprochenen Zertifikate und die für die Adhoc-Distribution vorgesehenen Geräte mit Ihrer UDID miteinander verbunden. Verkaufen Sie Ihre App über den Store, wird dort Ihre Signatur mit Ihrer App verknüpft.

Es geht noch weiter: Sie durchlaufen den letzten Punkt, die Distribution. Hier bekommen Sie Infos, wie Ihre App im App Store vertrieben wird. Danach werden Sie an iTunes Connect weitergeleitet.

iTunes Connect

Wenn Sie bei iTunes Connect gelandet sind, haben Sie die eigentliche Verkaufsplattform erklommen. Wenn Sie nun Ihre App verkaufen wollen und diese nicht kostenlos anbieten, wartet noch etwas Papierkram auf Sie. Darunter fällt die Angabe von Bank- und Adressinformationen. Sie stimmen noch einer steuerlichen Erfassung für die Vereinigten Staaten von Amerika zu und unterzeichnen den Paid-Apps-Contract mit Apple, der Ihre Geschäftsverbindung mit Apple regelt.



iTunes Connect: Hier verwalten Sie Ihre Apps.

In iTunes Connect verwalten Sie von nun an Ihre Apps. Sie können Ihre monatlichen Verkaufszahlen bestaunen oder haben eine Übersicht über die Rückgabe Ihrer Anwendung. Hier wird auch abgerechnet.

Kapitel 7

iOS 5 It just works, but better

Das iOS wird ständig weiter entwickelt. Während der Arbeit an diesem Buch wurde iOS 5 vorgestellt. Grund genug, dem neuen Betriebssystem ein eigenes Kapitel zu widmen. Die neuste Version des mobilen Betriebssystems weist 200 Neuerungen auf und ist für den Herbst 2011 angekündigt worden.

Benachrichtigungen

Das neue iOS 5 bietet eine Nachrichtenzentrale, in der alle eingehenden Nachrichten angezeigt werden können, ohne den Benutzer zu stören. Dabei kann man sich die Neuigkeiten, die man sehen möchte, zusammenstellen. Neue News werden auf dem Bildschirm des Gerätes kurz dargestellt. Möchte man das komplette Nachrichtenzentrum sehen, streicht man auf dem Bildschirm einfach nach unten, egal, wo man sich gerade befindet, und es erscheint auf der Oberfläche. Dort erhält man dann übersichtlich eine Liste der einzelnen Meldungen, die das Gerät empfangen hat. Tippt man auf eine Nachricht, gelangt man zu der dazugehörigen App.

Die neuen Features des Benachrichtigungssytems

iMessage

Mit iMessage macht Apple der klassischen Kurznachrichten wie SMS Konkurrenz. Der neue News-Dienst ist in die Nachrichten-App integriert. Der Versand funktioniert mit Texten, Fotos und Videos und erfolgt auf iPod touch, iPad und iPhone über WI-FI oder den Mobilfunk der dritten Generation. Es ist möglich,

eine Unterhaltung auf einem Gerät zu beginnen und auf einem anderen Gerät fortzusetzen. Die Nachrichten werden dabei verschlüsselt und kostenlos im Internet übertragen. Dieser neue Dienst ist für Entwickler sehr interessant, weil sich damit viele neue Apps programmieren lassen, die einen kostenlosen Versand von Bild- und Video-Dateien ermöglichen.

Mit iMessage macht Apple der SMS Konkurrenz.

Zeitungskiosk

Im Zeitungskiosk werden Zeitungs-Apps und Magazine zentral verwaltet. Der Kiosk kommt als eigene App mit Namen Newsstand auf iPhone, iPad und iPod touch. Wie iBooks präsentiert Apple auch die digitalen Magazine in gediegener Holzregal-Optik. Ist eine neue Ausgabe der Online-Zeitung verfügbar, wird das dem Benutzer direkt angezeigt. Kauft man sich ein neues Magazin im App Store, landet es im App-Kiosk auf dem Gerät und wird dort angezeigt. Für Entwickler ist es notwendig, dass sie ihre Apps für den Kiosk anpassen. Das SDK für iOS bietet dafür eine Programmierschnittstelle an. Ist die API implementiert, arbeitet sie im Hintergrund der App und füllt automatisch den Zeitungskiosk.

Der Zeitungskiosk verbindet direkt in den App Store.

Merklisten

In Merklisten können ortsbezogene Erinnerungen erstellt werden. Möchte man erinnert werden, dass man Schokoriegel im Supermarkt kaufen sollte, trägt man das in eine To-do-Liste ein. Ist man in der Nähe des Supermarktes, wird daran erinnert, dass man den Riegel dort kaufen wollte. Diese Erinnerungen funktionieren auch in iCal, Outlook und iCloud.

Die Merklisten funktionieren mit der Kalender-Funktion.

Twitter

Der Kurznachrichten-Dienst Twitter wurde im neuen Betriebsystem systemweit eingebettet. Der Benutzer meldet sich nur ein Mal an und kann alle Features benutzen, die Twitter ausmacht.. So können Fotos, Videos oder Karten getwittert werden. Die Kontakte-App erkennt die Twitter-Benutzer und Profilbilder der Freunde. Selbst der Aufenthaltsort kann in die Twitter-Nachricht eingebunden werden.

Der Twitter-Account wird nur ein Mal angelegt.

Safari

Der Browser Safari wurde weiter optimiert. Apple verspricht eine Leistungssteigerung auf allen iOS-5-Geräten. Neu ist eine Leseleiste, die Web-Artikel ohne Werbung anzeigt. Man kann sich die Artikel in iCloud abspeichern und hinterher lesen. Hat man ein iPad, so kann man in Safari mittels Tabs die Webseiten wechseln.

Der Browser läuft nun mit einer Nitro-Engine, die deutlich mehr Geschwindigkeit bringt. Dadurch hat Safari in punkto Schnelligkeit die Nase wieder vorn und hängt seine Windows-Konkurrenten Mango ab.

Die Leseliste in Safari

PC Free und iCloud

Mit PC Free entfällt die Aktivierung des iOS-Gerätes durch einen Computer. Die Installation des Betriebssystems geschieht drahtlos. Für Backups und die Wiederherstellung des Gerätes gibt es iCloud, einen Webservice von Apple. In iCloud kann der Benutzer alle Inhalte seines iOS-Gerätes im Web speichern wie (daher der Name) in einer Wolke. Meldet sich der Benutzer bei iCloud an, so erhält er 5 GB Speicherplatz im Web kostenlos. Dort kann er seine persönlichen Daten, E-Mails, Dokumente, Aufnahmen, Account-Daten, Einstellungen und andere App-Daten speichern und jederzeit wieder auf sein iPhone laden. Es ist möglich, dass Programmierer auf iCloud zugreifen, um Daten ihrer App in der Wolke zu speichern. Der umgekehrte Weg funktioniert auch. Apps können jederzeit auf die iCloud-Daten zugreifen. Der Benutzer hat so Zugriff auf größere Datenmengen. Er kann sie für alle Geräte, die auf seine iCloud zugreifen, zugänglich machen.

Setup auf einem iPhone ohne einen PC

Das waren jetzt nur die wichtigsten Features des neuen iOS 5. Insgesamt wurden laut Apple, wie bereits erwähnt, 200 Neuerungen in das neue Betriebssystem gepackt.

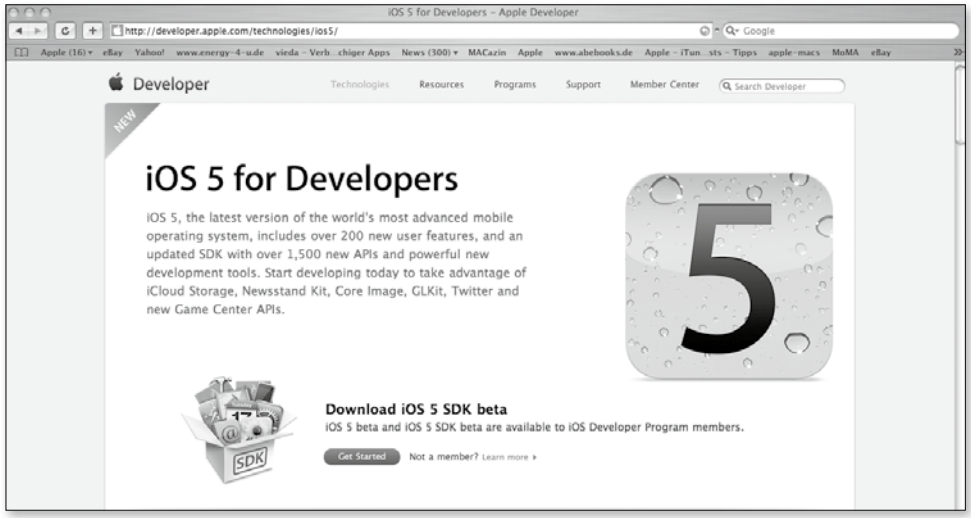
Das Programm Mail wurde verbessert. Der Kalender wurde überarbeitet. Auf dem iPad erhält man eine Jahresübersicht. Durch iCloud können Sie die Daten Ihres Kalenders mit anderen Personen austauschen und synchronisieren.

Wi-Fi-Sync, eine Funktion, mit der per WLAN über Nacht das Gerät mit iTunes synchronisiert wird.

Es wurden neue Multitasking-Gesten für das iPad entwickelt und die Bedienungshilfen verbessert. Besitzt man das iPad 2, kann der Inhalt des iPad-Bildschirms auf einen HD-Fehrnseher gespiegelt werden. Dies passiert mit AirPlay, es wird dafür aber ein Apple TV benötigt. Beide Komponenten arbeiten zusammen.

iOS 5 für Entwickler

Das SDK für iOS 5 bekommen Sie unter dem Link: http://developer.apple.com/technologies/ios5/. Im Lieferumfang hält das SDK für den Entwickler iCloud-Storage, Zeitschriften-Kit, Core-Image, GL-Kit, Twitter und neue Game-Center APIs bereit. Insgesamt verspricht Apple, dass über 1500 neue APIs dort bereit stehen.



Download-Bereich für das neue iOS 5 SDK

Im Download-Bereich für Entwickler befindet sich eine Beta-Version von Xcode 4.2, die 4,2 GB groß ist und nach ihrer Installation 8,4 GB Platz auf Ihrer Festplatte in Anspruch nimmt. Diese Version beinhaltet auch das neue SDK für iOS 5. Systemvoraussetzung ist Mac OS 10.6.6.

Hier wartet Xcode 4.2 auf Ihr Projekt.

Neue APIs

Das SDK bietet für die Neuerungen im iOS 5 neue APIs. Die neuen Schnittstellen für die Programmierung sind für die folgenden Bereiche:

iCloud-Storage

ermöglicht das drahtlose, automatische Versenden von Nachrichten auf alle Geräte des Benutzers.

Für den Entwickler stehen aber noch weitere Funktionen zur Verfügung. Es können Dokumente und Daten direkt aus der Benutzer-App in iCloud gespeichert und verändert werden. Dazu wurde das Foundation-Framework um folgende Klassen erweitert:

NSFileManager Class
enthält Methoden, um die Benutzerdokumente in iCloud zu speichern.

NSFileCoordinator Class und NSFilePresente Protocol
werden benutzt, wenn Dateien in iCloud verändert werden.

NSFileVersion class
wird benutzt, wenn es zu Konflikten beim Abspeichern von Dokumenten in iCloud kommt.

NSURL class
benutzt der Programmierer, um Daten zwischen der App und iCloud zu synchronisieren.

UIManagedDocument
ist Bestandteil des Core-Data-Frameworks und eine Unterklasse von UIDocument. Die Klasse ist für das Management von Dateien und Datenbanken zuständig, die in iCloud gespeichert werden sollen.

Notification Center

Die Implementierung von Push-Notifications wird vereinfacht und ist in Xcode integriert. In der Dokumentation der OS-Library lässt sich dieses Framework unter dem Suchbegriff »MessageUIFramework« finden.

iMessage

iMessage, der neue News-Dienst, der für alle Benutzer über WLAN und 3G funktioniert, lässt sich im SDK aufrufen und in Ihrer App einbauen. So kann aus der App für einzelne Benutzer oder Gruppen dieser neue Nachrichtendienst eingebaut werden. Das eröffnet dem Entwickler neue Möglichkeiten, da nun auch große Dateien wie Videos gratis versendet werden können. Dieser Dienst ist mit iCloud verzahnt.

Newsstand

Die Abo-Aktualisierung von Online-Zeitungen lässt sich direkt in Xcode programmieren. Das Zeitungs-Kit ist in Xcode integriert. Es lässt sich also auch in Ihrer App einbauen. Themen werden im Hintergrund aktualisiert und dann automatisch bereitgestellt. So ist der Benutzer immer auf dem neusten Stand der herausgegebenen Zeitung. Das NewsstandKit.framework arbeitet dabei weitgehend automatisch. Ist es einmal in die App eingebunden, verwaltet das Framework das ganze Handling des Herausgebens von Zeitungen selber. Ordert der Benutzer eine Zeitung über den Kiosk, lädt das Framework im Hintergrund Zeitungen nach, die neu erschienen sind.

Automatic Reference Counting

(kurz ARC) arbeitet mit dem LLVM-Compiler zusammen und übergibt die Speicherverwaltung dem Compiler. Dadurch sollen die Apps stabiler laufen. Apple verspricht sich davon die Reduktion von Speicher-Löchern. Insgesamt sollen die Applikationen dadurch schneller werden und weniger häufig abstürzen.

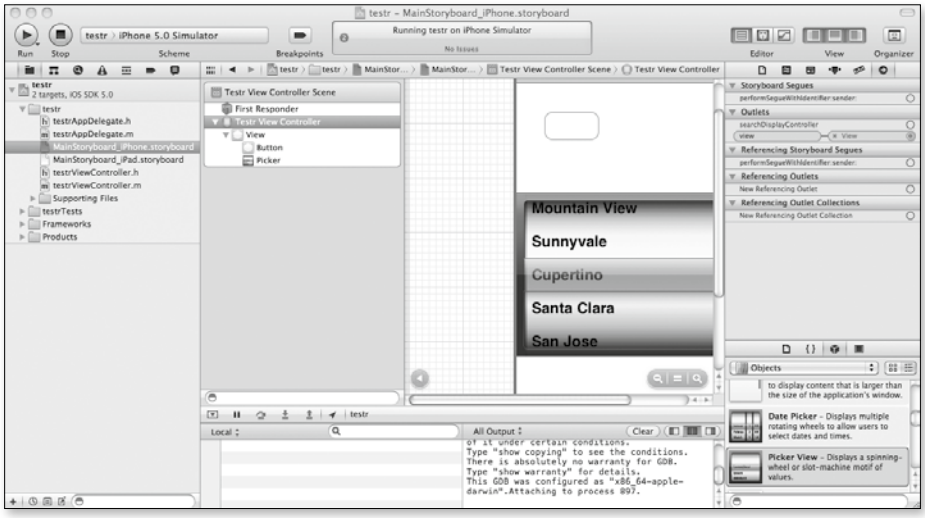
Twitter Integration

Durch die Twitter-API besteht die Möglichkeit, Twitter-Tweets aus den Anwendungen zu versenden. Dabei können die bestehenden Adressdaten, der Standort und Fotos in die Tweets eingebunden werden. Planen Sie eine App, die Kurznachrichten mit Fotos versenden soll, können Sie sich der Twitter-API bedienen. Das neue Framework heißt »Twitter.Framework«. Die Twitter-API sorgt dabei für den Inhalt der Twitter-Nachricht. Das Zusammenstellen der Twitter-News erfolgt über die TWTweetComposeViewController-Klasse. Diese Klasse ist ein View-Controller, der es dem Benutzer erlaubt, seine Nachricht vor dem Versenden zu bearbeiten.

Storyboards

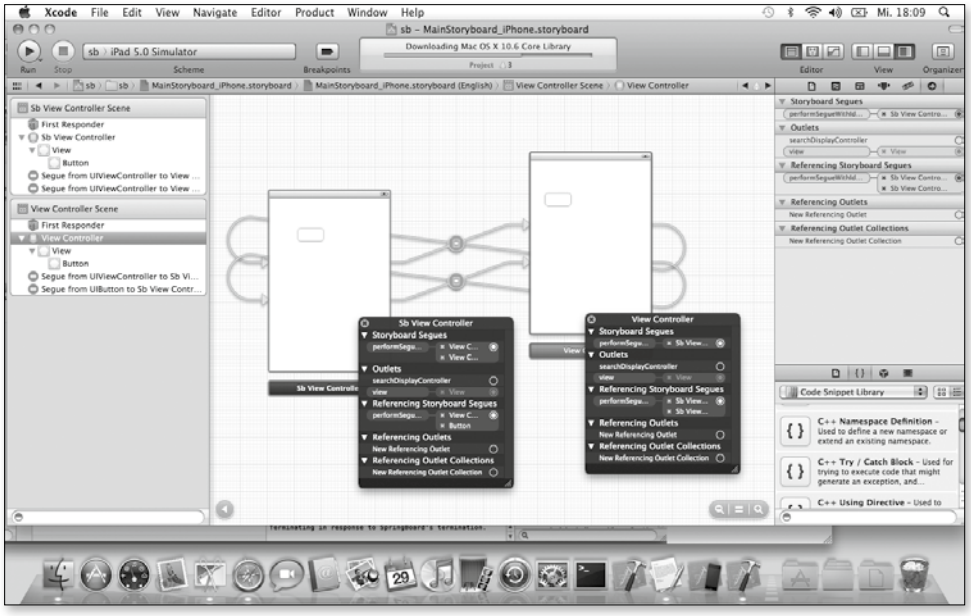
Ein Storyboard wurde in die Design-Tools von Xcode eingebaut. Es erleichtert das Handling mit dem View-Controller, so dass das Hin- und Herschalten zwischen den Ansichten vereinfacht wird. Dies gilt auch für das Handling von Reitern. Es ist eine Erleichterung für den Programmierer, der die Steuerung der einzelnen Views nun nicht mehr selber programmieren muss.

Ein Storyboard ist in Xcode 4.2 als Programmiervorlage enthalten. Starten Sie in Xcode 4.2 ein neues Projekt. Sie sehen links im Projektbaum, dass dort zwei neue Dateien existieren, MainStoryboard_iPhone.storyboard und MainStoryboard_iPad.storyboard. Wenn Sie auf eine der Dateien klicken, springt ein Fenster, wie in der Abbildung unten dargestellt, auf. In dieser Arbeitsumgebung haben Sie den kompletten Zugriff auf die Steuerung Ihrer Views. Im rechten Teil des Fensters gestalten Sie die Übergänge zwischen den einzelnen Views. Die Storyboard-Segues können Sie durch Klicken und Ziehen auf die einzelnen Views bewegen und so ganz einfach eine Steuerung erzeugen.



Storyboard-Bearbeitung in Xcode4.2

In der nächsten Abbildung sehen Sie das Erstellen von zwei Views mit dem neuen Feature Storyboard. Die Verbindungen der Views werden grafisch dargestellt. In der linken Seite des Fensters haben Sie die Übersicht darüber, welche Elemente und Verbindungen in den einzelnen Views zum Einsatz gebracht wurden.



Die Elemente des Storyboards

AirPlay

Diese API ermöglicht das Spiegeln von Inhalten des iPad 2 auf HDTV-Bildschirmen mittels Apple TV. Apps, die das AV Foundation Framework eingebunden haben, können diesen Dienst nutzen. Dadurch wird das iPad 2 zur Medienzentrale. Spezielle Apps, die diese Schnittstelle für Vorträge und Präsentationen nutzen, sind dadurch möglich.

Programmierer, die das AV Foundation Framework benutzen, binden die Klasse AVPlayer im Code ein. Diese Klasse ermöglicht das Streamen von Audio Video-Daten über AirPlay. Die UIWebView-Klasse unterstützt nun auch die Präsentation von Multi-Mediainhalten über AirPlay.

Core-Image

ermöglicht das Optimieren von Fotos und Videos. Das Framework beschleunigt die Hardware und bietet eine Anzahl von Filtern für Farbeffekte, Verzerrungen und Übergänge. Es bietet in den erweiterten Funktionen Rote-Augen-Reduktion und Gesichtserkennung an.

Über die CIImage class können verschiedene Filter aufgerufen werden, die die Qualität von Fotos verbessern. Diese Filter können erweitert und angepasst werden. Der Entwickler findet weitere Informationen in der CoreImageReference Collection.

OpenGL-ES

Dieses Framework wurde ebenfalls überarbeitet. Es ist optimiert für die Spiele-Entwicklung durch erweiterte Rendering- und Textur-Techniken. Die neuen Features setzen auf Schnelligkeit im Bildaufbau und nutzen dafür hardwarebeschleunigende mathematische Operationen. Ein OpenGL ES-Debugger ist in Xcode integriert und untersucht speziell OpenGL-Code.

Location-Simulation

Der neue Simulator verspricht, für standortbasierte Funktionen Geodaten zur Verfügung zu stellen, so dass Sie diese nun im Simulator testen können.

Kapitel

8

Ein Programm für alles: Xcode 4

avaxhome & boerse.bz

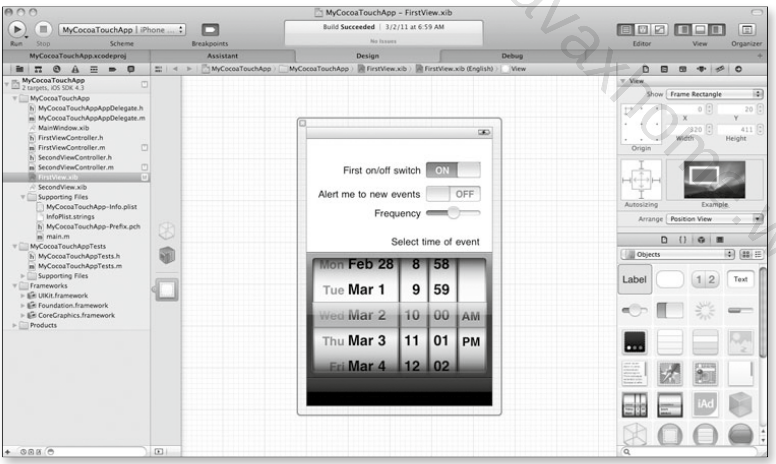
Noch während dieses Buch in Arbeit war, wurde Xcode 4.1 veröffentlicht. Das Programm ist nun, im Vergleich zu seinen Vorgängerversionen, nicht mehr kostenlos. Besonders teuer ist es aber auch nicht: Es lässt sich im App Store http://itunes.apple.com/de für 3,99 US-Dollar laden. Für Mitglieder des kostenpflichtigen Dev-Programms, das 99 US-Dollar im Jahr kostet, ist Xcode 4 sogar nach wie vor gratis. Auch beim Kauf eines Macs wird das Programm nicht mehr auf DVD ausgeliefert. Entwickler bekommen es unter www.developer.apple.com.
Die Systemvoraussetzungen für das neue Entwicklerprogramm Xcode 4.1 sind ein Intel-Mac und OS X 10.7 Lion. Das Installationspaket ist 4,1 GB groß.
Xcode 4 öffnet problemlos alle Projekte, die unter Xcode 3.x angelegt wurden.

Die neuen Features

Xcode 4 bietet dem Entwickler viele neue Features, die das Projekthandling erleichtern. Das Programm ist zusammen mit dem Interface Builder in einem Fenster untergebracht. Der Kompiler läuft schneller und die Fehlerbehandlung wird einfacher.

Single Window

Öffnet man das neue Xcode, erscheint ein Single-Fenster. Es beinhaltet die Entwicklerumgebung Xcode und, wie bereits erwähnt, den Interface Builder. Apple vereint somit in Version 4 alle Entwicklungsumgebungen.



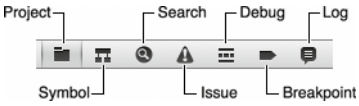
Der Interface Builder ist in Xcode integriert.

avaxhome.ws & boerse.bz

Die Drag & Drop-Funktionalität des Interface Builders wurde erweitert. So erlaubt es das Programm, wie gewohnt die *UI-Elemente* auf das View zu ziehen. Neu ist die Möglichkeit, grafische Elemente mit dem Code zu verbinden. So lassen sich Instanzvariablen in der *.h*-Datei schnell in Code umwandeln. Im unteren Beispiel wird eine Variable vom Typ *UI-Switch* mit dem Namen *MySwitch* in den Quelltext eingefügt.

Die einzelnen Objekte werden in Code umgewandelt.

Die einzelnen Programmteile werden über *Navigators* angesteuert. Sie befinden sich in der linken oberen Ecke von Xcode. Wenn Sie einen Navigator anwählen, wird zu dem entsprechenden Programmteil verzweigt. So kann beispielsweise der Debugger gestartet oder ein Suchbegriff verwendet werden.



Die Navigators im neuen Xcode

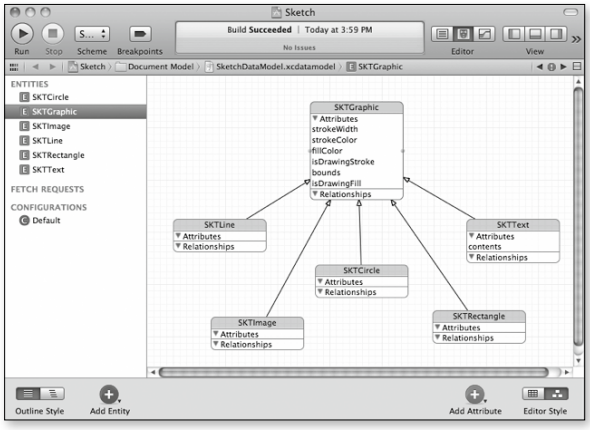
Jump-Bar

Dem Benutzer steht eine Jump-Bar zur Verfügung, mit der nun die Verzeichnisstruktur des Projektbaumes angeklickt werden kann. Sie erinnert an die Navigationsbar des Finders oder des Explorers unter Windows 7. Die Jump-Bar zeigt dem Benutzer, auf welcher Hierarchieebene des Projektes er sich gerade befindet. Diese Art der Navigation bezeichnet man auch als »Brotkrümel-Navigation«, weil sie wie Brotkrümel aneinander gereiht dem Benutzer einen Pfad aufzeigt.

Die Jump-Bar dient der Navigation durch den Projektbaum.

Assistant

Ein weiteres Werkzeug, welches dem Entwickler die Arbeit leichter macht, ist der *Assistant*. In ihm wird die logische Struktur des Programms dargestellt. Das Erscheinungsbild des Assistants verändert sich, je nachdem, welche Inhalte Sie abfragen. In der unteren Abbildung zeigt er etwa eine Konzeption zu einem Datenbankmodell an. Verändert man ein Objekt oder eine Funktion, wird die Veränderung grafisch dargestellt. Wenn Sie eine Klasse aus einem Framework in Ihren Code einbeziehen wollen, zeigt der *Assistant* Ihnen den dazugehörigen Quelltext.



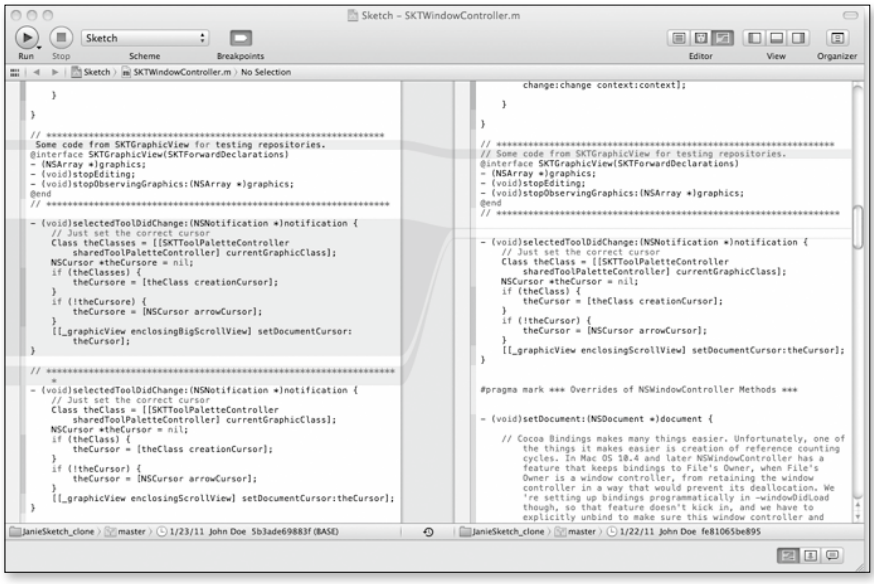
Der Assistant in Aktion

angelfire.ws & bgerse.bz

Version-Editor

Mit dem neuen Version-Editor ist es möglich, zwei beliebige Versionen Ihres Quelltextes nebeneinander zu betrachten. Dabei können Sie bei einem Dokument in der Zeit zurückreisen: Durch einen Schieberegler zwischen den Dokumenten veranlassen Sie einen Rückblick auf den Projektstand, der in der Vergangenheit liegt.

Im Editor können Sie zwei Versionen des Quelltextes miteinander vergleichen.



Hier haben Sie den Überblick über zwei Versionen Ihres Codes im neuen Editor.

avaxhome.ws & boerse.bz

Der Compiler und der Debugger

Der neue Compiler, der Open-Source-Compiler LLVM 2.0, soll doppelt so schnell arbeiten wie sein Vorgänger. Er unterstützt dabei C, C++ und Objective-C. Er arbeitet permanent im Hintergrund und überprüft währenddessen den geschriebenen Code. Kommt es zu Fehlern bei der Eingabe des Codes, werden während des Schreibens Verbesserungsvorschläge eingeblendet. Die Fachausdrücke dafür heißen *Fix-it* und *Live-Issues*.

Der neue Debugger des Compilers verspricht dabei, Fehler selber zu lösen. Wenn das nicht funktioniert, werden intelligente Vorschläge zum Beheben der Fehler eingeblendet.

Vorschläge des Debuggers zur Fehlerbeseitigung

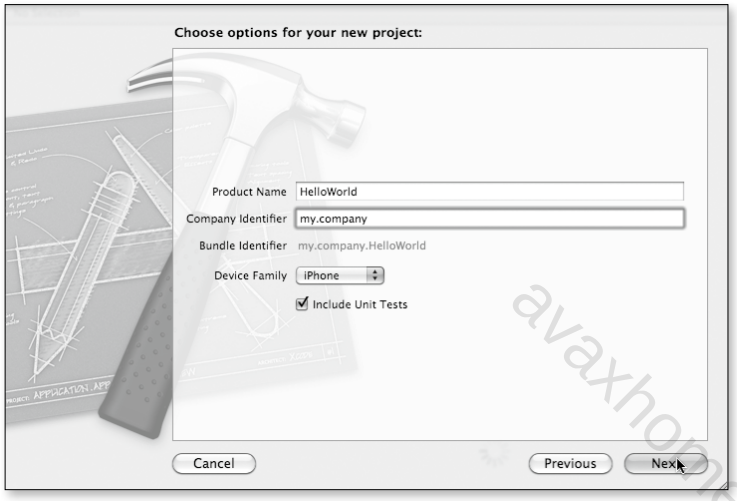
Xcode 4: der Schnell-Start

Starten Sie Xcode und führen Sie die ersten Schritte wie gewohnt durch. Im ersten Fenster wählen und benennen Sie ein Xcode-Projekt und speichern es unter einem Projektnamen ab. Danach springt ein weiteres Fenster auf. Dort treffen Sie die Auswahl für die Art des Projektes und entscheiden, für welche Plattform Ihr Projekt sein soll. Bis hier hin nichts Neues.

Das Startfenster beinhaltet keine Neuerungen.

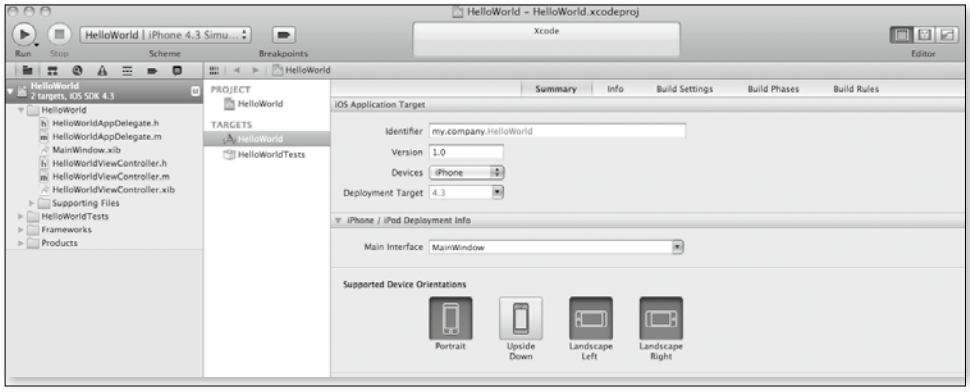
Ein Programm für alles: Xcode 4

Danach haben Sie die Möglichkeit, das Projekt abzuspeichern. Vorher, und das ist neu, können Sie einen Produktnamen und Ihren Company-Identifier (Ihre Firmenkennung) eintragen. Danach klicken Sie unten im Fenster auf *Next*.



Der Company- Identifier kann hier eingegeben werden.

Nach dieser Vorarbeit gelangen Sie in die neue Benutzerumgebung von Xcode. Sie erinnert ein wenig an iTunes. Sie sehen links den Projektbaum mit allen bereits angelegten Dateien. Darüber finden Sie die *Navigators*, mit denen Sie die einzelnen Programmteile starten können. Im rechten Teil des Fensters bekommen Sie Informationen über Ihr Projekt angezeigt. Aktuell sehen Sie hier alle Infos zu den iOS-Targets.

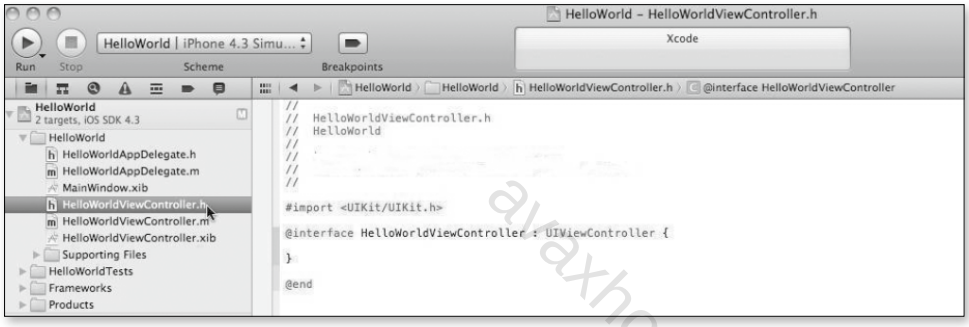


Weitere Infos können unter dem Menüpunkt Targets eingegeben werden.

avaxhome.ws & bigwise.bz

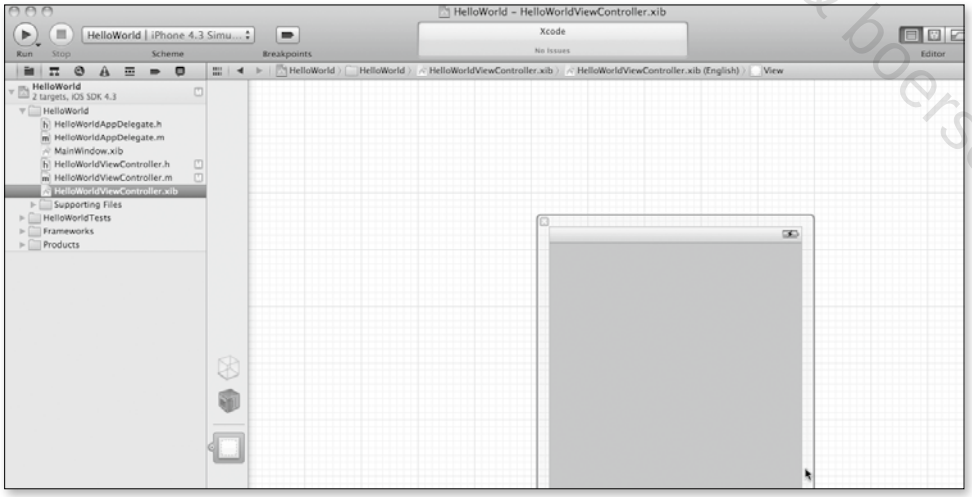
Wenn Sie Code eingeben wollen, klicken Sie einfach in die linke Hälfte des Projektbaums. Hier klicken Sie auf die Datei *HelloWorldViewController.h*. Danach startet automatisch der Editor.

In der Leiste über dem Editor sehen Sie die *Jump-Bar*. Mit ihr können Sie zusätzlich im Projektbaum navigieren.



Das Fenster zur Quellcode-Eingabe

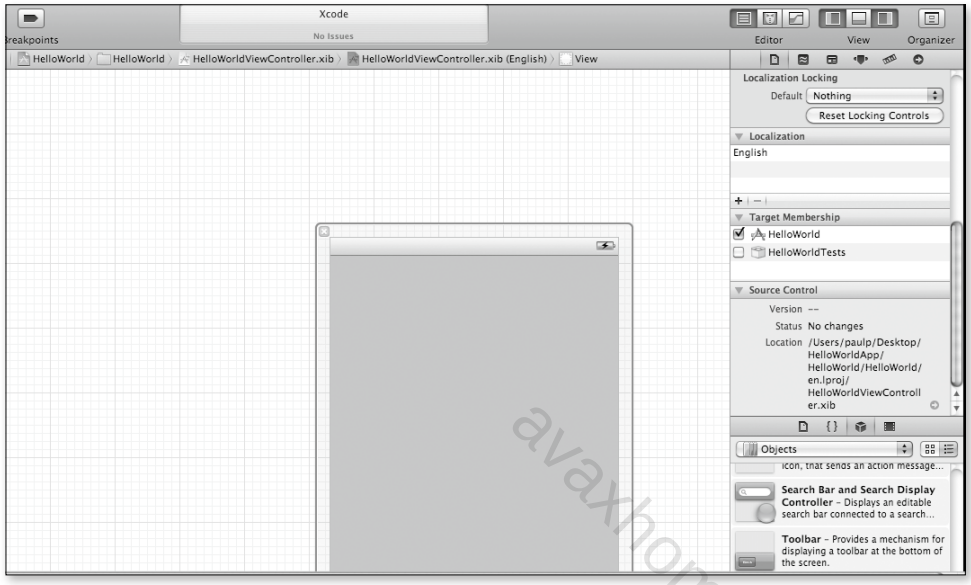
Um nach der Quellcodeeingabe den Interface Builder aufzurufen, klicken Sie einfach auf die Datei *HelloWorldViewController.xib*.



Der Interface Builder im Single-Window

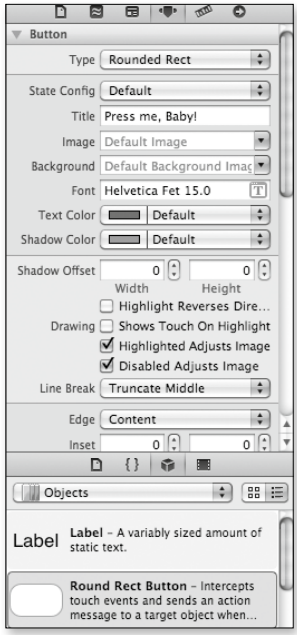
Die einzelnen Elemente des Interface Builders lassen sich in der oberen Leiste anwählen. Die Library befindet sich im gleichen Fenster. Dort können die einzelnen Objekte wieder ausgewählt und durch Klicken und Ziehen auf das View bewegt werden.

avaxhome.ws & boerse.bz



Die Library und die Inspectors sind im rechten Teil des Fensters untergebracht.

In der unteren Abbildung sehen Sie den ausgewählten *Attribute-Inspector*. Darunter schließt sich die Library an. In der Menüleiste darüber wählen Sie die *Inspectoren* über die *Inspector-Selector-Bar* aus.

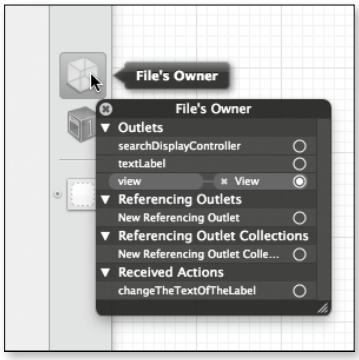


Der Attribute-Inspector

Ein Programm für alles: Xcode 4

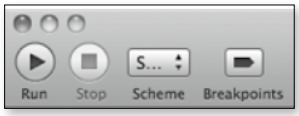
avaxhome.ws@boerse.bz

Im nächsten Schritt legen Sie die Connections fest. Dazu brauchen Sie den *Files-Owner*. Mit ihm werden *Aktionen* und *Outlets* mit den View-Elementen verbunden. Das *Files-Owner-Menu* wird im Interface Builder an der linken Seite des Fensters vor dem Projektbaum in einem Dock aufgerufen. Durch einen Klick in die runden *Radio-Buttons* stellen Sie dann die Verbindung zu den View-Elementen her.



Der Files-Owner erscheint in einem Dock neben dem Projektbaum.

Nach vollbrachter Arbeit können Sie Ihr Projekt kompilieren. Dazu klicken Sie auf den *Run-Button* links oben im Hauptfenster, der den Debugger startet.



Der Run-Button startet den Debugger.

avaxhome.ws & boerse.bz

avaxhome.ws & boerse.bz

Workshop

1

Der Krieg der Knöpfe

avaxhome & boerse.bz

In diesem Workshop lernen Sie, wie man Benutzereingaben mit Buttons steuert und kanalisiert und wie Sie die verschiedenen Steuerelemente in Xcode und dem Interface Builder anlegen. Im zweiten Teil erhalten Sie ein mächtiges Werkzeug in Form einer App aus der iOS-Library, die Ihnen eine komplette Übersicht über alle UI-Buttons und die UI-Klasse bietet. UI steht für User Interface. Dieses Framework beinhaltet alle Klassen der Benutzerführung. Das sind Schaltflächen, Eingabefelder und Steuerungselemente, die der Benutzer braucht, um eine App zu bedienen und Daten einzugeben.

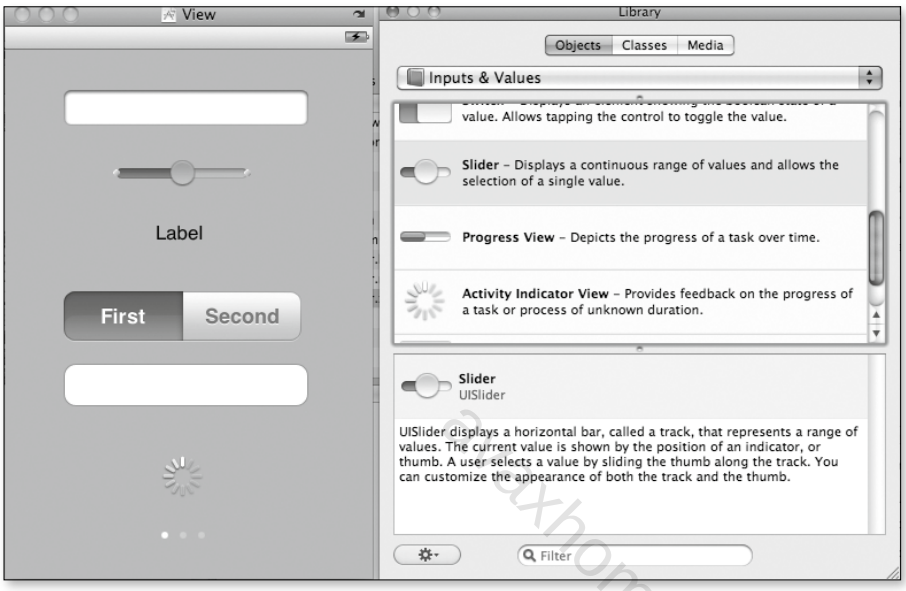
Der Krieg der Knöpfe

UI Buttons und Eingabefelder erstellen

1. Starten Sie ein neues *Xcode-Projekt* auf der Grundlage *View-based Application*. Nennen Sie die Datei Knopf. Wenn Ihr Xcode-Projektfenster erscheint, starten Sie den Interface Builder, indem Sie auf die Datei *KnopfViewController.xib* doppelklicken. Wenn der Interface Builder gestartet ist, öffnen Sie die Library und anschließend darin den Ordner *Input & Values*.

2. Im Order *Input & Values* erhalten Sie eine Auswahl an Eingabefeldern, Fortschrittsbalken und Buttons. Durch Klicken & Ziehen platzieren Sie die einzelnen Felder auf der *View-Umgebung*. Stellen Sie die Umgebung im *View*, wie in der unteren Abbildung zu sehen, zusammen. Wenn Sie ein Element in der Library anwählen, wie hier den *Slider*, erhalten Sie im unteren Fenster der Bibliothek eine kurze Zusammenfassung über die Eigenschaften des Objektes.

avexhome.ws & boerse.bz



3. Nachdem Sie die einzelnen Elemente auf dem *View* zusammengestellt haben, springen Sie in Xcode zurück und öffnen die Datei *KnopfViewController.h*. In dieser Header-Datei legen Sie Instanzvariablen der Objekte an, die Sie vorher im Interface Builder angelegt haben. Die Instanzvariable gehört zu dem Objekt, welches Sie vorher im Interface Builder erzeugt haben. Sie wird nur dem Objekt zugeordnet, das vorher angelegt wurde. Es genügt, wenn Sie diese drei Variablen definieren:

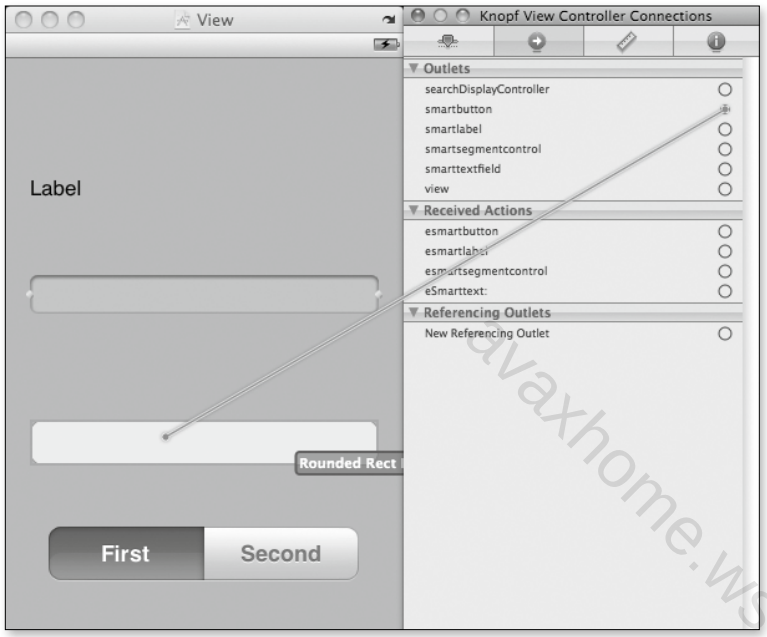
```
#import <UIKil/UIKit.h>
@interface knopfViewController: UIViewController {
    IBOutlet UITextfield *smarttextfield;
    IBOutlet UILabel *smartlabel;
    IBOutlet UIButton*smartbutton;
}
@end
```

4. Nach der Eingabe der Variablen speichern Sie diese in Xcode ab und springen in den Interface Builder zurück. Dort verknüpfen Sie die Steuerelemente und die angelegten Instanzvariablen. Die Variablen sind im Connection-Inspector aufgelistet. Neben der Bezeichnung ist ein runder Kreis zu sehen. Dort ist Ihr Startpunkt. Sie sehen in der Abbildung, wie der Smartbutton mit dem *Round Rect Button* miteinander verbunden wird. Sie können den Smartbutton nicht mit dem Label verbinden. Verbindungen lassen sich nur erstellen, wenn die Instanzvariable zu dem Steuerelement im *View* passt.

galileocomputing.de/buecherse.bz

Der Krieg der Knöpfe

Verbinden Sie nun noch die Instanzvariablen *smartlabel* und *smarttextfield* mit den Steuerelementen im *View*.



5. Nachdem die Verknüpfung stattgefunden hat, können Sie schon einiges mit den Steuerelementen machen. Kompilieren Sie Ihr Projekt in Xcode mit *Build and Run* und schauen Sie sich im iPhone-Simulator an, was bis jetzt geschehen ist. Wenn Sie in das Textfeld klicken, springt die Tastatur an. Der Button ändert die Farbe, wenn Sie ihn berühren. So weit so gut. Die Steuerelemente liefern aber noch keine Daten an das Programm zurück. Um das zu veranlassen, müssen noch Ereignisbehandlungsroutinen programmiert werden. Springen Sie dazu nach Xcode zurück und öffnen Sie die Header-Datei mit der Endung .h. Dort geben Sie den untenstehenden Code ein. Die Deklarationen fangen alle mit *IBAction* an. Der Editor ergänzt übrigens das Wort, während Sie schreiben. Hat er das richtige Wort gefunden (in diesem Fall *IBAction*), können Sie einfach die Return-Taste drücken und sich somit Tipparbeit sparen.

```
#import <UIKil/UIKit.h>
@interface knopfViewController: UIViewController {
    IBOutlet UITextfield *smarttextfield;
    IBOutlet UILabel *smartlabel;
    IBOutlet UIButton*smartbutton;
}
-(IBAction) eSmarttext;
```

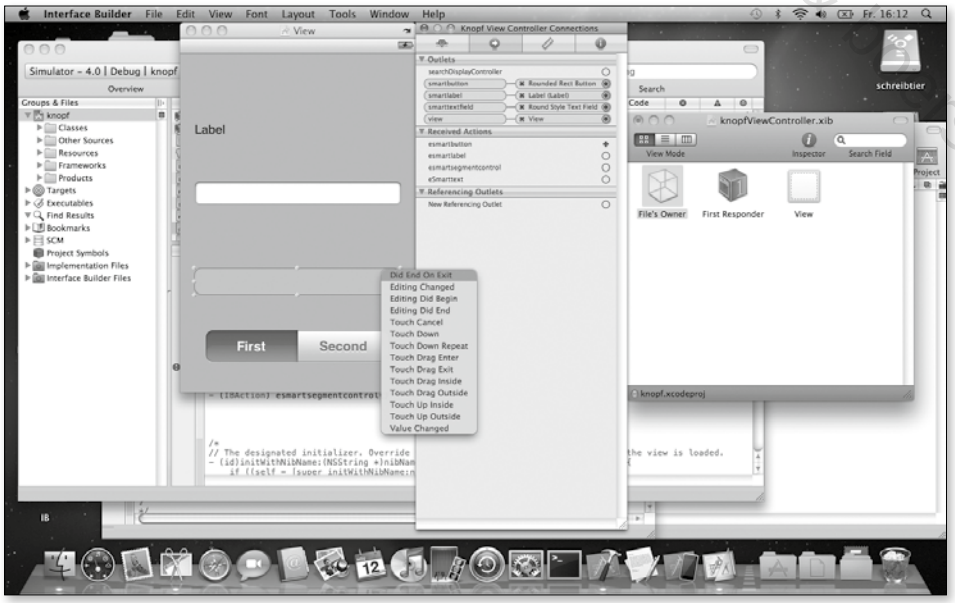
javaxhome.ws & kverse.bz

```
-(IBAction) esmartlabel;
-(IBAction) esmartbutton;
-(IBAction) esmartsegmentcontrol;
@end
```

6. Die Variablen müssen Sie dann noch in der *.h-Datei* Ihres Projektes deklarieren:

```
-(IBAction) eSmarttext {}
-(IBAction) esmartlabel {}
-(IBAction) esmartbutton {}
-(IBAction) esmartsegmentcontrol {}
@end
```

7. Im letzten Schritt werden die Ereignisroutinen miteinander verknüpft. Springen Sie dazu wieder in den Inteface-Builder. Rufen Sie den Connection-Inspector auf. Die vorher deklarierten Variablen tauchen nun auch im Feld *Received Actions* auf. Verbinden Sie dann wieder die Behandlungsroutinen mit den Schaltflächen. Wenn Sie die Maustaste loslassen, erscheint ein kleines Menü, in dem Sie mögliche Ereignisse auswählen können. Sie können so beispielsweise einem Button ein bestimmtes Verhalten bei der Berührung zuordnen.



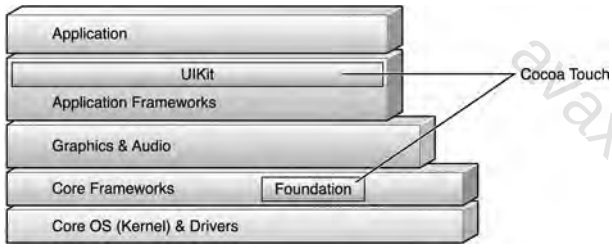
Wenn Sie mit der Eingabe fertig sind, haben Sie ein allgemeingültiges Konzept für Ihr Projekt erstellt. Es bestehen alle Verbindungen zwischen Variablen und

Der Krieg der Knöpfe

Steuerelementen, die nötig sind, um nun mittels Programmcode den Benutzer zu führen und Berechnungen anzustellen.

Das UIKit-Framework

Im vorherigen Beispiel haben Sie das UIKit-Framework näher kennengelernt. Es ist ein mächtiger Bestandteil der Cocoa-Touch-Programmierumgebung. In der unteren Abbildung sehen Sie, wie es in das Schichtenmodell des iOS eingebettet ist.



Einbettung des UIKit-Framework in das iOS; Quelle: Apple, iOS-Library

awaxhome.ws & kbverse.bz

Mit dem UIKit wird das Benutzerverhalten gesteuert. Es ist verantwortlich für alles, was mit der grafischen Darstellung der App zu tun hat. Um Ihnen einen kompletten Überblick darüber zu verschaffen, was damit gemeint ist, schauen Sie sich bitte das *UIKit* in Form einer App an.

Laden Sie sich hierzu unter dem Link

http://developer.apple.com/library/ios/#samplecode/UICatalog/Introduction/Intro.html

den Ordner *UI Catalog* herunter. Starten Sie das Xcode-Projekt und kompilieren Sie es gleich. Danach sehen Sie den UI-Catalog im iPhone-Simulator. Im Startfenster erfahren Sie, was das *UI-Kit* bietet. In Xcode sehen Sie zu den einzelnen Beispielen den Quellcode und erhalten so den Überblick darüber, wie welche Beispiele programmiert worden sind.

Der UI-Catalog im Überblick

Ein paar von diesen Steuerelementen wie *Buttons*, *Controls* oder *Textfields* haben Sie schon kennengelernt. Das *UIKit* beinhaltet aber auch das Handling mit Fotos und deren grafischen Übergängen. *Transitions* sind spezielle Methoden in einem View, durch das ein Bild gegen ein anderes ausgetauscht wird. Dem Programmierer stehen dabei die Funktionen *Drehen* und *Umblättern* zur Verfügung.

Das UIKit steuert auch durch Transitions grafische Übergänge in einem Fotokatalog.

avaxhome.ws/e-boerse.bz

Der Krieg der Knöpfe

Die Klasse der Picker

Ein weiteres großes Feld des *UIKits* sind die Picker. Der Picker ist eines der schönsten Bedienelemente, die es in dem Kit gibt. Das Bedienelement finden Sie natürlich auch in der Library des Interface Builders. Er wird via Drag&Drop auf das View gezogen. Mit einem Flick bringt man die Walze ins Rotieren. Das grau unterlegte Element gilt dann als ausgewählt. Sie können den Picker im Source-Code durch die Protokolle *UIPickerViewDelegate* und *UIPickerView* ansprechen. Sie finden dazu noch einen Workshop in diesem Buch, der speziell die Picker behandelt.

Picker View aus der Kategorie UIDatePicker

avaxhome.ws & boerse.bz

Die Klasse der Controls

Die nächste Kategorie des UIKits ist die der Controls. Hiermit sind Schieberegler, mehrteilige Schaltflächen und sonstige Steuerelemente gemeint. Das Definieren und Ansprechen der Controls haben Sie in diesem Workshop schon kennengelernt. Sie sind in der Vaterklasse UIControl zusammengefasst. Die Steuerelemente reagieren sehr genau auf Events, die der Benutzer mit der Bedienung dieser Elemente auslöst.

Die Steuerelemente der UIKlasse UIControl

Die Klasse der Alerts

Alerts sind Felder, die kleine Nachrichten und Hinweise beinhalten. Dies kann ein Hinweis darauf sein, dass die Netzwerkverbindung unterbrochen wurde oder dass eine Eingabe des Benutzers falsch ist. Sie sind meistens transparent gestaltet und werden überblendet. *Alerts* enthalten einen Alarmtext und ein oder zwei Buttons.

Die Klasse UI ActionSheet

avaxhome.ws a bverse.bz

Das war ein kleiner Überblick über die UI-Classes. Die App *UI-Catalog*, die Sie zur Hand haben, stammt aus der iOS-Library. Es macht einfach Spaß, in der App zur stöbern. Es gibt dort noch anderes zu entdecken, denn Sie in ihr den gesamten Überblick über alle möglichen Steuerelemente des UI- Kits. In Xcode haben Sie darüber hinaus die Möglichkeit, sich den Quellcode der einzelnen Steuerelemente anzuschauen.

Der Krieg der Knöpfe

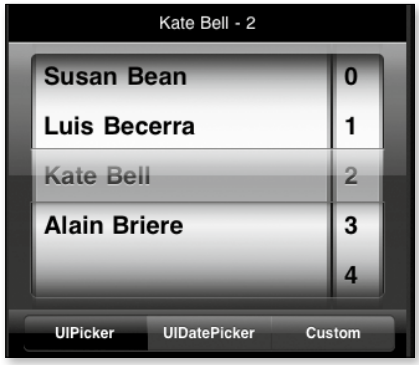
avaxhome.ws & boerse.bz

Die Picker-Bande

avaxhome & boerse.bz

Die Elemente der Picker-Bande sind eigentlich kleine Blender. Jeder Picker ist nur eine Auswahlliste in Form einer Walze. Nichts desto trotz machen die Picker optisch viel her und sind intuitiv bedienbar. Durch einen Wisch bringt man die Walze zum Rotieren. Im grau unterlegten Feld kommt die Walze zum Stehen. Der Inhalt gilt dann als ausgewählt. Die Picker erinnern an die einarmigen Banditen in den Casinos. Sie eignen sich vorzüglich dazu, dass der Benutzer nur die Daten eingibt, die gewollt sind. Durch die klare Definition der Walzen mit einem eindeutigen Inhalt können falsche Eingaben von Daten nahezu ausgeschlossen werden.

Die Picker-Bande



Picker aus dem UI Catalog

Date-and-time-Picker aus dem UI Catalog

Einen Picker können Sie einfach aus der Library des Interface Builders auf die View-Umgebung ziehen. Es gibt zwei Arten von Pickern: Eine, in der man vordefinierte Auswahlen festlegt, und solche, in denen man Datum und Uhrzeit verarbeitet.

Die Programmierroutinen finden Sie in diesen Protokollen in der iOS-Reference-Library unter den folgenden Schlagwörtern:

UIPickerViewDelegate Protocol Reference

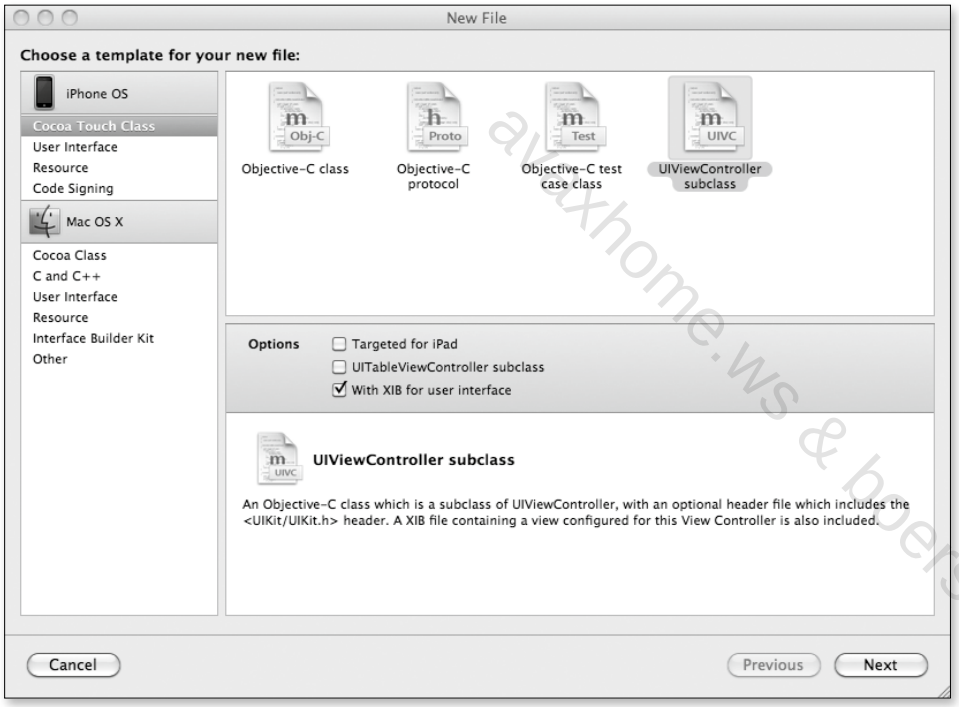
UIPickerViewDataSource Protocol Reference

UIDatePicker Class Reference

awaxhome.ws & boerse.bz

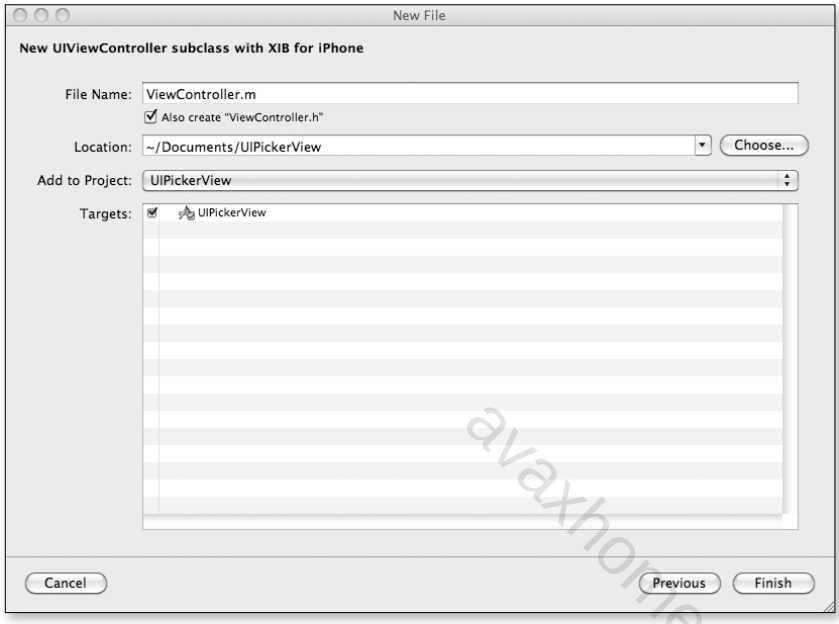
Im Folgenden werden Sie einen simplen Picker erstellen, der die Fibonacci-Zahlen in der Walze anzeigt und die ausgewählte Zahl auf ein Feld ausgibt.

1. Dazu legen Sie in Xcode ein *Windows-Based Application*-Projekt an. Benennen Sie das Projekt als *UIPickerView*. Wenn sich Ihr neues Xcode-Projekt aufgebaut hat, wählen Sie aus dem Menü *Cocoa Touch Class* die Datei *UIViewController subclass* aus. Unter *Options* wählen Sie *With XIB for User interface*. Quittieren Sie danach mit *Next*.

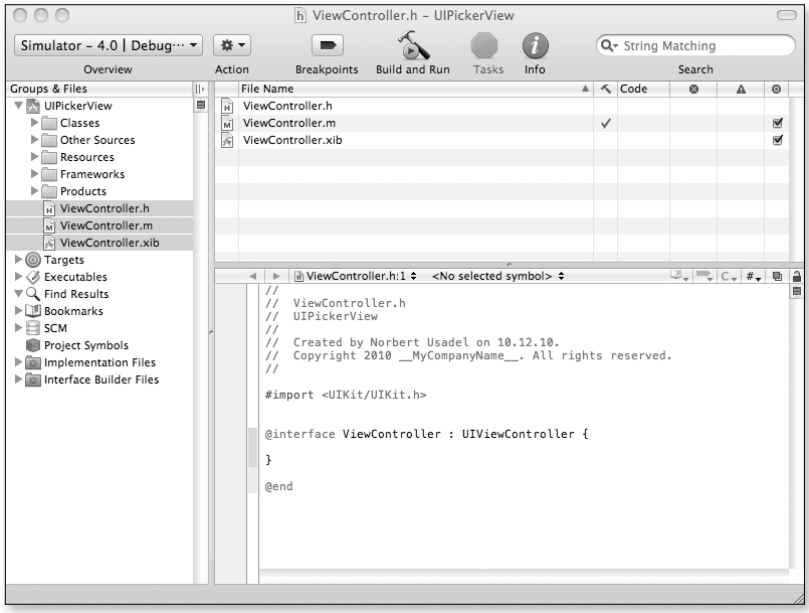


2. Benennen Sie nun diese Datei in *ViewControler.m*.

avaxhome.ws & boerse.bz



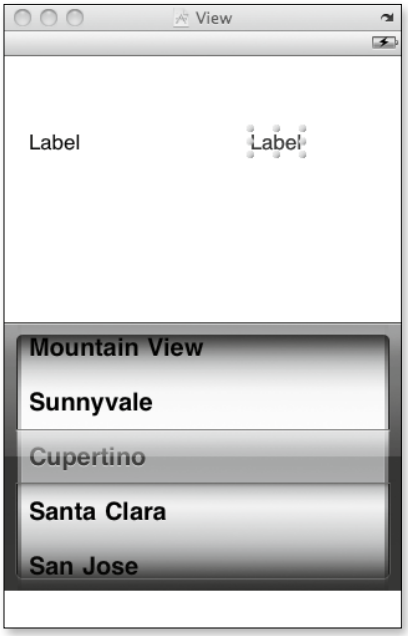
3. Sie sehen nun, dass Xcode das neue Projekt mit den Dateien *ViewController.h*, *ViewController.m* und *ViewController.xib* angelegt hat. In diese Dateien geben Sie nun den Quelltext ein.



Die Quelltexteingabe findet in den von Ihnen angelegten Dateien statt.

avaxhome.ws & boerse.bz

4. Im nächsten Schritt starten Sie per Doppelklick auf die Datei *ViewControler.xib* den Interface Builder. Dort positionieren Sie in der View-Umgebung zwei Label-Felder und den Picker. Die Elemente ziehen Sie aus der Library auf das *View*.



5. Danach speichern Sie die Datei im Interface Builder ab und springen nach Xcode zurück. Dort definieren Sie das Label, ein *NSMutableArray*, und die *UIPickerView*-Methode. Geben Sie nun in der Datei *Viewcontroler.h* folgenden Code ein:

```
//
//  ViewController.h
//  UIPickerView
//
//  Created by Norbert Usadel on 31/07/10.
//

#import <UIKit/UIKit.h>


@interface ViewController : UIViewController <UIPickerView-
Delegate, UIPickerViewDataSource>
{
```

avaxhome.ws & boerse.bz

```
    IBOutlet UILabel *mlabel;
    NSMutableArray *arrayzahl;
    IBOutlet UIPickerView *pickerView;

}
@property (nonatomic, retain) UILabel *mlabel;
@end
```

6. Im nächsten Schritt geben Sie in der Datei *ViewController.m* folgenden Code ein. Die Ausgabe der Fibunacci-Zahlen definieren Sie in einem Array. Dies können Sie jederzeit erweitern, indem Sie eine neue Zeile mit Code hinzufügen.

avaxhome.ws & boerse.bz

```
//
//  ViewController.m
//  UIPickerView
//
//  Created by Norbert Usadel on 31/07/10.
//

#import "ViewController.h"

@implementation ViewController
@synthesize mlabel;

- (void)viewDidLoad
{
    [super viewDidLoad];
    arrayzahl = [[NSMutableArray alloc] init];
    [arrayzahl addObject:@" 1 "];
    [arrayzahl addObject:@" 1 "];
    [arrayzahl addObject:@" 2 "];
    [arrayzahl addObject:@" 3 "];
    [arrayzahl addObject:@" 5 "];
    [arrayzahl addObject:@" 8 "];
    [arrayzahl addObject:@" 13 "];
    [arrayzahl addObject:@" 21 "];
    [arrayzahl addObject:@" 34 "];
    [arrayzahl addObject:@" 55 "];
    [arrayzahl addObject:@" 89 "];
    [arrayzahl addObject:@" 144 "];
    [arrayzahl addObject:@" 233 "];
    [arrayzahl addObject:@" 377 "];
```

```
    [arrayzahl addObject:@" 610  "];

    [pickerView selectRow:1 inComponent:0 animated:NO];
    mlabel.text= [arrayzahl objectAtIndex:[pickerView selected-
    RowInComponent:0]];
}

- (NSInteger)numberOfComponentsInPickerView:(UIPickerView *)
  pickerView;
{
    return 1;
}

- (void)pickerView:(UIPickerView *)pickerView didSelectRow:(
  NSInteger)row inComponent:(NSInteger)component
{
    mlabel.text=  [arrayzahl objectAtIndex:row];
}

- (NSInteger)pickerView:(UIPickerView *)pickerView numberOfRows-
  InComponent:(NSInteger)component;
{
    return [arrayzahl count];
}

- (NSString *)pickerView:(UIPickerView *)pickerView titleFor-
  Row:(NSInteger)row forComponent:(NSInteger)component;
{
    return [arrayzahl objectAtIndex:row];
}


/*
 // The designated initializer.  Override if you create the con-
    troller programmatically and want to perform customization
    that is not appropriate for viewDidLoad.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:
  (NSBundle *)nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil bundle:nib-
    BundleOrNil]) {
        // Custom initialization
    }
```

awxhome.ws & boerse.bz

Die Picker-Bande

```
    return self;
}
*/

/*
// Implement viewDidLoad to do additional setup after loading the
   view, typically from a nib.
- (void)viewDidLoad {
    [super viewDidLoad];
}
*/

/*
// Override to allow orientations other than the default portrait
   orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterface-
  Orientation)interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientation-
    Portrait);
}
*/

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload
{
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}


- (void)dealloc {
    [super dealloc];
}

@end
```

avaxhomeews & boerse.bz

7. Springen Sie nun in den Interface Builder zurück, indem Sie auf die Datei *ViewController.xib* doppelklicken. Sie bestimmen nun die Verbindungen der drei Felder im View. Dort verknüpfen Sie die *Outlets* der einzelnen Felder. Die Outlets erscheinen dort, weil sie vorher im Quelltext definiert worden sind. Sie stehen dann im Interface Builder zur Verfügung. Verknüpfen Sie *mlabel* mit *File´s Owner* und die drei Felder des Pickers wie in der Abbildung.

8. Nun springen Sie zurück nach Xcode, öffnen die Datei *UIPickerAppDelegate.m*. und geben diese Zeilen Code ein:

```
mviewController = [[ViewController alloc] initWithNib-
Name:@"ViewController" bundle:[NSBundle mainBundle]];
[window addSubview:mviewController.view];
```

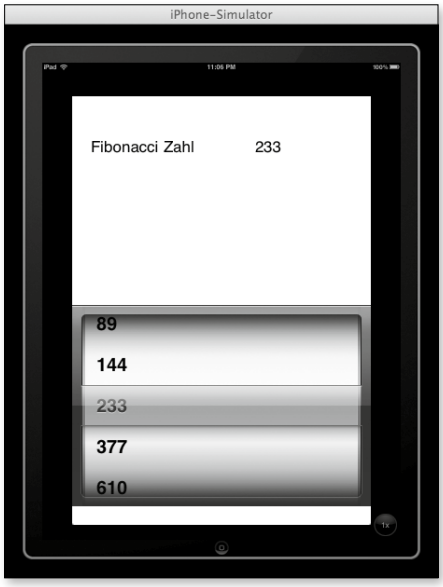
avahome.ws & boerse.bz

Der Quelltext der Datei *UIPickerAppDelegate.m* sieht dann so aus:

```
//
// UIPickerViewAppDelegate.m
// UIPickerView
//
// Created by Norbert Usadel on 31/07/09.
//

#import "UIPickerViewAppDelegate.h"
#import "ViewController.h"


@implementation UIPickerViewAppDelegate

@synthesize window;
@synthesize mviewController;

- (void)applicationDidFinishLaunching:(UIApplication *)applicati-
on
{
 mviewController = [[ViewController alloc] initWithNibNa-
me:@"ViewController" bundle:[NSBundle mainBundle]];
[window addSubview:mviewController.view];
[window makeKeyAndVisible];
}

- (void)dealloc
{
[window release];
[ViewController release];
[super dealloc];
}


@end
```

avaxhome.ws & boerse.bz

9. Nun sind Sie mit der Dateneingabe fertig. Kompilieren Sie Ihr Projekt mit *Build and Run* in der Toolbar von Xcode:



Die Auswahl der Zahlen passiert in einem Picker und wird dann oben in einem Feld ausgegeben.

NSMutableArray

Im letzten Beispiel wurde ein Array definiert. Ein Array ist eine Variable, die über einen Index angesprochen wird. Hier ist zum Beispiel eine Zahlenreihe: 2, 4, 6, 8, 10. In einem Array spricht man dann die Zahl sechs als Zahl[4]an. Das wäre in diesem Fall die Zahl 6. Das Array fängt bei Null an.

In dem Beispiel wurden in einem NS Mutable Array die Fibunacci-Zahlen gespeichert:

```
[arrayzahl addObject:@" 377 "];
```

Dadurch erscheint die Zahl 377 in der Walze des Pickers. Wenn Sie mehr Zahlen in der Walze darstellen wollen, fügen Sie einfach eine neue Zeile im Array ein. Soll die Zahl 610 auf die Zahl 377 folgen, heißt die nächste Zeile im Array

```
[arrayzahl addObject:@" 610 "];
```

UIDatepicker

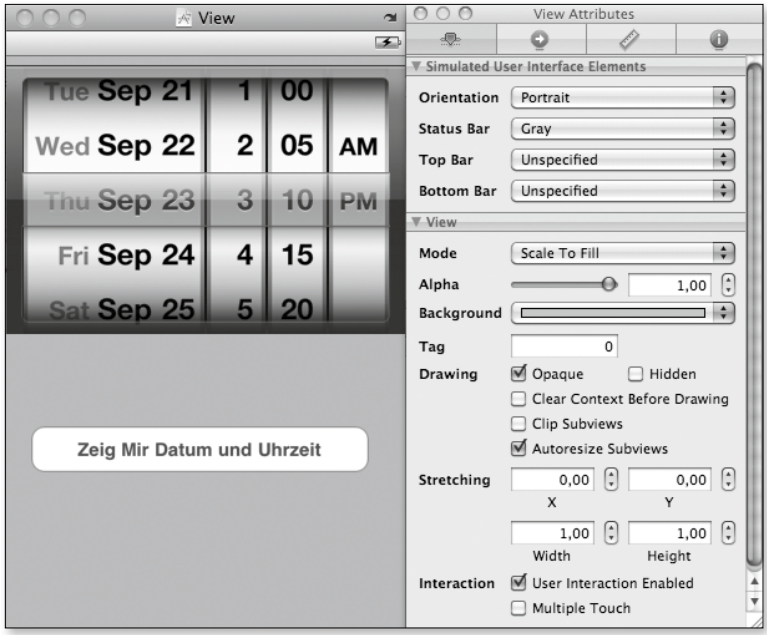
Im nächsten Beispiel lernen Sie Klasse der *UIDatepicker* kennen. In diesem Beispiel können Sie Datum und Uhrzeit auswählen und sich beides per *Alert-View* anzeigen lassen.

avaHome.wsp.boerse.bz

Die Picker-Bande

Ihr neues Projekt mit einem Date-and-Time-Picker

1. Hierzu erstellen Sie in Xcode ein *windows viewed*-Projekt. Nennen Sie die Datei *DatePickerViewController*. Danach springen Sie aus Xcode in den Interface Builder, indem Sie die Datei *DatePickerViewController.xib* doppleklicken. Dort ziehen Sie sich aus der Library einen Datenpicker und einen *Round rect Buton* auf das *View*. Beschriften Sie den Button mit *Zeig mir Datum und Uhrzeit*. Dazu klicken Sie in *Button* und geben einfach den Text ein.



avaxhome.ws/boerse.bz

2. Danach springen Sie zu Xcode zurück. Hier definieren Sie den Picker, der Ihnen zur Dateneingabe dient. Geben Sie in der *.h Datei* folgenden Code ein:

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

@interface DatePickerViewControllerViewController : UIView-
Controller {
    IBOutlet UIDatePicker *picker;
}
-(IBAction)showdate:(id)sender;

@end
```

3. In der *.m Datei* definieren Sie einen Alarmknopf vom Typ *UIAlertView* mit dem Inhalt "AUSGEWÄHTES DATUM UND UHRZEIT". Die Bestätigung heißt hier "NA GUT".

   Davor wird ein *roundrectbutton* mit dem Inhalt "Zeig mir Datum und Uhrzeit sind" definiert.

   In Quelltext umgesetzt, sieht das so aus:

```
#import "DatePickerViewControllerViewController.h"

@implementation DatePickerViewControllerViewController

-(IBAction)showdate:(id)sender
{
    NSLog(@"Button Pressed");
    NSDate *selected = [picker date];
    NSString *message = [[NSString alloc] initWithFormat:@"Datum
    und Uhrzeit sind: %@", selected];
    UIAlertView *alert = [[UIAlertView alloc] initWith-
    Title:@"AUSGEWÄHLTES DATUM UND UHRZEIT" message:message dele-
    gate:nil cancelButtonTitle:@"Na Gut" otherButtonTitles:nil];
    [alert show];
    [alert release];
    [message release];
}
```

avxhome.ws & boerse.bz

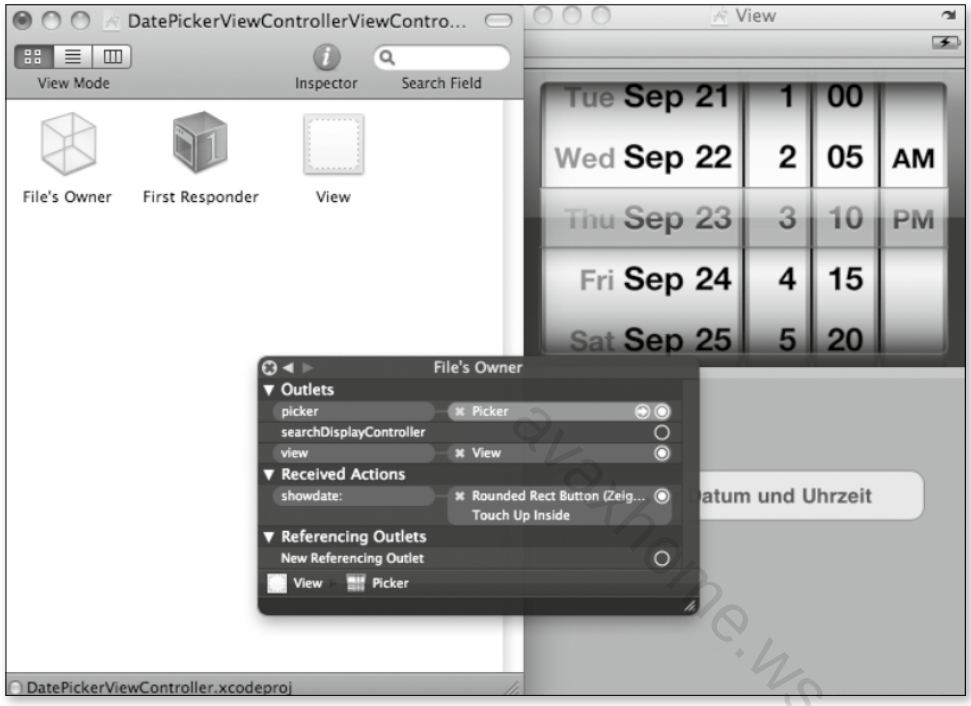
```
// The designated initializer. Override to perform setup that is
   required before the view is loaded.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)
  nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil bundle:nib-
    BundleOrNil]) {
        // Custom initialization
    }
    return self;
}


/*
// Implement loadView to create a view hierarchy programmatical-
   ly, without using a nib.
- (void)loadView {
}
*/



// Implement viewDidLoad to do additional setup after loading the
   view, typically from a nib.
- (void)viewDidLoad {
    NSDate *now = [NSDate date];
    [picker setDate:now animated:YES];
    [super viewDidLoad];
}



// Override to allow orientations other than the default portrait
   orientation.
- (BOOL)shouldAutorotateToInterfaceOrientation:(UIInterface-
  Orientation)interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation == UIInterfaceOrientation-
    Portrait);
}
```

awaxhome.ws & boerse.bz

```
- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}


- (void)dealloc {
    [super dealloc];
    [picker release];
}

@end
```

4. Nach der Eingabe des Codes sichern Sie die Dateien in Xcode und springen zum Interface Builder zurück. Dort legen Sie die Connections fest. Die Connections sind Verbindungen Ihrer Variablen, die Sie vorher im Quelltext definiert haben. Im Interface Builder regeln Sie durch das Festlegen der Connections den Datenfluss Ihrer Variablen und verknüpfen diese mit den Feldern auf der View-Umgebung. Die Connections wurden hier erstellt, indem Sie mit der Maus auf das Icon *File`s Owner* klicken. Danach erscheint das *File´s Owner*-Menü, das die festgelegten Variablen zeigt. Diese verknüpfen Sie mit den Elementen auf dem View, indem Sie in die Kreiselemente klicken und die Verbindungen auf die Elemente des View ziehen.

avaxhome.ws



Die Picker-Bande
avaname.ws & boerse.bz

Workshop

3

Tabledance

avaxhome & boerse.bz

Dieser Workshop beschäftigt sich mit dem Erstellen von Tabellen für das iPhone. Die kompletten Darstellungsmöglichkeiten sind in der Klasse *UITableView* zusammengestellt. Sie ermöglichen den Zusammenbau und die Darstellung von Listen. Diese Listen können weiter konfiguriert werden. So können Kopf und Fußteile von Listen erstellt, Schalter eingebaut oder grafische Elemente zusammengefasst werden. Sie dienen dazu, dem Benutzer auf kleinsten Raum innerhalb von verschachtelten Tabellen Informationen zu vermitteln. In den folgenden drei Abbildungen erhalten Sie einen Überblick, was damit gemeint ist. Die App, die das Periodensystem der Elemente darstellt, finden Sie auch in der OS Library unter dem Stichwort *TheElements*.

Tabledance

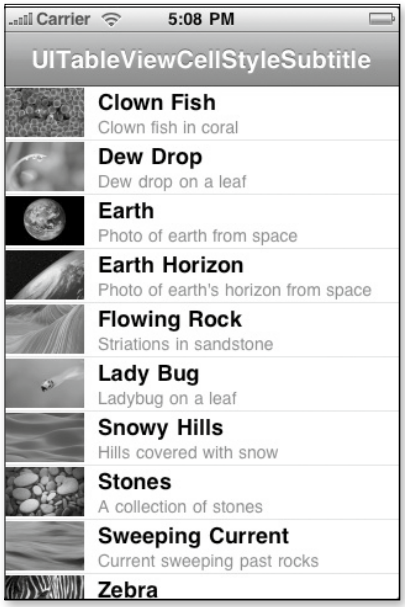
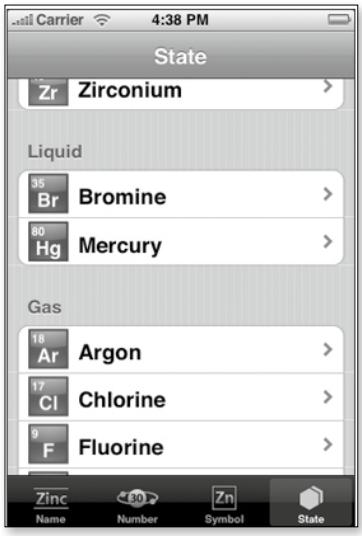
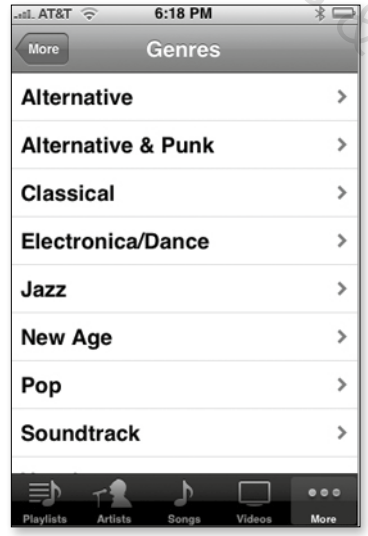


Table View mit eingebundenden Grafiken



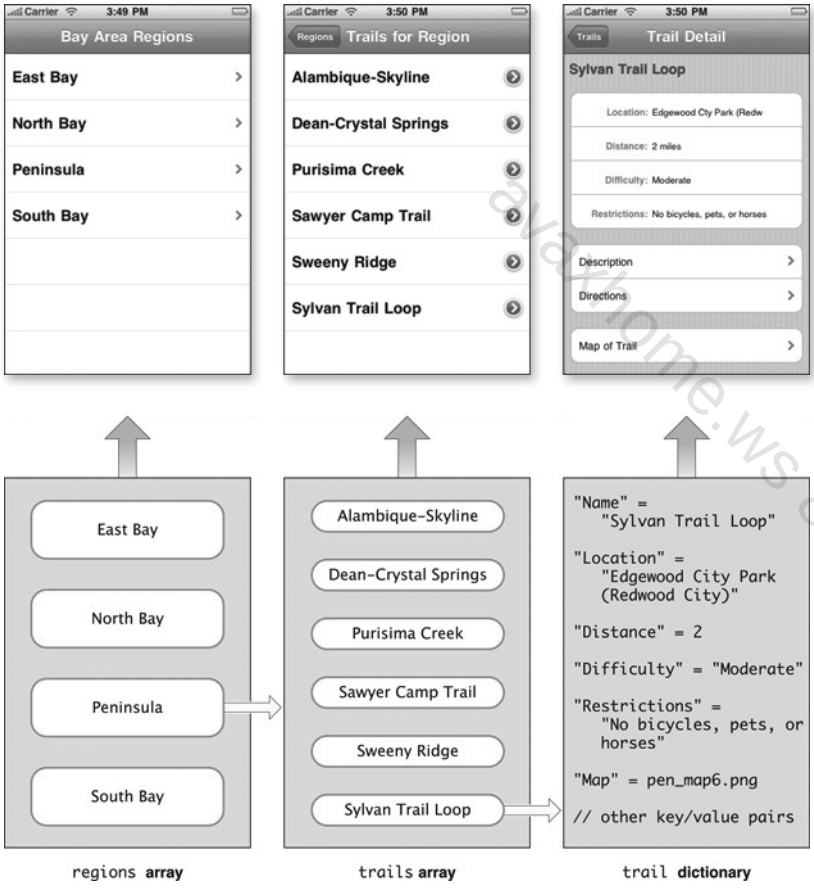
Beispiel für das Periodensystem der Elemente „TheElements" aus der OS Library



Beispiel für ein Table View aus dem iPod-Bereich

avaxhome.ws & boerse.bz

Es geht darum, dem Benutzer auf engsten Raum eines iPhone so viele Informationen wie möglich zukommen zu lassen. Dies geschieht in ineinander verschachtelten Listen. In der nächsten Abbildung erhalten Sie eine Übersicht, wie dort vorgegangen werden kann. Der Benutzer wird so geführt, dass er aus vier Regionen Trails auswählen kann. In der letzen Auswahl erhält er dann Detailinformationen über den Trail.

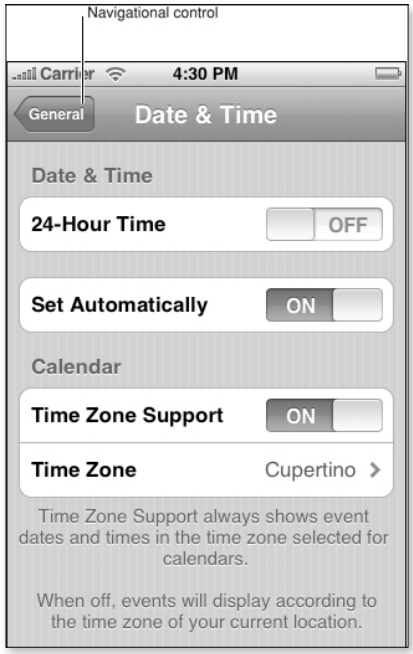


Ein Beispiel für verschachtelte Table Views. Quelle: Apple, iOS-Library

In den Table Views können Steuerelemente mit den Listen kombiniert werden.

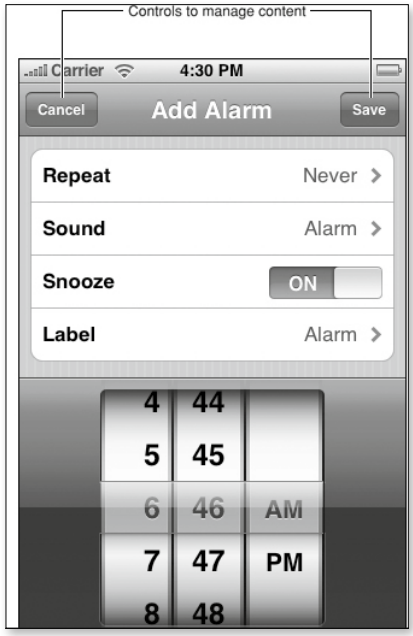
avaxhome.ws & boerse.bz

Tabledance



Hier eine typische Kombination von Schaltflächen und Table Views

Eine beliebte Kombination ist es auch, die des Table Views mit einem Picker und anderen Buttons aus der *UIButton Klasse* zu verbinden.



Hier das Beispiel für einen Wecker, in dem viele Elemente miteinander verbunden wurden

avaxhome.ws & boerse.bz

Ein simples Table View erstellen:

Das nächste Beispiel zeigt Ihnen ein simples Table-View-Beispiel. Im ersten View haben Sie eine Übersicht über die Reviervereine. Wählen Sie einen Verein aus, wechselt die Ansicht, und der Verein wird einzeln aufgelistet. Mittels eines Schalters navigieren Sie zur ersten Ansicht zurück.

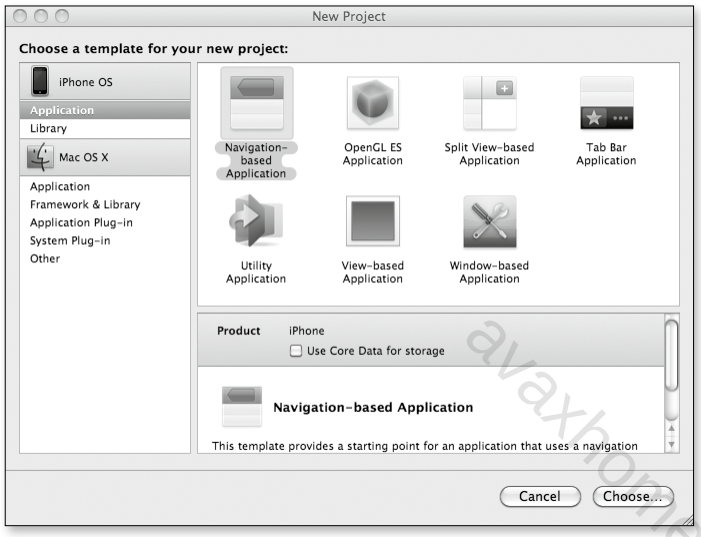
Der Startscreen der Beispiel-App

Der zweite Screen der Beispiel-App

avaxhome.ws & boerse.bz

Tabledance

1. Los gehts. Im ersten Schritt legen Sie in Xcode eine *Navigation based Application* an. Speichern Sie Ihr Projekt unter dem Namen *SimpleTable* ab.

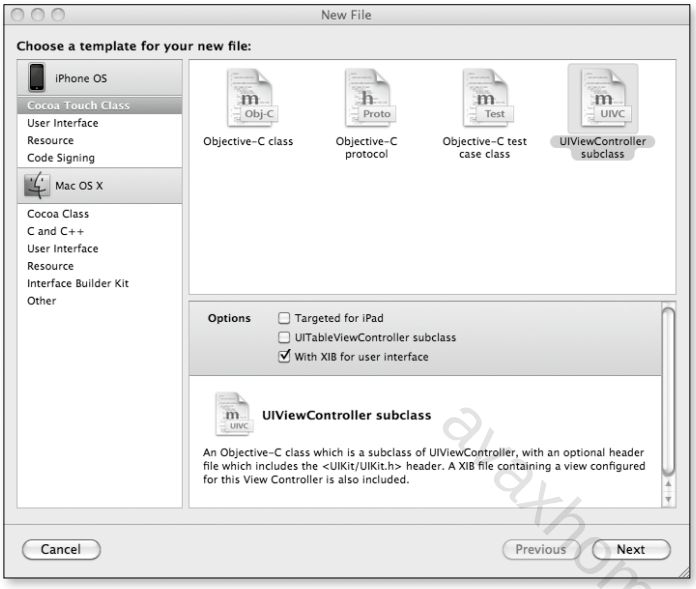


2. Öffnen Sie nun in Xcode die Datei *SimpleTableAppDelegate.m* und tragen Sie dort diese Zeilen Code ein:

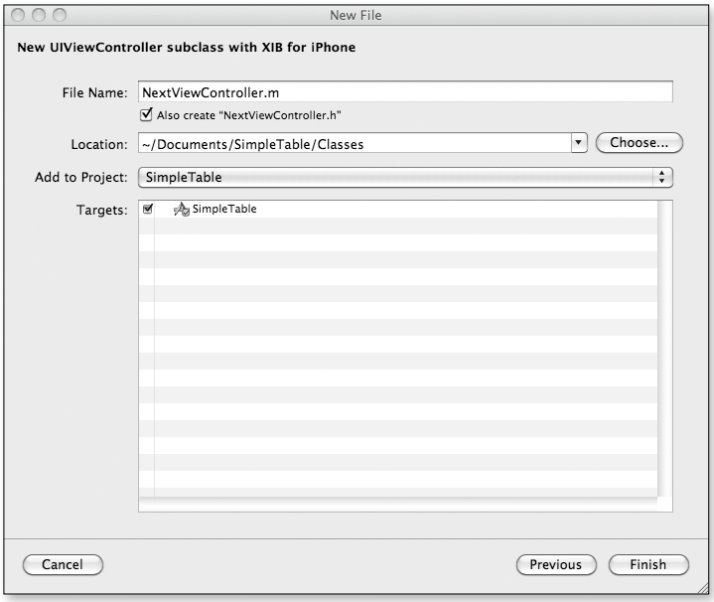
```
- (void)applicationDidFinishLaunching:(UIApplication *)
  application {
UINavigationController *navController = [[UINavigationController
alloc] initWithRootViewController:viewController];
// Override point for customization after app launch
[window addSubview:navController.view];
[window makeKeyAndVisible];
}
```

3. Danach erstellen Sie in Xcode eine neue Datei, die Sie Ihrem Projekt hinzufügen. Sie dient dazu, das zweite View der App festzulegen. Dazu wählen Sie den Menüpunkt *New File* in dem Xcode Menüpunkt an. Nun wählen Sie aus dem Fenster die Datei *UIViewControllersubclass* aus.

avaxhome.ws & boerse.bz



4. Speichern Sie die neue Datei als *NextViewcontroller.m* ab. Achten Sie auf das Häkchen *create NextViewcontroller.h*. Zwei Dateien mit den Endungen *.h* und *.xib* werden nun zusätzlich in Ihrem Projekt angelegt.

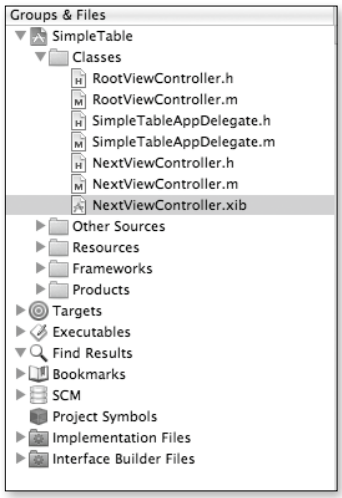


Hier legen Sie die Dateien zur Quellcodeeingabe an.

alexhumeniks & boerse.bz

Tabledance

5. Sie haben jetzt alle Dateien angelegt, die für das Projekt gebraucht werden. Im linken Fenster *Group & Files* von Xcode finden Sie die Übersicht.



6. Nach dieser Vorarbeit geht es nun daran, den Quelltext einzugeben. In der neu angelegten Datei *NextViewcontroller.h* geben Sie gleich am Anfang folgenden Quelltext ein. Sie definieren dort ein Label mit dem Namen *lblProductTex*t und einen String, in dem eine Zeichenkette ausgegeben werden kann.

```
@interface NextViewController : UIViewController {
IBOutlet UILabel *lblProductTxt;
}

- (IBAction) changeProductText:(NSString *)str;
```

7. In der Datei *NextViewcontroller.m* geben Sie ebenfalls am Anfang folgenden Code ein:

```
- (IBAction) changeProductText:(NSString *)str{
lblProductTxt.text = str;
}
```

8. Die Datei *SimpleTableViewController.m* beschicken Sie mit diesem Code. Hier wird wieder ein Array vom Typ *NSArray* angelegt, das dem Benutzer die Fußballvereine anzeigt, die ausgewählt werden können.

```
#import "SimpleTableViewController.h"
#import "NextViewController.h"
```

avaxhome.ws & boerse.bz

```
@implementation SimpleTableViewController



/*
// The designated initializer. Override to perform setup that is
required before the view is loaded.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle
*)nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil
bundle:nibBundleOrNil]) {
        // Custom initialization
    }
    return self;
}
*/

/*
// Implement loadView to create a view hierarchy
programmatically, without using a nib.
- (void)loadView {
}
*/



// Implement viewDidLoad to do additional setup after loading the
view, typically from a nib.
- (void)viewDidLoad {
    arryData = [[NSArray alloc] initWithObjects:@"VFL
Bochum",@"Borussia Dortmund",@"FC Schalke 04",@"MSV
Duisburg",nil];
    self.title = @"Reviervereine";
    [super viewDidLoad];
}



/*
// Override to allow orientations other than the default portrait
orientation.
```

avaxhome.ws & boerse.bz

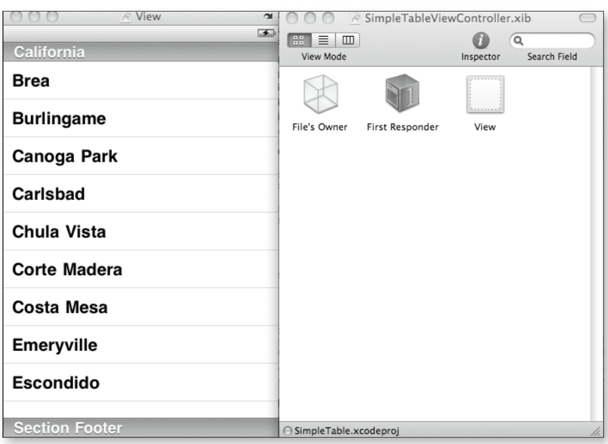
Tabledance

```
- (BOOL)
shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation ==
UIInterfaceOrientationPortrait);
}
*/


- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning]; // Releases the view if it
doesn't have a superview
    // Release anything that's not essential, such as cached data
}


- (void)dealloc {
    [super dealloc];
}

#pragma mark Table view methods

- (NSInteger)numberOfSectionsInTableView:(UITableView *)tableView
{
    return 1;
}


// Customize the number of rows in the table view.
- (NSInteger)tableView:(UITableView *)tableView
numberOfRowsInSection:(NSInteger)section {
    return [arryData count];
}


// Customize the appearance of table view cells.
- (UITableViewCell *)tableView:(UITableView *)tableView
cellForRowAtIndexPath:(NSIndexPath *)indexPath {

    static NSString *CellIdentifier = @"Cell";
```

avaxhome.ws & beerse.bz

```
    UITableViewCell *cell = [tableView
dequeueReusableCellWithIdentifier:CellIdentifier];
    if (cell == nil) {
        cell = [[[UITableViewCell alloc] initWithFrame:CGRectZero
reuseIdentifier:CellIdentifier] autorelease];
    }

    // Set up the cell...
    cell.text = [arryData objectAtIndex:indexPath.row];
    return cell;
}

- (void)tableView:(UITableView *)tableView
didSelectRowAtIndexPath:(NSIndexPath *)indexPath {
    NextViewController *nextController = [[NextViewController
alloc] initWithNibName:@"NextView" bundle:nil];
    [self.navigationController pushViewController:nextController
animated:YES];
    [nextController changeProductText:[arryData
objectAtIndex:indexPath.row]];
}

@end
```

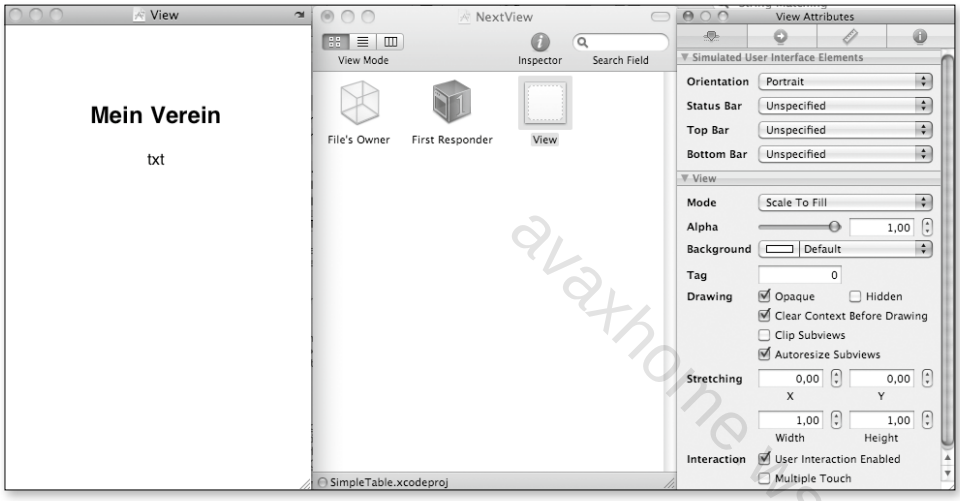
9. Im nächsten Schritt springen Sie in den Interface Builder. Durch einen Doppelklick auf die Datei *SimpleViewController.xib* gelangen Sie dorthin. Ziehen Sie nun aus der Library ein Tableview auf das View und speichern Sie die Datei ab.



Hier ist auf der Viewumgebung ein Table View mittels Drag & Drop installiert.

avaxhome.ws & boerse.bz

10. Der letzte Schritt besteht darin, die zweite Tabelle für die Ansicht festzulegen. Klicken Sie dazu auf die Datei *NextViewController.xib* in Xcode. Der Interface Builder öffnet sich. Dort ziehen Sie aus der Library ein Label und ein Textfeld auf das View. Das Label benennen Sie mit *mein Verein*, das Textfeld hat bereits den Namen. Danach springen Sie zurück zu Xcode und kompilieren Ihr Projekt.

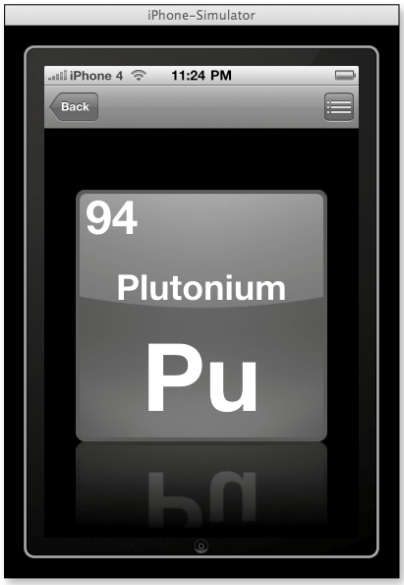


TheElements

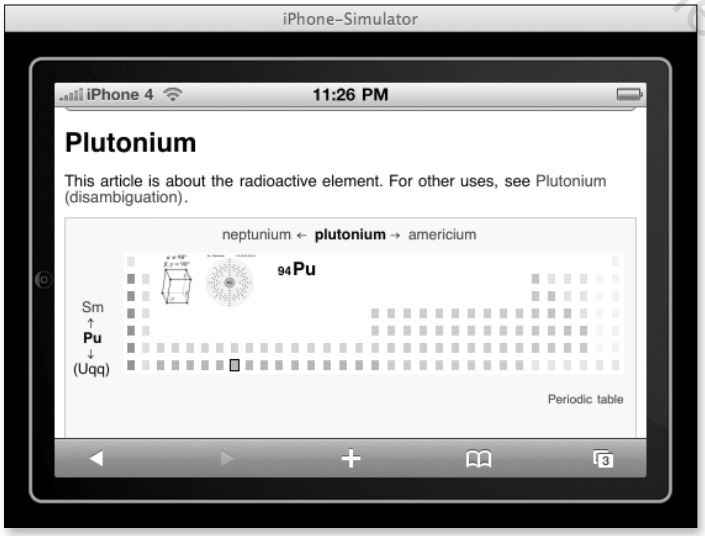
Was alles möglich ist mit dem *UITableview* lesen Sie im Folgenden. In der iOS- Library befindet sich für den Download eine App mit dem Namen *TheElements*. In ihr verbirgt sich das Periodensystem der Elemente. Die App ist ein schönes Beispiel dafür, was man mit dem Tableview alles anstellen kann. Also laden Sie sich die App aus der *iOS-Library* herunter. Nach dem Start in Xcode sehen Sie den Startscreen der App, in der Sie den Quelltext einsehen können.

Der Start der App TheElements

avaxhome.ws & bookzz.org



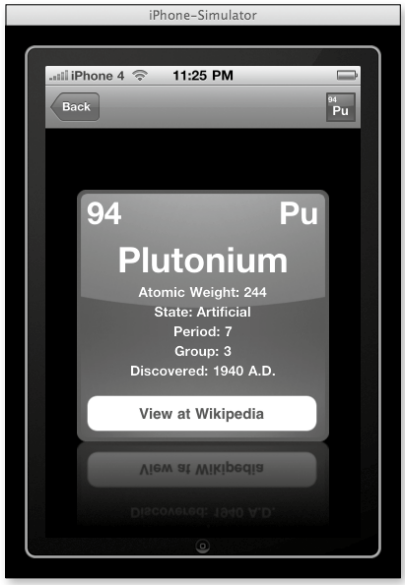
Die Detailansicht: das Element Plutonium



Die weitere Detailansicht mit einem Link zur Wikipedia

avaxhome.ws & boerse.bz

Tabledance



Die Ansicht des Elements wird in der Wikipedia dargestellt.

avaxhome.ws & boerse.bz

Workshop

4

The heat is on

avaxHome & boerse.bz

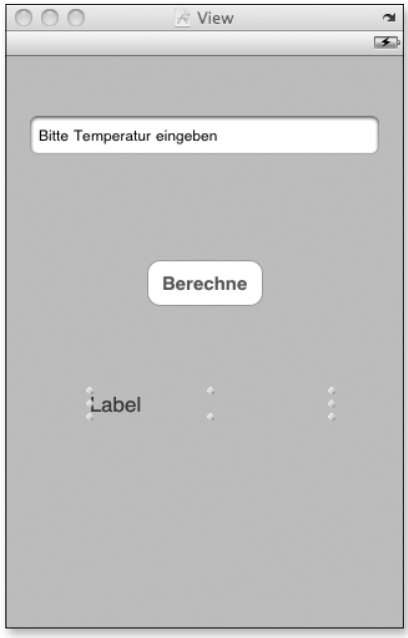
In diesem Workshop erstellen Sie einen Temperaturkonverter, der die Temperatur von Fahrenheit in Celsius umrechnet.

The heat is on

Die App rechnet die Temperatur von Fahrenheit in Celsius um.

avaxhome.ws & boerse.bz

1. Erstellen Sie in Xcode eine *View-based Application* und speichern Sie das Projekt unter dem Namen *Converter* ab. Danach starten Sie den Interface Builder, indem Sie auf die Datei *convertorViewComtroller.xib* klicken. In der View-Umgebung legen Sie drei Felder an: ein Textfeld vom Type *Round Style Text Field*, einen *Rounded Rect Button* und ein Label.



Die drei Felder, die zur Temperaturumrechnung dienen

2. Wenn Sie mit der Eingabe fertig sind, speichern Sie das Projekt im Interface Builder ab und kehren nach Xcode zurück. Hier erfolgt nun die Eingabe des Codes. Fangen Sie mit der Datei *convertorViewComtroller.h* an und geben Sie dort den unten stehenden Code ein. Sie definieren zwei Variablen mit den Namen *tempText* und *resultLabel* und eine Methode, die einen berechneten Wert zurückliefert:

```
(IBAction)convertTemp:(id)sender.
#import <UIKit/UIKit.h>

@interface convertorViewController : UIViewController {
    UITextField     *tempText;
    UILabel         *resultLabel;
}
@property (nonatomic, retain) IBOutlet UILabel *resultLabel;
@property (nonatomic, retain) IBOutlet UITextField *tempText;
- (IBAction)convertTemp:(id)sender;

@end
```

3. Nach dieser Eingabe wechseln Sie in die Datei *convertorViewComtroller.m*. Sie sehen im folgenden Quelltext, dass dort zwei Variablen für die Temperaturumrechnung definiert wurden. `double farenheit und double celcius.` Die Umrechnung der Temperatur erfolgt in dieser Zeile: `double celcius = (farenheit - 32) / 1.8`

Das Ergebnis wird in einem NSString, in einer Zeichenkette, ausgegeben.

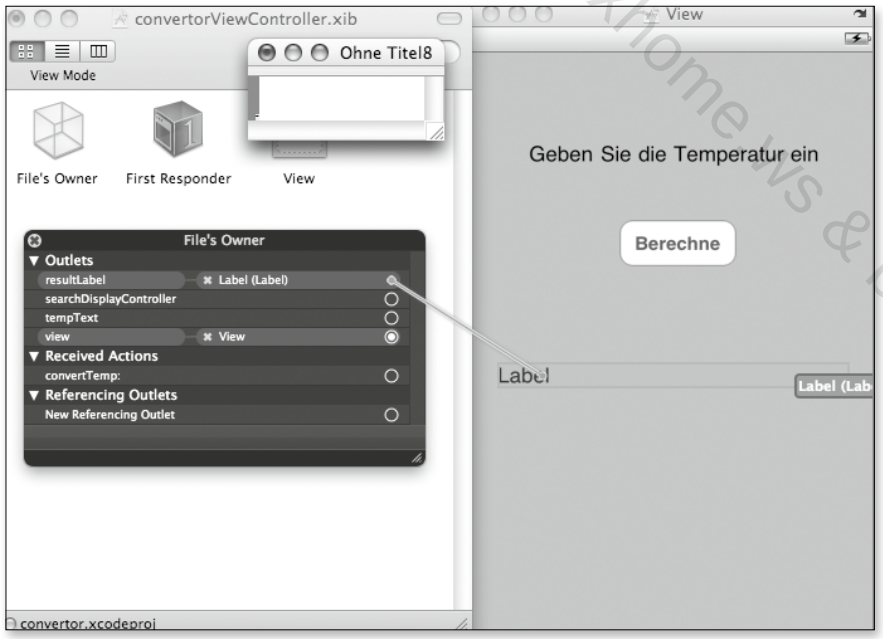
```
@implementation convertorViewController

@synthesize resultLabel, tempText;

- (void) convertTemp: (id) sender {
    double farenheit = [tempText.text doubleValue];
    double celcius = (farenheit - 32) / 1.8;

    NSString *resultString = [[NSString alloc] initWithFormat:
    @"Celcius %f", celcius];
    resultLabel.text = resultString;
    [resultString release];
}
```

avaxhome.ws/boterse.bz

The heat is on

In dieser Passage erzeugt man durch den synthesize-Befehl eine Methode. Dadurch kann man die definierten Variablen der Methode dynamischer einsetzen und man spart sich die Eingabe von Code. Dabei geht es in der weiteren Implementierung um get- und set-Methoden. Hier werden die Variablen *resultlabel* und *tempText* dazu gebraucht, in die Berechnung einzufließen und das Ergebnis in einen String auszugeben.

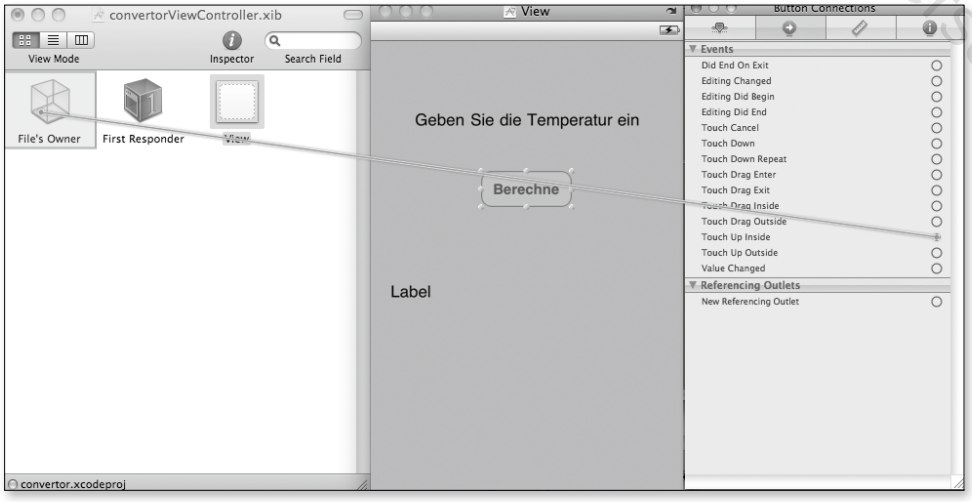
4. Im nächsten Schritt speichern Sie die Dateien in Xcode ab und springen in den Interface Builder zurück. Dort legen Sie die Connections fest. Im Fenster *convertorViewComtroller.xib* klicken Sie dazu das File´s Owner-Icon an. Wenn Sie dazu die Maustaste gedrückt lassen, klappt das *File´s Owner*-Menü auf. Dies enthält die von Ihnen deklarierten Variablen. Verbinden Sie nun, wie in der Abbildung dargestellt, die Variablen, die Sie vorher deklariert haben, mit den Elementen im View.



5. Die beiden anderen Variablen aus dem File`s Owner-Menü werden ebenfalls mit den Elementen im View verbunden. Wenn Sie damit fertig sind, sehen Sie in dem Menü, dass die Punkte hinter den Variablen mit einem weißen Kreis gefüllt sind. Die Verbindungen sind somit festgelegt.

awaxhome.ws & boerse.bz

6. Im letzten Schritt bestimmen Sie die letzte Verbindung Ihres Projektes. Hierzu klicken Sie ein Mal auf den Schalter, den Sie im View definiert haben. Sie sehen dann, dass der Connection Inspector eine Liste von Attributen für den Schalter anzeigt. Wählen Sie *Touch up Inside* und ziehen Sie die Verbindung, wie in der Abbildung dargestellt, auf das File`s Owner-Fenster.



avaxhome.ws/c boerse.bz

The heat is on

7. Zum Schluss habe ich die Elemente im View noch einmal umgestellt und dem Textfeld einen anderen vordefinierten Inhalt gegeben. Das Label, in dem die Berechnung ausgegeben wird, wurde nach oben gerückt, weil es sonst von der Tastatur verdeckt würde.

avaxhome.ws & boerse.bz

Workshop

5

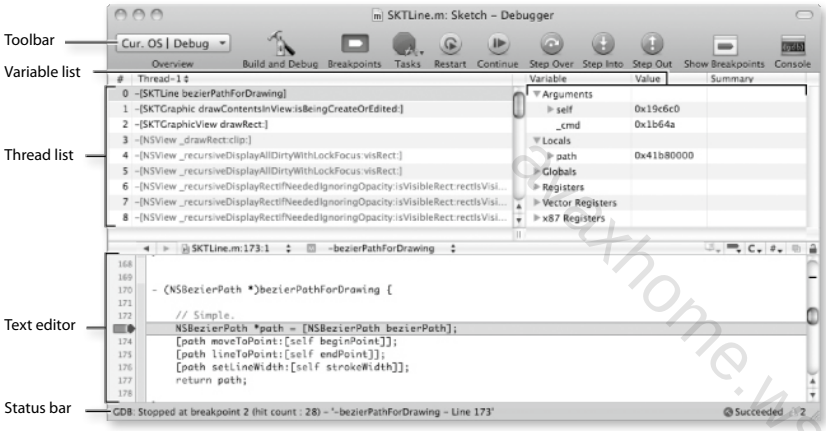
Alle Fehler schon gemacht

avaxim & boerse.bz

In diesem Workshop geht es um das Debugging Ihrer Projekte in Xcode. Eine sehr gute Dokumentation über die Fehlersuche finden Sie in der iOS-Reference-Library. Rufen Sie diese auf und geben Sie unter der Suche *Xcode Debugging Guide* ein.

Alle Fehler schon gemacht

Den Debugger rufen Sie in Xcode über die *Toolbar* auf. Starten Sie in Xcode eine Anwendung mit dem *Build-and-Go*-Befehl, so wird Ihre Anwendung automatisch im Debug-Modus gestartet. In diesem Modus erkennt der Debugger Fehler, die dem Programmcode zugrunde liegen. Mögliche Fehler sind zum Beispiel falsch deklarierte Variablen oder falsch geschriebener Programmcode. Findet der Debugger Fehler, so weist er darauf hin und zeigt den von ihm gefundenen Fehler im Programmcode.



Der Debugger in Aktion

Der Debugger ist in fünf Bereiche unterteilt:

1. Toolbar
2. Variablen-Liste
3. Thread-Liste
4. Text-Editor
5. Status-Bar

Toolbar

Die *Toolbar* ist für das Ausführen des Programms zuständig. Durch die Schaltelemente starten und stoppen Sie das Programm.

Variablen -Liste

Die Variablen-Liste zeigt Ihnen alle deklarierten Variablen mit deren Inhalt an.

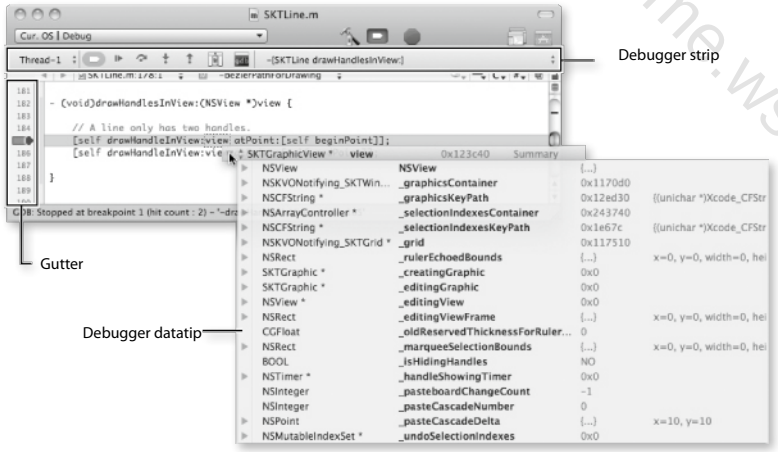
galileoxhome.ws & boerse.bz

Thread-Liste

Die Thread-Liste zeigt Ihnen die aktuellen Themen und Methoden an, die sich gerade im Quellcode befinden. Dort erhalten Sie einen Überblick darüber, wie sich die Syntax der Methode anwenden lässt.

Text-Editor

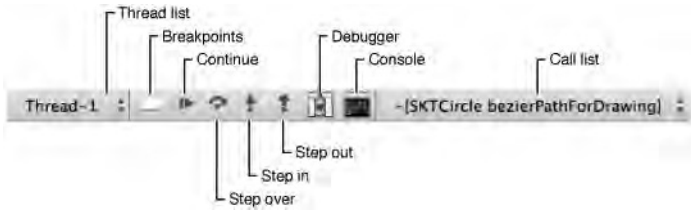
Der Text-Editor zeigt Ihnen den Quellcode an. Er ist das Hauptarbeitsfeld, wenn es um die Fehlerbeseitigung geht. In ihm können Sie Haltepunkte, die Breakpoints, setzen. Diese sind dazu da, Teilbereiche des Programms näher zu untersuchen. Setzen Sie einen Haltepunkt, hält der Debugger an der Stelle des Haltepunkts an. Das Setzen der Haltepunkte erreichen Sie, indem Sie neben dem Quelltext in den Gutter klicken. Der Gutter befindet sich links vor dem Quellcode. Er zeigt Ihnen die Zeilennummern des Quelltextes an. Der aktive Breakpoint wird durch einen blauen Pfeil sichtbar gemacht. Klicken Sie erneut auf den Pfeil, wird der Breakpoint deaktiviert. Wenn Sie den Breakpoint löschen wollen, ziehen Sie ihn einfach nach links aus dem Gutterbereich. Er löst sich dann in eine Staubwolke auf.



Debugging mit gesetztem Breakpoint im Texteditor

Am oberen Ende des Editorfensters befindet sich der Debugger-Strip. Mit den Buttons veranlassen Sie das Programm, weiterzulaufen, es zu stoppen oder nur bestimmte Teilbereiche laufen zu lassen.

entwickler.de & boerse.bz



Der Debugger-Strip in Xcode

Status-Bar

In der Status-Bar, die sich unten im Debugger-Fenster befindet, sehen Sie den aktuellen Status der Aktionen, die im Debugger ablaufen. So wird dort die Anzahl der gefundenen Fehler angezeigt, oder man kann ablesen, wo das Programm gerade an welchem Breakpoint gestoppt wurde.

Alle Fehler schon gemacht

awaxhome.ws & boerse.bz

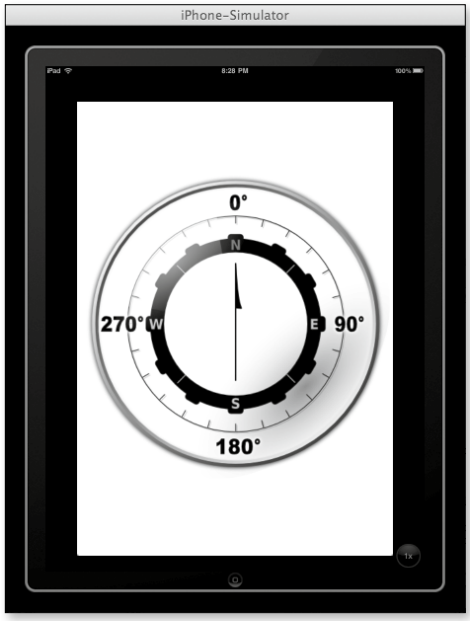
Workshop

6

Der Kompass

avaxim & boerse.bz

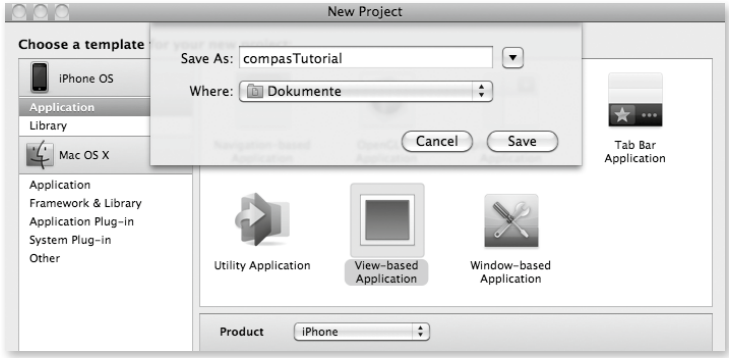
In diesem Workshop werden Sie einen Kompass für das iPhone erstellen. Hierzu werden GPS-Signale in der App verarbeitet, um den Kompass zu simulieren. Sie lernen ebenso, Grafiken (hier eine Kompassrose) in eine App einzubinden. Es gibt dabei ein kleines Problem: Sie können den Kompass im iPhone-Simulator nicht testen, weil er keine GPS-Daten empfängt. Um diese App testen zu können, müssen Sie sie auf einem iPhone installieren.



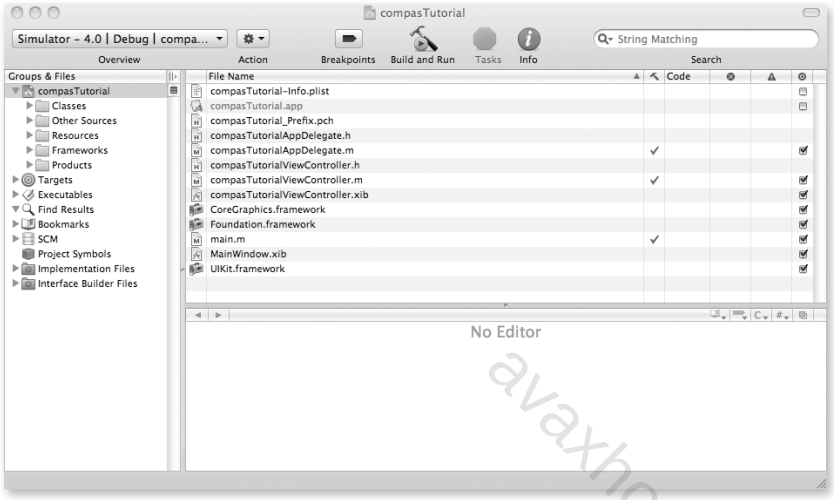
Die fertige App weist Ihnen den Weg nach Norden.

avaxhome.ws & boerse.bz

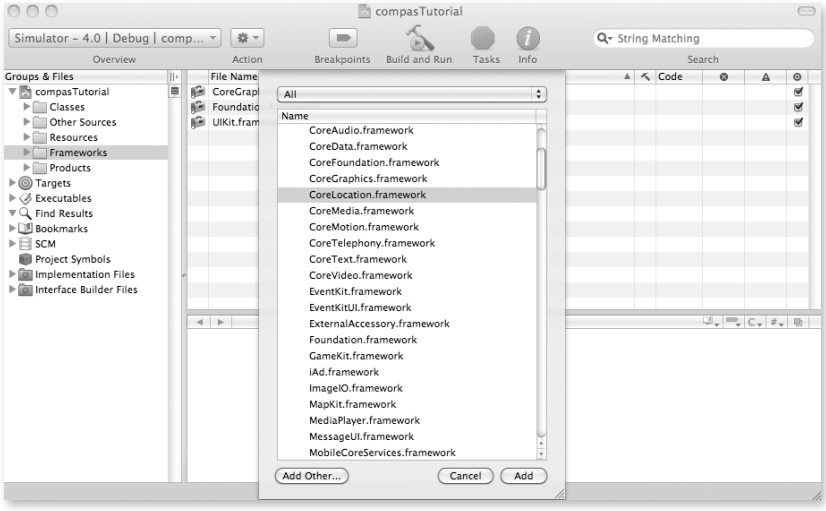
1. Auf gehts. Erstellen Sie in Xcode eine *View-based Application* und nennen Sie Ihr Projekt *compasTutorial*.



2. In Xcode sieht Ihr Projekt dann so aus:



3. Im nächsten Schritt fügen Sie das *Core Location Framework* zu Ihrem Projekt hinzu. Dieses Framework enthält Methoden, in denen GPS-Daten weiter verarbeitet werden können. Dazu doppelklicken Sie unter *Group and Fi*les im linken Fenster von Xcode den Ordner *Frameworks*. Alle bestehenden Frameworks erscheinen nun im rechten Fenster. In der *Toolbar* finden Sie eine Schaltfläche mit einem Zahnrad, die mit *Action* untertitelt ist. Wählen Sie in dieser Schaltfläche den Menupunkt *Add* an. Eine Auswahl aller Frameworks erscheint. Wählen Sie nun das Core Location Framework aus.

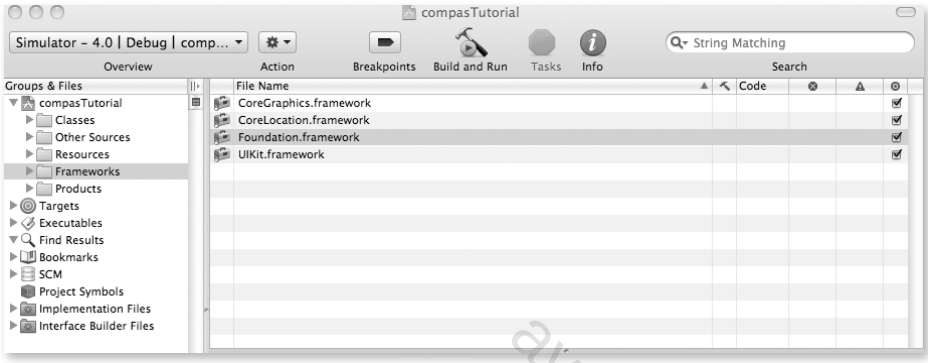


Hier wird das CoreLocation.framework zu Ihrem Projekt hinzugefügt.

avaxhome.ws/boerse.bz

Der Kompass

4. Nachdem Sie das neue Framework zu Ihrem Projekt hinzugefügt haben, sieht die Arbeitsoberfläche, wenn Sie auf den Ordner *Frameworks* klicken, so aus:

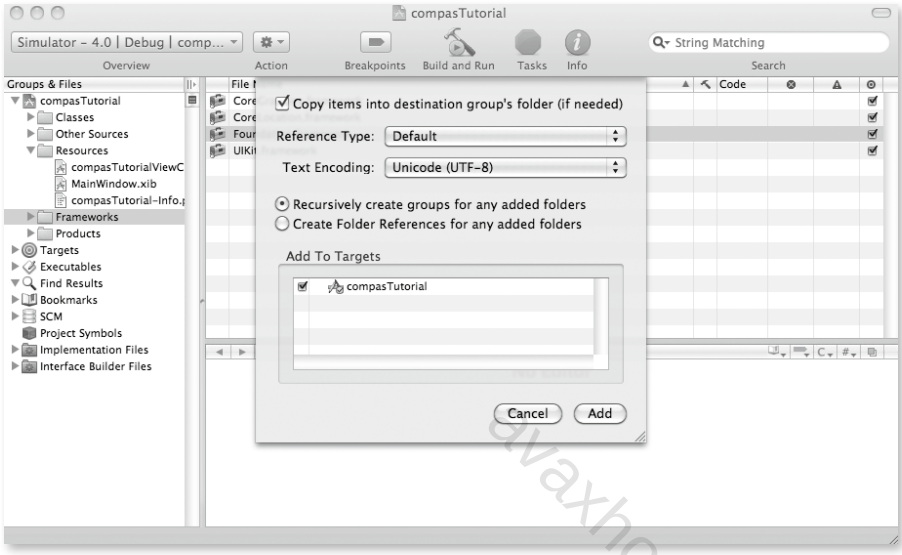


Im Ordner Frameworks befinden sich alle Frameworks Ihres Projektes.

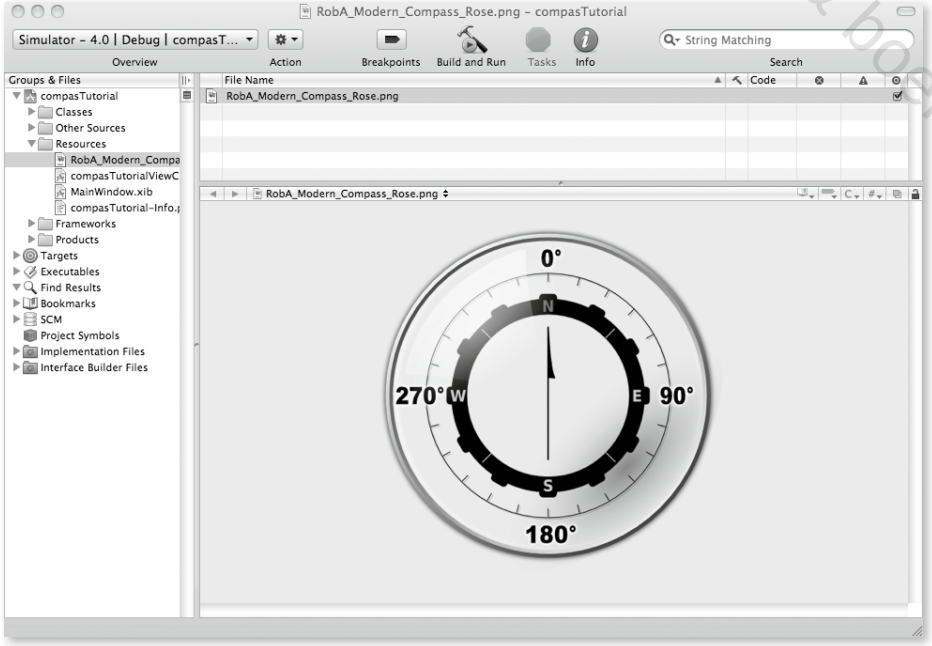
5. Im nächsten Schritt fügen Sie Ihrem Projekt eine Grafik, die Kompassrose, hinzu. Diese muss als *.png*-Datei vorliegen. In der Beispiel-App sieht das Bild so aus:

6. Nun beziehen Sie die Grafik in das Projekt ein. Dazu greifen Sie sich die Bilddatei außerhalb von Xcode und befördern sie via Drag & Drop in den Ordner *Ressources*. Dazu geht ein Fenster auf. Setzen Sie dazu das Häkchen in der linken oberen Ecke. Danach ist die Grafik in Ihr Projekt implementiert.

awaxhome.ws & boerse.bz



7. Ist der Transport des Bildes erfolgreich gewesen, sehen Sie dies in der Arbeitsumgebung von Xcode, indem Sie im Ordner *Resources* die geladene Datei doppelklicken. Es wird Ihnen auch der Name der *.png*-Datei angezeigt. In diesem Fall heißt die Datei *RabA_Modern_Compass_Rose.png*.



Hier wurde die Grafik erfolgreich Ihrem Projekt hinzugefügt.

avaxhome.ws & boarse.bz

Der Kompass

8. Nun ist es an der Zeit, Programmcode einzugeben. Zunächst öffnen Sie die Datei *CompasTutorialViewController.h* und geben die paar Zeilen Code ein. Durch die Anweisung `IBOutlet UIImageView *compassRose` definieren Sie eine Instanzvariable, die im Interface Builder weiterverarbeitet werden kann.

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

@interface CompasTutorialViewController : UIViewController {
    IBOutlet UIImageView *compassRose;
}
@property(nonatomic,retain) IBOutlet UIImageView *compassRose;
@end
```

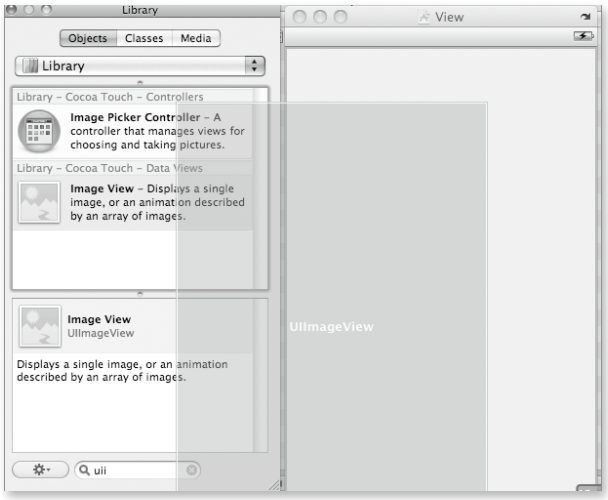
9. Im nächsten Schritt öffnen Sie die Datei *CompasTutorialViewController.m*. Dort geben Sie den nachfolgenden Quelltext ein. Hier wird die Methode rotate Compass definiert. Die Kompassrose wird dadurch erst im Uhrzeigersinn um 45 Grad gedreht, um sich dann wieder auf ihren Ursprung zurückzudrehen. Alles andere regelt das Core Animation-Framework im Hintergrund.

```
#import "CompasTutorialViewController.h"
@interface CompasTutorialViewController(private)
- (void) rotateCompass: (double) degrees;
@end

@implementation CompassTutorialViewController
@synthesize compassRose;



- (void)viewDidLoad {
compassRose.transform = CGAffineTransformMakeRotation(45.0);
 [self rotateCompass: 0.0];
 [super viewDidLoad];

}

- (void) rotateCompass: (double) degrees{
    [UIView beginAnimations:nil context:NULL];
    [UIView setAnimationDuration:2.0];
compassRose.transform = CGAffineTransformMakeRotation(degrees);
    [UIView commitAnimations];
}
```

ausdrucke.ws & boerse.bz

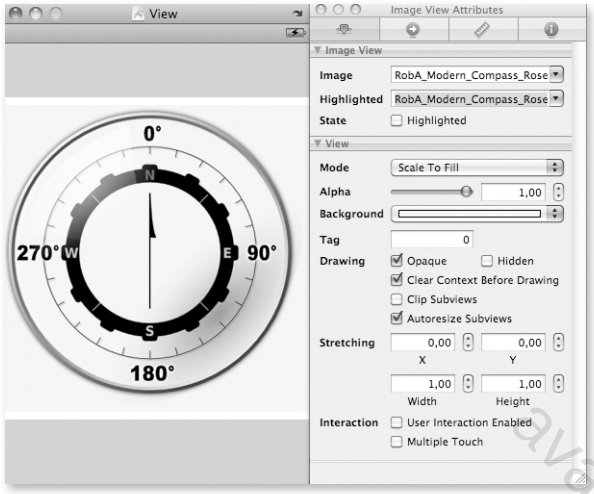
```
- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning]; // Releases the view if it
    doesn't have a superview
// Release anything that's not essential, such as cached data
}


- (void)dealloc {
    [compassRose release];
    [super dealloc];
}

@end
```

10. Nach vollbrachter Code-Eingabe springen Sie in den Interface Builder zurück, indem Sie auf die Datei *CompasTutorialViewController.xib* klicken. Dort legen Sie das User-Interface fest. Hierzu gestalten Sie die Oberfläche des Views weiß und positionieren die Kompassrose. Aus der Library wählen Sie ein *Image-View* aus und ziehen es auf die Fläche Ihres Views.

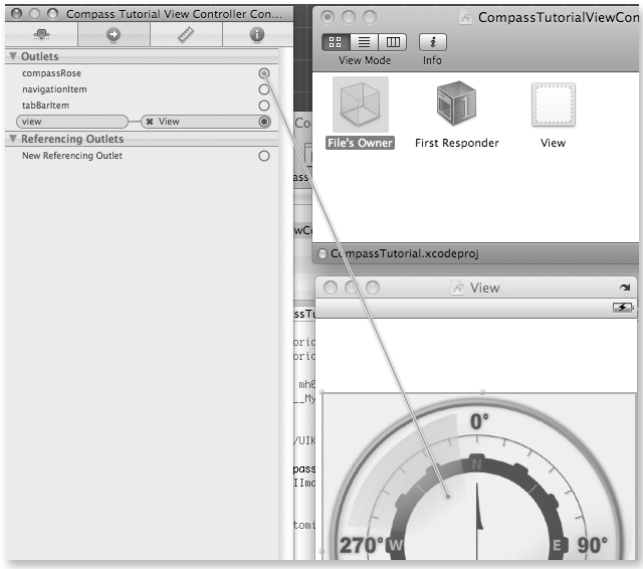


11. Als Nächstes wählen Sie im Attribute-Assistenten die Kompassrose aus. Sie sehen im unteren Fenster, dass dort ein Menü *Image* ist, in dem Sie die Kompassrose auswählen können. Nachdem Sie die Datei angewählt haben, erscheint sie in dem *UIImage-View*, welches Sie vorher auf das Fenster-View positioniert haben.

avaxhome.ws & boerse.bz



12. Zum Schluss stellen Sie noch eine Verbindung zu der Kompassrose her. Danach springen Sie zu Xcode zurück und kompilieren Ihr Projekt.



Hier wird die Verbindung zu Ihrer Instanzvariable Compasrose hergestellt.

javaxhome.ws & boerse.bz

Workshop

7

Die fliegende Kuh

avaxhome & boerse.bz

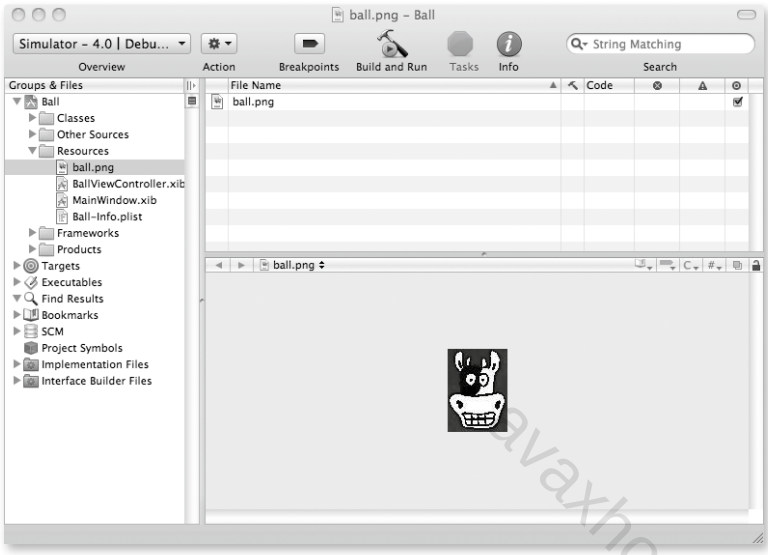
In dieser App lassen Sie eine Kuh über den Bildschirm des iPhone fliegen. Immer wenn Sie an den Bildschirmrand stößt, ändert die Kuh ihre Richtung. Mit ein paar Tricks des *NSTimer* und dem Interface Builder erlernen Sie schnell, das Grundgerüst für Arcade-Games zu erstellen.

Die fliegende Kuh

avaxhome.ws & boerse.bz

1. Let's go. Erstellen Sie in Xcode ein *View-based Application*-Projekt mit dem Namen *Ball* und speichern Sie es ab.

2. Nun bewegen Sie das Bild der Kuh via Drag & Drop in den *Resources-Ordner*. Ich habe die Datei *ball.png* genannt, da stellvertretend für die Kuh auch ein Ball genommen werden könnte. Nachdem die Bilddatei in Ihrem Ordner gelandet ist, können Sie einen Doppelklick darauf machen. Das Bild erscheint dann im unteren Projektfenster von Xcode.



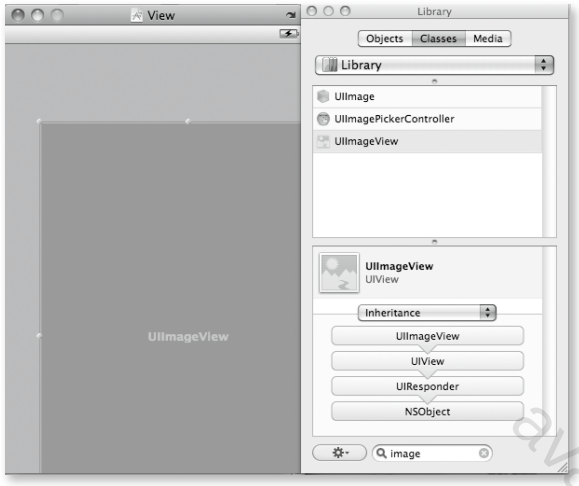
3. Jetzt wird es Zeit, in der Datei *BallViewController.h* Code einzugeben. Das IBOutlet definiert wieder die Variable, die im Interface Builder weiterverarbeitet wird.

```
@interface BallViewController : UIViewController {
    IBOutlet UIImageView *ball;
    CGPoint pos;
}

@property(nonatomic,retain) IBOutlet UIImageView *ball;

@end
```

4. Springen Sie nun in den Interface Builder mit einem Doppelklick auf die Datei *BallViewController.xib*. Dort angekommen bewegen Sie ein *UIImageView* aus der Library auf die View-Umgebung. Das UIImageView nimmt hinterher die Grafik der Kuh auf. Wenn Sie nicht wissen, wo sich das UIIImageView in der Library befindet, können Sie im unteren Feld der Library das Element suchen lassen.

javaxhome.ws & boerse.bz

Die fliegende Kuh



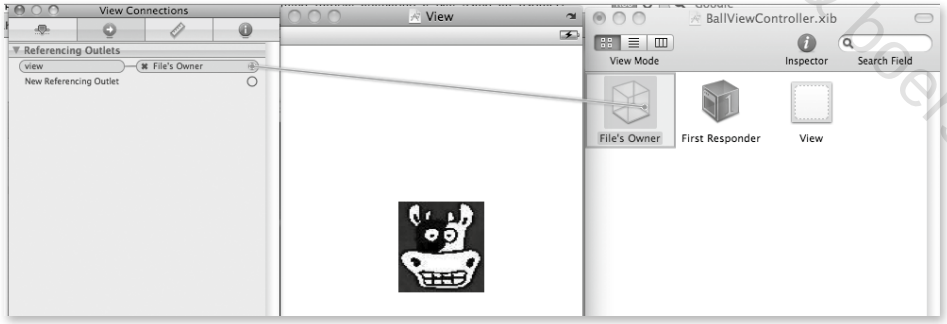
Das UIImage wird auf die View-Umgebung gezogen,

5. Nun positionieren Sie das UIImageView auf der Umgebung des View in der Mitte und ziehen es kleiner, so dass es hinterher das Bild der Kuh aufnimmt.

6. Klicken Sie nun in das Fenster *View* und legen Sie im *View-Attributes*-Fenster die Hintergrundfarbe fest. In der Abbildung steht die Hintergrundfarbe noch auf Grau. Wechseln Sie die Farbe zu Weiß. Die Kuh laden Sie in das *UIIImage-View*, indem Sie darauf klicken. Ein anderes Fenster klappt nun auf, in dem Sie oben im Fenster ein Auswahlmenü bekommen, von dem aus Sie die *Ball.png*-Datei anwählen. An dieser Stelle sei darauf hingewiesen, dass Sie beliebige Grafiken in das *UIIImage-View* einbinden könnten. Diese würden sich dann alle so verhalten wie ein Ball oder hier die Kuh, die in einem Kasten hin und her springt.

awaxhomeows & boerse.bz

7. Der letzte Schritt. Dazu brauchen Sie den *Connection-Inspector* des Views. Gehen Sie wie in der unteren Abbildung vor und legen Sie die Verbindung fest. Wenn Sie auf das *File's Owner*-Symbol in dem Fenster *BallViewController.xib* treffen, erhalten Sie in einem Menü eine Auswahl, in der Sie die Variable *Ball* als Menüpunkt auswählen.



8. Jetzt ist die Arbeit im Interface Builder zunächst getan. Speichern Sie das Projekt im Interface Builder ab und springen Sie zu Xcode zurück. Hier geben Sie in der Steuerdatei *BallViewController.m* den weiteren Code ein. Hier diejenige Passage im Code, die die Kuh springen lässt:

```
if(ball.center.x > 320 || ball.center.x < 0)
        pos.x = -pos.x;
    if(ball.center.y > 460 || ball.center.y < 0)
        pos.y = -pos.y;
```

avaxhome.ws & boerse.bz

Die Werte 320 und 460 in den if-Abfragen ergeben sich durch die Größe des iPhone-Displays. Es besteht aus 320 x 460 Pixeln. Somit wird der Rand begrenzt. Die Position der Kuh wird durch x- und y-Koordinaten angegeben. Immer wenn der Wert der Variablen größer als 320 und 460 wird, wird der positive x- oder y-Wert negiert. Das Spiel oder die Bewegung des Objektes kann dann von vorne beginnen.

```
#import "BallViewController.h"

@implementation BallViewController

@synthesize ball;

/*
// Override initWithNibName:bundle: to load the view using a
   nib file then perform additional customization that is not
   appropriate for viewDidLoad.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)
  nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil
    bundle:nibBundleOrNil]) {
        // Custom initialization
    }
    return self;
}
*/

/*
// Implement loadView to create a view hierarchy programmatically.
- (void)loadView {
}
*/

-(void) onTimer {
    ball.center = CGPointMake(ball.center.x+pos.x,ball.center.
    y+pos.y);

    if(ball.center.x > 320 || ball.center.x < 0)
        pos.x = -pos.x;
    if(ball.center.y > 460 || ball.center.y < 0)
        pos.y = -pos.y;
}
```

avaxhome.ws & boerse.bz

```
// Implement viewDidLoad to do additional setup after loading the
   view.
- (void)viewDidLoad {
    pos = CGPointMake(14.0,7.0);

    [NSTimer scheduledTimerWithTimeInterval:0.05 target:self
    selector:@selector(onTimer) userInfo:nil repeats:YES];
}



- (BOOL)shouldAutorotateToInterfaceOrientation:
  (UIInterfaceOrientation)interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation ==
    UIInterfaceOrientationPortrait);
}


- (void)didReceiveMemoryWarning {
    [super didReceiveMemoryWarning]; // Releases the view if it
    doesn't have a superview
    // Release anything that's not essential, such as cached data
}


- (void)dealloc {
    [super dealloc];
}

@end
```

Fertig!

alexhome.ws & boerse.bz

avaxhome.ws & boerse.bz

Workshop

8

Digitale Zeiten

avaxim & boerse.bz

In diesem Workshop erstellen Sie innerhalb von circa 15 Minuten eine Digitaluhr. Sie greifen dazu auf das NSDate-Framework zurück. Dieses beinhaltet eine Sammlung von Methoden zur Zeitberechnung.

Digitale Zeiten

1. Auf gehts. Erstellen Sie in Xcode ein *view-based-Application*-Projekt und nennen Sie es *Uhr*. Dann geben Sie in Xcode in der *Header-Datei .h* folgenden Code ein:

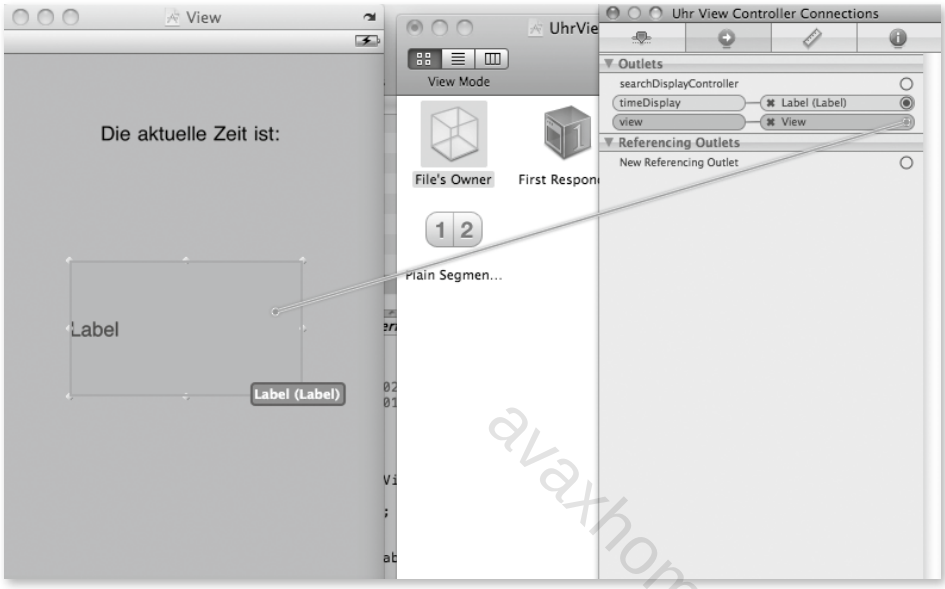
```
@interface TimeViewController :  UIViewController {
        NSTimer *timer;
        IBOutlet UILabel *timeDisplay;
}
@property (nonatomic, retain) UILabel *timeDisplay;
-(void)time;
```

Danach haben Sie Ihre Variablen für den Interface Builder definiert. Springen Sie nun in den Interface Builder und platzieren Sie auf dem View zwei Label. Im oberen Label geben Sie den statischen Text: "Die aktuelle Zeit ist." ein. Das untere Label dient zur Datenaufnahme des *NSTimers*. Dort wird die digitale Uhr angezeigt.

anonymous & boerse.bz

Hier werden im View zwei Label platziert.

2. Jetzt legen Sie die Connections fest. Dazu verknüpfen Sie im Interface Builder, wie in der Abbildung unten dargestellt, die Verbindung von der definierten Variable *timeDisplay* zu Ihrem Label in der View-Umgebung.



3. Nach vollbrachter Arbeit im Interface Builder geben Sie in Xcode unten stehenden Quelltext ein. Sie definieren damit eine Methode, mit der die Zeit berechnet wird:

```
[timeDisplay setFont:[UIFont fontWithName:@"DBLCDTempBlack"
size:128.0]];
    timer = [NSTimer scheduledTimerWithTimeInterval:(1.0)
    target:self selector:@selector(time) userInfo:nil
    repeats:YES];
```

Danach wird sie in einem String, einer Zeichenkette, ausgegeben: `timeDisplay.text = [NSString stringWithFormat:@"%02d:%02d", hourOfDay, minuteOfHour];`

```
#import "TimeViewController.h"

@implementation TimeViewController

//@synthesize hour;
//@synthesize min;
```

avaxhome.ws & boerse.bz

Digitale Zeiten

```
/*
// The designated initializer. Override to perform setup that is
   required before the view is loaded.
- (id)initWithNibName:(NSString *)nibNameOrNil bundle:(NSBundle *)
  nibBundleOrNil {
    if (self = [super initWithNibName:nibNameOrNil
    bundle:nibBundleOrNil]) {
        // Custom initialization
    }
    return self;
}
*/

/*
// Implement loadView to create a view hierarchy
   programmatically, without using a nib.
- (void)loadView {
}
*/



// Implement viewDidLoad to do additional setup after loading the
   view, typically from a nib.
- (void)viewDidLoad {
    [timeDisplay setFont:[UIFont fontWithName:@"DBLCDTempBlack"
    size:128.0]];
     timer = [NSTimer scheduledTimerWithTimeInterval:(1.0)
     target:self selector:@selector(time) userInfo:nil
     repeats:YES];
    [super viewDidLoad];
}

-(void)time
{
    NSDate *now = [NSDate date];
    int hourOfDay = [[now dateWithCalendarFormat:nil
    timeZone:nil] hourOfDay];
    int minuteOfHour = [[now dateWithCalendarFormat:nil
    timeZone:nil] minuteOfHour];
    timeDisplay.text = [NSString stringWithFormat:@"%02d:%02d",
    hourOfDay, minuteOfHour];
```

avaxhome.ws & boerse.bz

```
}

/*
// Override to allow orientations other than the default portrait
   orientation.
- (BOOL)
  shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)
  interfaceOrientation {
    // Return YES for supported orientations
    return (interfaceOrientation ==
    UIInterfaceOrientationPortrait);
}
*/

- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}


- (void)dealloc {
    [super dealloc];
}

@end
```

4. Zum Schluss kompilieren Sie Ihr Projekt. Die Zeit wird wie unten abgebildet dargestellt.

avaxhome.ws & boerse.bz

Digitale Zeiten

avaxhome.ws & boerse.bz

Workshop

Der Höhenmesser 9

avaxim & boerse.bz

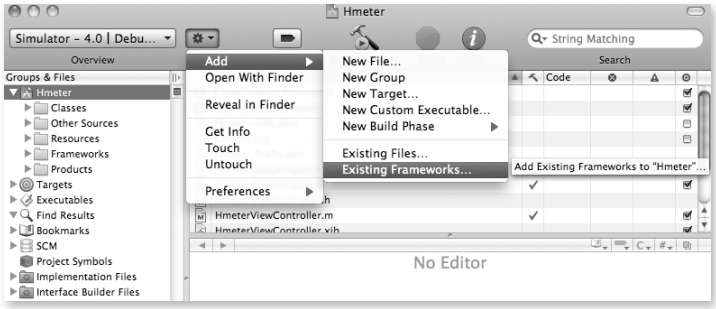
In diesem Workshop erstellen Sie einen Höhenmesser. Dazu wird das Core Location Framework in das Projekt eingebunden, damit GPS-Daten weiterverarbeitet werden können. Die App ist recht schnell erstellt.

Die App benutzt das Core Location Framework für die Höhenmessung.

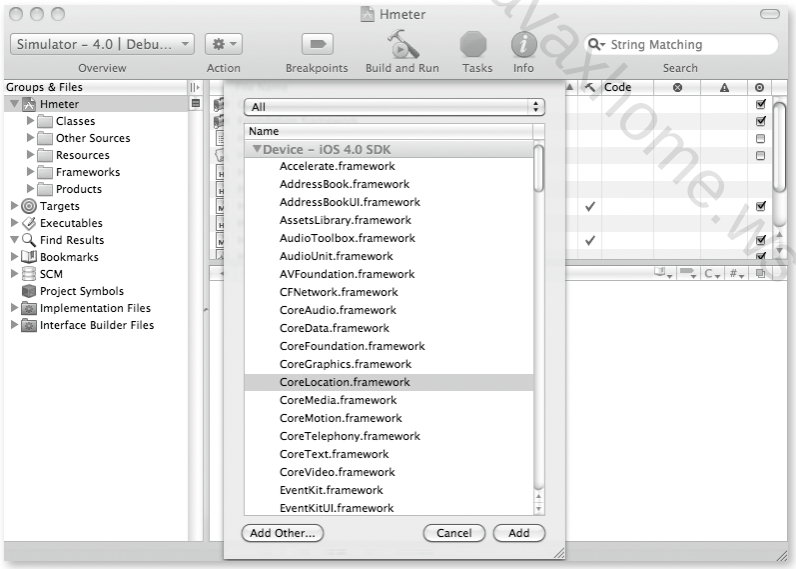
avaxhomeows & boerse.bz

1. Beginnen Sie wieder mit einem *View-based Application*-Projekt und speichern Sie es unter dem Namen *Hmeter* ab.

2. Im nächsten Schritt binden Sie das *Core Location Framework* in Ihr Projekt ein. Dazu klicken Sie in der *Xcode*-Toolbar auf das Symbol mit dem Zahnkranz. Wählen Sie zuerst *Add* und dann *Existing Framework* an.



3. Wählen Sie nun aus der Liste das Core Location Framework aus und fügen es Ihrem Projekt hinzu.



avaxiome.ws & boerse.bz

Die Einbindung des CoreLocation.framework in Ihr Projekt

Im Folgenden öffnen Sie in Xcode die Datei *MmeterViewController.h*. Dort definieren Sie die Variablen, die Sie hinterher im Interface Builder für die Connections brauchen. Sie definieren ein Label zur Ausgabe der Höhenmessung und eine Methode, die zur Höhenmessung dient.

```
@interface HeightMeasurementViewController : UIViewController {
  IBOutlet UILabel *heightMesurement;
  CLLocationManager  *locmanager;

}
```

```
- (void)locationManager:(CLLocationManager *)manager
  didUpdateToLocation:(CLLocation *)newLocation
  fromLocation:(CLLocation *) oldLocation ;

- (void)locationManager:(CLLocationManager *)manager
  didFailWithError:(NSError *) error;
```

4. Danach springen Sie in den Interface Builder und platzieren dort zwei Label. In einem, dem oberen, tragen Sie einen statischen Text ein mit dem Inhalt "Höhenmeter in m über NN". Das zweite Label platzieren Sie darunter. Es dient dazu, die Höhenmessung anzuzeigen. Die untere Abbildung fällt etwas spärlich aus. Sie sollten versuchen, die Connections selber festzulegen. Sie müssen von File´s Owner eine Verbindung zum großen Label herstellen.

@waxhome.ws & boerse.bz

5. Im nächsten Schritt geben Sie in der Datei *HmeterViewController.m* wieder den Programmcode ein. Im Programmcode definieren Sie eine Methode, die der Höhenmessung dient, und einen String, in dem die Messung angezeigt werden kann. Die Definition der Zeichenkette liest sich so:

```
heightMesurement.text = [NSString stringWithFormat: @"%.2f m",
newLocation.altitude];
```

Die Methode zur Höhenmessung wird so eingegeben:

```
(void)locationManager:(CLLocationManager *)manager
didUpdateToLocation:(CLLocation *)newLocation
fromLocation:(CLLocation *)oldLocation
```

Ergänzen Sie nun den Quelltext und kompilieren Sie Ihr Projekt.

```
#import "HeightMeasurementViewController.h"

@implementation HeightMeasurementViewController
//@synthesize heightMesurement;


-(void)awakeFromNib {

    locmanager = [[CLLocationManager alloc] init];
    [locmanager setDelegate:self];
    [locmanager setDesiredAccuracy:kCLLocationAccuracyBest];

    [locmanager startUpdatingLocation];

}

- (void)locationManager:(CLLocationManager *)manager
  didUpdateToLocation:(CLLocation *)newLocation
  fromLocation:(CLLocation *)oldLocation
{

    heightMesurement.text = [NSString stringWithFormat: @"%.2f m",
    newLocation.altitude];
}

- (void)locationManager:(CLLocationManager *)manager
  didFailWithError:(NSError *)error
{
    heightMesurement.text = @"0.00 m";
}


- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
     [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
```

awaxhomeews & boerse.bz

```
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}


- (void)dealloc {
    [super dealloc];
}

@end
```

6. Im letzten Schritt wechseln Sie nochmals in den Interface Builder zurück und legen noch eine Verbindung fest. Dort verankern Sie die im Quelltext vorher definierte Methode *highMesurement*t mit Ihrem Label der Höhenmessung. Fertig.

Der Höhenmesser

alexhome.ws & boerse.bz

Workshop

10

Die Web-App

avaxhome & boerse.bz

In diesem Workshop erstellen Sie eine Web-App. Die App ruft nach ihrem Start automatisch Seiten von Google auf. Diese Webseiten sind für die Darstellung auf dem iPhone optimiert. Mit ein paar Zeilen Code zaubern sie für den Benutzer wirklich Nützliches auf das Display.

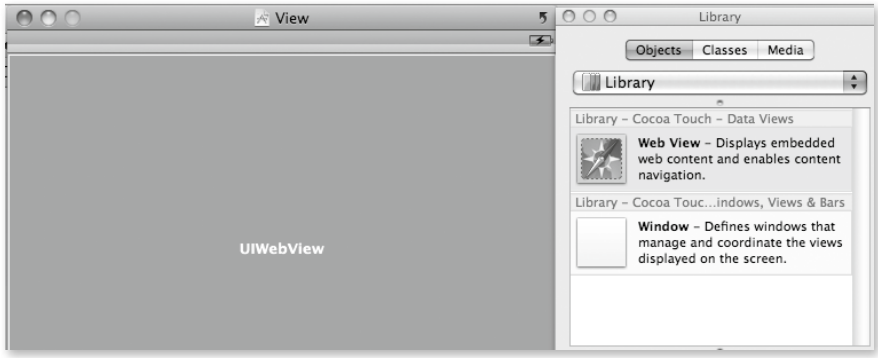
Die Grenzen zwischen einer nativen App und einer Web-App verschwimmen immer mehr. Durch die Always-on-Mentalität sind Smartphones fast immer mit dem Web verbunden, dadurch lassen sich einfacher Web-Apps realisieren, weil sie sofort vom Benutzer aufgerufen werden können. Die Produktion einer Web-App ist einfacher und kostengünstiger. Sie liegt im Web und kann jederzeit aktualisiert werden.

1. Fangen Sie also an. Erstellen Sie ein *View-based Application*-Projekt in Xcode und nennen Sie es *MapDisplay*. In der Datei *MapDisplayViewController.h* geben Sie folgenden Code ein:

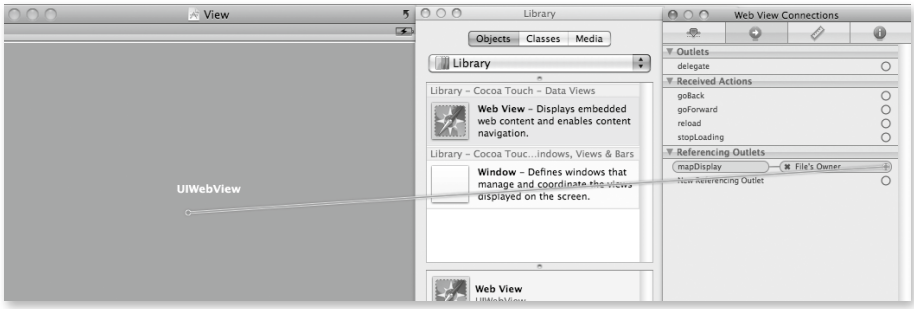
```
#import <UIKit/UIKit.h>

@interface MapDisplayViewController : UIViewController {

        IBOutlet UIWebView *mapDisplay;

}

@property(nonatomic ,retain) UIWebView *mapDisplay;

@end
```

avaxhome.ws & bgcrise.bz

2. Danach springen Sie in den Interface Builder, indem Sie auf die Datei *MapDisplayViewController.xib* doppelklicken. Dort ziehen Sie aus der Library ein *UIWebView* auf die Fläche Ihrer View-Umgebung.



3. Der nächste Schritt besteht darin, eine Verbindung, wie in der unteren Abbildung dargestellt, zu erstellen. Sie verbinden dort die Variable *mapDisplay* mit dem *UIWebView*.



Hier wird die Instanzvariable mapDisplay mit dem UIWebView verknüpft.

4. Der letzte Schritt besteht einfach darin, die Seiten von Google aufzurufen. Die Anweisung sieht so aus: `NSString *urlAddress = @"http://maps.google.co.in";` Nach geglückter Codeeingabe kompilieren Sie das Projekt und schauen, was im iPhone-Simulator passiert.

```
#import "MapDisplayViewController.h"

@implementation MapDisplayViewController

@synthesize mapDisplay;



// Implement viewDidLoad to do additional setup after loading the
   view, typically from a nib.
- (void)viewDidLoad {

    NSString *urlAddress = @"http://maps.google.co.in";

    //Create a URL object.
    NSURL *url = [NSURL URLWithString:urlAddress];

    //URL Requst Object
    NSURLRequest *requestObj = [NSURLRequest requestWithURL:url];

    //Load the request in the UIWebView.
    [mapDisplay loadRequest:requestObj];

    [super viewDidLoad];
}
```

openhome.ws & boerse.bz

```
- (void)didReceiveMemoryWarning {
    // Releases the view if it doesn't have a superview.
    [super didReceiveMemoryWarning];

    // Release any cached data, images, etc that aren't in use.
}

- (void)viewDidUnload {
    // Release any retained subviews of the main view.
    // e.g. self.myOutlet = nil;
}


- (void)dealloc {
    [mapDisplay release];
    [super dealloc];
}

@end
```

avaxhome.ws/blogs/doerse.bz

Haben Sie erfolgreich Ihr Projekt erstellt, stellt Google den Rest der App zur Verfügung. Es handelt sich hierbei um eine typische Web-App. Durch den Aufruf der App werden im *UIWebView* die für das iPhone optimierten Webseiten dargestellt. In ihr werden alle Features dargestellt, die Google auch sonst zu bieten hat.

Die App startet mit Ihrer Positionsangabe.

Verwendet Ihre App den aktuellen Standort, zeigt sie diesen auch an.

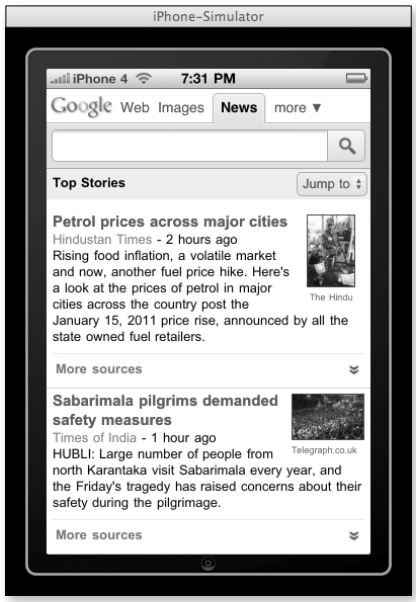
Hier wird Ihre aktuelle Position angezeigt.

Google stellt aber noch andere Features für Ihre Web-App bereit: So wird auch ein Routenplaner für das iPhone optimiert angeboten.

Innerhalb der App kann ein Routenplaner aufgerufen werden.

avaxhome.ws & boerse.bz

Von Google-Maps lässt sich eine Verbindung zu der Suchabfrage von Google schalten:



Hier sehen Sie die Verzweigung zur Suchmaschine von Google.

avaxhome.ws & boerse.bz

Abspann

avaxhome & boerse.bz

Herzlichen Glückwunsch! Sie haben es geschafft, das Buch zu Ende zu lesen. Ich hoffe, Sie hatten Spaß dabei und werden ein guter X-Coder. Abschließend finden Sie hier noch ein paar Links und Buchtipps, die Ihnen das Weitermachen erleichtern sollen.

Abspann

Links

Apple
http://developer.apple.com/library/mac/navigation/index.html

https://itunesconnect.apple.com

Shortcuts für Xcode 3.2 im html Format
http://cocoasamurai.blogspot.com/2009/08/xcode-shortcuts-updated-for-xcode-32-on.html

Der Video-Blog für Einsteiger, kommt aus dem Pott (Castrop-Rauxel)
http://www.x02100.de/

Der Cocoa-Guru und seine Ranch
http://www.bignerdranch.com/

Die Webseite des Autors
http://www.appzitty.de/

avaxhome.ws & boerse.bz

Literatur

Das Kultbuch
Aaron Hillegass: Cocoa Programmierung für Mac OS X

Gibt es in diesem Verlag:
Amin Negm-Awad: Objective-C und Cocoa Band 1: Grundlagen

Ders./ Christian Kienle: Objectiv C und Cocoa Band 2: Fortgeschrittene

Ayaz, Hameister, Meißner: Cocoa Kochbuch (mitp Professional)

Index

avaxhome & boerse.bz

Index

awaxinne.ws & boerse.bz

J

K

L

M

N

O

P

Q

R

S

avaxhome.ws & boorse.bz

T

U

V

W

X

Z

plexhome.ws & boerse.bz