2. Auflage

iOS-Apps programmieren mit Swift

Christian Bleske

Der leichte Einstieg in die Entwicklung für iPhone, iPad und Co. – inkl. Apple Watch und Apple TV

dpunkt.verlag

Christian Bleske ist Autor, Trainer und Entwickler. Sein Arbeitsschwerpunkt ist die Entwicklung von Client/Server- und mobilen Anwendungen. In vielen namhaften Entwicklerzeitschriften erscheinen seine Fachaufsätze. Er lebt in Witten im Ruhrgebiet.

Christian Bleske

iOS-Apps programmieren mit Swift

Der leichte Einstieg in die Entwicklung für iPhone, iPad und Co. – inkl. Apple Watch und Apple TV

2., aktualisierte und erweiterte Auflage

dpunkt.verlag

Christian Bleske
cb.2000@hotmail.de
christianbleske.wordpress.com

Lektorat: René Schönfeldt, Sandra Bollenbacher
Copy-Editing: Petra Kienle, Fürstenfeldbruck
Satz: III-satz, Husby
Herstellung: Susanne Bröckelmann
Umschlaggestaltung: Helmut Kraus, www.exclam.de
Druck und Bindung: M.P. Media-Print Informationstechnologie GmbH, 33100 Paderborn

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.

ISBN:
Print 978-3-86490-438-7
PDF 978-3-96088-074-5
ePub 978-3-96088-075-2
mobi 978-3-96088-076-9

2., aktualisierte und erweiterte Auflage

Wieblinger Weg 17
69123 Heidelberg

5 4 3 2 1 0

Vorwort

Liebe Leserinnen und Leser,

mobile Geräte wie Smartphones, Tablets und Smartwatches erobern den Erdball, gleichzeitig stagniert der Absatz von PCs oder ist sogar rückläufig. Durch den massenhaften Absatz mobiler Geräte wird auch der Hunger nach Anwendungen, die darauf ausgeführt werden können, immer größer – ein spannender und manchmal sogar lukrativer Markt für Entwickler.

Im Jahr 2011 schrieb ich mein erstes Buch für Entwickler, die sich diesen Markt über Windows Phone erschließen wollten (»Windows Phone 7 Apps entwickeln«). Im Jahr darauf erschien das zweite Buch zur Entwicklung von Apps, dieses Mal für Android (»Java für Android«). Nun, mit dem vorliegenden Buch zur iOS-Programmierung, schließt sich der Kreis.

iOS und Swift

iOS ist das universelle Betriebssystem von Apple, das sowohl die unterschiedlichen iPhone- und iPad-Modelle sowie iPod Touch, CarPlay, Apple TV (tvOS) und Apple Watch (Basis von watchOS ist iOS) antreibt.

Neben den unterschiedlichen Geräten gibt es seit nunmehr zwei Jahren bei Apple auch eine passende moderne Programmiersprache, die die Entwicklung für die iOS-Plattform erleichtern soll: Swift. Swift ist der Nachfolger von Objective-C, das mit seinen fast 30 Jahren langsam, aber sicher abgelöst werden soll. Viele Entwickler setzen ihre neuen Projekte deshalb nur noch mit Swift um.

In diesem Jahr ist Swift in der Version 3.0 erschienen und es enthält viele Änderungen. Dies ist die zweite Auflage des Buches, neben der Aktualisierung auf Swift 3.0 finden Sie in dieser Auflage auch neue Themen. So ist ein Kapitel zum Thema Apple TV 4 (tvOS) hinzugekommen. In diesem erfahren Sie, wie Apps für diese Plattform geschrieben werden. Außerdem gibt es im Buch jetzt ein Kapitel zum Thema parallele Verarbeitung. Auch die Programmierung von Apps mit Unterstützung für Handoff, Spotlight und 3D Touch wird demonstriert. Zuletzt wird nun auch das Thema Drucken aus Apps heraus behandelt.

Leser

Konzentrieren werde ich mich auch in Bezug auf die Leserschaft dieses Buches. Es spricht Leser an, die bisher noch keine iOS-Anwendungen entwickelt haben und auch mit Swift nicht vertraut sind. Auch mit Apples Entwicklungsumgebung Xcode müssen Sie sich nicht auskennen, denn die lernen Sie hier ausführlich kennen. Sie sollten aber grundlegende Kenntnisse in einer beliebigen anderen Programmiersprache haben und über einen Mac-Computer als Entwicklungsrechner verfügen.

Damit ausgestattet sollten Sie kaum Probleme haben, meinen Ausführungen zu folgen und die Beispiele nachzuvollziehen. Nach der Lektüre werden Sie dann genügend über das Betriebssystem iOS, die Entwicklungsumgebung Xcode und die Programmiersprache Swift gelernt haben, um eigene Apps damit zu programmieren.

Beispielcode

Wie in meinen Büchern üblich, so finden Sie auch hier wieder viele Beispiele. Sie sind allesamt mit Swift 3 und Xcode 8 entwickelt worden, und ihren Code können Sie online von der Buch-Webseite herunterladen unter

www.dpunkt.de/swift2

Als Alternative hierzu ist der Code ebenso über meinen GitHub-Account *https://github.com/christianbleske* verfügbar. Dort können die Beispiele auch einzeln heruntergeladen werden. Wenn neue Swift-Versionen erscheinen, dann werde ich an dieser Stelle aktualisierte Versionen der Beispiele zum Download bereitstellen.

Den Rückmeldungen der Leserinnen und Leser zu meinen bisherigen Büchern habe ich entnommen, dass sich viele etwas komplexere Beispiele und eine »richtige« App wünschen. Diesem Wunsch habe ich versucht in diesem Buch gerecht zu werden. So werden die meisten Themen anhand einer kompletten App, etwa einer App zur Wettervorhersage, erläutert.

Darüber hinaus zieht sich eine besondere Beispiel-App durchs ganze Buch: eine Passwortverwaltung, die in verschiedenen Kapiteln immer wieder aufgegriffen und themenbezogen weiterentwickelt wird.

Ein weiteres besonderes Beispiel ist der Schnelleinstieg, in dem ich Ihnen in kompakter Form die wichtigsten Dinge zeige, die die iOS-Entwicklung mit Swift ausmachen. Das komplette Einstiegsbeispiel finden Sie zusätzlich in Form eines kleinen Videos auf der Buch-Webseite.

Das Blog zum Buch

Begleitend zum Buch gibt es außerdem ein Blog, in dem ich Ihnen zusätzliche Informationen sowie ggf. Fehlerkorrekturen zum Buch bereitstelle und auf Änderungen in Xcode und der iOS-API hinweise. Sie finden es unter

http://christianbleske.wordpress.com

Über das Blog oder per E-Mail unter *cb.2000@hotmail* dürfen Sie mich auch gerne persönlich ansprechen, wenn bei der Lektüre zu diesem Buch Fragen auftreten sollten.

Christian Bleske, im September 2016

Inhaltsübersicht

Inhaltsverzeichnis

1 Einleitung

Viele Normalbenutzer waren wohl relativ enttäuscht, als auf der *World Wide Developers Conference* (WWDC) 2014 keine Uhr, kein Telefon bzw. kein neuer Rechner vorgestellt wurde – die Entwicklergemeinde freute sich dafür umso mehr. Denn im Juni 2014 wurde auf der WWDC eine neue Programmiersprache nebst passender Integration in Apples Entwicklungswerkzeug Xcode präsentiert. *Swift* heißt die neue Sprache, was man unter anderem als »Mauersegler« übersetzen kann, und der gilt als ganz schneller Vogel. Aber warum eine neue Programmiersprache? Mit Objective-C hatte man doch eigentlich die Mutter aller Sprachen im Programm – oder etwa doch nicht?

Um diese Frage zu beantworten, sehen wir uns folgenden Objective-C-Code-Auszug an: Hier wird einer String-Variablen eine URL zugewiesen und anschließend in einem Browser-Control aufgerufen:

```
NSString *urlString=@"http://";
urlString = [urlString stringByAppendingString:Id];
NSURL *url = [NSURL URLWithString:[NSString
   stringWithFormat:@"%@",urlString]];
NSURLRequest *requestObj = [NSURLRequest
   requestWithURL:url];
[uiWebView loadRequest:requestObj];
```

Selbst wenn man ein Fan von Objective-C ist, wird man zugeben müssen, dass die Syntax dieser Sprache im Vergleich zu anderen (Assembler einmal ausgenommen) gewöhnungsbedürftig ist. Gerade Ein- und Umsteigern bereitet die komplexe Syntax von Objective-C oft Probleme. Apple bzw. die Geräte des Unternehmens zeichnen sich dadurch aus, dass sie besonders einfach zu bedienen sind. Von der Haus- und Hofsprache Objective-C kann man das leider nicht sagen.

Es musste also etwas Neues her, um sicherzustellen, dass der Strom von neuen Apps für die iOS-Plattform und die damit verbundenen Geräte auch zukünftig nicht abreißt. So kam man bei Apple vor einigen Jahren auf die Idee, Objective-C noch eine zweite Sprache zur Seite zu stellen – eine moderne Sprache mit einem zeitgemäßen Funktionsumfang (z. B. Generics), die leicht zu erlernen sein soll.

Man kann durchaus festhalten, dass es Apple mit Swift gelungen ist, genau die »Problemzonen« von Objective-C anzugehen und trotzdem kompatibel zum Rest von Apples SDK zu bleiben. Denn neben den bereits genannten Vorzügen von Swift gibt es auch eine Brücke in die »alte« (Objective-C-)Welt. Natürlich ist auch das passende API (Cocoa Touch) in Swift-Apps nutzbar. So muss nicht alles neu entwickelt werden.

1.1 iPhone, iPad & Co.

Eine Programmiersprache allein – und sei sie noch so gut – bringt nichts, wenn die zugrunde liegende Plattform bzw. das Betriebssystem keinen Erfolg hat. Das kann man von iOS nicht sagen, denn es ist auf vielen Geräten im Einsatz.

Daher wird in diesem Buch bewusst nicht allein das iPhone angesprochen, denn iOS treibt darüber hinaus natürlich auch das iPad und den iPod Touch, Apple TV und die Apple Watch an. Seinen Ursprung hat iOS in Apples Mac-Betriebssystem macOS (OS X). Im Prinzip handelt es sich um eine abgespeckte Variante dieses Betriebssystems, die für die Bedienung mit der Hand und für die Ausführung der Software auf schwächerer Hardware optimiert wurde.

An dieser Stelle sollen Sie aber nicht mit der Historie rund um Apples Hard- und Software gequält werden. Die ganze Story ist sicherlich bekannt.

Hinweis
Seit der WWDC 2016 heißt das Betriebssystem für Macs (wieder) macOS 10.12 anstatt OS X.

1.2 Für wen ist dieses Buch gedacht?

Dieses Buch ist für Einsteiger mit Programmierkenntnissen gedacht. Fachbegriffe aus dem Bereich der Softwareentwicklung, wie z.B. IDE (Entwicklungsumgebung), Variable oder Schleife, sollten Sie also kennen. Ideal wäre es, wenn Sie bereits mit einer anderen Programmiersprache (z.B. C, C# oder Java) gearbeitet haben. Es wird also vorausgesetzt, dass Sie sich mit dem Thema Programmierung bereits beschäftigt haben. Leser ohne diese Kenntnisse werden es schwer haben, mit dem Buch zurechtzukommen.

Außerdem wird davon ausgegangen, dass Sie die Bedienung von macOS beherrschen, also beispielsweise wissen, wie man Dateien kopiert oder Anwendungen aus dem App Store installiert.

Die Entwicklungsumgebung Xcode müssen Sie nicht kennen.

1.3 Benötigte Hard- und Software

Eine wichtige Frage ist, welche Hard- und Software man benötigt, um Apps für iOS zu entwickeln – vor allem, wenn man bisher auf einer anderen Plattform, z. B. Windows, gearbeitet hat.

1.3.1 Welcher Mac genügt?

Haben Sie bereits einen Mac oder suchen Sie noch ein passendes Gerät? Generell gilt: Jeder Mac, auf dem Yosemite oder El Capitan läuft, kann zur Entwicklung von iOS-Apps verwendet werden. Die neueste Version von macOS 10.12 (Sierra) kann auf den folgenden Macs installiert werden:

- iMac (Late 2009 oder neuer)
- MacBook (2009 oder neuer)
- MacBook Air (Late 2010 oder neuer)
- MacBook Pro (Mid 2010 oder neuer)
- Mac mini (Mid 2010 oder neuer)
- Mac Pro (Mid 2010 oder neuer)

Ferner werden mindestens 2 Gigabyte Arbeitsspeicher (besser 4) und 8 Gigabyte Festplattenspeicher benötigt. Sehen Sie diese Werte als Minimalwerte an – mehr ist immer besser!

Ein Tipp, wenn Sie noch keinen Mac haben und einen kaufen möchten: Schauen Sie sich einmal einen Mini an. Warum? Wenn Sie bereits einen anderen Computer (PC) zu Hause haben, dann können Sie vorhandene Hardware (z. B. Monitor, USB-Tastatur, USB-Maus) weiterverwenden. Sofern Sie nicht auf gebrauchte Hardware setzen, ist das die günstigste Variante (ab ca. 600 Euro), um in die Mac-Welt einzusteigen.

1.3.2 Welche macOS-Version?

Yosemite, El Capitan oder Sierra – diese Versionen können Sie zur Entwicklung von Apps mit Swift verwenden. Ältere Versionen von macOS, z. B. Mountain Lion, können *nicht* zur Entwicklung verwendet werden. Welche Version ist besser? Sierra unterstützt zurzeit die größte Anzahl an APIs. Das liegt daran, dass Sierra die aktuellste macOS-Version ist. Man sollte aber generell darauf achten, dass es mit der neusten Version von macOS nicht zu Stabilitätsproblemen kommt. Konservative Zeitgenossen (der Autor zählt auch dazu) warten deshalb immer auch darauf, dass die aktuellste macOS-Version mindestens ein oder zwei Updates erhalten hat, bevor ein Umstieg erfolgt.

Hinweis

Wenn Sie aus der Windows-Welt kommen, dann sind Sie es (bisher) in der Regel nicht gewohnt, ständig (also mindestens einmal im Jahr) auf eine neue Betriebssystem-Version zu aktualisieren. Unter macOS ist es mittlerweile quasi Standard, dass einmal pro Jahr ein Major-Update, also eine neue macOS-Version, erscheint. Normalerweise funktioniert das Update auf die neueste Version auch problemlos. Trotzdem sollte man warten, bis mindestens die Version XX.1 oder besser XX.2 erreicht ist.

Benötigen Sie ein iOS-Gerät?

Nicht zwingend, aber es erleichtert die Entwicklung von Apps doch sehr. Außerdem kann man nur auf einem »echten« Gerät bestimmte Funktionen (z.B. den Beschleunigungssensor) auch aus dem Code heraus ansprechen. Ob Sie sich für ein iPhone, ein iPad oder gar einen iPod touch entscheiden, das bleibt Ihrem Geschmack überlassen. Möchten Sie in Ihrer App auch auf Telefoniedienste zugreifen, so muss das Testgerät natürlich ein iPhone sein. Achten Sie auch auf darauf, wie lange es noch Updates für das jeweilige Gerät gibt. Nichts ist ärgerlicher, als viel Geld für ein Gerät auszugeben, um dann festzustellen, dass das Gerät vom Hersteller nicht mehr mit neuen Versionen des Betriebssystems versorgt wird.

1.4 Xcode und das iOS-SDK

Apple hat eine Entwicklungsumgebung, die für die Entwicklung von Anwendungen sowohl für iOS als auch für macOS verwendet wird. Diese IDE trägt den Namen Xcode. Es gibt sie seit vielen Jahren (genauer gesagt seit 2003), und mittlerweile ist Xcode relativ umfangreich. Xcode unterstützt nicht nur eine Programmiersprache, denn neben Swift können Sie damit auch Programme in Objective-C, C++ und C schreiben. Im Gegensatz zu den bekannten Java-IDEs (Eclipse/Net Beans) gibt es Xcode nur für den Mac bzw. macOS.

Xcode besteht aus mehreren Komponenten. Zum einen ist da natürlich der Quellcodeeditor, in dem Sie den Code schreiben. Wenn eine grafische Oberfläche für eine Anwendung erstellt werden muss, dann kann man dafür den grafischen Designer von Xcode verwenden. Er trägt den Namen *Interface Builder*. Mit dem Interface Builder lassen sich Oberflächen von Anwendungen via Drag & Drop zusammenstellen. Das bedeutet: Ähnlich wie in Microsofts Visual Studio gibt es einen Bereich mit vorgefertigten Controls, die auf einem Formular (View) abgelegt werden können.

1.4.1 Download und Installation von Xcode

Neben Xcode benötigen Sie für die Entwicklung von Apps für iOS noch das entsprechende SDK. Es handelt sich dabei aber nicht um einen separaten Download neben Xcode, sondern Sie laden einfach die neueste Xcode-Version aus dem App Store herunter. Diese Version enthält dann alle benötigten Komponenten.

Es gibt mehrere Wege, um Xcode auf Ihren Mac herunterzuladen. Der einfachste Weg soll Ihnen an dieser Stelle vorgestellt werden. Er führt über den App Store von macOS zum Ziel. Öffnen Sie den App Store, und geben Sie im Suchfeld einfach »Xcode« ein. Nach Aktivierung der Suche sollte gleich der erste Treffer in den Suchergebnissen Xcode sein.

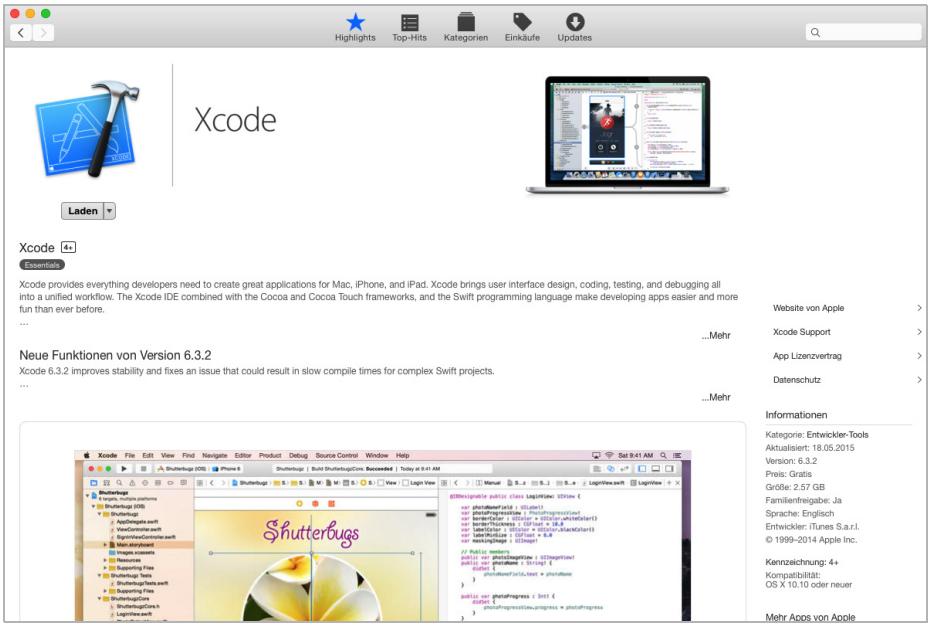

Abb. 1–1 *Download und Installation von Xcode über den App Store*

Wenn Sie den Xcode-Link auswählen, kommen Sie auf die Xcode-Homepage im App Store. Zum Download und zur Installation müssen Sie einfach den *Laden*-Button einmal anklicken und anschließend die grüne Schaltfläche mit der Beschriftung *App installieren* betätigen. Der Rest läuft vollautomatisch ab. Das heißt, nach dem Download wird Xcode auch gleich automatisch installiert. Informationen zur Größe des Downloads und zur aktuellen Versionsnummer finden Sie ebenfalls auf der Xcode-Homepage. Nach dem Download und der Installation können Sie Xcode über das Launchpad starten. Dort wurde ein entsprechendes Icon angelegt. Alternativ ist natürlich der Start aus dem Verzeichnis *Programme* möglich. Alle notwendigen Dateien von Xcode befinden sich im Xcode-Package. Das bedeutet: Wenn Sie Xcode wieder loswerden möchten, müssen Sie

nur das entsprechende Paket aus dem *Programme*-Verzeichnis auf den Papierkorb ziehen.

Hinweis

In Abschnitt 1.7 wird Ihnen noch eine alternative Möglichkeit zur Installation von Xcode vorgestellt.

1.4.2 Xcode im Detail

Nachdem Start von Xcode gibt es zwei Möglichkeiten, wie sich Xcode meldet: Entweder zeigt Xcode den Willkommens-Dialog an (siehe Abb. 1–2) oder es öffnet automatisch das bzw. die zuletzt geöffneten Projekte. Direkt nach der Installation wird der Willkommens-Dialog angezeigt.

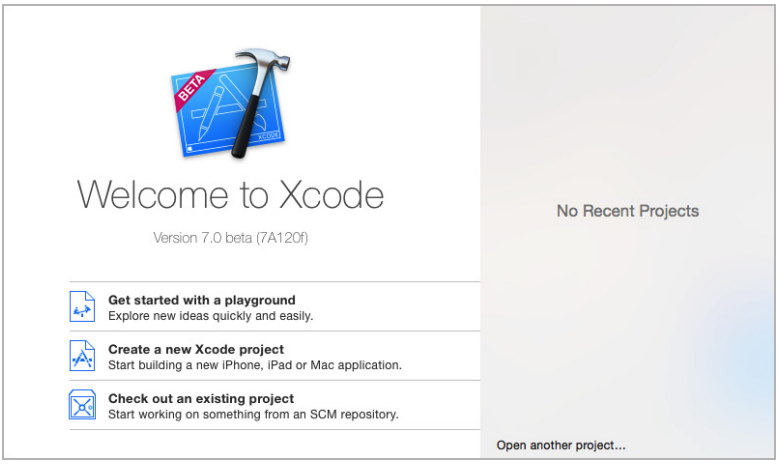

Abb. 1–2 *Der Willkommens-Dialog von Xcode*

Der Willkommens-Dialog von Xcode bietet unterschiedliche Optionen. Im linken Bereich werden unter anderem folgende Punkte angeboten:

- Neues Playground-Projekt
- Neues Xcode-Projekt (App)
- Ein Projekt aus einem angeschlossen Versionskontrollsystem auschecken

Wenn man bereits ein paar Projekte mit Xcode erstellt bzw. bearbeitet hat, dann werden zusätzlich im rechten Bereich die bisher geöffneten Projekte angezeigt. Wird ein neues Projekt angelegt, so zeigt Xcode das Projekt anschließend in der Übersicht an. In dieser Ansicht sind nun mehrere Bereiche von Xcode gut sichtbar (siehe Abb. 1–3).

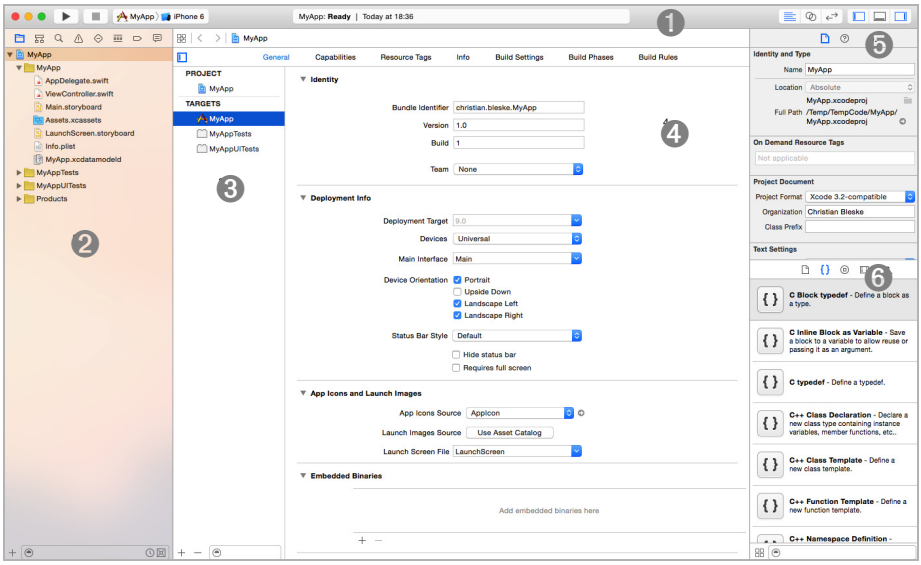

Abb. 1–3 *Die Bereiche von Xcode mit einem geöffneten Projekt*

Im oberen Bereich von Xcode (❶ in Abb. 1–3) befindet sich die Toolbar von Xcode. In der Toolbar wird mittig das gerade geöffnete Projekt angezeigt. Direkt unterhalb des Projektnamens befindet sich ein Textfeld, dem Sie den aktuellen Status des Projekts entnehmen können. Wenn Sie beispielsweise ein Projekt starten, so zeigt hier ein Fortschrittsbalken (blau), wie lange es noch dauert, bis das Projekt übersetzt und gestartet wird.

Im linken Bereich der Toolbar befindet sich die *Build and Run*-Schaltfläche, mit der das Projekt (im Simulator oder auf einem angeschlossenen iOS-Gerät) gestartet wird. Direkt danach kommt die Schaltfläche zum Beenden eines laufenden Projekts. Dann folgen der Projektname und das Gerät, auf dem die App gestartet wird. Auch wenn es nicht offensichtlich ist: Es handelt sich hierbei um ein Auswahlfeld. Aufgelistet finden Sie hier die möglichen (simulierten) Geräte sowie ein eventuell angeschlossenes iOS-Gerät.

Im rechten Bereich der Toolbar befinden sich sechs Schaltflächen. Die erste (von links nach rechts) aktiviert den *Standard Editor*. Darin werden die Projektdetails angezeigt, z. B. die Versionsnummer oder Informationen zum Deployment (Zielversion von iOS oder Zielgerät).

Unterhalb der Toolbar befindet sich im linken Bereich (❷ in Abb. 1–3) der *Project Navigator*. Diesem können Sie die aktuelle Struktur des Projekts entnehmen. Es handelt sich hierbei allerdings nicht um eine reine Ansicht auf Dateiebene, sondern um eine logische Ansicht des Projekts. Änderungen, die an dieser Stelle vorgenommen werden können, müssen sich aber nicht auf die physikalische Struktur des Projekts auswirken. Abhängig vom im *Project Navigator* ausgewählten Element des Projekts ändert sich auch die Ansicht in Xcode. Wählt

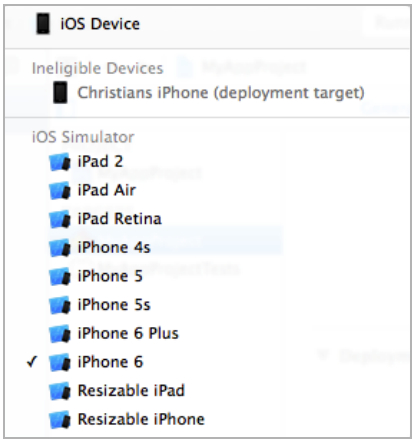

Abb. 1–4 *Auswahl des iOS-Geräts zur Ausführung des Projekts*

man an dieser Stelle beispielsweise eine Codedatei aus (ihr Name endet auf *.swift*), dann wird zentral das Codefenster von Xcode geöffnet.

Von links folgt dann die *Project and Target*-Übersicht (siehe ❸ in Abb. 1–3). Darin können Sie zwischen den Einstellungen für das Projekt selbst (*Project*) und der Konfiguration zur Erstellung der Anwendung (*Target*) umschalten. Im Target-Bereich befindet sich auch die Konfiguration der Anwendungstests.

In der Mitte (❹ in Abb. 1–3) sehen Sie das zentrale Fenster, in dem beispielsweise die Projektkonfiguration, der Quellcodeeditor oder auch der Interface Builder angezeigt werden.

Am rechten Rand von Xcode (❺ und ❻ in Abb. 1–3) befinden sich dann noch zwei Bereiche mit unterschiedlichen Funktionen. Im Bereich von ❺ werden die sogenannten Inspektoren angezeigt. Je nach Auswahl im Project Navigator oder Interface Builder werden in einem Inspektor unterschiedliche Informationen angezeigt bzw. können dort bearbeitet werden. Der letzte Bereich (❻ in Abb. 1–3) enthält ebenfalls unterschiedliche Werkzeuge. Hier befindet sich unter anderem die *Object Library* (enthält die Controls), die *File Template Library* (Vorlagen für Dateien) oder auch die *Code Snippet Library* (Codevorlagen).

1.4.3 Projektübersicht

Im letzten Abschnitt wurde unter anderem der *Project Navigator* bereits kurz vorgestellt. Dieser erlaubt die Navigation im Projekt. Im *Project Navigator* ausgewählte Elemente haben Einfluss auf das, was zentral in Xcode angezeigt wird. Wählt man in diesem Bereich z.B. eine Quellcodedatei aus, so wird automatisch der Codeeditor von Xcode geöffnet. Aber auch der Interface Builder lässt sich hier aktivieren. Hierzu müssen Sie im *Project Navigator* nur eine passende Datei auswählen, die den Interface Builder zur Darstellung verwendet. Sie erkennen diese Dateien an der Endung **.storyboard* und **.xib*.

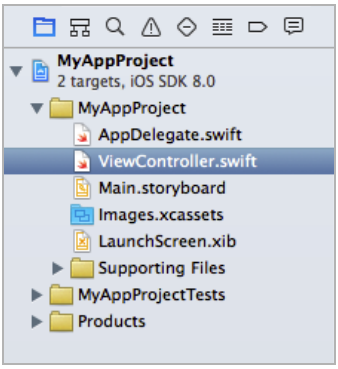

Abb. 1–5 *Der Project Navigator von Xcode*

Neben dem *Project Navigator* gibt es aber auch noch andere Funktionen, die in diesem Teil der IDE angezeigt werden. In der Leiste direkt oberhalb des Project Navigator gibt es dafür noch weitere Schaltflächen. Das erste Symbol (Ordner) aktiviert den Project Navigator. Der zweite Button aktiviert den *Symbol Navigator*, in dem die Klassen eines Projekts angezeigt werden. Es folgt der *Find Navigator*, über den sich im Projekt suchen lässt. Das nächste Icon aktiviert den *Issue Navigator*, der Informationen zu Problemen im Projekt enthält. Danach folgen der *Test Navigator* (Übersicht der Tests im Projekt), der *Debug Navigator*, der *Breakpoint Navigator* und der *Report Navigator*.

1.4.4 Quellcodefenster

Wählt man im *Project Navigator* eine Quellcodedatei (diese Dateien enden auf **.swift*) aus, so wird automatisch der Quellcodeeditor geöffnet.

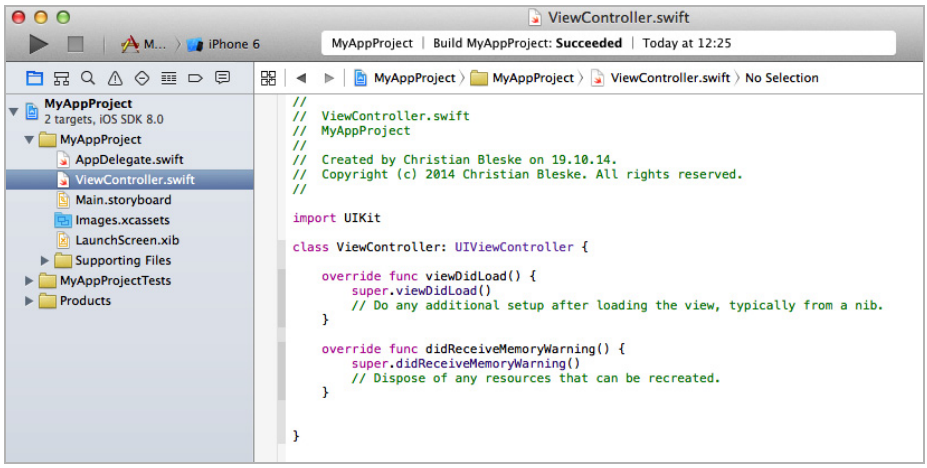

Abb. 1–6 *Der Codeeditor von Xcode*

Im Editor stehen alle bekannten Funktionen (Bearbeiten, Kopieren, Einfügen) zur Bearbeitung von Texten zur Verfügung. Die gerade geöffnete Quellcodedatei kann nicht nur dem *Project Navigator* entnommen werden, sie wird auch direkt oberhalb des Editors angezeigt. Eine nützliche Funktion ergibt sich in Verbindung mit der *Code Snippet Library*.

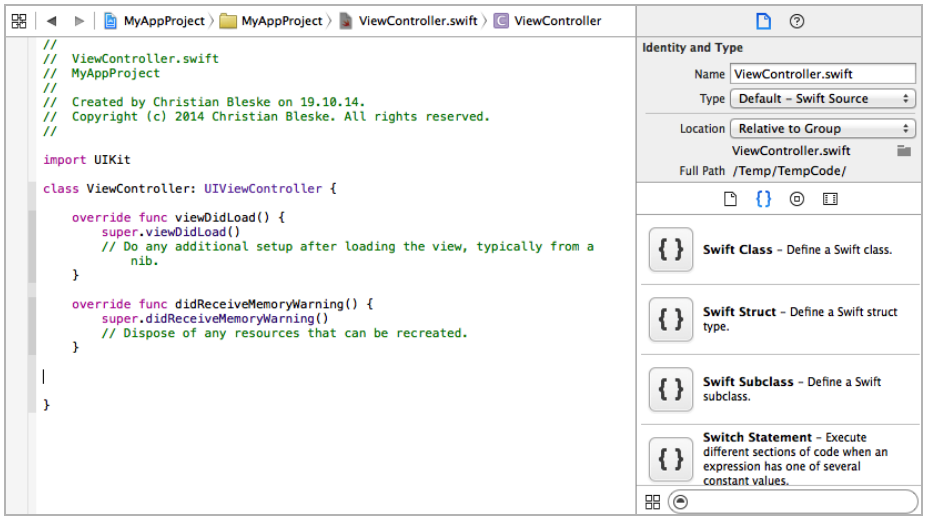

Abb. 1–7 *Code Snippet Library und Codeeditor*

Aus der *Code Snippet Library* können Codeblöcke via Drag & Drop im Editor abgelegt werden. An der abgelegten Stelle wird dann der jeweilige Codeblock (z. B. eine Klasse) automatisch eingefügt.

1.4.5 Interface Builder

Der Interface Builder ist das Werkzeug in Xcode, um Oberflächen für Anwendungen zu entwickeln. Ähnlich wie in anderen Entwicklungsumgebungen, so wird auch in Xcode eine Anwendung Formular für Formular entwickelt. Controls – beispielsweise Textfelder oder Schaltflächen – werden aus der *Object Library* (siehe auch Abschnitt 1.4.6) via Drag & Drop in das gerade im Interface Builder geöffnete View eingefügt.

In Abbildung 1–8 ist links neben dem *Interface Builder* das sogenannte *Document Outline* zu sehen. Nachdem das Label-Control in Abbildung 1–8 in das View eingefügt wurde, wird neben der Ansicht im Interface Builder auch das *Document Outline* aktualisiert. Darin werden die im View eingefügten Elemente innerhalb einer hierarchischen Ansicht angezeigt. Das *Document Outline* kann so auch zur Selektion von Elementen in einem View verwendet werden, wenn die direkte Auswahl im Interface Builder vielleicht nicht möglich ist – beispielsweise weil ein anderes Control das auszuwählende verdeckt. Um Eigenschaften eines Views oder eines Controls zu bearbeiten, werden die Inspektoren von Xcode verwendet.

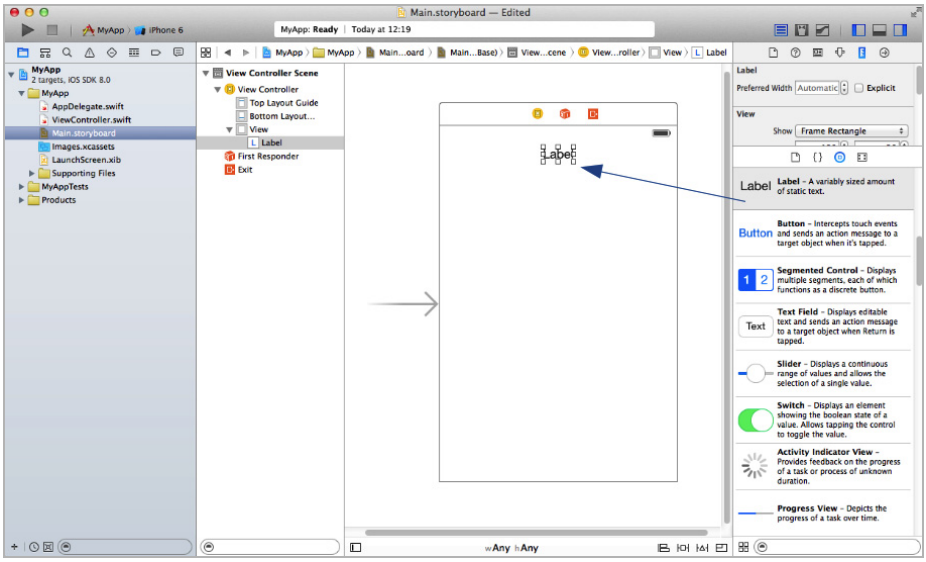

Abb. 1–8 *Der Interface Builder von Xcode*

1.4.6 Object Library & Co.

Im letzten Abschnitt wurde bereits eine Funktion angesprochen, die im rechten Bereich der IDE angesiedelt ist. Die *Object Library* enthält unter anderem Elemente (Controls), die innerhalb eines Views eingefügt werden können. Das ist aber längst nicht alles.

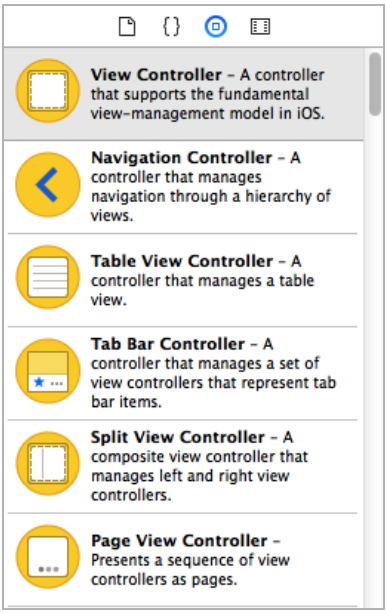

Abb. 1–9 *Die Object Library von Xcode*

In Abbildung 1–9 ist gut zu sehen, dass beispielsweise auch ein View Controller innerhalb der *Object Library* vorhanden ist. Wenn man eine App erstellt, die über mehrere Views verfügt, dann können in einer **.storyboard*-Datei zusätzliche Views eingefügt werden. Es ist also nicht erforderlich, für jedes neue View eine eigene Datei anzulegen.

Ein weiteres Fenster in diesem Bereich ist die bereits vorgestellte *Code Snippet Library,* die kurze Codeschnipsel zum Einfügen in den Xcode-Texteditor enthält. Neben diesen beiden Bibliotheken gibt es auch noch die *File Template Library*. In ihr sind Vorlagen für Dateien enthalten, die in ein Projekt eingefügt werden können. Auch das geschieht via Drag & Drop. Hierbei ist aber nicht der Interface Builder das Ziel der Drop-Aktion, sondern der *Project Navigator*. Um beispielsweise eine neue Swift-Code-Datei anzulegen, müssen Sie nur die entsprechende Vorlage (*Swift-File*) aus der *File Template Library* in das aktuelle Projekt einfügen, das gerade im *Project Navigator* angezeigt wird.

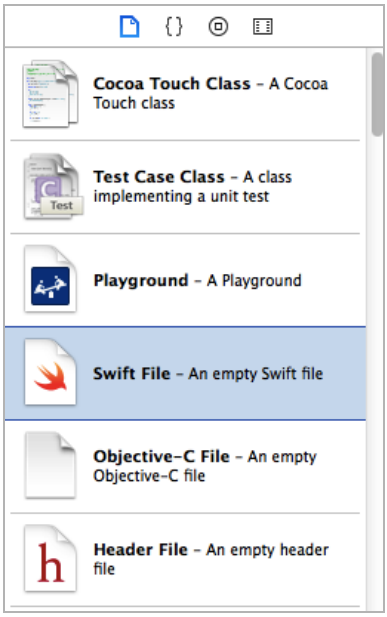

Abb. 1–10 *Die File Template Library von Xcode*

Hinter dem vierten Button in diesem Bereich verbirgt sich die *Media Library*. Hierbei handelt es sich um einen Shortcut auf die Medien-Elemente (z. B. Bilder, Audio- und Videodateien), die Sie einem Projekt hinzugefügt haben.

1.4.7 Inspektoren

Wichtig für die Arbeit in Xcode sind die sogenannten Inspektoren. Diese werden in der rechten oberen Ecke von Xcode angezeigt. Abhängig davon, was im Project Navigator oder im Interface Builder markiert wurde, wird automatisch der Inspektor geöffnet, mit dem sich die Eigenschaften des zuvor markierten Elements bearbeiten lassen. In Abbildung 1–11 wurde beispielsweise im Interface Builder ein Button-Control markiert.

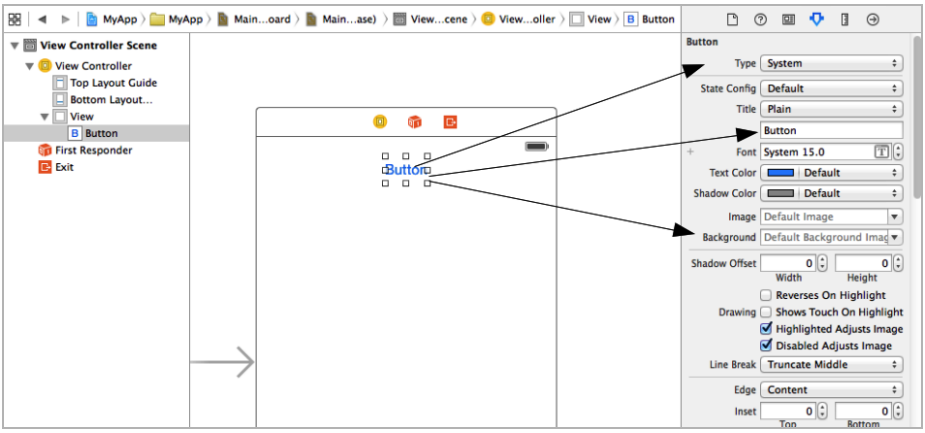

Abb. 1–11 *Der Attributes Inspector von Xcode*

Wird nun der *Attributes Inspector* geöffnet (siehe Abb. 1–11), so können die Eigenschaften des Buttons (z.B. seine Beschriftung oder ein Bild oder die Hintergrundfarbe) über den Inspektor konfiguriert werden.

Insgesamt gibt es sechs Inspektoren:

- Der erste (von links nach rechts) ist der *File Inspector*. Wie es die Bezeichnung schon vermuten lässt, enthält er Informationen zur im *Project Navigator* ausgewählten Datei. Hierbei stehen vor allem die Projektdatei sowie Storyboard- und Xib-Dateien im Fokus. Ist eine Storyboard-Datei ausgewählt, so werden im *File Inspector* beispielsweise Informationen angezeigt, ob die Option *AutoLayout* verwendet wird oder mit welcher Version von Xcode sich die Datei öffnen lässt.
- Es folgt der *Help Inspector*. Er zeigt eine Beschreibung des markierten Elements an und gibt ferner Auskunft darüber, seit welcher iOS-Version das entsprechende Element verfügbar ist.
- Im *Idenity Inspector* werden Metadaten zum markierten Objekt (z.B. der Klassenname) angezeigt.
- Als Nächstes kommt der (bereits besprochene) *Attributes Inspector*.
- Im *Size Inspector* werden Informationen zur Größe, Breite und ggf. zur Position eines gewählten Elements angezeigt.
- Der *Connections Inspector* enthält Angaben zu den Verbindungen eines Objekts. Mit »Verbindungen« ist hier beispielsweise die Verknüpfung einer Schaltfläche (Button) mit einem Stück Quellcode gemeint. In Kapitel 2 finden Sie ein Beispiel hierzu.

Sie dürften jetzt eine rudimentäre Vorstellung davon haben, was sich hinter einigen der Funktionen von Xcode verbirgt. In den folgenden Abschnitten werden einige Funktionen vertiefend besprochen.

1.5 Vorlagen in Xcode

Nach dieser Einführung wird es Zeit, sich etwas mit den in Xcode enthaltenen Vorlagen zur Erstellung von Apps zu beschäftigen. Eine App muss nämlich (analog zur anderen Entwicklungswerkzeugen) nicht komplett neu entwickelt werden. Xcode stellt Vorlagen bereit, mit denen sich auf Knopfdruck das Gerüst für einen bestimmten Typ von Anwendung erstellen lässt.

Bevor allerdings die Vorlagen zur Erstellung von Apps vorgestellt werden, besprechen wir einen für uns besonders interessanten Vorlagentyp, der seit Xcode 6 existiert: den Playground.

1.5.1 Schnellstart: Hallo Playground

Playground ist ein Typ von Vorlage, der mit Swift eingeführt wurde und auch nur mit Swift als Sprache funktioniert. Mit der *Playground*-Vorlage wird auch kein Gerüst für eine Anwendung erzeugt, sondern ein »Bereich«, um schnell und direkt Swift-Code ausprobieren zu können. Diese Umgebung ähnelt sicherlich ein wenig den früheren Basic-Interpretern. Startet man Xcode, so gibt es zwei Möglichkeiten, die *Playground*-Vorlage aufzurufen.

Im *Welcome to Xcode*-Dialog gibt es direkt einen entsprechenden Bereich *Get startet with a playground*, um einen neuen Playground anzulegen. Klickt man in den entsprechenden Bereich, so wird man aufgefordert, eine Bezeichnung einzugeben sowie zu entscheiden, ob der Playground in Verbindung mit iOS oder macOS angelegt werden soll. Nach Betätigung des *Next*-Buttons muss man sich noch entscheiden, in welchem Verzeichnis der neue Playground gespeichert werden soll. Ein Klick auf den *Create*-Button öffnet dann das Playground-Formular.

Alternativ hierzu kann über das *File*-Menü von Xcode der Punkt *New* und dann *Playground* aufgerufen werden. Im Formular ist schon etwas Code vorhanden. So findet sich eine *import*-Anweisung, die dafür sorgt, dass die Bibliothek UIKit geladen wird. In einer weiteren Zeile wird eine Zuweisung vorgenommen. Der lokalen Variablen *str* wird eine Zeichenkette zugewiesen. Im rechten Bereich des Playground-Formulars sieht man den Inhalt der Variablen nach der Zuweisung, und zwar zur Laufzeit. In einem Playground-Formular können Sie direkt Eingabentätigen, d.h., Programmcode schreiben. Probieren Sie es doch direkt einmal aus. Erfassen Sie die folgende Anweisung:

```
print("Hallo Playground")
```

Direkt nach der Eingabe der Anweisung wird im rechten Bereich eine Ausgabe angezeigt. Im rechten Bereich werden aber nur die Inhalte von Variablen bzw. Anweisungen angezeigt. Um Ausgaben im klassischen Sinne (z.B. Grafiken) anzuzeigen, muss der Ausgabebereich aktiviert werden. Dies geschieht über das Hauptmenü von Xcode mit dem Menüpfad *View* → *Assistant Editor* → *Show*

Assistant Editor. Nach Auswahl des Menüpunkts wird im Playground-Formular nun ein weiterer Bereich eingefügt.

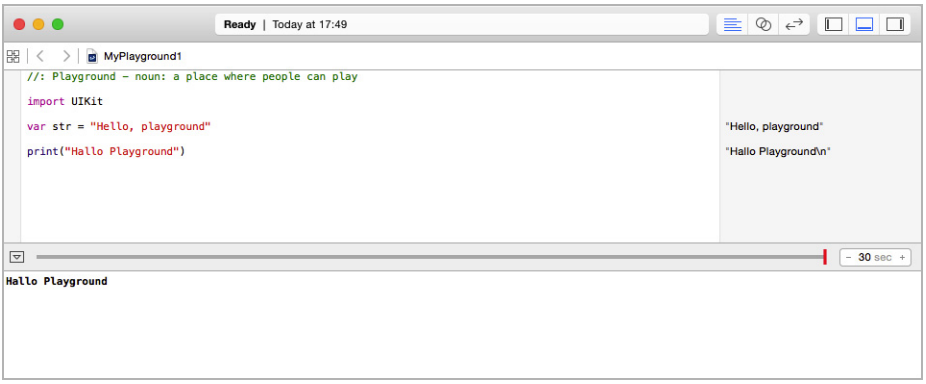

Abb. 1–12 *Das Playground-Formular*

Fehlermeldungen werden in der *Debug Area* angezeigt. Diese Ansicht aktiviert man über die Menüpunkte *View* → *Debug Area* → *Show Debug Area*. Im Playground können Sie also direkt Swift-Code ausprobieren, ohne erst ein Projekt anlegen zu müssen. In den Kapiteln 3 und 4 werden wir von dieser Möglichkeit reichlich Gebrauch machen.

1.5.2 Die Projektvorlagen von Xcode für iOS-Apps

Xcode enthält mehrere Vorlagen für iOS-Apps. Die Auswahl beginnt mit einer *Master Detail Application* und endet mit einer *Tabbed Application*. All diese Vorlagen sind für einen bestimmten Einsatzbereich gedacht – je nachdem, für welchen Einsatzzweck eine App vorgesehen ist. In den folgenden Abschnitten werden die vorhandenen Projektvorlagen vorgestellt, und es wird erläutert, wofür eine bestimmte Vorlage verwendet werden kann.

1.5.3 Schnellstart: Die Master-Detail-Application-Vorlage

Die Vorlage *Master Detail Application* ist für Apps gedacht, die Daten in Form einer Liste anzeigen. Typische Vertreter dieses Anwendungstyps sind z.B. die Mail-, SMS- oder Kontakte-App. Im Hauptdialog werden die Informationen in einer tabellarischen Ansicht angezeigt. Wählt man eine bestimmte Zelle aus, so wird im folgenden Schritt ein Formular angezeigt, das die Detailinformationen enthält. Neben der Master-Detail-Aufteilung ist in diesem Anwendungstyp automatisch ein Navigation Controller enthalten. Hierbei handelt es sich um den oberen Bereich der App, in dem Schaltflächen vorhanden sind, mit denen man sich

innerhalb der App vorwärts und rückwärts bewegen kann. Auswählen können Sie diesen Vorlagentyp über das Hauptmenü von Xcode: *File* → *New* → *Project* ...

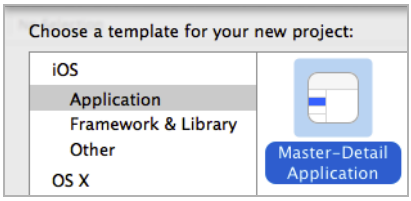

Abb. 1–13 *Die Master-Detail-Application-Vorlage*

Nach Auswahl der Vorlage wird der Anwender aufgefordert, einen Produktnamen einzugeben. Hier können Sie jetzt einmal direkt *MasterDetailApp* eingegeben und anschließend die *Next*-Schaltfläche betätigen. Im folgenden Dialog, in dem der Speicherort festgelegt wird, muss nun nur noch die *Create*-Schaltfläche betätigt werden. Die neue App wird dann automatisch angelegt. Nach Anlage des Projekts zeigt Xcode zunächst die Projektübersicht an.

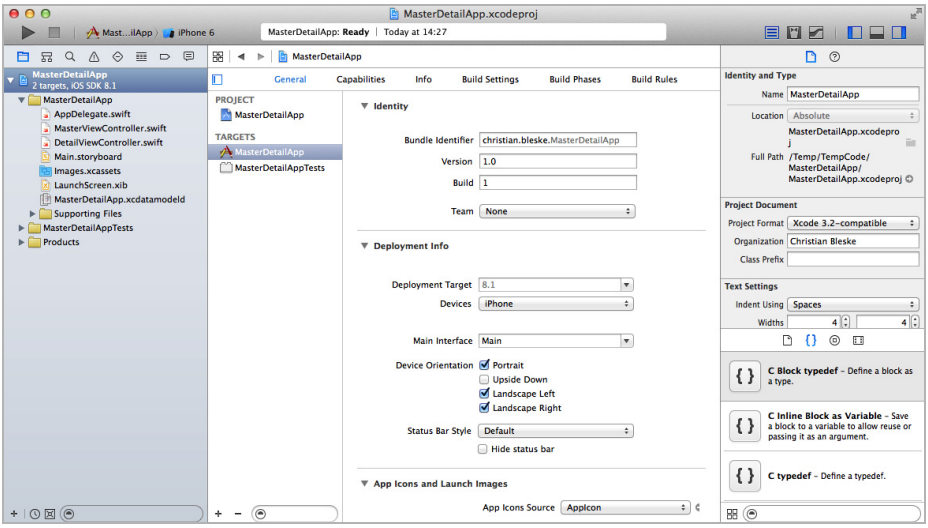

Abb. 1–14 *Die Master-Detail Application in der Projektübersicht*

Spielen Sie ruhig ein wenig mit der erstellten App. Um beispielsweise die Bezeichnung zu ändern, die in der Navigationsbar der App angezeigt wird, sind nur ein paar Handgriffe notwendig. Markieren Sie die Datei *Main.storyboard* im *Project Navigator* von Xcode. Markieren Sie den zweiten (!) Eintrag *Master Scene* im *Document Outline*, und erweitern Sie diesen Punkt, indem Sie das kleine Dreieck davor anklicken. Die nächste Ebene wird eingeblendet. Hier muss nun der Punkt *Master* markiert werden. Anschließend wird das zugehörige View im Interface Builder angezeigt. Erweitert man nun den Punkt *Master,* so werden zwei unterge-

ordnete Punkte (*Table View* und wieder *Master*) angezeigt. Markieren Sie den Punkt *Master* jetzt erneut (siehe Abb. 1–15).

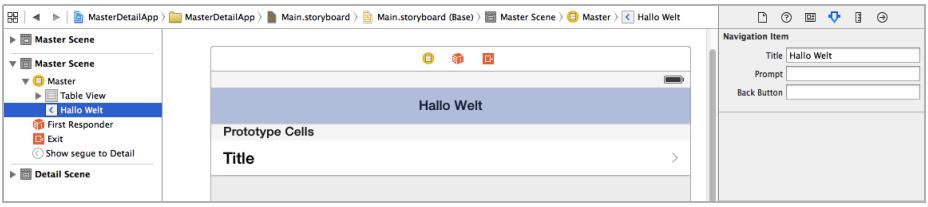

Abb. 1–15 *Das markierte Label-Control in einer Zelle (Cell)*

Öffnen Sie dann den Attributes Inspector (wenn er nicht bereits offen ist) von Xcode, suchen Sie die *Title*-Eigenschaft (sie befindet sich ganz oben), und überschreiben Sie den vorhandenen Text »Master« mit »Hallo Welt«. Starten Sie die App nun mit der *Build and run*…-Schaltfläche. Die App sollte starten, und in der Navigationsbar sollte der Text »Hallo Welt« stehen. Die App kann aber noch etwas mehr. Betätigen Sie einmal die +-Schaltfläche.

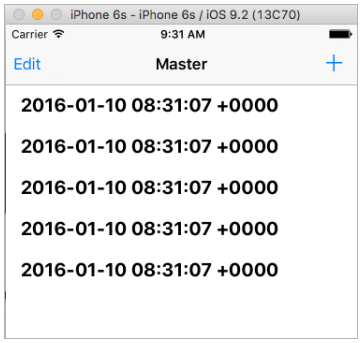

Abb. 1–16 *Die Master-Detail App im Einsatz (iPhone)*

Sie werden feststellen, dass automatisch neue Einträge mit Datum und Uhrzeit eingefügt werden. Wenn Sie einen Eintrag auswählen, dann wird sogar eine Detailansicht geöffnet. Auch das Löschen von Einträgen ist bereits möglich. Hierzu muss nur die *Edit*-Schaltfläche betätigt werden. Es ist also durch die Vorlage bereits eine vollständig funktionierende Master Detail Application erzeugt worden. Seit Xcode 7.1 verwendet die Vorlage noch zusätzlich einen SplitView-Controller. Welche Aufgabe hat der SplitViewController? Der SplitViewController sorgt auf einem iPad für eine alternative Darstellung der App im Gegensatz zur Anzeige auf einem iPhone.

Anstatt von der tabellarischen Übersicht in den Detailbereich zu wechseln (nur noch auf dem iPhone), wird auf einem iPad die tabellarische Übersicht

zusätzlich im linken Bereich neben einem Detailview eingeblendet. In Kapitel 7 wird die Vorlage noch einmal detailliert vorgestellt.

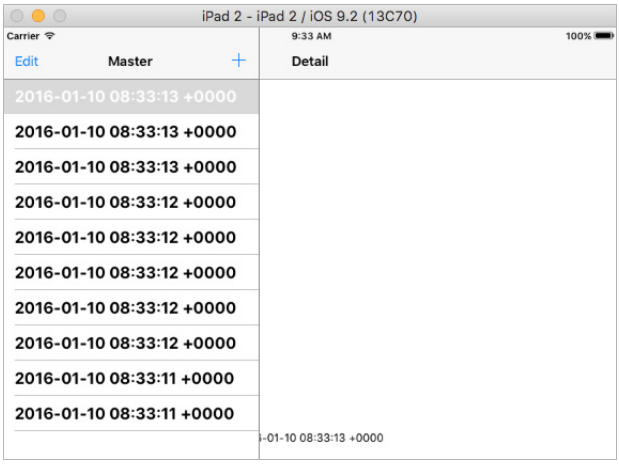

Abb. 1–17 *Die Master-Detail App im Einsatz (iPad)*

1.5.4 Schnellstart: Die Page-Based-Application-Vorlage

In einer *Page-Based Application* werden unterschiedliche Inhalte immer auf demselben Wege angezeigt. Denken Sie beispielsweise an ein Buch oder eine PDF-Datei. Ein gutes Beispiel aus den Standard-Apps von iOS ist iBooks. Navigiert wird in diesen Apps, indem die Seiten »umgeschlagen«, also via Swipe-Geste umgeblättert werden. Dieser Anwendungstyp verwendet im Gegensatz zur Master-Detail Application keine Navigationbar.

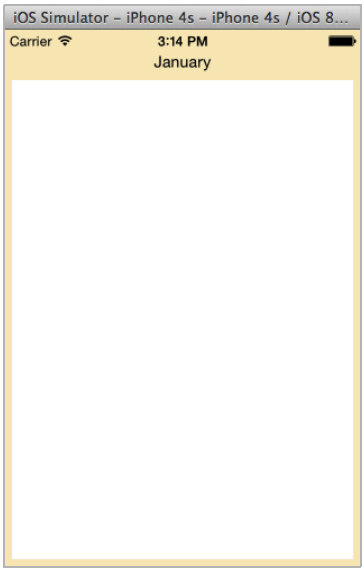

Abb. 1–18 *Eine Page-Based Application*

Ein Page-Based-Application-Projekt wird durch Auswahl der entsprechenden Vorlage angelegt. Startet man die App, so kann man vorwärts und rückwärts durch die Seiten »blättern«. Die Vorlage ist so weit ausgebaut, dass der Inhalt des oberen Label-Controls mit jedem Seitenwechsel erneut beschrieben wird.

1.5.5 Schnellstart: Die Single-View-Application-Vorlage

Die Single-View-Application-Vorlage ist die flexibelste Vorlage von allen. Nach Anlage eines Projekts enthält es nur ein (leeres) View. Wie schon bei den Vorlagen zuvor wird auch eine neue *Single View Application* durch Auswahl des entsprechenden Icons im Template-Auswahl-Dialog von Xcode angelegt.

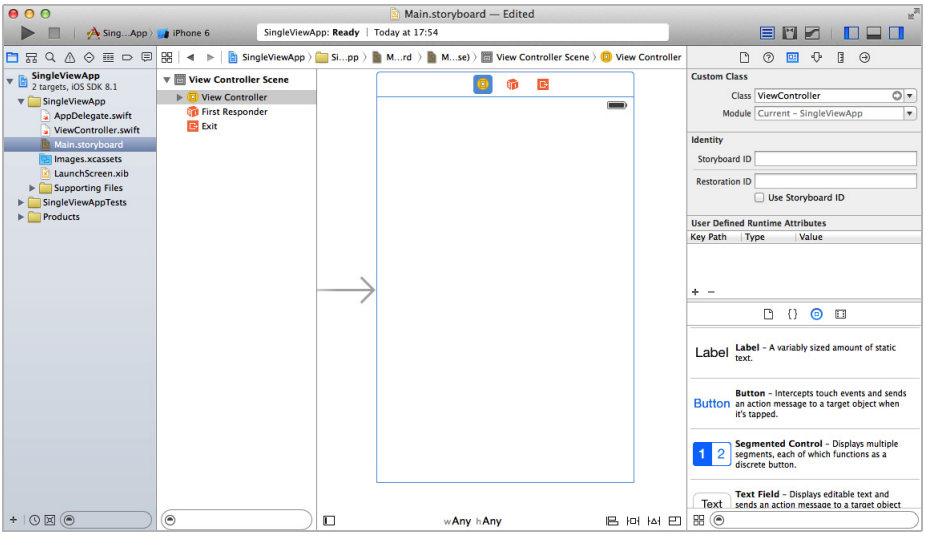

Abb. 1–19 *Eine Single View Application*

Die Single-View-Application-Vorlage wird ein zentraler Punkt der App sein, die wir in Kapitel 2 entwickeln. Aus diesem Grund finden Sie weitere Erläuterungen zu diesem Vorlagentyp im nächsten Kapitel.

1.5.6 Schnellstart: Die Tabbed-Application-Vorlage

Die letzte Vorlage, mit der sich eine typische iOS-App entwickeln lässt, ist die *Tabbed Application*. Dieser Vorlagentyp wird verwendet, wenn eine App mindestens zwei Views anzeigen soll, die über eine Registerlasche aktiviert werden können. Typische Vertreter dieser Gattung sind beispielsweise die *iTunes Store*- oder auch die *App Store*-App. Nach dem Anlegen des Projekts lohnt ein Blick ins Storyboard. Hier ist sehr gut zu sehen, wie dieser App-Typ aufgebaut ist (siehe Abb. 1–20).

Der sogenannte *Tab Bar Controller* bildet den Rahmen der App. Am unteren Rand des Tab Bar Controllers sind zwei Schaltflächen vorhanden, mit denen die beiden bereits integrierten Views aktiviert werden können. Startet man die App, so ist diese bereits vollständig funktionsfähig. Je nach Auswahl des entsprechenden Buttons wird auch das zugehörige View aktiviert und angezeigt.

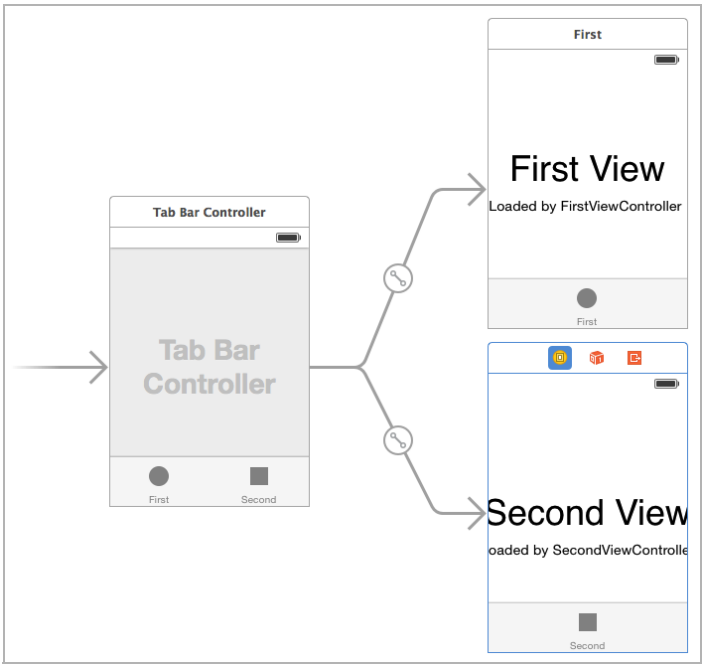

Abb. 1–20 *Eine Tabbed Application*

1.5.7 Bestandteile eines Projekts

Alle vorgestellten Vorlagen haben eines gemeinsam: Die erzeugten Apps verwenden in vielen Bereichen identische Projektdateien. In diesem Abschnitt wird ein Blick auf die Bestandteile eines Projekts geworfen und die Bedeutung der einzelnen Dateien vorgestellt.

Im *Project Navigator* werden die Bestandteile eines Projekts aufgeführt. Die aufgeführten Dateien und Verzeichnisse sind in der Regel Abbilder der Dateien, die im entsprechenden Projektverzeichnis auf der Festplatte zu finden sind. Dies wird deutlich, wenn man eine Datei markiert und über einen Mausklick das Kontextmenü aufruft. Wählt man hier den Menüeintrag *Show in Finder,* beispielsweise die Projektdatei, so wird das Projektverzeichnis im Finder von macOS angezeigt. Hier wird direkt deutlich, dass sich die Ansicht in *Project Navigator* nicht sehr von der Ansicht im Finder unterscheidet.

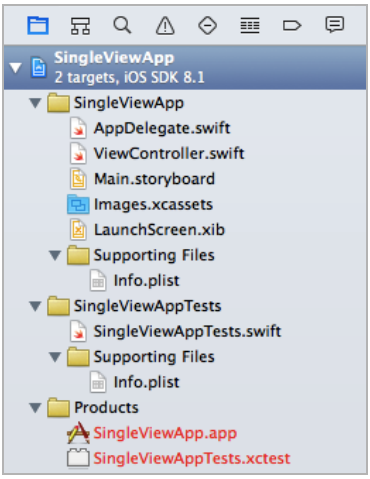

Abb. 1–21 *Ein Projekt im Project Navigator*

An der einen oder anderen Stelle gibt es aber Unterschiede. So sind beispielsweise die *Storyboard*-Datei oder auch *LaunchScreen.xib* im Dateisystem in einem eigenen Verzeichnis (*Base.lproj*) untergebracht. Diese Information kann man dem *Project Navigator* so nicht entnehmen.

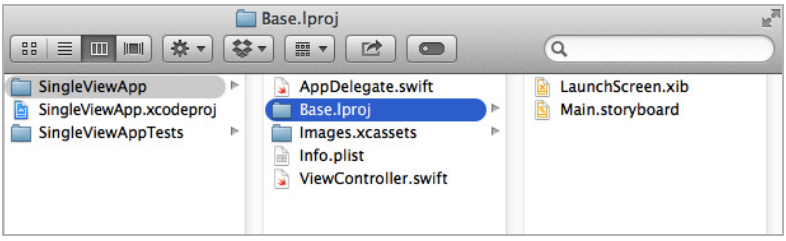

Abb. 1–22 *Dateien eines Projekts*

Zu einem Projekt gehört natürlich immer die Projektdatei, die den gewählten (Projekt-)Namen trägt und die Kennung **.xcodeproj* verwendet. Die Projektdatei enthält die wichtigsten Einstellungen zum Projekt. Hier werden unter anderem die Versions- und Buildnummer eingetragen. Außerdem legen Sie hier fest (*Deployment Target*), für welche iOS-Version das Projekt (minimal) kompiliert werden soll, und geben den Gerätetyp (iPhone/iPad oder Universal) an, der von der App unterstützt werden soll. Die Projektdatei besteht aus mehreren Abschnitten. Die gerade beschriebenen Optionen befinden sich im Abschnitt *General*.

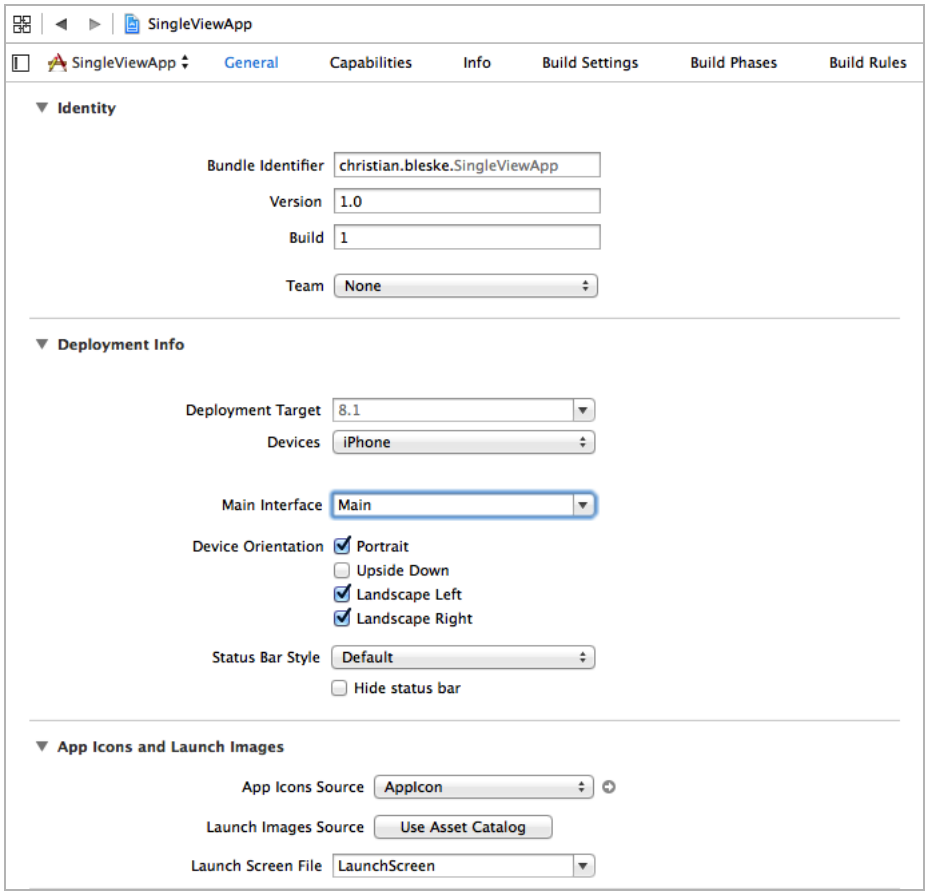

Abb. 1–23 *Die Projektdatei enthält die Konfiguration der App.*

Als Nächstes folgt der Abschnitt *Capabilities*. In diesem Bereich befinden sich ausschließlich Optionsschaltflächen, die ein- und ausgeschaltet werden.

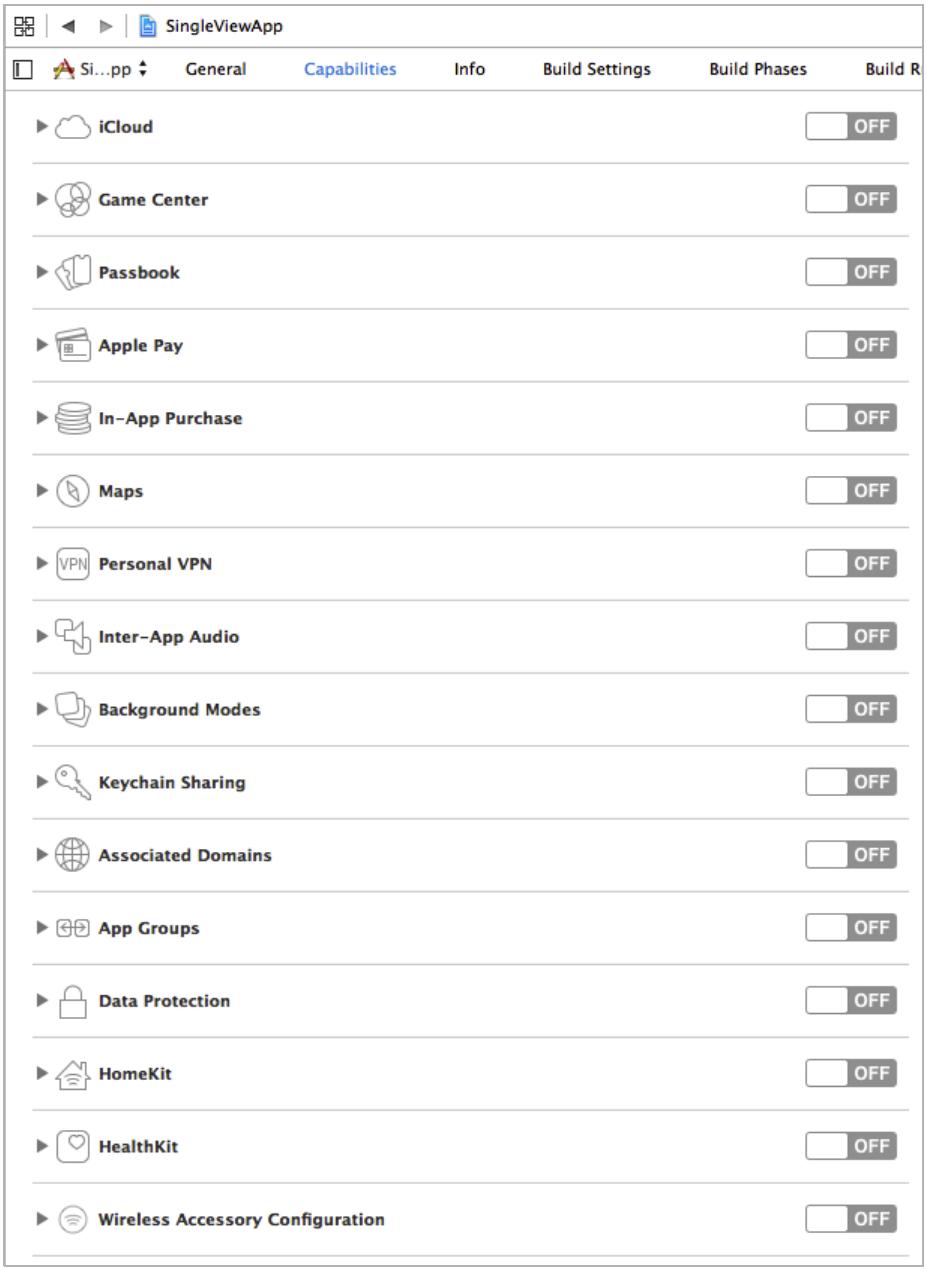

Abb. 1–24 *Der Capabilities-Abschnitt in der Projektdatei*

Die Schalter tragen den Namen eines Themas, das von der App unterstützt werden soll bzw. aus dem Funktionen integriert sind. Sobald Sie einen Schalter aktivieren, wird die App entsprechend markiert. Außerdem wird das erforderliche

Framework automatisch im Projekt verknüpft. Wenn Sie also eine App mit iCloud-Support entwickeln wollen, muss der entsprechende Schalter hier aktiviert werden. In Kapitel 9 wird dieses Thema noch einmal angeschnitten.

Im Abschnitt *Info* haben Sie die Möglichkeit, weitere Eigenschaften des Projekts, wie z.B. den ausführbaren Dateinamen (*Executable file*), festzulegen (wenn dieser sich vom Projektnamen unterscheiden soll) oder aber den Wert für die Basissprache (*Localization native development region*) der App (Englisch – natürlich) zu ändern. Möchten Sie Ihre App mit einem Dateityp verknüpfen (z.B. PDF), so können Sie das in diesem Abschnitt im Bereich *Document Types* festlegen.

Es folgen die *Build Settings, Build Phases* und *Build Rules*. Die Bezeichnungen dieser Abschnitte lassen es bereits vermuten: Hier werden Änderungen bzw. Einstellungen zur Erstellung der App vorgenommen. Diese Abschnitte sind äußerst umfangreich und beinhalten viele Optionen. So befindet sich beispielsweise im Abschnitt *Build Settings* ein Bereich (*Code Signing*), in dem festgelegt wird, welche Zertifikate zur Signierung einer App verwendet werden sollen. In der Regel müssen Sie sich mit diesem Bereich nicht oft auseinandersetzen, da viele Einstellungen automatisch vorgenommen werden. Manchmal lohnt es sich aber zu wissen, wo bestimmte Optionen zu finden sind.

Der Projektdatei sind drei Verzeichnisse untergeordnet. Das erste enthält die zur App gehörenden Code- (**.swfit*) und Interface-Builder-Dateien (**.xib und *.storyboard*). Auch dem Projekt zusätzlich hinzugefügte Dateien (wie z.B. Bild- oder Sounddateien) sind hier zu finden. Eine wichtige Datei befindet sich in einem untergeordneten Verzeichnis (*Supporting Files*). Sie heißt *Info.plist*; es handelt sich dabei um eine XML-Datei, in der Konfigurationsinformationen zum Projekt festgehalten werden. Beispielsweise findet man hier die Info, welche Ausrichtungen (Quer- und/oder Hochformat) die App unterstützt. Dass es sich um eine XML-Datei handelt, kann man im Übrigen auf den ersten Blick nicht sehen. Wenn Sie die Datei aber markieren, das Kontextmenü öffnen und dort den Menüpunkt *Open As → Soucre Code* aktivieren, dann wird die XML-Ansicht sichtbar. Der nächste Ordner unterhalb der Projektdatei trägt den Namen des Projekts mit dem angehängten Kürzel *Tests*. In diesen Dateien können Sie Unit-Tests programmieren. Im Ordner *Products* sind die übersetzten Dateien (App und Tests) zu finden.

1.6 Apps ausführen

Wie eine App aus Xcode heraus gestartet wird, wurde bereits erläutert. Sie wählen das Zielgerät in Xcode aus und betätigen die *Build and Run...*-Schaltfläche. Je nachdem, ob Sie als Ziel den Simulator oder ein angeschlossenes iOS-Gerät ausgewählt haben, wird die App nun automatisch dorthin kopiert und gestartet. Details, wie die Verwendung von unterschiedlichen iOS-Versionen oder Geräten, wurden noch nicht vorgestellt.

1.6.1 App im Simulator

Eine neue Hauptversion von Xcode wird ja in der Regel veröffentlicht, wenn es eine neue Version von iOS und/oder macOS gibt. Im Falle von iOS wird dann auch immer ein passender Simulator installiert, und sobald die App aus Xcode heraus gestartet wird, wird die aktuellste iOS-Version zur Ausführung der App im Simulator verwendet – natürlich nur, sofern der Simulator das Ziel der Installation ist. Tabelle 1–1 zeigt die Xcode-Versionen in Verbindung mit dem jeweiligen Betriebssystem.

Xcode-Version	Betriebssystem
Xcode 4.5	iOS 6
Xcode 4.6	iOS 6.1
Xcode 5.0	iOS 7 und OS X 10.9
Xcode 5.1	iOS 7.1
Xcode 6.0	iOS 8
Xcode 6.1	iOS 8.1 und OS X 10.10
Xcode 6.2	iOS 8.2 und OS X 10.10
Xcode 6.3	iOS 8.3 und OS X 10.10
Xcode 6.4	iOS 8.4 und OS X 10.10
Xcode 7.0	iOS 9 und OS X 10.11
Xcode 8.0	iOS 10 und macOS 10.12

Tab. 1–1 *Welche Xcode-Version für welches Betriebssystem?*

Gelegentlich kann es vorkommen, dass man eine App auch unter einer älteren Version von iOS testen möchte. Das kann beispielsweise dann der Fall sein, wenn sich ein API von einer iOS-Version zur nächsten geändert hat und man testen möchte, ob die App unter beiden Versionen noch funktioniert. Das kommt öfter vor, als man glaubt. Ein Beispiel hierfür ist die Änderung der Freigabe zur Verwendung der Ortungsdienste von iOS 7 zu iOS 8. In solchen Fällen ergibt es Sinn, eine App sowohl unter iOS 7 als auch unter iOS 8 zu testen, denn nicht alle (Apple-)Anwender aktualisieren das Betriebssystem direkt am ersten Tag. Man benötigt also in solchen Fällen mehrere Simulatoren in Xcode, um die App jeweils unter der entsprechenden Betriebssystemversion testen zu können. Woher aber kommen diese zusätzlichen Simulatoren? Die Antwort ist einfach: Man kann sie installieren. Ruft man in Xcode das *Preferences*-Menü auf, so wird ein Dialog von Xcode angezeigt, in dem sich Funktionen zur Anpassung von Xcode befinden. Unter anderem gibt es hier den Punkt *Downloads*. Aktiviert man ihn, so werden die zur Verfügung stehenden Downloads angezeigt.

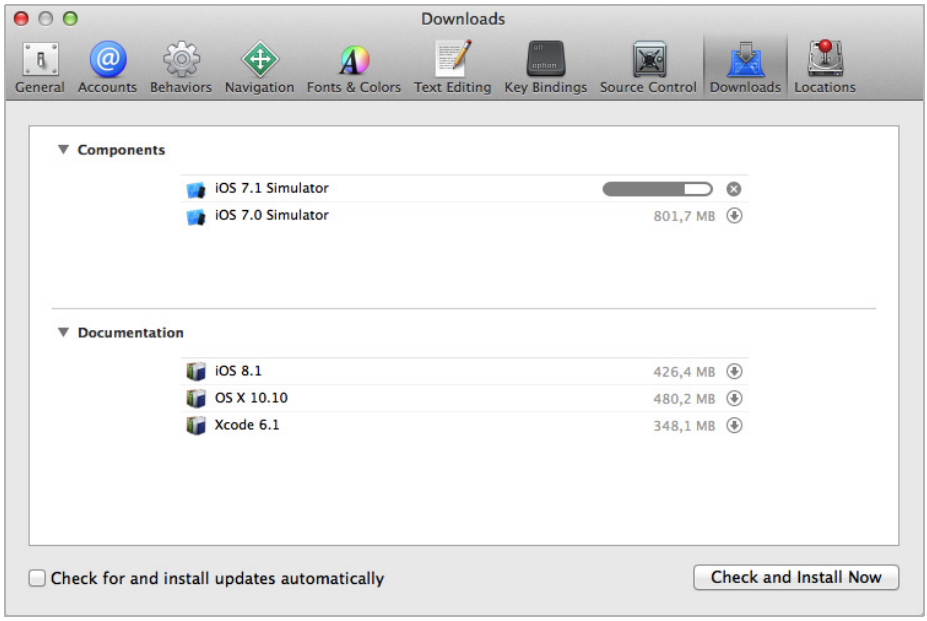

Abb. 1–25 *Der Download-Bereich von Xcode*

In der Auflistung findet man auch ältere (Simulator-)Versionen von iOS. Nach der Auswahl wird die entsprechende Version heruntergeladen und steht dann zur Verfügung. Die Liste zur Auswahl der Simulatoren ist nach Installation des Pakets natürlich entsprechend größer.

Abb. 1–26 *Auswahlliste von iOS-Simulatoren*

Neben den Standardgeräten (iPhone, iPad) mit den unterschiedlichen iOS-Versionen gibt es auch noch die Möglichkeit, als Ziel für die App ein *Resizable* iPhone oder iPad auszuwählen. Diese beiden Simulatoren bieten die Möglichkeit, die Breite und Höhe des Geräts zur Laufzeit (der App) festzulegen und so direkt zu beobachten, wie die App auf Änderungen der Größe reagiert.

1.6.2 Die App auf dem iOS-Gerät

Wie bereits in Abbildung 1–26 zu sehen ist, werden direkt an den Mac angeschlossene iOS-Geräte zur Auswahl angeboten, wenn eine App aus Xcode heraus gestartet werden soll. Hierfür ist es erforderlich, dass das entsprechende iOS-Gerät zuvor für die Entwicklung von Apps freigeschaltet wurde. Das bedeutet: Es ist ein Apple Developer Account erforderlich, bevor man die App auf das eigene oder ein fremdes iPad oder iPhone kopieren kann. Alle anderen Schritte unterscheiden sich nicht von der Verwendung eines Simulators: einfach das Gerät auswählen und die *Build and Run*-Schaltfläche betätigen. Anschließend wird die App automatisch auf das iOS-Gerät kopiert, dort installiert und zuletzt gestartet.

Hinweis

Ab Xcode 7.0 ist es erstmals möglich, eine App direkt auf das eigene an den Mac angeschlossene iOS-Gerät zu übertragen, ohne einen bezahlten Apple Developer Account zu haben. Wenn Sie also nur für den eigenen Bedarf und somit für die eigenen Geräte programmieren möchten, müssen Sie künftig kein Geld mehr ausgeben.

1.6.3 Der iOS-Simulator im Detail

In den meisten Fällen wird man zu Beginn der Entwicklung wohl den iOS-Simulator verwenden. Er bietet eine anständige Performance und ist gut in Xcode integriert. Der Simulator kann nicht nur in Verbindung mit einer App gestartet werden, auch der separate Aufruf ist möglich. Im Hauptmenü von Xcode finden Sie den Simulator auf folgende Weise: *Xcode → Open Developer Tools → iOS Simulator*. Nach Auswahl wird der Simulator in der zuletzt gewählten Konfiguration gestartet.

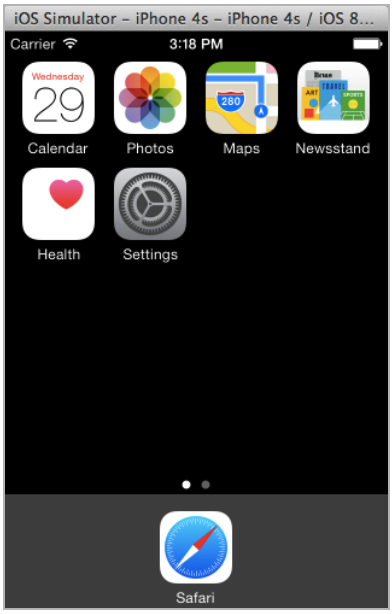

Abb. 1–27 *Der iOS-Simulator*

Da es sich bei dem Simulator um eine »normale« Anwendung handelt, verfügt diese auch über ein Menü. Über das Menü stehen nützliche Funktionen zur Verfügung. Beispielsweise kann via *File → Save Screen Shot* ein Bild des Simulators bzw. des gerade angezeigten Inhalts angefertigt werden.

Eine wichtige Funktion ist die Möglichkeit, das Gerät zurückzusetzen. Nach einigen Installationen oder auch dann, wenn man das System restlos verbogen hat, kann es nützlich sein, das Gerät zurückzusetzen. Dies geht über das Hauptmenü mit *iOS Simulator → Reset Content and Settings*. Im Anschluss erfolgt noch eine Sicherheitsabfrage, und dann wird der iOS-Simulator neu initialisiert. Auf dem iOS-Simulator installierte Apps verbleiben dort, auch nachdem Xcode beendet wurde. In diesen Belangen verhält sich der iOS-Simulator wie ein echtes Gerät.

Weitere interessante Funktionen gibt es unterhalb des Menüs *Hardware*. Der Menüpunkt *Device* bietet die Möglichkeit, zur Laufzeit des Simulators das »Gerät« umzuschalten. Das heißt, man kann den Simulator von einem iPhone 7 auf ein iPhone SE umschalten. Im Prinzip wird dabei nur die Auflösung des Simulators geändert. Es stehen dieselben Geräte zur Auswahl wie in Xcode inklusive des *resizable* iPhone und iPad. Der Simulator kann natürlich auch gedreht werden. Hierzu gibt es ebenfalls unterhalb des Menüpunkts *Hardware* entsprechende Menüpunkte (*Rotate Left & Right*). Um eine Schüttelgeste zu simulieren, gibt es die Funktion *Shake Gesture*. Der *Home*-Button wird mit der *Home*-Funktion

simuliert, und über die Funktion *Simulate Memory Warning* kann knapper Hauptspeicher simuliert werden.

Der Simulator enthält allerdings nicht alle Standard-Apps. So fehlt beispielsweise die Telefon-App oder auch die SMS-App. Was kann man mit dem Simulator nicht machen? Es liegt auf der Hand: Alles, was in Verbindung mit Sensoren verarbeitet wird (z.B. Kamera oder Kompass), lässt sich im Simulator eher schlecht oder auch gar nicht testen. In solchen Fällen kommt man um ein *iDevice* nicht herum.

Hinweis

Wenn Sie den Simulator zur Entwicklung von Apps nutzen, empfiehlt es sich, die Landeseinstellungen und die Sprache auf Deutsch festzulegen. Das wird direkt im Simulator über das bekannte Settings-Menü (Einstellungen) erledigt.

1.7 Mehrere Xcode-Versionen parallel verwenden

Mehrere Versionen des iOS-Simulators sind also kein Problem. Manchmal kann es aber auch nützlich sein, unterschiedliche Versionen von Xcode vorzuhalten. Jedes Jahr erscheinen neue Versionen von iOS und macOS und natürlich auch von Xcode. In der Regel kann man davon ausgehen, dass pro Jahr mindestens zwei neue Major-Versionen nebst Bugfixes erscheinen. Das sind dann mindestens vier Versionen im Jahr. Zusätzlich gibt es auch noch Beta-Versionen von zukünftigen Xcode-Versionen. Dass sich bei diesem Tempo auch der ein oder andere Fehler in eine neue Xcode-Version einschleicht, ist fast normal. Als Entwickler sollte man daher darauf achten, dass man mindestens eine stabile Version vorhält, um notwendige Updates einer App im Store ausliefern zu können. Zusätzlich hat man (wahrscheinlich) mindestens eine weitere Version installiert, um neue Funktionen ausprobieren zu können. Man hat dann also immer mindestens zwei Versionen auf dem Mac. Hier sollte man sich eine kleine Strategie zurechtlegen, damit man nicht durcheinanderkommt.

1.7.1 Ältere Versionen von Xcode finden

Sollten Sie eine benötigte Xcode-Version zu früh gelöscht haben, dann können Sie auch diese (ältere) Version wieder problemlos installieren. Allerdings ist im Mac App Store immer nur die neueste Version verfügbar. Sie müssen also wissen, wo man ältere Versionen von Xcode findet. Im ersten Schritt müssen Sie hierfür das Apple Developer Center im Browser aufrufen. Suchen sie dann den Link mit dem Namen Develop und klicken Sie darauf. Anschließend werden mehrere Bereiche angezeigt, wählen Sie hier Xcode aus.