O'REILLY®

Durchstarten mit Swift

SCHRITT FÜR SCHRITT ZUM ERFOLGREICHEN iOS-PROJEKT

Stefan Popp & Ralf Peters

Inhalt

KAPITEL 5

Hello World

Ein iOS-Projekt erstellen

Ein iOS-Projekt kann ebenso leicht erstellt werden wie ein Playground. Neben dem Shortcut ⌘+⇧+N haben wir im *Xcode File Menu* das Submenü *New*, unter dem sich der Menüeintrag *New Project* befindet. Zusätzlich existiert im Willkommensfenster, das beim Starten von Xcode erscheint, der *Create a new Xcode Project*-Button. Verwenden Sie eine der drei Methoden, um den Wizard zum Erstellen des neuen Projekts zu öffnen.

Über das Tastenkürzel ⌘+⇧+1 lässt sich das Willkommen-Fenster öffnen, vorausgesetzt, Xcode ist die aktuell aktive Applikation.

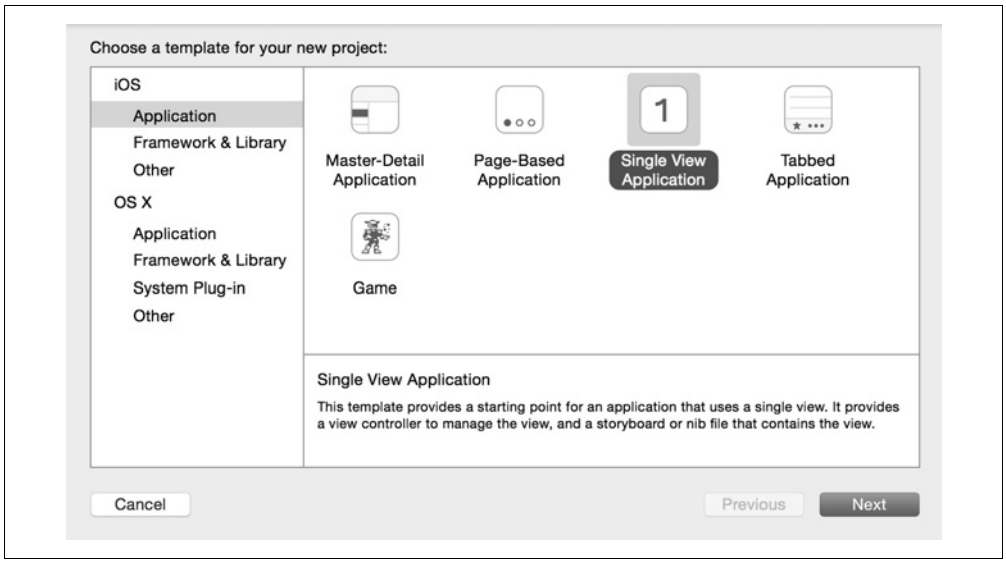


Abbildung 5-1: Maske, um ein Xcode-Projekt zu erstellen

Die Maske zum Erstellen des Projektes (siehe Abbildung 5-1) beinhaltet verschiedene Zielsysteme. Stellen Sie sicher, dass Sie den Punkt *iOS* oder den darunter liegenden Punkt *Application* gewählt haben. Für iOS stellt Xcode verschiedene Templates zur Entwicklung von Apps oder Bibliotheken zur Verfügung. Dies kann z. B. eine statische Bibliothek oder auch ein Template-Projekt zum Erstellen von Spielen mithilfe von *SpriteKit* sein. Für unsere erste App entscheiden wir uns für die *Single View Application*. Dieses Template erzeugt ein Storyboard, das exakt einen ViewController enthält sowie die dazugehörige View-Controller-Klasse. Über einen ViewController wird die Funktionalität der Ansicht sichergestellt. Er übernimmt die Aufgaben des Controllers im MVC-Architekturmuster. Details zu diesem Konzept finden Sie in der iOS Developer Library. Nachdem wir uns für dieses Template entschieden und dies mit dem *Next*-Button bestätigt haben, folgt eine Eingabemaske, in der verschiedene Projekteigenschaften festgelegt werden müssen. Der *Product Name* entspricht unserem App-Namen. Das Feld *Organization Name* kann Ihren oder den Namen Ihrer Firma enthalten. Der *Organization Identifier* setzt sich im Normalfall aus einer Domain zusammen, die aber mit der Top-Level-Domäne beginnt. In Abbildung 5-2 ist diese als *de.swift-blog* festgelegt. Der *Bundle Identifier* setzt sich automatisch aus dem *Product Name* und dem *Organization Identifier* zusammen.

Daten wie den *Bundle Identifier* oder auch den *Product Name* kann man später noch in der *Info.plist*-Datei ändern, die automatisch in jedem Projekt erzeugt wird.

Im Dropdown-Menü *Language* wird die Entwicklungssprache festgelegt, mit der das Projekt umgesetzt werden soll. Für unser Projekt verwenden wir Swift, es ist aber später möglich, andere Programmiersprachen in einem Projekt zu mischen, sofern dies nötig ist. Im Dropdown-Menü *Device* legt man fest, für welche Produktfamilie die App spezialisiert sein soll. Hier definieren Sie, ob es sich um eine reine *iPad-* oder *iPhone-App* handelt oder um eine sogenannte *Universal App*, die beide Targets enthält. Für unsere Einsteiger-App werden wir uns auf das iPhone festlegen. Die Checkbox *Use Core Data* wird von uns deaktiviert. Core Data ist ein Objektgraph, den Sie später noch kennenlernen, aber in diesem Projekt nicht benötigen.

Nachdem alle Eigenschaften und Felder gesetzt sind, beenden wir den Wizard mit einem Klick auf den *Next*-Button und wählen zu guter Letzt noch den Speicherort für unsere App aus.

In Xcode-Versionen vor Xcode 6 wurde für das iPad und das iPhone jeweils ein Storyboard erstellt. Seit Version 6 teilen sich beide Plattformen ein Storyboard. Mithilfe von *Auto Layout* und *Size Classes* wird dieses Verfahren ermöglicht. Es ist aber kein Bestandteil dieses Buchs.

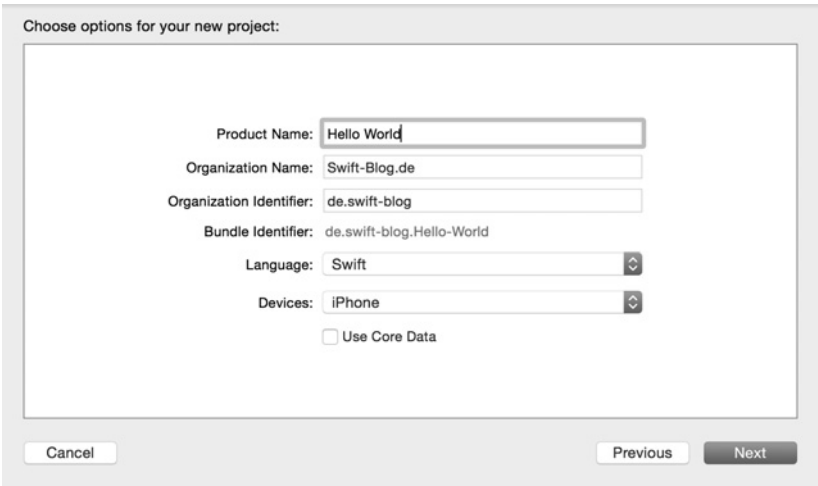


Abbildung 5-2: Die Eingabemaske nachdem Ausfüllen

In dem Auswahlfenster für den Speicherort befindet sich unter der Dateiauflistung eine Checkbox, um unser Projekt mit einer Git-Versionskontrolle zu erstellen (siehe Abbildung 5-3). Sie können diese Funktion gern aktivieren, jedoch werden wir in diesem Projekt keine Versionsverwaltung benötigen.

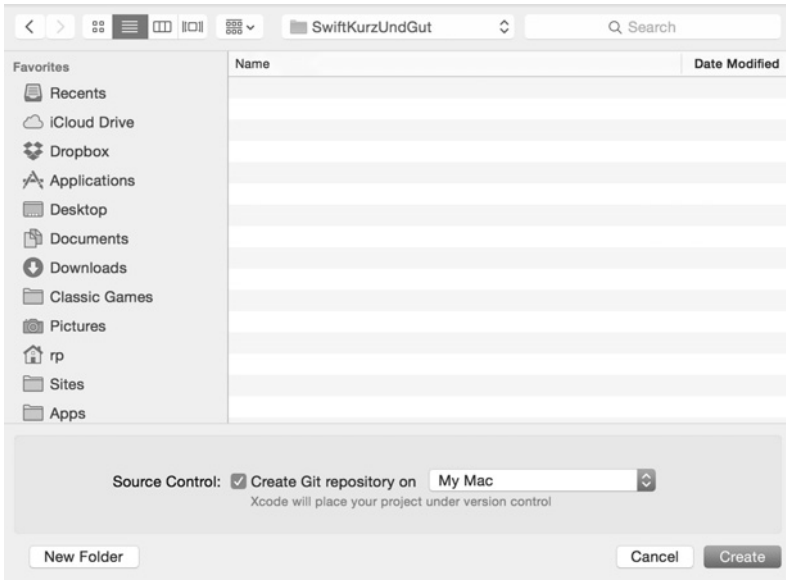


Abbildung 5-3: Auswahl des Speicherorts mit aktivierter Versionskontrolle

Sobald das Projekt erstellt ist, befindet sich links eine Auflistung der Dateien und Ressourcen des Projektes. Die Datei *AppDelegate.swift* ist eine Klasse, die verschiedene Funktionen beinhaltet, die aufgerufen werden, wenn die App verschiedene Zustände annimmt oder ein Benutzer bestimmten Berechtigungen, wie den Push Notifications, zugestimmt hat. In der Datei *ViewController.swift* befindet sich die Klasse, in der wir unserer App ein bisschen Leben einhauchen werden. Das Storyboard befindet sich in der Datei *Main.storyboard*, die eine visualisierte Darstellung der App ist. Sie enthält neben den View Controllern auch Interaktionen und Verläufe, bei denen man gut erkennen kann, wie innerhalb der App navigiert wird. Die Ressourcendatei *Image.xcassets* soll möglichst alle Grafiken beinhalten, die in unserer App genutzt werden. Sie veranschaulicht nicht nur, welche Grafiken existieren; mit Ihrer Auflistung nach verschiedenen Geräteklassen kann man auch gut und schnell erkennen, für welche Displays noch keine Grafiken vorhanden sind, was bei der Benutzung dieser Datei auch durch Warnungen während der Laufzeit unterstützt wird.

Die ersten UI-Elemente hinzufügen

Kaum etwas macht einem Entwickler mehr Spaß, als schnell eine App zusammenklicken zu können, um schnell einen Gedanken, eine Idee vom Stammtisch oder auch einfach nur ein Hirngespinst zu realisieren. Das Storyboard nimmt uns viele Aufgaben ab, die auf anderen Plattformen ohne Routine selbst für einfache Fenster eine halbe Ewigkeit benötigen.

Unsere erste App wird sehr einfach sein, beinhaltet aber bereits wichtige Schritte, um Ideen visualisieren zu können. Wir starten unser Storyboard, das im Interface Builder geöffnet wird.

In unserem Beispielprojekt möchten wir nicht mit Size Classes arbeiten, da diese genutzt werden, um gemeinsame Oberflächen für iPads und iPhones in einem Storyboard zu erstellen. Dies ist ein eigenes Thema, das sehr fortgeschritten ist. Weil es auch nicht Bestandteil dieses Buches ist, werden wir diese Option deaktivieren. Dafür deaktivieren wir im File-Inspector-Tab des Utilities-Bereichs die Option *Use Size Classes*, wie in Abbildung 5-4 zu sehen ist. Dabei wird ein Dialog angezeigt, in dem gefragt wird, zu welcher Produktreihe die bestehenden Views konvertiert werden sollen. Wir wählen hier das iPhone.

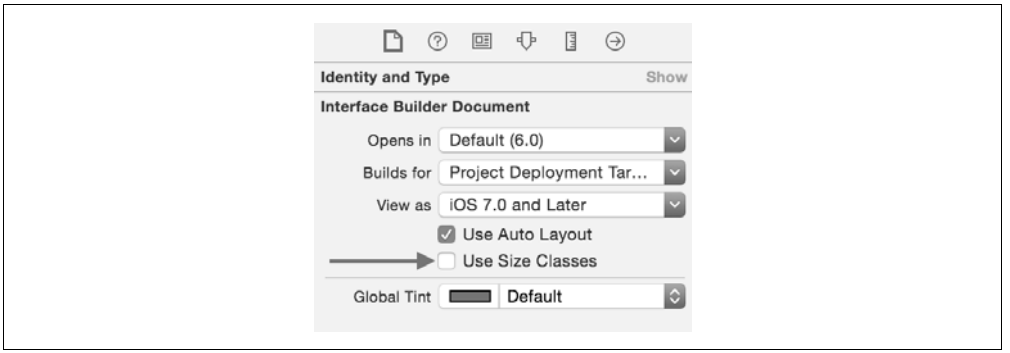


Abbildung 5-4: Deaktivieren der Size Classes

Wir betrachten jetzt unseren ersten ViewController, der beim Erstellen des Projekts automatisch erzeugt wurde. Er sollte wie in Abbildung 5-5 leer sein. Der Pfeil, der von links auf den View Controller zeigt, teilt uns mit, dass es sich bei diesem ViewController um den Einstiegspunkt des Storyboards (den *Initial Starting Point*) handelt. Dieser ViewController wird als erster instanziiert und angezeigt, sobald die App geöffnet wird. Ohne diesen Einstiegspunkt wird die App nichts anzeigen, und eine Navigation oder Interaktion wäre nicht möglich.

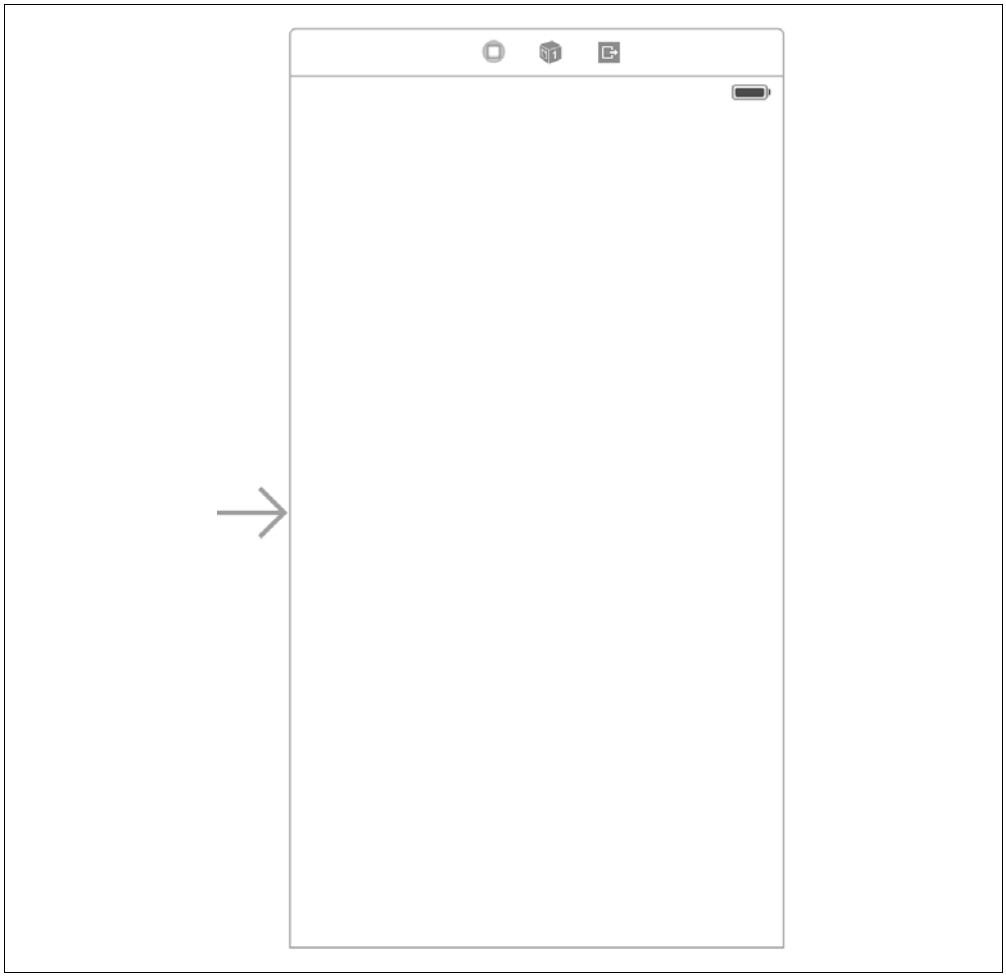


Abbildung 5-5: Ansicht eines leeren ViewControllers

Im Utilities-Bereich befindet sich im unteren Teil eine Bibliothek, aus der man mithilfe von Drag-and-Drop Elemente schnellstmöglich in den ViewController oder in das Storyboard platzieren kann. UI-Elemente befinden sich im Reiter OBJECT LIBRARY, wie in Abbildung 5-6 dargestellt. Diese Bibliothek umfasst nicht nur einfache Elemente (wie ein

Label oder einen *Button*), sondern neben normalen ViewControllern auch spezialisierte Elemente, wie einen *CollectionViewController*, verschiedene *Gesture Recognizer* oder auch *Container Views*. Eine Filterung gezielt nach dem Elementnamen ist über das Suchfeld unterhalb der Objektübersicht möglich.

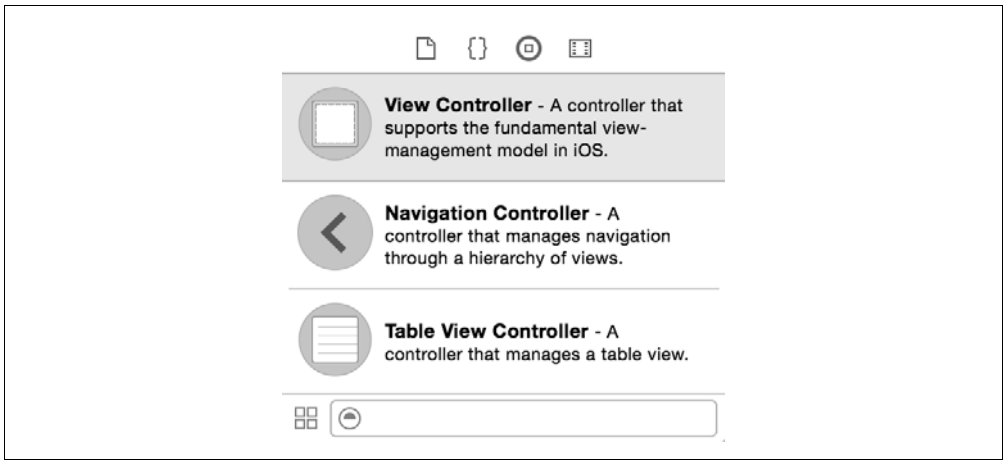


Abbildung 5-6: Übersicht der Object Library

Das erste Element, das wir platzieren möchten, ist ein sogenanntes Label. Ein Label repräsentiert einen statischen Text und hat verschiedene Eigenschaften, wie z. B. Farbe, Schriftart oder auch Schriftgröße. Am einfachsten finden Sie das Label, indem Sie den Objektfilter unterhalb der Objektbibliothek benutzen und das Wort »Label« eingeben, wie in Abbildung 5-7 zu sehen ist.

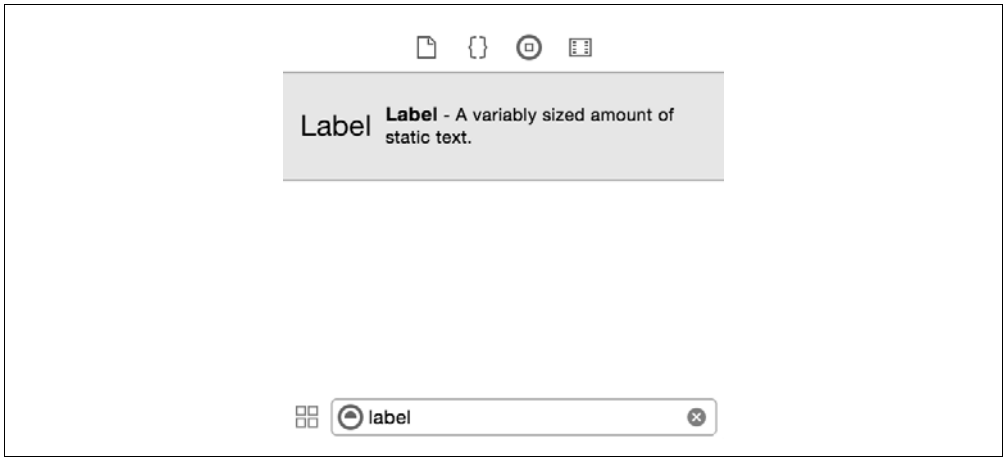


Abbildung 5-7: Gefiltertes Element in der Objektbibliothek

Um das Label zu platzieren, müssen Sie es einfach via Drag-and-Drop aus der Bibliothek herausziehen und auf dem View an der gewünschten Stelle ablegen. Der Interface Builder bietet uns beim Platzieren der Elemente Hilfslinien an, mit denen wir das Objekt einfacher ausrichten können. Zusätzlich erleichtern sie das Ausrichten der Elemente, vor allem dann, wenn bereits andere Elemente vorhanden sind.

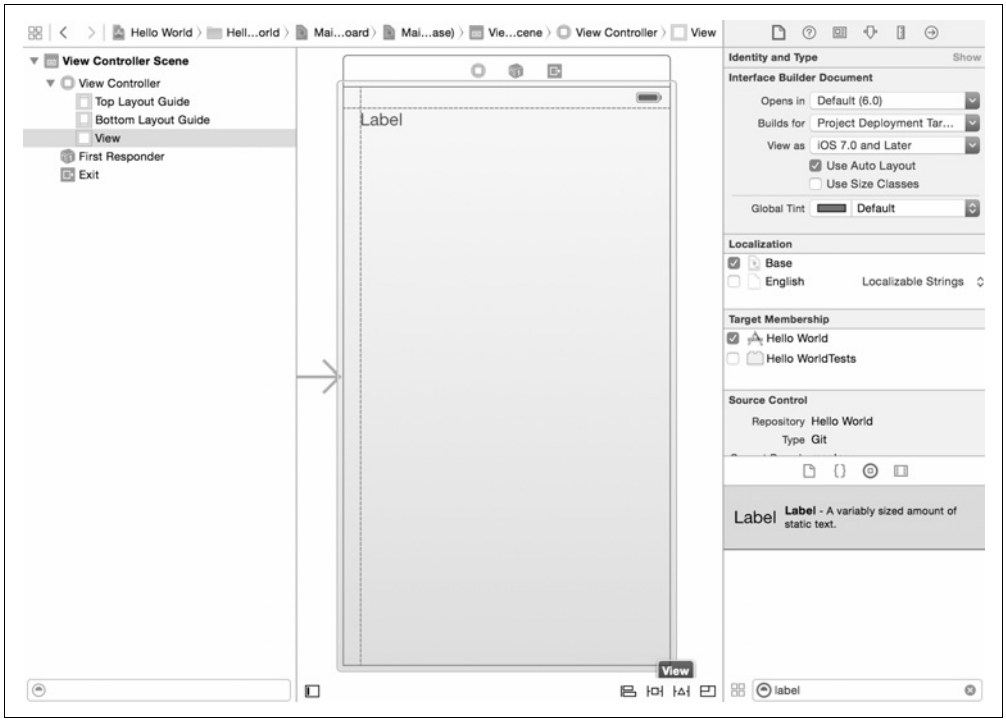


Abbildung 5-8: Objekt mit Drag-and-Drop platzieren

Damit wir den Text des Labels ändern können, bietet uns der Interface Builder die Möglichkeit, einfach auf das Label doppelzuklicken. Alternativ kann man das Element auch wie in Abbildung 5-9 mit einem einzigen Klick selektieren und den Wert über den *Attribute Inspector* ändern, der sich im Utilities-Bereich von Xcode befindet. Im Attribute Inspector finden Sie neben der Möglichkeit, den Labeltext anzupassen, auch Optionen zum Einstellen der Textfarbe, der Ausrichtung oder auch der Eigenschaften, die das Rendering-Verhalten beeinflussen können. Unser Label soll den Text »Hallo Welt« darstellen. Sollte das Label zu klein sein, können Sie es einfach selektieren und mit einem einzigen Klick nach Belieben vergrößern. Im Utilities-Bereich findet sich auch der SIZE INSPECTOR, in dem Sie die Möglichkeit haben, die Größe der Elemente pixelgenau einzustellen.

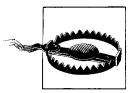
Abbildung 5-9: Verändern der Labeleigenschaften im Attribute Inspector

Nachdem das Label platziert ist, fügen wir noch ein weiteres Element hinzu. Wir suchen oder filtern in der Objektbibliothek nach dem Element *Text Field* und ziehen dieses unter das von uns erzeugte Label auf unseren View Controller. Ihnen wird dabei auffallen, dass das Text Field in der Nähe des Labels automatisch an verschiedenen Positionen einrastet, damit Sie es am Label ausgerichtet platzieren können. Durch einen Doppelklick auf das Text Field können Sie einen Text eingeben, der standardmäßig das Text Field füllt. Dieser Text ist aber kein Platzhalter und verschwindet später nicht automatisch, sobald ein Benutzer auf das Text Field drückt. Einen Platzhaltertext können Sie im Storyboard nur über den ATTRIBUTE INSPECTOR festlegen.

Wir haben die ersten Schritte für unsere UI abgeschlossen. Ehe wir der App ein wenig Leben einhauchen, beschäftigen wir uns zunächst mit dem Simulator. Wenn Sie es nicht zu eilig haben, empfehlen wir Ihnen, ein paar Elemente nach Belieben hinzuzufügen und ihre Eigenschaften im ATTRIBUTE INSPECTOR oder SIZE INSPECTOR zu verändern. Sie können Änderungen jederzeit über das EDIT-Menü von Xcode oder mit der Tastenkombination ⌘+Z rückgängig machen. Probieren Sie doch einmal, die Hintergrundfarbe des Views unseres View Controllers blau zu färben.

Der erste Start im Simulator

Der Simulator ist für uns während der Entwicklung der schnellste und einfachste Weg, um ein geschriebenes Programm zu testen. Leider gibt es bei Simulatoren jedoch auch Einschränkungen. Bei iOS sind diese aber recht gering und betreffen zumeist eher Beschränkungen wie das Fehlen der iPhone- oder iPad-GPU bei der 3D-Entwicklung. Ein weiterer großer Unterschied des Simulators gegenüber den Endgeräten ist, dass man nicht wie beim Endgerät mit einem ARM-Prozessor arbeitet, sondern mit der Intel-CPU des Entwicklungsrechners.



Sie sollten stets im Hinterkopf behalten, dass der Simulator gewisse Funktionen nicht bieten kann. So steht Ihnen z.B. keine Kamera zur Verfügung, und auch die GPS-Daten werden, sofern Sie diese nicht ändern, standardmäßig auf die Apple-Firmenzentrale gesetzt. Auch das Telefonieren und Verfassen von SMS sind mit dem Simulator nicht möglich.

Bevor man die App startet, muss man sich zunächst entscheiden, mit welchem Simulator man die App testen möchte. In Xcode 6.1 bietet sich hier eine breite Palette an Geräten an, unter anderem die verschiedenen iPads und die iPhones der Serie 4s bis 6 Plus. Die Auswahl des Simulators erfolgt in Xcode, indem man auf das aktuelle Build Target klickt, das sich oben links wie in Abbildung 5-10 befindet. Nachdem man auf es geklickt hat, bekommt man eine Auswahl an verschiedenen Testgeräten. Die Auswahl setzt sich aus den verschiedenen Möglichkeiten zusammen, die die Projekteinstellungen erlauben. Sollte z.B. eine alte Prozessorarchitektur in den Build Settings nicht vorhanden sein, so kann es passieren, dass der iPhone-4s-Simulator nicht mehr angezeigt wird.

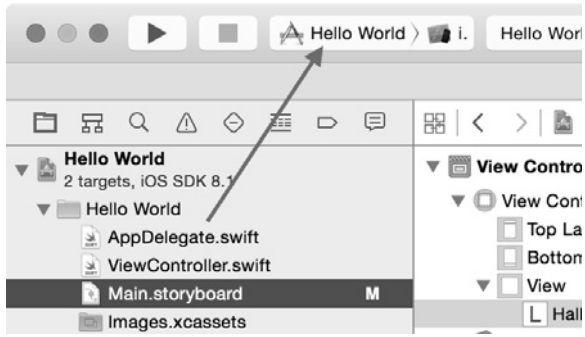
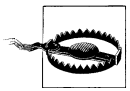


Abbildung 5-10: Öffnen der Geräteübersicht anhand des Build Targets

Bei der Auswahl des Simulators werden ebenfalls angeschlossene iPhones oder iPads angezeigt. Möchte man auf den Endgeräten testen, erfolgt die Auswahl in diesem Menü so, wie in Abbildung 5-11 dargestellt. Wir wählen für unseren ersten Start den iPhone-4s- oder iPhone-5-Simulator aus.

Abbildung 5-11: Übersicht der Simulatoren



Da wir am Anfang des Kapitels in unserem Storyboard die Size Classes und das Auto Layout deaktiviert haben, kann es sein, dass Elemente auf kleineren Displaygrößen nicht sichtbar sind, wenn diese nicht innerhalb der möglichen Dimensionen des Testgeräts liegen.

Nachdem die Auswahl des Simulators erfolgt ist, können Sie die App jetzt ausführen, indem Sie auf den Start-Knopf neben dem Target oben links klicken oder die Tastenkombination ⌘+R drücken. Die App wird dann kompiliert. Sollte der Build-Prozess aus irgendeinem Grund fehlschlagen, wird Xcode dies entsprechend anzeigen. Im Erfolgsfall wird der Simulator gestartet und nach wenigen Sekunden auch die App.

Gelegentlich schlägt das Starten des Simulators mit den unterschiedlichsten Fehlermeldungen fehl. Im Normalfall reichen das Beenden des Simulators und der erneute Start oder ein Clean der Build-Artefakte mit der Tastenkombination ⌘+⇧+K, um dieses Problem zu beheben. In selteneren Fällen kann es aber sein, dass bereits eine andere App im Simulator ausgeführt wird. Diese muss zunächst beendet werden, damit eine andere App gestartet werden kann.

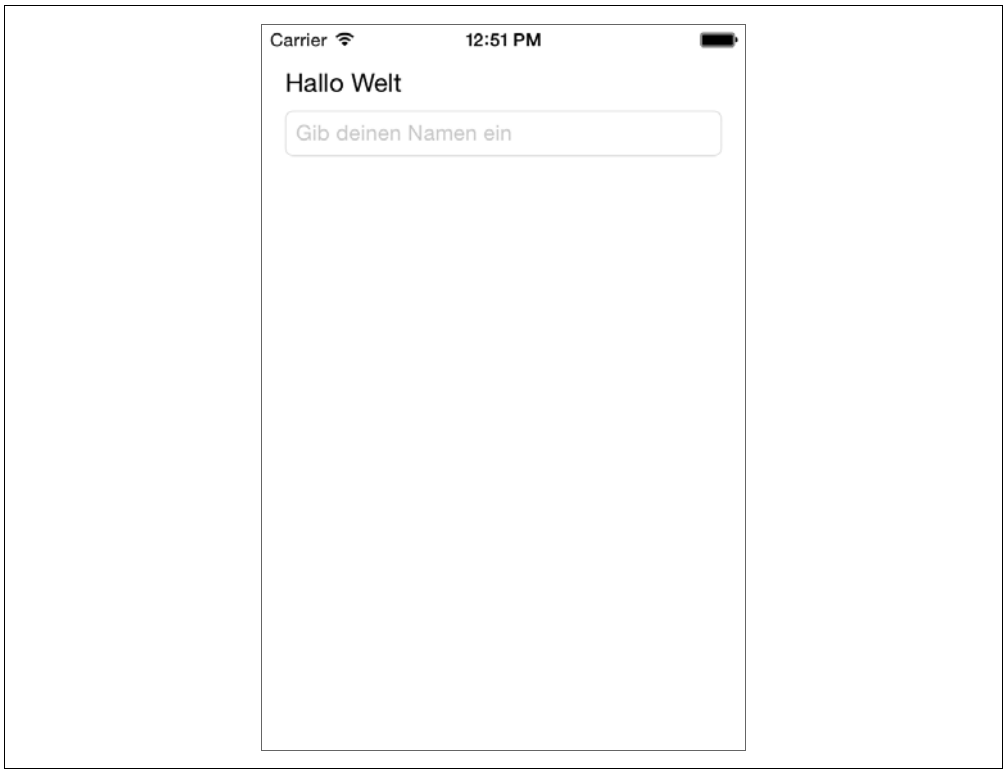


Abbildung 5-12: Der erste Start des Simulators

Sobald der Simulator (siehe Abbildung 5-12) unsere App anzeigt, können wir einige Möglichkeiten des Simulators ausprobieren. Generell können Sie die komplette App mit Ihrer Maus steuern. Alle Gesten, die Sie mit den Fingern machen würden, werden jetzt durch die Maus ersetzt. Ein Klick entspricht einem *Touch*, ein langer Klick ist einem *Long Touch* gleichgestellt. Mit gedrückter Alt-Taste können Sie Zoom-Gesten, wie z.B. in Map Views, simulieren. Der Simulator zeigt entsprechend zwei Kreise an, die die Touchpunkte von zwei Fingern darstellen sollen.

Reicht einmal der Platz auf Ihrem Monitor nicht aus, können Sie die Gesamtgröße des Simulators beeinflussen, indem Sie entweder im Simulator den Menüpunkt WINDOW aufrufen und das Untermenü ZOOM verwenden oder indem Sie die Tastenkombinationen ⌘+1–3 drücken. Entsprechend der Zahl 1–3 wird der Simulator dann zwischen 50% und 100% skaliert. Diese Größenkontrolle ist vor allem dann hilfreich, wenn man mit einem Retina-iPad-Simulator testen möchte und nur über ein MacBook Air verfügt.

Im Menüpunkt HARDWARE befinden sich noch einige Punkte, die Sie häufig verwenden werden, unter anderem die Rotation und Ausrichtung des Gerätes sowie die Möglichkeit, verschiedene Hardware- oder Softwarezustände zu simulieren. Dies kann z.B. das Sperren oder ein simuliertes Schütteln des Gerätes sein.

Der Menüpunkt DEBUG bietet zusätzlich einige verschiedene Rendering-Einstellungen an, die die gezielte Suche nach Rendering-Problemen vereinfachen. Zudem beinhaltet dieses Menü auch einen Menüpunkt, der die Synchronisation mit der iCloud auslöst. Um einen Screenshot vom aktuellen Simulatorinhalt zu erstellen, bietet sich die Tastenkombination ⌘+S oder der entsprechende Menüpunkt zum Erstellen von Screenshots im FILE-Menü an. Die Screenshots werden auf Ihrem Desktop gespeichert.

Um den Simulator zu beenden, reicht es nicht aus, nur den Stopp-Button in Xcode zu klicken. Der Simulator ist ein eigener Prozess und muss entsprechend über sein Menü oder über die Tastenkombination ⌘+Q beendet werden.

Den Simulator und alle darauf vorinstallierten Daten, inklusive Ihrer Applikationen, finden Sie unter *Macintosh HD* ▸ *Users* ▸ *<username>* ▸ *Library* ▸ *Developer* ▸ *CoreSimulator*.

Outlets und Actions hinzufügen

Damit unsere App ein wenig an Interaktivität gewinnt, können wir mithilfe eines Buttons und der Eingabe eines Textes durch den Benutzer ein wenig Action hineinbringen. Unser Ziel wird es sein, durch einen Button-Klick und die Eingabe des Benutzernamens eine Grußnachricht auszugeben. Nach dem Klick auf den Button soll das Label »Hallo Welt« in »Hallo <name>« geändert werden.

Um per Code auf die Eigenschaften eines Labels zugreifen zu können, müssen wir es mit unserer Klasse »verbinden«. Hierfür erzeugen wir ein *Outlet* für das entsprechende Label,

und zwar mithilfe des Interface Builders in unserer Klasse *ViewController*. Bei der Arbeit mit Xcode ist es üblich, dass man in einem Split-View-Modus das Storyboard und die betreffende Klasse gleichzeitig in einem Fenster geöffnet hat. Öffnen Sie als Erstes Ihr Storyboard und anschließend die Datei *ViewController*. Wir klicken diese Datei aber mit gedrückter Alt-Taste an. Beide Dateien sollten jetzt in einem Fenster sichtbar sein, so wie in Abbildung 5-13 dargestellt.

Abbildung 5-13: Storyboard und ViewController im Split-View-Modus

Das Outlet für das Label kann jetzt erzeugt werden, indem wir das Element zuerst mit einem einfachen Klick selektieren. Vergewissern Sie sich, dass es sich tatsächlich um das gewünschte Element handelt, indem Sie prüfen, ob die Markierung nach dem Klick auch tatsächlich auf dem gewünschten Element aktiv ist. Als Nächstes klicken Sie das Element mit der linken Maustaste und gedrückter Ctrl-Taste an. Wenn Sie jetzt Ihren Mauszeiger von dem Objekt entfernen, sollten Sie eine blaue Linie sehen, die Ihrem Mauszeiger folgt. Bewegen Sie Ihren Mauszeiger in den Editor, der die *ViewController*-Klasse anzeigt, und platzieren Sie diesen im Leerraum zwischen der Zeile `class ViewController: UIViewController` und der nächsten Codezeile, die mit `override func` beginnt. Hierbei ist es wichtig, dass Sie sich im Leerraum zwischen den Zeilen befinden. Xcode hilft Ihnen dabei, einen Leerraum zu finden, indem es eine blaue Linie in der Klasse anzeigt und damit verdeutlicht, dass man diese Verbindung an dieser Stelle platzieren kann. Wenn Sie den Leerraum gefunden haben, können Sie die Maustaste loslassen, und es öffnet sich eine Eingabemaske. Abbildung 5-14 veranschaulicht, wie die Linie sich darstellt und wo sich der gewünschte Zwischenraum befindet.

Sollten Sie versehentlich im Assistant Editor die falsche Klasse geöffnet haben und versuchen, ein Outlet zu verbinden, das für eine andere Klasse bestimmt ist, so wird Xcode das Erstellen nicht zulassen. Xcode wird schlichtweg keine blaue Hilfslinie in den Leerräumen anzeigen, wodurch gleichzeitig auch das »Ablegen« des Elements unterbunden wird. Sie können folglich Actions und Outlets nur für Elemente eines View Controllers aus dem Storyboard in der entsprechenden Klasse des View Controllers erstellen.

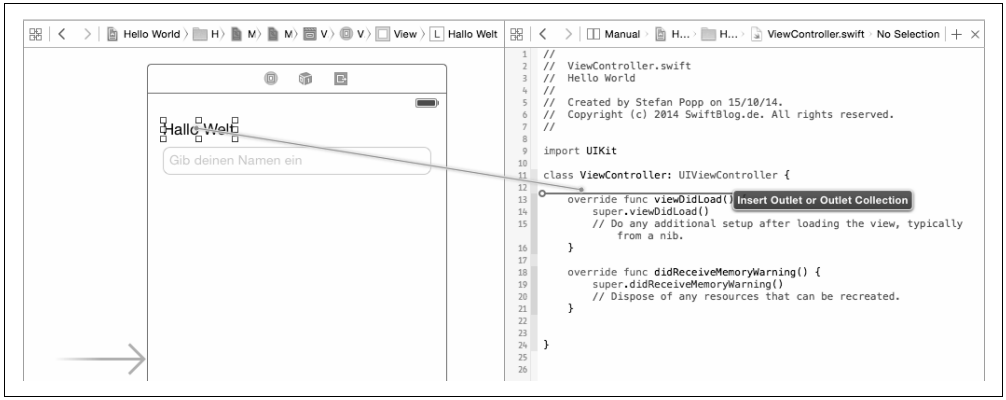


Abbildung 5-14: Erstellen des Outlets mithilfe des Interface Builders

In der jetzt angezeigten Eingabemaske stellen wir sicher, dass in dem Dropdown-Menü für *Connection* der Wert *Outlet* steht. In das *Name*-Feld tragen wir einen Variablennamen ein, mit dem wir innerhalb der *ViewController*-Klasse auf das Element zugreifen können. Wir entscheiden uns hierbei für den Variablennamen `helloLabel`, der (wie die späteren Variablen auch) der Namenskonvention *CamelCase* folgt. Beim Benennen von Klassen und anderen Datentypen verwendet man üblicherweise *Pascal Case*. Der *Type* sollte der Klasse des entsprechenden Elements entsprechen. Für ein Label ist dies unter iOS *UILabel*. Wir bestätigen unsere Eingaben mit einem Klick auf den *Connect*-Button.

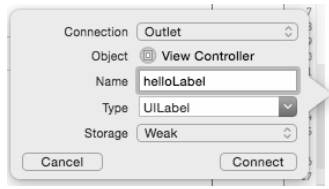


Abbildung 5-15: Ausgefüllte Parameter für das Label-Outlet

Nachdem der von uns bestätigte Dialog verschwunden ist, erzeugt Xcode auf Basis unserer Angabe eine neue Codezeile, die mit `@IBOutlet` beginnen sollte. Das gleiche Verfahren wiederholen wir auch noch für das Text Field, aus dem wir später den eingegebenen Namen auslesen werden. Wir ziehen wieder unsere Linie, dieses Mal von dem Text Field in die Klasse. Sie können gerne ein paar Leerzeilen nach dem Label-Outlet einfügen, um ein bisschen Platz zu schaffen. Die Hilfslinien von Xcode werden Ihnen aber auch dabei helfen, das Element zu platzieren, wenn einmal kein Freiraum zwischen Outlets und Methoden ist. Das Text Field soll den Namen `nameTextField` erhalten. Alle anderen voreingestellten Optionen müssen, wie beim Label zuvor, nicht von uns verändert werden.

Damit eine Aktion vom Benutzer ausgelöst werden kann, braucht unsere Oberfläche noch einen Button. Wir suchen in der Objektbibliothek den Button und ziehen ihn auf unsere

Oberfläche. Sie können wie bei einem Label den Text des Buttons ebenfalls mit einem Doppelklick oder über den ATTRIBUTE INSPECTOR verändern. Für den neu erstellten Button benötigen wir kein Outlet, da wir keine Eigenschaften von ihm verändern oder auslesen müssen. Allerdings möchten wir darauf reagieren, wenn ein Benutzer den Button drückt, und hierfür müssen wir eine sogenannte Action erstellen. Wir ziehen wieder unsere blaue Linie vom Button in unsere Klasse hinein, dieses Mal aber soweit unten wie möglich, am besten vor die letzte schließende geschweifte Klammer in der Klasse. Später kann man selbstverständlich natürlich Outlets und Actions platzieren, wo man möchte. Dieses Mal ändern wir in dem *Connection*-Dropdown-Menü den Wert von *Outlet* auf *Action*. Hierdurch wird eine Methode erzeugt, die beim Drücken des Buttons ausgelöst wird. Der Name für die Methode soll `pressMeTouched` lauten. In dem Dropdown-Menü bei *Type* selektieren wir statt *AnyObject* den *UIButton*. Dies ist normalerweise nicht nötig, in Kapitel 12 gehen wir aber nochmals darauf ein. Alle anderen Felder können unberührt bleiben, und wir klicken erneut auf den *Connect*-Button. Xcode hat für uns jetzt eine Methode mit genau den Eigenschaften erzeugt, die wir festgelegt haben.

Dieses Prinzip zum Erstellen von Actions und Outlets ist nahezu komplett identisch für jedes UI-Element aus der Objektbibliothek.

Ein bisschen Code muss sein

Als letzten Feinschliff muss unsere frisch erzeugte Action noch dafür sorgen, dass der Label-Text ersetzt wird, sobald der Button gedrückt wird. Wir müssen den hierfür erforderlichen Programmcode zwischen die geschweiften Klammern der `pressMeTouched()`-Methode platzieren. Wir versuchen uns noch einmal logisch das Ziel vorzustellen und überlegen, welche Reihenfolge an Aktionen wir benötigen, um dies zu erreichen. Der Benutzer gibt seinen Namen ein, er drückt den Button, und wir zeigen statt »Hallo Welt« in dem Label den Wert »Hallo <name>« an. In unserem Programmcode kann man dies fast eins zu eins widerspiegeln, indem man folgendermaßen formuliert:

1. Beschaffe und speichere den aktuellen Wert aus `nameTextField` in einer Konstante.
2. Erzeuge eine Zeichenkette »Hallo«, und füge den Wert des Text Fields am Ende an.
3. Füge den Text der Zeichenkette in dem `nameLabel` ein.

Wenn wir das Ganze in Form von Programmcode dann in unserer `pressMeTouched()`-Methode schreiben, ergeben sich auch drei Anweisungen. Die vollständige Methode sollte dann so wie Beispiel 5-1 aussehen.

Beispiel 5-1: Vollständige pressMeTouched()-Methode

```
@IBAction func pressMeTouched(sender: UIButton) {
    let nameString = nameTextField.text!
    let greetingString = String(format: "Hallo %@", nameString)
    helloLabel.text = greetingString
}
```

Nachdem wir die Methode vervollständigt haben, können wir den Simulator starten und in das Text Field klicken. Geben Sie einen Namen mithilfe der Tastatur oder mit dem Mauszeiger auf der Simulatortastatur ein, und drücken Sie den »Drück mich!«-Button. Das Label oberhalb des Text Fields sollte die neue Zeichenkette mit dem von Ihnen eingegebenen Text anzeigen. Sollten Sie nicht den vollständigen Text sehen, kann das daran liegen, dass Ihr Label nicht breit genug ist. Vergrößern Sie dieses einfach bei Bedarf im Storyboard.

iOS 8 bietet für Labels eine automatische Längenfunktion an, die sich am Textinhalt orientiert. ACHTUNG: Diese Funktion ist nicht abwärtskompatibel zu iOS 7.

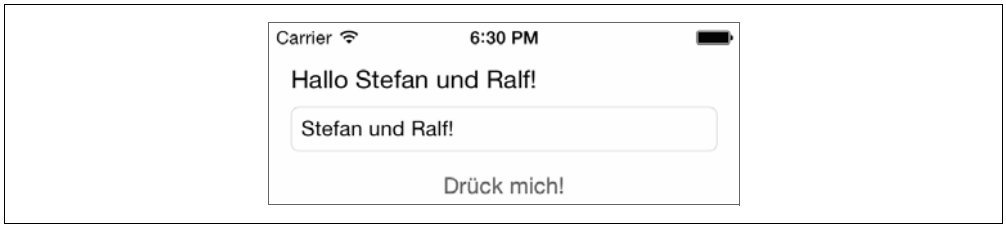


Abbildung 5-16: Fertige App im Test