

[Apps für iOS 9 professionell entwickeln](#)

Sauberen Code schreiben mit Objective-C und Swift. Stabile Apps programmieren. Techniken & Methoden von Grund auf verstehen

Bearbeitet von
Thomas Sillmann

2., aktualisierte und erweiterte Auflage 2015. Buch. 718 S. Hardcover

ISBN 978 3 446 44566 6

Format (B x L): 18,4 x 24,7 cm

Gewicht: 1407 g

[Weitere Fachgebiete > EDV, Informatik > Programmiersprachen: Methoden > PDA & Handheld Programmierung](#)

schnell und portofrei erhältlich bei


DIE FACHBUCHHANDLUNG

Die Online-Fachbuchhandlung beck-shop.de ist spezialisiert auf Fachbücher, insbesondere Recht, Steuern und Wirtschaft. Im Sortiment finden Sie alle Medien (Bücher, Zeitschriften, CDs, eBooks, etc.) aller Verlage. Ergänzt wird das Programm durch Services wie Neuerscheinungsdienst oder Zusammenstellungen von Büchern zu Sonderpreisen. Der Shop führt mehr als 8 Millionen Produkte.



Leseprobe

Thomas Sillmann

Apps für iOS 9 professionell entwickeln

Sauberen Code schreiben mit Objective-C und Swift. Stabile Apps programmieren. Techniken & Methoden von Grund auf verstehen

ISBN (Buch): 978-3-446-44566-6

ISBN (E-Book): 978-3-446-44553-6

Weitere Informationen oder Bestellungen unter

<http://www.hanser-fachbuch.de/978-3-446-44566-6>

sowie im Buchhandel.

Inhalt

Vorwort	XVII
1 Über iOS	1
1.1 Was ist iOS?	1
1.1.1 iOS und OS X	2
1.1.2 Besonderheiten der iOS-Plattform	2
1.2 iOS für Entwickler	3
1.2.1 Hardware für Entwickler	4
1.2.2 Software für Entwickler	5
1.2.3 Das Apple Developer Program	6
1.3 Der Aufbau von iOS	8
1.3.1 Die vier Schichten von iOS	8
1.4 Die perfekte iOS-App	10
1.4.1 Apple Human Interface Guidelines	11
2 Die Programmiersprache – Objective-C	13
2.1 Über Objective-C und objektorientierte Programmierung	13
2.2 Grundlagen der Programmierung	14
2.2.1 Objekte	14
2.2.2 Primitive Datentypen	15
2.2.3 Variablen	16
2.2.4 Operatoren	17
2.2.5 Abfragen und Schleifen	18
2.2.6 Kommentare	22
2.3 Aufbau einer Klasse	23
2.3.1 Die Header-Datei	23
2.3.2 Die Implementation-Datei	25
2.3.3 Los geht's: Unsere erste Klasse!	26
2.4 Methoden	30
2.4.1 Aufbau von Methoden	30
2.4.2 Methoden in Header- und Implementation-Dateien einer Klasse	32

2.4.3	Implementierung von Methoden	33
2.4.4	Methoden aufrufen	36
2.4.5	Klassen- und Instanzmethoden	37
2.5	Instanzvariablen	38
2.6	Properties	39
2.6.1	Aufbau einer Property	40
2.6.2	Die Punktnotation	42
2.6.3	Optionen	43
2.6.4	Direktzugriff auf Properties	44
2.6.5	Setter und Getter überschreiben	46
2.7	Konstanten	49
2.7.1	Deklaration von Konstanten	49
2.8	Namenskonventionen	50
2.8.1	Klassen	50
2.8.2	Methoden	51
2.8.3	Properties	51
2.9	Strukturen	51
2.9.1	enum	52
2.9.2	typedef	52
2.10	Initialisierung von Objekten	53
2.10.1	alloc und init	54
2.10.2	Zeiger	56
2.11	init im Detail	57
2.11.1	Erstellen mehrerer init-Methoden	59
2.11.2	Designated Initializer	60
2.12	Vererbung	62
2.12.1	Methoden der Superklasse überschreiben	64
2.13	Kategorien	66
2.13.1	Aufbau von Kategorien	66
2.13.2	Kategorien in Xcode erstellen	67
2.14	Erweiterungen	69
2.14.1	Aufbau von Erweiterungen	69
2.14.2	Erweiterungen innerhalb der Implementation-Datei	70
2.14.3	Erweiterungen in Xcode erstellen	71
2.15	Protokolle	72
2.15.1	Aufbau von Protokollen	73
2.15.2	Zuweisen eines Protokolls zu einer Klasse	74
2.15.3	Vererbung in Protokollen	74
2.15.4	Protokolle in Xcode erstellen	75
2.16	#import und @class	76
2.16.1	#import	76
2.16.2	@class	77

2.17	Blöcke	79
2.17.1	Ja zu Blöcken!	79
2.17.2	Was sind Blöcke?	79
2.17.3	Aufbau eines Blocks	80
2.17.4	Blockvariablen	84
2.17.5	Globale Blöcke	84
2.18	Singletons	85
3	Der Neue im Club – Swift	89
3.1	Programmierst du noch oder swiftst du schon?	89
3.1.1	Über Swift	89
3.1.2	Voraussetzungen zur Nutzung von Swift	89
3.1.3	Swift und Objective-C	90
3.1.4	Playgrounds	90
3.2	Grundlagen der Programmierung	92
3.2.1	Fundamental Types	92
3.2.2	Variablen und Konstanten	93
3.2.3	Operatoren	95
3.2.4	Abfragen und Schleifen	96
3.2.5	Kommentare	102
3.2.6	print	103
3.3	Fundamental Types im Detail	105
3.3.1	Strings	105
3.3.2	Arrays	108
3.3.3	Dictionaries	114
3.3.4	Any und AnyObject	117
3.4	Aufbau einer Klasse	117
3.4.1	Erstellen einer Instanz einer Klasse	120
3.4.2	Zugriff auf Eigenschaften einer Klasse	120
3.5	Methoden	121
3.5.1	Methoden mit Rückgabewert	122
3.5.2	Methoden mit Parametern	123
3.5.3	Local und External Parameter Names	126
3.5.4	Aufruf von Methoden einer Klasse	128
3.5.5	Zugriff auf andere Eigenschaften und Methoden einer Klasse	129
3.5.6	Klassen- und Instanzmethoden	130
3.6	Properties	132
3.6.1	Computed Properties	132
3.6.2	Property Observers	135
3.6.3	Type Properties	136
3.7	Vererbung	137
3.7.1	Überschreiben von Eigenschaften und Methoden der Superklasse	139
3.7.2	Zugriff auf Eigenschaften und Methoden der Superklasse	140

3.8	Optionals	141
3.8.1	Deklaration von Optionals	141
3.8.2	Zugriff auf Optionals	142
3.9	Initialisierung	145
3.9.1	Schreiben von Initializern	146
3.9.2	Designated und Convenience Initializer	150
3.9.3	Initializer und Vererbung	151
3.9.4	Deinitialisierung	153
3.10	Type Casting	154
3.10.1	Typ prüfen	154
3.10.2	Downcasting	156
3.11	Enumerations	158
3.11.1	Zusätzliche Werte in Member einer Enumeration speichern	159
3.12	Structures	161
3.13	Error Handling Model	162
3.14	Extensions	164
3.15	Protocols	165
3.15.1	Protocol Type	166
3.16	Access Control	168
4	Programmierung für iOS	171
4.1	Grundlagen	171
4.2	Foundation Framework	171
4.2.1	Die wichtigsten Klassen aus dem Foundation Framework und ihre Funktionen	172
4.3	UIKit Framework	176
4.4	Speicherverwaltung mit ARC	177
4.5	Besonderheiten von Objective-C	180
4.5.1	Kurzschreibweisen zum Erstellen von Objekten	181
4.5.2	Vergleichen der Werte von verschiedenen Objekten	183
4.5.3	Schlüsselwörter zum Zusammenspiel mit Optionals	184
4.6	Besonderheiten von Swift	185
4.6.1	Zusammenspiel zwischen Fundamental Types und Foundation-Framework-Klassen	185
4.6.2	Playgrounds im Detail	186
4.7	Objective-C und Swift vereint	190
4.7.1	Objective-C-Code in Swift verwenden	190
4.7.2	Swift-Code in Objective-C verwenden	191
4.8	NSError	192
4.8.1	Eigene Methode mit NSError-Parameter erstellen	193
4.9	Doxygen-Dokumentation	194
4.9.1	Besonderheiten bei Methoden	196
4.9.2	Doxygen-Dokumentation in Xcode	197

4.10	Nebenläufigkeit mit Grand Central Dispatch	198
4.10.1	Parallel laufenden Code erstellen	199
4.11	Grundlegende Struktur einer App	201
4.11.1	main.m	201
4.11.2	Info.plist	201
4.11.3	App Delegate	202
4.12	Lebenszyklus einer iOS-App	203
4.12.1	Start einer App	203
4.12.2	Lebenszyklus einer App	204
4.12.3	Die Methoden des App Delegate	205
4.12.4	Start der App	206
4.13	Tipps für die tägliche Arbeit	208
4.13.1	Die netten Kleinigkeiten	208
4.13.2	Fast Enumeration in Objective-C	209
4.13.3	Type Casting in Objective-C	209
4.13.4	Xcode-Beispielprojekte	210
5	Die Entwicklungsumgebung – Xcode	211
5.1	Willkommen bei Xcode!	211
5.1.1	Was ist Xcode?	212
5.1.2	Interface Builder und Xcode – endlich vereint!	212
5.2	Arbeiten mit Xcode	213
5.2.1	Dateien und Formate eines Xcode-Projekts	213
5.2.2	Umgang mit Dateien und Ordnern in Xcode	218
5.3	Der Aufbau von Xcode	221
5.3.1	Die Toolbar	221
5.3.2	Die Navigation Area	223
5.3.3	Die Editor Area	227
5.3.4	Die Utilities Area	229
5.3.5	Die Debug Area	230
5.4	Einstellungen in Xcode	231
5.4.1	Anpassen von Xcode	231
5.4.2	General	231
5.4.3	Accounts	232
5.4.4	Behaviors	233
5.4.5	Navigation	233
5.4.6	Fonts & Colors	234
5.4.7	Text Editing	235
5.4.8	Key Bindings	235
5.4.9	Source Control	236
5.4.10	Downloads	237
5.4.11	Locations	237

5.5	Projekteinstellungen	238
5.5.1	Grundlagen	238
5.5.2	Einstellungen am Projekt	240
5.5.3	Einstellungen am Target	243
5.5.4	Einstellungen am Scheme	249
5.6	Grafiken und Asset-Bundles	252
5.7	Lokalisierung mit Localizable.strings	254
5.7.1	Grundlagen	254
5.7.2	NSLocalizedString	254
5.7.3	Erstellen der Localizable.strings-Datei	255
5.7.4	Localized String mit Parameter	257
5.7.5	Alle Localized Strings automatisch auslesen	258
5.8	Der iOS-Simulator	259
5.8.1	Grundlagen	259
5.8.2	Funktionen und Möglichkeiten des Simulators	259
5.8.3	Performance und Einschränkungen des Simulators	263
5.9	Dokumentation	263
5.9.1	Nichts geht über die Dokumentation!	263
5.9.2	Das Documentation-Window	265
5.9.3	Direktes Aufrufen der Dokumentation aus Xcode heraus	268
5.10	Devices	269
5.11	Organizer	271
5.12	Projects	274
5.13	Debugging in Xcode	275
5.13.1	Grundlagen – über das Debugging	275
5.13.2	Die Debug Area	275
5.13.3	Die Arbeit mit dem Debugger – NSLog und Breakpoints	277
5.13.4	Debug Navigator	285
5.14	Refactoring	286
5.14.1	Grundlagen	286
5.14.2	Refactoring-Funktionen in Xcode	287
5.15	Instruments	289
5.15.1	Über Instruments	289
5.15.2	Nächste Schritte	292
5.16	Tipps für die tägliche Arbeit	293
5.16.1	Man lernt immer was dazu!	293
5.16.2	Code Snippets	293
5.16.3	Open Quickly	294
5.16.4	Caller einer Methode feststellen	295
5.16.5	Speicherorte für Ordner und Dateien ändern	296
5.16.6	Shortcuts für die Navigation Area	296
5.16.7	Run Without Building	297
5.16.8	Clean Build	297

6	MVC – Model-View-Controller	299
6.1	MVC... was?	299
6.2	MVC in der Praxis	301
6.3	Kommunikation zwischen Model und Controller	301
6.3.1	Key-Value-Observing	302
6.3.2	Notifications	308
6.4	Kommunikation zwischen View und Controller	310
6.4.1	Target-Action	310
6.4.2	Delegates	312
7	Die Vielfalt der (View-)Controller	315
7.1	Alles beginnt mit einem View-Controller	315
7.2	UIViewController – die Mutter aller View-Controller	317
7.2.1	Wichtige Methoden von UIViewController	318
7.2.2	UIView – fester Bestandteil eines jeden UIViewControllers	322
7.3	View-Controller-Hierarchien	323
7.4	View-Controller erstellen mit dem Interface Builder	325
7.4.1	View-Controller mit Nib-File	326
7.5	Storyboards	357
7.5.1	Über Storyboards	357
7.5.2	Das Storyboard-Projekt	358
7.5.3	Die Klasse UIStoryboard	367
7.5.4	Segues	369
7.5.5	Zugriff über den App Delegate	372
7.5.6	Quo vadis – Storyboard oder Nib-File?	373
7.6	Auto Layout	374
7.6.1	Setzen und Konfigurieren von Constraints	375
7.6.2	Constraints bearbeiten und weiter anpassen	376
7.6.3	„Optimale“ Constraints automatisch setzen lassen	378
7.7	UIViewController und seine Subklassen	379
7.7.1	UINavigationController	379
7.7.2	UITabBarController	385
7.7.3	UITableViewController	389
7.7.4	UICollectionViewController	396
7.7.5	UISplitViewController	398
8	Views erstellen und gestalten	401
8.1	Über Views in iOS	401
8.2	UIView – die Mutter aller Views	401
8.3	Arbeiten mit UIView	402
8.3.1	Programmatisches Erstellen einer UIView	402
8.3.2	View-Hierarchien	404
8.3.3	Weiterführendes zu UIView	408

8.4	Views erstellen mit dem Interface Builder	409
8.4.1	Grundlagen	409
8.4.2	View-Klasse mit Nib-File erstellen	410
8.4.3	Beliebiges Nib-File laden und verwenden	414
8.4.4	Nib-File nachträglich erstellen	415
8.4.5	Unterschiedliche Nib-Files für iPhone und iPad erstellen	417
8.5	Die wichtigsten Views und ihre Funktionen	419
8.5.1	Grundlagen	419
8.5.2	UILabel	419
8.5.3	UIButton	419
8.5.4	UISwitch	420
8.5.5	UISegmentedControl	420
8.5.6	UITextField	420
8.5.7	UIImageView	421
8.5.8	UIAlertView	421
8.5.9	UIActionSheet	423
8.5.10	UIPickerView	423
8.5.11	UIDatePicker	424
8.5.12	UIWebView	424
8.5.13	UIMapView	425
8.5.14	UIScrollView	425
8.5.15	UITextView	426
8.5.16	UITableView	427
8.5.17	UICollectionView	427
8.5.18	Wichtig und unerlässlich: die Dokumentation!	427
8.5.19	Views und der Interface Builder	428
8.6	Die Grundlage gut gestalteter Views	428
9	Das Model und die Datenhaltung	431
9.1	Die Logik Ihrer App	431
9.2	Benutzereinstellungen sichern und nutzen	432
9.2.1	ÜberNSUserDefaults	432
9.2.2	Standardeinstellungen festlegen	434
9.2.3	NSUserDefaults zurücksetzen	435
9.3	Zugriff auf das Dateisystem	435
9.3.1	Das Dateisystem von iOS	435
9.3.2	NSFileManager	436
9.3.3	File-Sharing-Funktion nutzen	443
9.4	Core Data	444
9.4.1	Datenbankverwaltung mit Core Data	444
9.4.2	Wie funktioniert Core Data?	445
9.4.3	Die Klassen und Bestandteile von Core Data	446
9.4.4	Aufbau eines Standard-Core Data Stacks	447
9.4.5	Der Core Data-Editor	450

9.4.6	Erstellen eines neuen Managed-Objects	458
9.4.7	Löschen eines Managed-Objects	459
9.4.8	Laden von Managed-Objects	459
9.4.9	Was kommt als Nächstes?	460
10	Local und Push Notifications	461
10.1	Was sind Notifications?	461
10.2	Registrieren von Notification Types	463
10.3	Registrieren von Notification Categories und Actions	465
10.3.1	Erstellen einer Action	466
10.3.2	Erstellen einer Kategorie	467
10.3.3	Reagieren auf eine Action	469
10.4	Local Notifications	471
10.4.1	Festlegen des Ausführungstermins	472
10.4.2	Konfiguration des Alerts	473
10.4.3	Konfiguration des Sounds	475
10.4.4	Konfiguration des Badge Values	476
10.4.5	Speichern zusätzlicher Informationen in einer Local Notification	476
10.4.6	Registrieren von Local Notifications im System	477
10.4.7	Abbrechen bereits registrierter Local Notifications	478
10.4.8	Reagieren auf Feuern und Auswahl einer Local Notification	478
10.5	Push Notifications	479
10.5.1	Versand von Push Notifications	480
10.5.2	Erstellen einer Push Notification	484
10.5.3	Quality of Service	486
11	Extensions	487
11.1	Verfügbare Typen von Extensions	487
11.2	Erstellen von Extensions in Xcode	489
11.3	Aufbau einer Extension	492
11.4	Funktionsweise einer Extension	492
11.5	Wichtige Klassen und Objekte	493
11.6	Unterstützte Dateitypen für Share- und Action-Extensions festlegen	494
11.7	Today	496
11.7.1	Today-Extension testen	498
11.8	Share	499
11.9	Action	500
11.9.1	Action mit User Interface	500
11.9.2	Action ohne User Interface	501
11.10	Photo Editing	502
11.10.1	Festlegen der unterstützten Typen zur Bearbeitung	506
11.11	Document Provider	507
11.11.1	Document Provider-Extension	508

11.11.2	File Provider-Extension	511
11.11.3	Document Provider aufrufen	513
11.12	Custom Keyboard	515
11.12.1	Erstellen eines Custom Keyboards	515
11.12.2	Arbeit mit der Klasse UINavigationController	515
11.12.3	Bearbeiten und Setzen von Text	517
11.12.4	Mehrsprachige Keyboards	518
11.13	Content Blocker	519
11.13.1	Konfiguration eines Content Blockers	520
11.13.2	Aktualisieren eines Content Blockers	523
11.13.3	Die Klasse ActionRequestHandler	523
11.14	Shared Links	524
11.14.1	Erstellen eines NSExtensionItem	526
11.14.2	NSExtensionItem als Shared Link bereitstellen	527
11.15	Watch App	528
11.16	App Groups	528
11.16.1	Registrieren einer App Group im Apple Developer Portal	528
11.16.2	Registrieren einer App Group innerhalb einer App	530
11.16.3	Zugriff auf eine App Group	530
12	App-Entwicklung für die Apple Watch	533
12.1	Apples neues großes Ding: Die Apple Watch	533
12.2	Möglichkeiten, Einschränkungen, Unterschiede	534
12.3	Das WatchKit SDK	536
12.3.1	WKInterfaceController	536
12.3.2	WKInterfaceObject	537
12.3.3	WKExtensionDelegate	538
12.3.4	Weitere Klassen	538
12.4	Aufbau und Funktionsweise von Apple Watch-Apps	538
12.4.1	iPhone-App	539
12.4.2	WatchKit Extension	539
12.4.3	WatchKit App	539
12.4.4	Verbindung von WatchKit Extension und WatchKit App	540
12.4.5	Notification Scene und Glance Scene	540
12.4.6	Complications	541
12.5	Erstellen einer WatchKit App mitsamt WatchKit Extension	541
12.5.1	Dateien der WatchKit Extension	544
12.5.2	Dateien der WatchKit App	544
12.5.3	Ausführen und Testen der Apple Watch-App	545
12.6	Lebenszyklus einer WatchKit App	546
12.7	Der WKInterfaceController im Detail	548
12.7.1	Initialisierung	548

12.7.2	Activation Events	549
12.7.3	Setzen des Titels	550
12.7.4	Ein- und Ausblenden von Interface-Controllern	550
12.7.5	Umsetzen eines Navigation Stacks	552
12.7.6	Reaktion auf Storyboard-Events	553
12.7.7	Weitere Attribute	554
12.7.8	Weitere Funktionen von WKInterfaceController	555
12.8	Arbeiten mit dem Interface-Storyboard einer WatchKit App	555
12.8.1	Erstellen und Konfigurieren eines WKInterfaceController	556
12.8.2	Hinzufügen und Konfigurieren von Interface-Elementen	557
12.8.3	Positionierung und Anordnung von Interface-Elementen	558
12.8.4	Ändern der Größe von Interface-Elementen	559
12.8.5	Unterschiedliche Konfigurationen für verschiedene Apple Watch-Größen	561
12.8.6	Gruppierung von Interface-Elementen mittels WKInterfaceGroup	563
12.8.7	Verbindung von Storyboard und Code	564
12.8.8	Zusammenfassen mehrerer Interface-Controller zu einem page-based Interface	567
12.8.9	Erstellen und Konfigurieren von Segues	567
12.9	Erstellen von Tabellen	570
12.9.1	Hinzufügen einer Tabelle im Storyboard	570
12.9.2	Konfiguration einer Zelle	571
12.9.3	Konfiguration einer Tabelle	575
12.9.4	Verwenden verschiedener Zellen in einer Tabelle	577
12.9.5	Zellen hinzufügen und entfernen	579
12.9.6	Direkt zu Zelle scrollen	581
12.9.7	Aktuelle Anzahl an Zellen auslesen	581
12.9.8	Auf Auswahl einer Zelle reagieren	581
12.9.9	Segues von Zellen einer Tabelle über das Storyboard konfigurieren	582
12.10	Erstellen von Menüs	584
12.10.1	Erstellen eines Menüs im Storyboard	585
12.10.2	Erstellen eines Menüs im Code	588
12.10.3	Fazit: Menüerstellung im Storyboard oder Code?	591
12.10.4	Mischen von Menüpunkten aus Storyboard und Code	591
12.11	Eingabe von Text	591
12.12	Glance Scene	593
12.12.1	Erstellen einer Glance Scene	595
12.12.2	Lebenszyklus einer Glance Scene	596
12.12.3	Angepasster App-Start mittels Glance Scene	596
12.12.4	Testen einer Glance Scene	598
12.13	Notification Scene	599
12.13.1	Short-Look und Long-Look Interface	599

12.13.2	Long-Look Interface im Detail	600
12.13.3	Erstellen eigener Notification Scenes	601
12.13.4	Reaktion auf Action-Buttons	609
12.13.5	Testen einer Notification Scene	611
12.14	Complications	611
12.14.1	Was sind Complications?	612
12.14.2	Das ClockKit Framework	612
12.14.3	Aufbau und Bestandteile von Complications	612
12.14.4	Vorbereiten des eigenen Projekts	616
12.14.5	Entwicklung einer Complication	617
12.14.6	Bereitstellen der Complication mittels CLKComplicationDataSource	621
12.15	Kommunikation und Datenaustausch zwischen iOS und watchOS	625
12.15.1	WatchConnectivity	626
12.16	Was sonst noch zu sagen und zu beachten ist	631
13	Unit-Tests	633
13.1	Unit-Tests in der iOS-Entwicklung	633
13.1.1	Grundlagen	633
13.1.2	Aufbau und Funktionsweise von Tests	637
13.1.3	Aufbau einer Test-Case-Klasse	639
13.1.4	Neue Test-Case-Klasse erstellen	640
13.1.5	Ausführen von Tests	641
13.1.6	Was sollte ich eigentlich testen?	642
13.2	Performance-Tests	643
13.3	UI-Testing	645
13.3.1	Klassen für UI-Testing	646
13.3.2	Aufbau von UI-Testing-Klassen	648
13.3.3	Erstellen von UI-Tests	649
13.3.4	Einsatz von UI-Tests	649
13.4	Test-Driven Development	650
14	Versionierung mit Git	651
14.1	Über Versionskontrolle	651
14.2	Basisfunktionen und -begriffe von Git	651
14.2.1	Begriffe	651
14.2.2	Funktionen	652
14.3	Source Control in Xcode	653
14.4	Version Editor und Source Control	657
15	Veröffentlichung im App Store	661
15.1	Zertifikate, Provisioning Profiles und Ihre App	661
15.1.1	Certificates, Identifiers & Profiles	663
15.1.2	Erstellen von ...	665

15.2	Testen auf dem eigenen Endgerät	678
15.2.1	Setzen des Teams	678
15.2.2	Auswahl Ihres iOS-Geräts	679
15.3	iTunes Connect und Veröffentlichung im App Store	681
15.3.1	Vorbereiten der App in iTunes Connect	682
15.3.2	Upload der App in den App Store	685
15.3.3	Wie geht es weiter?	687
Index	689

Vorwort

iOS ist und bleibt für Entwickler ein spannendes Feld, nicht zuletzt, da Apple mit dem App Store einen Weg geschaffen hat, mit dem auch einzelne Entwickler Software für einen internationalen Markt verbreiten können, ohne dass sie sich um Dinge wie Bezahlssysteme, Abrechnungen und Download Gedanken machen müssen. Ich selbst habe mich aufgrund dessen vor über fünf Jahren für die iOS-Entwicklung begeistern lassen und habe bis heute nichts von der Faszination für diese spannende und innovative Plattform verloren.

Beim Einstieg in die iOS-Entwicklung habe ich eines aber schmerzlich vermisst: einen einfachen, übersichtlichen und professionellen Einstieg. Ich habe mich mit viel Literatur auseinandergesetzt, war in vielen Foren unterwegs und habe schlicht und einfach viel ausprobiert. Da vieles von diesen Anfängen aber sehr umständlich oder im Nachhinein betrachtet sogar gänzlich falsch war, hat mich das viel Zeit und Lehrgeld gekostet. Und ich habe mir oft gewünscht, man hätte mich von Beginn an an die Hand genommen und mir nicht nur gezeigt, *wie* ich Apps für iOS entwickle, sondern *wie ich gute und professionelle* Apps entwickle, welche Besonderheiten, Best Practices und Design Patterns es gibt und wie ich effektiv und effizient mit der Entwicklungsumgebung arbeiten kann. Und dieser Wunsch hat den Grundstein für dieses Buch gelegt.

Dieses Buch vermittelt Ihnen alle essenziellen Grundlagen und Kenntnisse über die Entwicklung für iOS. Angefangen bei der Vorstellung des Betriebssystems selbst geht es weiter über die Programmiersprachen Objective-C und Swift, deren Struktur und jeweiligen Besonderheiten und all das, was sie ausmacht. Danach rückt die eigentliche Entwicklung für iOS in den Fokus. Sie erfahren alles über die wichtigsten Frameworks von Apple für die App-Entwicklung und lernen typische Best Practices kennen. Besonders wichtig ist hier auch die Vorstellung der Dokumentation, die für Sie als App-Entwickler Bestandteil Ihrer täglichen Arbeit sein wird. Denn letztlich beherbergt die Apple-Dokumentation alle Antworten auf die Fragen, wie Sie bestimmte Probleme in der iOS-Entwicklung angehen und welche Möglichkeiten Ihnen die einzelnen Frameworks von Apple liefern.

Nach der Vorstellung der Plattform und der Programmiersprache(n) geht es weiter mit der Entwicklungsumgebung Xcode, mit der wir unsere Apps für iOS entwickeln. Dabei war es mir besonders wichtig, den genauen Aufbau und die Struktur hinter Xcode vorzustellen sowie alle spannenden Möglichkeiten und Kniffe aufzuzeigen, die Xcode Ihnen bietet und Ihre tägliche Arbeit erleichtern. Auch der Umgang mit Grafiken oder die Lokalisierung Ihrer Anwendung sowie Debugging und Refactoring habe ich in dieses Kapitel mit aufgenommen.

Bis zu diesem Punkt habe ich mich rein mit den essenziellen Grundlagen beschäftigt und ich finde es wichtig, dass auch Sie diese Grundlagen verinnerlicht und verstanden haben, denn sie sind die Grundpfeiler für gute und erfolgreiche Apps. Dazu abschließend folgt im sechsten Kapitel die Vorstellung von MVC – Model-View-Controller. Dabei handelt es sich um eines der wichtigsten Design-Patterns in iOS und ist essenziell für die App-Entwicklung. Aufgrund dessen widme ich MVC ein eigenes Kapitel, stelle es im Detail vor und erkläre, wie und warum Sie es in Ihren eigenen Apps anwenden sollen.

Anschließend geht es im Speziellen um die Entwicklung für iOS und die Nutzung der Funktionen und Frameworks für Apple, unterteilt auf die drei Bereiche Controller, View und Model aus dem MVC-Pattern. Auch in diesen Kapiteln geht es darum, die grundlegenden Besonderheiten zu erläutern und aufzuzeigen, wie Sie mit den einzelnen Elementen arbeiten und diese in Ihren eigenen Apps verwenden. Sie lernen die wichtigsten Elemente aus den jeweiligen Bereichen kennen und erfahren, wie Sie selbstständig mit ihnen arbeiten können und worauf bei der jeweiligen Verwendung zu achten ist. Mit diesem Wissen gewappnet sind Sie imstande, sich selbst in neue Frameworks, Technologien und Themen anhand der Apple-Dokumentation einzuarbeiten und selbstständig Probleme zu lösen. Und genau das ist es, was einen guten und professionellen iOS-Entwickler ausmacht.

Kapitel 10 setzt sich im Detail mit den sogenannten Local und Push Notifications auseinander, die es Ihnen erlauben, Nachrichten aus Ihren Apps an Ihre Nutzer zu senden. Im darauffolgenden Kapitel erstelle ich Ihnen ergänzend dazu dann Extensions vor, die uns Entwicklern ganz neue und innovative Möglichkeiten an die Hand geben, unsere Apps zu erweitern und über das gesamte iOS-System heraus zugänglich zu machen.

Im Anschluss daran widme ich ein eigenes und umfangreiches Kapitel der Entwicklung für die Apple Watch. Die Apple Watch ist die neue spannende Plattform in Apples Ökosystem und ist – was die grundsätzliche App-Entwicklung betrifft – in vielen Dingen recht ähnlich zur iOS-Plattform. Da eine Apple Watch-App immer eine iPhone-App voraussetzt, war es nur passend, auch die Entwicklung für die Apple Watch in diesem Buch im Detail zu beleuchten und zu zeigen, wie Sie Ihre eigenen iPhone-Apps um ein Pendant für die Apple Watch erweitern können. Sie lernen alle Möglichkeiten und Einschränkungen kennen und werden so in die Lage versetzt, selbst mit der Entwicklung eigener Apple Watch-Apps zu beginnen.

Anschließend folgen die Themen Unit-Tests und Versionsverwaltung mit Git, die in jeder neuen Xcode-Version mehr und mehr in die Entwicklungsumgebung integriert und unterstützt werden. Ich zeige Ihnen dabei, was Unit-Tests sind, warum Sie sie in Ihren Apps verwenden sollten, wie Sie Unit-Tests schreiben und wie Sie Xcode in Sachen Unit-Tests unterstützt und Ihnen unter die Arme greift. Auch die zusammen mit iOS 9 neu eingeführten UI-Tests finden dabei Beachtung. Bei der Versionsverwaltung mit Git erfahren Sie alles über die integrierten Funktionen zur Arbeit mit Git in Xcode und wie Sie Änderungen im Code verfolgen und nachvollziehen können.

Zu guter Letzt geht es noch – wie könnte es anders sein? – um die Veröffentlichung Ihrer Apps im App Store und die integrierten Tools in Xcode, die Ihnen bei diesem Prozess unter die Arme greifen. Insbesondere erfahren Sie hier etwas über die Erstellung und Verwaltung Ihrer Apps in Apples iOS Developer Program.

Bei allen diesen Themen soll dieses Buch Sie unterstützen und Ihnen die Grundlagen und das essenzielle Praxiswissen vermitteln und mit auf den Weg geben. Es soll Ihnen nicht

Beispielprojekte aufzeigen und deren Verhalten und Eigenschaften erklären (davon gibt es nämlich von Apple selbst mehr als genug), sondern Ihnen das nötige Wissen mitgeben, um Sie in die Lage zu versetzen, Problemstellungen selbstständig zu lösen und zu verstehen, wie Sie gute und professionelle iOS-Apps entwickeln. Denn wenn Sie diesen Status erreicht haben, können Sie darauf aufbauen, experimentieren und eigene spannende und innovative iOS-Projekte umsetzen. Und ich bin gespannt, welche großartigen Apps wir von Ihnen erwarten dürfen.

Ich wünsche Ihnen viel Freude beim Lesen dieses Buches und viel Erfolg mit all Ihren iOS-Projekten.

Thomas Sillmann



thomassillmann.de/ios-buch

Unter dieser Adresse finden Sie einige Basis-Klassen, die aus den Code-Beispielen aus diesem Buch aufgebaut sind. Diese Basis-Klassen dienen einerseits dazu, Ihnen einzelne Code-Schnipsel für bestimmte Aufgaben zur Verfügung zu stellen, und sind andererseits aber auch dazu geeignet, in Ihre eigenen Projekte übernommen und dort verwendet zu werden. Sie enthalten die Logik für verschiedene grundlegende Funktionen und Aktionen in der iOS-Entwicklung und können Ihnen daher möglicherweise des Öfteren eine nützliche Unterstützung sein. Schauen Sie einfach mal vorbei!

1

Über iOS

■ 1.1 Was ist iOS?

Auch wenn diese Frage in der heutigen Zeit möglicherweise überflüssig erscheint (und auch in Anbetracht dessen, dass Sie sich dieses Buch gekauft haben), möchte ich zu Beginn doch zumindest kurz darauf eingehen, was eigentlich dieses iOS ist, für das ich mich – und Sie sich offensichtlich auch – als Entwickler so sehr interessiere. Dabei werde ich auch direkt den Spagat schlagen und Ihnen die Geräte vorstellen, auf denen iOS verfügbar ist, und beschreiben, wie sich das System im Laufe der Jahre entwickelt hat.

Zunächst einmal ist iOS ein Betriebssystem der Firma Apple. Seinen ersten großen Auftritt hatte es im Jahr 2007 zusammen mit der Vorstellung des allerersten iPhone, denn genau auf diesem System lief iOS (auch wenn es damals noch iPhone OS hieß). Mit dem iPhone kramelte sich der Markt der Smartphones maßgeblich um und heutzutage sieht man Touch-Smartphones mit dem Bildschirm als Hauptbedienelement allerorten.

Nach mehreren Hardware-Sprüngen des iPhone folgte im Jahr 2010 das nächste iOS-Device von Apple: Das iPad, welches – ebenso wie das iPhone zuvor den Smartphone-Markt – nun den Tablet-Markt ordentlich aufmischte und bis heute den Quasistandard im Bereich Tablets setzt. Auch auf dem iPad läuft Apples Mobil-Betriebssystem iOS (dessen Namensänderung ebenfalls im Jahr 2010 mit dem Erscheinen des iPad von iPhone OS zu iOS erfolgte).

Darüber hinaus läuft iOS auch auf dem iPod touch. Alle Apps, die Sie für das iPhone entwickeln, sind prinzipiell ebenfalls auf dem iPod touch lauffähig, lediglich die zugrunde liegende Hardware unterscheidet sich ein wenig; Telefonieren ist beispielsweise mit dem iPod touch nicht möglich. So kann sich aber ein iPod touch durchaus als günstiges Testgerät für iOS-Apps anbieten (das iPhone spielt da nun mal doch in einer etwas anderen Preisklasse).

Und direkt zu Beginn noch eine kleine Randnotiz: Auch auf dem Apple TV läuft eine angepasste Version von iOS, allerdings gibt es bisher für Entwickler keine Chance, eigene Anwendungen für dieses Gerät zu entwickeln und zu veröffentlichen. Aber vielleicht dürfen Sie und ich langfristig unsere Kenntnisse über die iOS-Entwicklung auch dafür nutzen, eigene Anwendungen für das Apple TV zu kreieren; warten wir ab, was die Zukunft noch so bringt (siehe Bild 1.1)!



Bild 1.1 iPhone und iPad sind die erfolgreichsten Geräte mit dem Betriebssystem iOS. Daneben verfügt auch Apples iPod touch über iOS als Betriebssystem. (Bild: developer.apple.com)

1.1.1 iOS und OS X

Zusammengefasst lässt sich also einfach sagen: iOS ist das Betriebssystem von Apples iPhone-, iPad- und iPod touch-Familie. Sicherlich wissen Sie aber auch, dass Apple nicht nur iOS-Geräte entwickelt und veröffentlicht (auch wenn das wohl aktuell das Geschäft ist, das Apple den größten Umsatz einbringt). Daneben gibt es noch die Mac-Familie, die Apples Produktplatte aus Notebooks und Desktop-PCs darstellt. Und besonders spannend ist hierbei, dass iOS auf OS X – dem Betriebssystem der Macs – basiert. So sind viele Frameworks, mit denen wir in der iOS-Entwicklung arbeiten werden, unter OS X in derselben oder in einer leicht abgewandelten Form verfügbar. Das bedeutet umgekehrt auch, dass der Einstieg in die OS X-Entwicklung leichter fällt, wenn Sie bereits für iOS entwickelt haben – und umgekehrt. Das aber nur als kleine Randnotiz und mögliche Motivation, sich nach der Lektüre dieses Buches eventuell auch mit der OS X-Entwicklung näher auseinanderzusetzen; Sie werden sehen, über das nötige Rüstzeug verfügen Sie dann. ☺

1.1.2 Besonderheiten der iOS-Plattform

Auch wenn iOS auf OS X basiert, so gibt es doch mannigfaltige Unterschiede zwischen den beiden Betriebssystemen (auch wenn sie sich unter der Haube relativ ähnlich sind).

Entscheidend anders sind die Bedienoberflächen und das Bedienkonzept gestaltet. Während OS X und jedes andere Desktop-Betriebssystem typischerweise mittels Maus und Tastatur

gesteuert werden, verfügen iOS-Geräte lediglich über einen Touchscreen, über den mittels Fingergesten und Berührungen alle Aktionen gesteuert werden. Hier gibt es also ganz neue Aspekte, auf die wir als Entwickler achten müssen, um gut funktionierende und intuitiv bedienbare Apps zu entwickeln. Denn ein Finger zum Bedienen eines Touchscreens ist nun mal etwas gänzlich anderes als eine Maus, die ich pixelgenau an jede Position bewegen kann. Ein Finger besitzt wesentlich mehr Fläche und allein das muss bereits beim Konzipieren und Entwickeln eigener Anwendungen für iOS maßgeblich beachtet werden.

Auch sind die Nutzer mit iPhone und iPad mobil unterwegs, was in heutigen Zeiten mit sehr gutem Ausbau des Mobilfunknetzes nichtsdestotrotz bedeutet: Nicht immer ist Internet verfügbar (mal ganz davon abgesehen, dass es das iPad auch in einer reinen WLAN-Version ohne Mobilfunkverbindung gibt) und den Nutzer dazu zu zwingen, eine Internet-Verbindung herzustellen, sollte nur wirklich dann erforderlich sein, wenn es gar nicht anders geht und ein Internet-Zugang zwingend für die Nutzung der eigenen App (oder der gerade benötigten Funktion) notwendig ist.

iPhone und iPad sind Mobilgeräte, und genau so werden sie auch genutzt, soll heißen: Viele Nutzer holen ihr Smartphone nur für den Bruchteil eines Augenblicks hervor, checken aktuelle Facebook- oder WhatsApp-Nachrichten und lassen das Handy dann wieder verschwinden. Auch für Sie als App-Entwickler gilt: Halten Sie den Nutzer bei dem, was er mit Ihrer App tun will, nicht auf. Weniger ist hier ganz klar mehr. Ihre App soll eine eindeutige Funktion erfüllen, bieten Sie diese darum dem Nutzer so komfortabel, übersichtlich und leicht zugänglich wie nur irgend möglich an.

Daneben gibt es noch einen weiteren wichtigen Aspekt, den wir als Entwickler immer berücksichtigen sollten: Schonender Umgang mit den Akku-Ressourcen. Wenn wir ununterbrochen Bluetooth in Beschlag nehmen und nach anderen Geräten suchen, saugen wir den Akku des Nutzers damit sehr schnell leer und dürfen uns wahrscheinlich im Umkehrschluss über schlechte Kritiken unserer App im App Store „freuen“. Hier gilt ganz klar: Weniger ist mehr, und Ihre App sollte sich immer auf genau die Aufgabe konzentrieren, für die sie gedacht ist.

Sie sehen also, Apps für iOS zu entwickeln besteht nicht nur darin, die Programmiersprache und SDKs zu beherrschen; es geht auch darum zu verstehen, wie die iOS-Gerätefamilie funktioniert, wie sie genutzt wird und wie Sie mit Ihren Apps den Nutzern das Leben erleichtern.

■ 1.2 iOS für Entwickler

Apple hat mit dem App Store und iOS eine großartige Infrastruktur für uns Entwickler geschaffen. Wir können uns damit voll und ganz auf die Entwicklung unserer Apps konzentrieren, alle sonstigen Modalitäten wie Bezahlmethoden, Zahlungseingang oder Vertrieb übernimmt Apple für uns. Auch wenn Apple dafür einen Obolus in Form eines jährlichen Mitgliedsbeitrags im iOS Developer Program als auch 30% der Erlöse pro verkaufter App fordert, so stellt der App Store doch eine großartige Möglichkeit dar, die eigene Anwendung binnen kürzester Zeit einem internationalen (und auch durchaus kauffreudigen) Nutzerkreis zum Download zur Verfügung zu stellen.

Was an dieser Stelle auch gleich gesagt sein muss: Es gibt von Apple keinen anderen vorgesehenen Weg zur Installation einer App auf einem iOS-Gerät außer dem offiziellen App Store. Höchstwahrscheinlich haben Sie bereits einmal von einem „Jailbreak“ gehört, der es ermöglicht, unter anderem den Weg über den App Store zu umgehen und dadurch Apps aus beliebigen Quellen (wie zum Beispiel direkt über die Website eines Anbieters) auf dem eigenen Gerät zu installieren; das sollte aber nicht Ihr bevorzugtes Vorgehen sein, wenn Sie Apps für iOS entwickeln möchten. Zum einen schiebt Apple den Lücken, über die sich ein solcher Jailbreak durchführen lässt, regelmäßig einen Riegel vor, zum anderen ist das schlicht und einfach nicht der Weg, diese Plattform zu nutzen. Apple hat die iOS-Geräte als relativ geschlossene und abgeschottete Systeme konzipiert, und genau so sollten sie auch betrachtet und genutzt werden. Denn auf der anderen Seite sollte nicht der Sicherheitsgewinn vergessen werden, den gerade iOS gegenüber anderen Mobil-Betriebssystemen innehat; Schadsoftware lässt sich nur schwer bis gar nicht auf den Geräten installieren. Möglicherweise halten Sie diese Einstellung für engstirnig und betiteln mich in Gedanken bereits als „Apple Fanboy“, ich versuche aber schlicht, die iOS-Plattform als das zu sehen, was sie ist, und sie so zu nutzen, wie es gedacht ist. Damit ist sowohl uns Entwicklern als auch all den Millionen iOS-Nutzern da draußen am meisten geholfen.

So weit, so gut, doch was benötige ich als Entwickler nun konkret, um mit der Entwicklung für iOS starten zu können?

1.2.1 Hardware für Entwickler

Um für iOS entwickeln zu können, benötigen Sie in jedem Fall einen gewissen Fuhrpark an Apple-Geräten. Zunächst wäre hier einmal der Mac genannt. Ja, ein Mac ist notwendig, um für iOS entwickeln zu können, denn nur unter OS X – dem Betriebssystem des Mac – stehen die SDKs und Frameworks zur Entwicklung für iOS zur Verfügung. Für welches Gerät Sie sich dabei im Detail entscheiden, ist gut und gerne Ihnen und Ihren persönlichen Vorlieben überlassen, leistungstechnisch eignen sich alle. Achten Sie im Idealfall am ehesten noch darauf, einen Mac mit mindestens 8 GB Arbeitsspeicher zu erstehen; für die Entwicklung und zum Kompilieren Ihrer Apps ist das ein sehr angenehmes Mindestmaß, glauben Sie mir (siehe Bild 1.2).



Bild 1.2 Die Macs von Apple eignen sich alle gleichermaßen zur Entwicklung für iOS, auch wenn man mindestens 8 GB Arbeitsspeicher beim Gerätekauf berücksichtigen sollte. (Bild: apple.com)

**Tipp**

Das MacBook Pro ist nicht nur ein extrem leistungsstarkes Notebook, es verfügt zudem auch über das sogenannte Retina-Display, welches Apple in dieser Form auch in seinen iPhones und iPads sowie dem neuen iMac verwendet. Damit werden für einen Bildpunkt die doppelte Anzahl Pixel verwendet, wodurch Bilder, Texte und Anwendungen gestochen scharf dargestellt werden. Nicht nur, dass dieses Display mit seiner Auflösung von 2560 × 1600 Pixeln eine technische Meisterleistung darstellt, nein, es erleichtert Ihnen auch den Umgang mit dem iOS-Simulator während der Entwicklung, denn dieser Simulator kann auch in der nativen Retina-Auflösung von iPhone und iPad dargestellt werden. Auf Bildschirmen ohne Retina-Display bedeutet das, dass beispielsweise für den iPad-Retina-Simulator eine Fläche von 2048 × 1536 Pixeln auf dem Bildschirm angezeigt werden muss; selbst auf einem iMac mit 21,5“-Bildschirm passt dieser Simulator im Hochformat nicht auf das komplette Display. Bei einem MacBook Air mit seinen noch kleineren Displays wird es dann wahrlich schwierig, mit diesen riesigen Simulatoren zu arbeiten. Wenn Sie also gänzlich unentschlossen bei der Wahl Ihres Entwickler-Mac sind, sollten Sie möglicherweise das MacBook Pro mit Retina-Display oder alternativ den neuen iMac mit Retina-Display in Betracht ziehen.

Neben Ihrem Mac sollten Sie auch mindestens ein iPhone und/oder iPad besitzen – eben je nachdem, ob Sie nur für eine oder für beide Plattformen Apps entwickeln. Zwar haben Sie die Chance, all Ihre entwickelten Apps auch in einem Simulator auszuführen und zu testen, Sie sollten aber in jedem Fall vor der Veröffentlichung Ihrer App im App Store diese auch auf einem richtigen iOS-Gerät ausführlich geprüft haben. Der Simulator nutzt nämlich die komplette Hardware-Power Ihres Mac, wodurch es passieren kann, dass zwar im Simulator alles schnell und fluffig läuft, aber auf einem richtigen (möglicherweise auch schon etwas betagteren) iOS-Gerät alles ruckelt oder sogar abstürzt.

1.2.2 Software für Entwickler

Das Programm, mit dem Sie als Entwickler die meiste Zeit verbringen werden, ist Xcode. Xcode ist kostenlos über den App Store Ihres Mac erhältlich und ist die komplette IDE von Apple zur Softwareentwicklung für OS X und iOS. Es liefert alle Werkzeuge, die Sie benötigen, inklusive aller SDKs und Simulatoren (sehen Sie dazu auch das fünfte Kapitel „Die Entwicklungsumgebung – Xcode“). Bild 1.3 zeigt das App-Icon von Xcode.



Bild 1.3

Xcode: Mit dieser IDE legen Sie in der iOS-Entwicklung richtig los!

Daneben gibt es noch weitere Software, die Ihnen in Ihrer täglichen Entwicklerarbeit das Leben erleichtern kann, zum Beispiel spezielle Clients zur Arbeit mit der Versionsverwaltung Git oder Apps zur Arbeit mit Datenbanken. Diese sind aber nicht notwendig und werden von mir an passender Stelle im Buch in der benötigten Tiefe vorgestellt.

1.2.3 Das Apple Developer Program

Wie eingangs erwähnt, ist es mit der Entwicklung einer App allein noch nicht mit der Veröffentlichung im App Store getan. Neben den 30% Verkaufserlös, die Apple automatisch von Ihren App-Verkäufen abzwackt, müssen Sie auch noch Mitglied im Apple Developer Program sein. Als Mitglied können Sie dann ein eigenes Entwicklerzertifikat beantragen und damit Ihre Apps signieren, was nötig ist, um diese in den App Store einzureichen als auch um sie auf einem eigenen iOS-Gerät testen zu können. Insgesamt gibt es drei verschiedene Formen des Apple Developer Program, die hier nun einmal in Kürze vorgestellt werden sollen:

Apple Developer Program Individual

Das ist das Entwicklerprogramm für alle, die alleine bzw. freiberuflich oder selbstständig Apps für iOS entwickeln und veröffentlichen möchten. Die Mitgliedschaft kostet 99 \$ pro Jahr und erlaubt den vollen Zugriff auf alle Entwicklerressourcen von Apple, einschließlich Vorabversionen der Entwicklungsumgebung Xcode als auch iOS selbst.

Apple Developer Program Company

Das Apple Developer Program Company entspricht im Großen und Ganzen dem Individual Program, nur dass dieses hier explizit für Firmen ausgelegt ist. Dieses Programm bietet daher auch die Möglichkeit, mehrere Teammitglieder und Entwickler mit verschiedenen Rollen anzulegen und zu verwalten. Die Kosten belaufen sich – ebenfalls wie bei Individual – auf 99 \$ im Jahr.

Apple Developer Enterprise Program

Das Enterprise Program ist für Firmen gedacht, die Apps für interne Geschäftszwecke entwickeln und nutzen möchten. So erlaubt dieses Programm das Veröffentlichen von Apps auf beliebig vielen im Programm registrierten Geräten, es ist allerdings keine Veröffentlichung von Apps in Apples App Store möglich. Die Kosten für dieses Programm liegen bei 299 \$ pro Jahr.

Apple Developer University Program

Das Apple Developer University Program ist das einzige kostenlose Entwicklerprogramm von Apple. Wie der Name bereits andeutet, richtet es sich an Studierende und Lehrkräfte von Hochschulen und Universitäten, um dort beispielsweise iOS-Entwicklung zu unterrichten (siehe Bild 1.4).

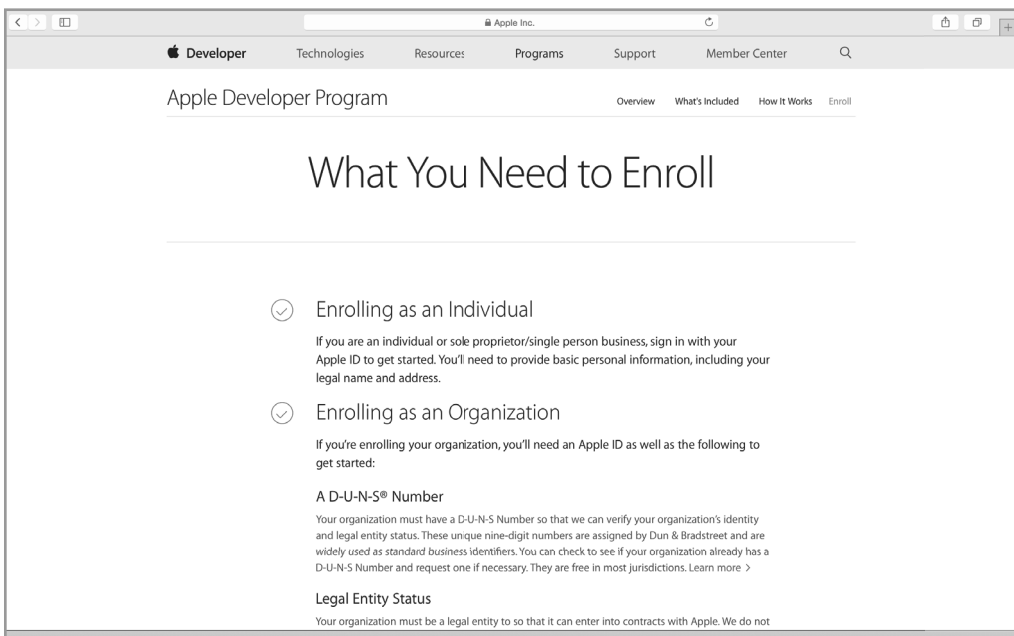


Bild 1.4 Auf <https://developer.apple.com/programs/enroll/> finden Sie weitere Informationen zum Apple Developer Program und gelangen von dort auch zur Registrierung. (Bild: developer.apple.com)

Weitere Informationen zum Apple Developer Program sowie das Formular für die Registrierung finden Sie unter <https://developer.apple.com/programs/enroll/>.

■ 1.3 Der Aufbau von iOS

Wir wissen nun also, was iOS ist, auf welchen Geräten es läuft und was diese Plattform ausmacht. Für uns als Entwickler ist aber besonders interessant, wie dieses System aufgebaut ist und wie die Architektur von iOS aussieht.

1.3.1 Die vier Schichten von iOS

iOS fußt auf insgesamt vier Schichten (den sogenannten Layern). Es gibt hier den Core OS Layer, den Core Services Layer, den Media Layer und den Cocoa Touch Layer (siehe Bild 1.5). All diese Schichten machen iOS zu dem, was es ist, und Sie als Entwickler nutzen die verschiedenen Funktionen der einzelnen Schichten, um Ihre Anwendungen zu entwickeln. Im Folgenden sollen diese Schichten einmal im Detail vorgestellt werden.

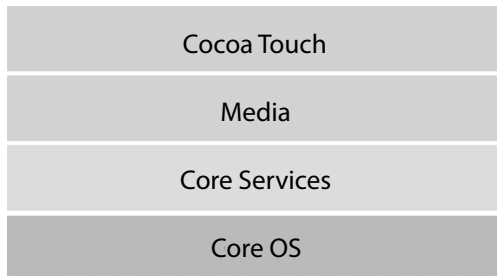


Bild 1.5

Die vier Schichten von iOS, chronologisch geordnet von unten nach oben. (Bild: Xcode-Dokumentation)

Core OS

An allererster Stelle steht das Core OS (und es ist somit die unterste Schicht des Systems). Es beherbergt die grundlegenden Funktionen und Frameworks zur Arbeit mit iOS, wobei wir die meiste Zeit über mit den darüber liegenden Schichten zu tun haben. Es ist aber wichtig zu wissen, dass es diese Funktionen und Frameworks gibt und wofür sie gut sind. Auf diesem Layer finden sich so beispielsweise das Core Bluetooth, das Security und das External Accessory Framework Netzwerkcommunication, Input/Output, Zugriffe auf das Dateisystem und mathematische Operationen.

So wichtig all diese Funktionen auch sind, so werden wir aber meist in unserer täglichen Arbeit – wie eben erwähnt – eher mit Frameworks arbeiten, die auf dem Core OS aufbauen und damit einen erleichterten und komfortableren Zugang auf diese Ressourcen erlauben. Sie sehen aber, dass Sie auch die Möglichkeit haben, selbst in diese Bereiche vorzudringen (sollte das nötig sein).

Core Services

Die nächste Schicht in der iOS-Systemarchitektur ist der Core Services Layer. Wie der Name bereits andeutet, stellt diese Schicht grundlegende Dienste zur Verfügung und setzt damit auf dem Core OS auf. Beispielsweise enthält dieser Layer die Peer-to-Peer-Services, mit

denen Verbindungen zwischen verschiedenen iOS-Geräten via Bluetooth hergestellt werden können. Wie Sie sehen, brauchen Sie für eine solche Aufgabe nicht zwingend das Core Bluetooth Framework aus dem Core OS zu nutzen; das entsprechende Multipeer Connectivity Framework aus dem Core Services Layer setzt darauf auf und bietet bereits Lösungen für den Verbindungsaufbau zwischen verschiedenen iOS-Geräten an.

Daneben enthält Core Services noch Frameworks zur Arbeit mit iCloud, die automatische Speicherverwaltung ARC (mehr dazu in Kapitel 4, „Programmierung für iOS“), Frameworks für In-App-Käufe und mehr.

Mit am wichtigsten für uns als Entwickler ist mit Sicherheit das Foundation Framework, welches grundlegende Datentypen wie Strings, Arrays etc., Funktionen zur Berechnung von Datum und Uhrzeit und weitere grundlegende Klassen liefert, die für die tägliche Arbeit unabdingbar sind, aber dazu später mehr.

Media

Hier wird es multimedial (im wahrsten Sinne des Wortes). Egal ob Frameworks zur Arbeit mit Bildern und Videos, zur Arbeit mit Animationen, zum Arbeiten mit Audio oder das Erstellen eigener Grafiken, im Media Layer findet sich all das Handwerkszeug, das Sie als Entwickler für alle Aufgaben rund um verschiedene Medien benötigen. Auch die Frameworks zur Arbeit mit AirPlay oder zum Verwalten eigener Game Controller finden sich auf dieser Schicht. Die meisten dieser Dinge werden aber ausschließlich für spezielle Einzelfälle benötigt, sodass wir in diesem Buch nur grundlegend auf das ein oder andere Framework näher eingehen werden (Sie werden aber sehen, dass Sie nach Lesen dieses Buches in der Lage sind, sich das nötige Know-how für all die anderen Media-Frameworks ohne Schwierigkeiten selbst anzueignen, versprochen).

Cocoa Touch

Mit Cocoa Touch erreichen wir die oberste Schicht in der iOS-Architektur und damit auch die Schicht, die den größten Unterschied zwischen iOS und OS X darstellt. Ganz grundlegend gesprochen stellt Cocoa Touch alle Frameworks und Funktionen zur Verfügung, um Apps für iOS zu entwickeln. Es enthält allen voran das UIKit Framework, welches verschiedene Views und Controller zum Bau von iOS-Apps enthält und womit dann das komplette Aussehen und Design einer App umgesetzt werden kann. Es enthält auch Frameworks und Funktionen für die grundlegende App-Infrastruktur, so beispielsweise ein System, um den Nutzer mittels Notifications zu informieren, oder die Arbeit mit verschiedenen Touch-Gesten. Viele enthaltene Frameworks bauen auf den Funktionen der darunter liegenden Schichten auf, womit Cocoa Touch einfach nutzbare und objektorientierte Schnittstellen bietet.

Zur Entwicklung von iOS-Apps ist dieses Framework essenziell und für grundlegende Apps auch ausreichend. Dennoch ist es wichtig, als Entwickler zu wissen, aus welchen Schichten sich iOS aufbaut und welche Funktionen Ihnen als Entwickler zur Verfügung stehen.

■ 1.4 Die perfekte iOS-App

O je, solch ein reißerischer Titel direkt zu Beginn dieses Buches? Nun ja, auch wenn ich Ihnen hier kein Patentrezept zu einer erfolgreichen und ertragreichen App geben kann, so kann ich Ihnen aber sagen, was Sie beachten müssen, wollen Sie mit Ihrer App-Idee wirklich erfolgreich sein. Und ich teile diese Weisheiten bereits mit Ihnen hier zu Beginn dieses Buches, weil es grundlegende Aspekte sind, die Sie bei jedem Schritt der Entwicklung (und bereits davor) im Hinterkopf behalten und absolut verstehen sollten.

Zunächst einmal: Achten Sie bei jedem Schritt der Entwicklung auf die Besonderheiten der iOS-Plattform (wie im vorigen Abschnitt beschrieben). Sie haben es hier mit Mobilgeräten zu tun, die die Nutzer in unregelmäßigen Abständen nutzen und mit denen sie meist gezielt eine bestimmte Aufgabe erfüllen möchten. Halten Sie den Nutzer nicht mit ewig langen Hinweisen oder Meldungen auf und achten Sie auf die Akku-Ressourcen und gar zu stromhungrige Funktionen, die Sie eigentlich gar nicht benötigen.

Wenn Sie das beherzigen (und zwar am besten bereits in der Planungs- und Konzeptionsphase, bevor die erste Zeile Code geschrieben ist), haben Sie bereits einen essenziellen Grundstein für eine erfolgreiche App gelegt. Denn Sie machen sich keine Vorstellung, bei wie vielen Apps genau diese Aspekte bereits schief laufen und zu entsprechend schlechten Kritiken im App Store führen. Darüber hinaus gibt es noch eine weitere Weisheit, die zwischen Erfolg und Misserfolg Ihrer App entscheiden kann und daher auch von Beginn an berücksichtigt werden sollte:

Konzentrieren Sie sich auf genau die eine Aufgabe, die Ihre App erfüllen soll! Denn dafür sind Apps für iOS gedacht: Für eine Aufgabe, die der Nutzer dank der App schnell, komfortabel und intuitiv lösen können soll. Was sich jetzt möglicherweise einfach und simpel anhört, ist in der Praxis bisweilen ein schmaler Grat und alles andere als einfach umzusetzen. Geben Sie sich daher im Vorhinein die Mühe und überlegen Sie, wie Sie sich selbst Ihre Traum-App vorstellen würden. Wie würde sie aussehen, welche Funktionen muss sie anbieten? Wie bedient sie sich, wie nutzt sie Animationen zum besseren Verständnis? Werden Sie sich über diese Dinge klar, bevor Sie beginnen, massiv Code zu schreiben, denn das geht meist nach hinten los. Und denken Sie daran, den Nutzer nicht mit unzähligen (und unnützen) Features zu überfrachten, die er gar nicht oder zumindest so gut wie gar nicht braucht; der Nutzer wird Ihnen diese Überfrachtung der App nicht danken. Deshalb betone ich es noch einmal, einfach weil es so wichtig ist und zwischen Erfolg und Misserfolg Ihrer Idee und Ihrer App entscheiden kann:



Wichtig

Konzentrieren Sie sich auf die eine Aufgabe, die Ihre App erfüllen soll ... und dann geben Sie Vollgas!

1.4.1 Apple Human Interface Guidelines

Bevor Sie dann mit der Entwicklung loslegen, ist eines auch ganz besonders wichtig: Werden Sie mit dem Betriebssystem und dem iPhone bzw. iPad als Zielplattform vertraut. Kennen Sie diese Geräte bisher lediglich aus der Presse und der Werbung und haben sich selbst noch nie mit ihnen über einen längeren Zeitraum beschäftigt, holen Sie das erst nach. Sonst kann es leicht passieren, dass Sie die Besonderheiten der Plattform und das typische Verhalten von iOS-Apps bei Ihrer Planung und Konzeption komplett missachten und schließlich eine App entwickeln, die sich vor schlechten Kritiken kaum retten kann. Eine große Hilfe an dieser Stelle sind die iOS Human Interface Guidelines, die wertvolle Tipps zum Aufbau und zur Gestaltung Ihrer App geben. In diesem Dokument hat Apple alle Informationen zu Design und Aufbau typischer für iOS optimierter Apps zusammengefasst, es ist daher nicht nur für Einsteiger in die iOS-Entwicklung einen Blick wert (siehe Bild 1.6).

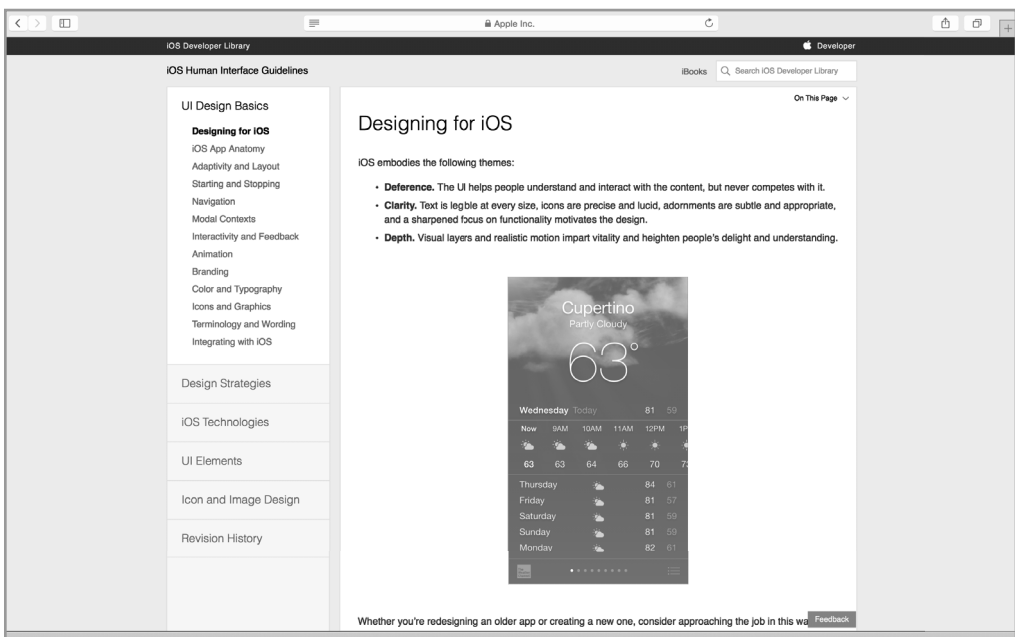


Bild 1.6 Die iOS Human Interface Guidelines sind eine sehr gute Anlaufstelle für die perfekte Gestaltung von iOS-Apps; selbst als erfahrener Entwickler lohnt ein Blick in dieses Entwicklerdokument von Apple. (Bild: <https://developer.apple.com/library/ios/documentation/UserExperience/Conceptual/MobileHIG/index.html>)

Berücksichtigen Sie bitte diesen Ratschlag, denn ich kann aus eigener Erfahrung sagen: Erst wenn Sie wissen, wie Sie mit iPhone und iPad effektiv arbeiten und wie Sie die verschiedenen Gesten und Strukturen der Apps nutzen, werden Sie imstande sein, selbst eine großartige und innovative App für genau diese Plattformen zu entwerfen und zu entwickeln. Und schließlich soll das doch das große Ziel sein, nicht wahr?

Index

Symbole

- @class 77
- @implementation 25
- #import 29, 76
- @interface 24
- @optional 73
- @protocol 73
- @selector 304
- @synthesize 45

A

- Abfragen 18, 96
- Access Control 168
- Accessibility 646
- Accounts 232
- Action 488, 500
 - mit User Interface 500
 - ohne User Interface 501
- Activation Events 549
- Alert 463
- alloc 54
- Any 117
- AnyObject 117
- APN 480
- App Delegate 202
- App Group 528
 - Registrierung 528
 - Zugriff 530
- App-Icon 253
- App ID 670
- App Name 684
- App Store 661
- App Store Review Guidelines 686
- Apple Developer Enterprise Program 7
- Apple Developer Program 6
- Apple Developer Program Company 6
- Apple Developer Program Individual 6

- Apple Developer University Program 7
- Apple Human Interface Guidelines 11
- Apple Push Notification Service 480
- Apple TV 1
- Apple Watch 533
- Apple Watch-Extension 491
- ARC 177
- Archiv 681
- Array 93, 108
- Asset-Bundle 252
- Atomarität 44
- Attributes Inspector 334
- Ausrichtung 321
- Auto Layout 374
- Autore sizing Masks 374

B

- Background Transfers 627
- Badge 463
- Badge Value 462
- Bedienkonzept 2
- Bedingung 18
- Beispielprojekte 210
- Benutzereinstellungen 432
- Block 79
 - Aufbau 80
 - Blockvariablen 84
 - globaler 84
 - als Parameter 82
 - als Property 83
- Bluetooth 3
- bool 15, 93
- Branch 652
- Breakpoints 279
- Breakpoint Navigator 283
- Build Phases 249
- Build Rules 249

Build Settings 248
 Bundle ID 684
 Bundle Identifier 216

C

Caller 295
 Capabilities 246
 char 16
 Child-View-Controller 324
 Clean Build 297
 Clean Code 300
 CLKComplicationDataSource 615, 621
 ClockKit Framework 612
 Cocoa Touch 9
 Code Signing 240
 Code Snippets 293
 Commit 652
 Complication 541, 611
 - Aufbau 612
 - Bestandteile 612
 - Template 612
 - Timeline 615
 Connections Inspector 336
 const 49
 Constraint 375
 Content Blocker 488, 519
 - Aktualisierung 523
 - Konfiguration 520
 Controller 316
 Convenience Initializer 150
 copy 43
 Core Bluetooth 8
 Core Data 444
 - Bestandteile 446
 Core Data-Editor 450
 Core OS 8
 Core Services 8
 Crashes 271
 Crash-Report 273
 CSR 666
 Custom Keyboard 488, 515
 - Erstellen 515
 - Mehrsprachigkeit 518

D

Dangling Pointer 179
 Data Source 313
 Dateisystem 435
 - Documents 435

- Library 436
 - tmp 436
 dealloc 180, 307
 Debug 262
 Debug Area 230
 Debug Navigator 285
 Debugging 275
 Deinitialisierung 153
 Delegation 312
 Deprecated Segues 366
 Designated Initializer 60, 150
 destinationViewController 370
 Development Profile 673
 Devices 216, 269, 665, 671
 Dictionary 93, 114
 Direkte Typzuweisung 95
 dispatch_once 86
 dispatch_once_t 86
 Document Provider 488, 507
 - Aufrufen 513
 Dokumentation 22, 194, 263, 427
 - Download 264
 Double 93
 Downcasting 156
 Downloads 237
 Doxygen 194
 Dynamic Notification Interface 601

E

Editor Area 227
 Elternklasse 24
 Embed In Application 490
 Entity 452
 Entwickler 3
 Entwickler-Account 662
 Entwickler-Einstellungen 680
 Entwicklerzertifikat 663
 - Erstellen 665
 enum 52
 Enumeration 52, 158
 Error Handling Model 162
 ErrorType 163
 Erweiterung 69
 Extensions 164, 487
 - Aufbau 492
 - Einschränkungen 493
 - Erstellen 489
 - Funktionsweise 492
 - Typen 487

External Accessory Framework 8
External Parameter Name 126

F

Fast Enumeration 209
File Inspector 330
File Provider 511
File-Sharing 443
File's Owner 329
First Responder 329
float 15, 93
for 21, 101
Force Touch 584
Foundation Framework 29, 171
Frameworks 245
Fundamental Types 92, 105

G

genstrings 258
Getter 40
Git 651
Glance Scene 540, 593
– App-Start 596
– Erstellen 595
– Lebenszyklus 596
– Test 598
Grand Central Dispatch 198

H

Hardware 4
Hilfslinien 338

I

IBAction 348
IBOutlet 344
IDE 211
Identifier 664
– Erstellen 670
Identity Inspector 333
if 18, 96
Image-Provider 620
Implicit Unwrapping 144
Info 247
Info.plist 201
init 54, 57
init-Methoden 59
Initialisierung 53, 145
Initializer 146
instancetype 57

Instanzvariable 38
Instruments 289
int 15, 93
Interactive Messaging 629
Interface Builder 212, 325
internal 168
Inverse Relation 456
iOS 1
IPA 273
iPad 1
iPhone 1
iPod touch 1
iTunes Connect 681, 682

J

Jailbreak 4
JavaScript 89

K

Kategorien 66
keyPath 304
Key-Value-Observing 302
Klasse 23, 117
– Header 23
– Implementation 25
Kommentare 22, 102
Konsole 276
Konstanten 49, 93
Koordinatensystem 403

L

Language 216
Launch Image 253
Layer 8
Lebenszyklus einer iOS-App 203
Lebenszyklus einer WatchKit App 546
Local Notifications 471
– Abbrechen 478
– Alert 473
– Ausführungstermin 472
– Badge Value 476
– Registrieren 477
– Sound 475
– zusätzliche Informationen 476
Local Parameter Name 126
Localizable.strings 254
Logik 431
Lokalisierung 254
Long-Look Interface 599, 600

M

Mac 4
 MacBook Pro 5
 Magic Numbers 49
 main.m 201
 Media 9
 Mehrfachvererbung 64
 Meine Apps 684
 Member Center 662
 Menü 584
 Methoden 30, 121

- Implementierung 33
- Instanzmethode 37, 130
- Klassenmethode 37, 130
- Methodenaufruf 36
- Methodennamen 32
- Method Signature Keyword 30
- Method Type Identifier 30
- Parameter 31
- Rückgabewert 30
- überschreiben 64

 Mobilfunknetz 3
 Multitasking 203
 Mutable 175
 MVC 299

N

Namenskonventionen 50
 Navigation Area 223
 Navigation Stack 552
 Nebenläufigkeit 198
 Netzwerkkommunikation 8
 new 55
 Nib 326, 410
 nil 42
 Notifications 308, 461
 Notification Action 465
 Notification Category 465
 Notification Scene 540, 599

- Erstellen 601
- Test 611

 Notification Type 463
 NSArray 173
 NSBundle 351
 NSData 175
 NSDate 175
 NSDictionary 174
 NSError 192
 NSExtensionItem 526
 NSFileManager 436

NSLayoutConstraint 379
 NSLocalizedString 254
 NSLog 277
 NSManagedObject 446

- Subklasse 457

 NSManagedObjectContext 446
 NSManagedObjectModel 447
 NSNotification 308
 NSNotificationCenter 308
 NSNumber 173
 NSObject 172
 NSPersistentStore 447
 NSPersistentStoreCoordinator 447
 NSSet 174
 NSString 173
 NSUserDefaults 432
 NULL 42

O

Object Library 337
 Objective-C 13
 Objective-C Bridging Header 190
 Objekt 14
 Objektorientierte Programmierung 13
 Open Quickly 294
 Operator 17, 95
 Optionals 141
 Organization Identifier 215
 Organization Name 215
 Organizer 271
 OS X 2

P

page-based Interface 567
 Parent-View-Controller 324
 Performance-Tests 643
 Photo Editing 488, 502
 Playgrounds 90
 PLIST-Datei 202
 Primärsprache 684
 Primitive Datentypen 15
 print 92, 103, 279
 private 168
 Product Name 215
 Projects 274
 Projekt 214
 Projekteinstellungen 238
 Properties (Swift) 132

- Computed Properties 132
- Property Observers 135

- Read-Only Computed Properties 134
- Type Properties 136
- Properties (Objective-C) 38, 39
 - Aufbau 40
 - Direktzugriff 44
 - Schreibbarkeit 43
- Protocol (Swift) 165
- Protocol (Objective-C) 72
 - Protokoll zuweisen 74
 - Vererbung in Protokollen 74
- Protocol Type 166
- Provider 481
- Provisioning Profile 664
 - Erstellen 673
- public 168
- Pull 652
- Punktnotation 42
- Push 652
- Push Notifications 479
 - Erstellen 484
 - Versand 480
- Python 89

Q

- Quality of Service 486
- Quick Help Inspector 332
- Quick Look 189, 282

R

- Refactoring 286
- Relationship 454
- Repository 651
- Resource Tags 247
- retain-Cycles 180
- return 34, 122
- rootViewController 316
- Rotation Events 320
- Run Without Building 297

S

- Scheme 218
- Schichten von iOS 8
- Schleife 18, 96
- Schlüsselbundverwaltung 666
- SDKs 5
- Security Framework 8
- Segue 369, 567
- self 41, 130
- Set 93

- Setter 40
- Share 487, 499
- Shared Links 488, 524
- Shortcuts 296
- Shorthand External Parameter Name 127
- Short-Look Interface 599
- Simulator 259
 - Hardware 260
- Singletons 85
- Size Classes 331
- Size Inspector 335
- Skriptsprache 14, 89
- SKU 684
- Software 5
- Sound 463
- Source Control 653, 657
- sourceViewController 370
- Speicherverwaltung 43, 177
- Static Notification Interface 600
- Storyboard 357
- Storyboard ID 368
- Strings 93, 105
- strong 43, 178
- Structures 161
- Strukturen 51
- Struktur einer App 201
- Subklasse 64, 138
- Subview 405
- super 65
- Superklasse 24, 64, 138
- Superview 405
- Swift 14, 89
 - Grundlagen 92
 - Voraussetzungen 89
- switch 20, 98

T

- Tab Bar Items 389
- Tabellen 570
- Table Row Controller 573
- Target 217
- Target-Action 310
- Team 678
- Test-Case-Klasse 639
- Test-Driven Development 650
- Text-Provider 620
- Today 487, 496
- Toolbar 221
- Type Casting 154, 209
- typedef 52

U

UIActionSheet 423
 UIAlertView 421
 UIButton 419
 UICollectionViewController 396, 427
 UICollectionViewController 396
 UICollectionViewFlowLayout 397
 UIControl 346
 UIDatePicker 424
 UIImageView 421
 UIInputViewController 515
 UIKit 176
 UILabel 419
 UIMapView 425
 UINavigationController 379
 UINib 413
 UIPickerView 423
 UIScrollView 425
 UISegmentedControl 420
 UISplitViewController 323, 398
 UIStoryboard 367
 UISwitch 420
 UITabBarController 385
 UITableView 389, 427
 UITableViewCell 390
 UITableViewController 389
 UITableViewDataSource 391
 UITableViewDelegate 394
 UI-Testing 645
 UITextField 420
 UITextView 426
 UIUserInterfaceIdiom 356
 UIUserInterfaceIdiomPad 356
 UIUserInterfaceIdiomPhone 356
 UIView 401
 UIViewController 317
 UIWebView 424
 Uniform Type Identifiers 510
 Unit-Tests 633
 unsigned int 15
 Utilities Area 229, 330

V

Variablen 16, 93
 – Globale Variable 35
 Variables View 276
 Vererbung 137

Veröffentlichung 681
 Version 684
 Version Editor 657
 Versionskontrolle 651
 Views 401
 View-Controller 315
 View-Controller-Hierarchien 323
 View-Hierarchien 404
 View Lifecycle 318
 void 30

W

Watch App 488, 528
 WatchConnectivity 626
 WatchKit App 539
 WatchKit Extension 539
 WatchKit SDK 536
 watchOS 533
 weak 43, 178
 Wertebereich 15
 while 21, 100
 Wiederverwendbarkeit 299
 WKExtensionDelegate 538
 WKInterfaceController 536, 548
 – Attribute 554
 – Initialisierung 548
 WKInterfaceDevice 538
 WKInterfaceGroup 563
 WKInterfaceObject 537
 WKInterfaceTable 570
 WKUserNotificationInterfaceController 607
 Workspace 214
 WWDC 14

X

Xcode 5, 26, 211
 – Aufbau 221
 – Einstellungen 231
 Xcode-Generated Header 191
 XCT-Bedingung 638
 XCTest Framework 633
 XIB 326, 410

Z

Zeiger 54, 56
 Zukunftssicherheit 300