

Erica Sadun


ADDISONWESLEY

iOS 5

Podręcznik programisty

Najlepsze przepisy dla każdego dewelopera!



 Helion

Tytuł oryginału: The iOS 5 Developer's Cookbook:
Core Concepts and Essential Recipes for iOS Programmers (3rd Edition)

Tłumaczenie: Robert Górczyński (wstęp, rozdz. 2 – 15), Łukasz Suma (rozdz. 1)

ISBN: 978-83-246-5121-4

Authorized translation from the English edition, entitled: THE IOS 5 DEVELOPER'S COOKBOOK: CORE CONCEPTS AND ESSENTIAL RECIPES FOR IOS PROGRAMMERS, Third Edition; ISBN 0321832078; by Erica Sadun; published by Pearson Education, Inc, publishing as Addison Wesley. Copyright © 2012 by Pearson Education, Inc.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from Pearson Education, Inc.

Polish language edition published by HELION S.A., Copyright © 2013.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Wydawnictwo HELION dołożyło wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie bierze jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Wydawnictwo HELION nie ponosi również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: helion@helion.pl
WWW: <http://helion.pl> (księgarnia internetowa, katalog książek)

Pliki z przykładami omawianymi w książce można znaleźć pod adresem: <ftp://ftp.helion.pl/przyklady/ios5pp.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/ios5pp>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

- [Kup książkę](#)
- [Poleć książkę](#)
- [Oceń książkę](#)

- [Księgarnia internetowa](#)
- [Lubię to! » Nasza społeczność](#)

Spis treści

Podziękowania	21
O autorce	22
Wstęp	23
1 Wprowadzenie do iOS SDK	33
Programy twórców iOS	33
Online Developer Program	33
Standard Developer Program	34
Developer Enterprise Program	35
Developer University Program	35
Rejestracja	35
Rozpoczęcie pracy	35
Pobieranie SDK	36
Urządzenia do tworzenia aplikacji	37
Ograniczenia symulatora	38
Podłączenie urządzenia do komputera	40
Różnice pomiędzy modelami	41
Wymiary ekranu	41
Kamera	42
Dźwięk	42
Telefonia	43
Różnice związane z Core Location i Core Motion	43
Obsługa wibracji i zbliżenia	44
Szybkości procesorów	44
OpenGL ES	44
Ograniczenia platformy	45
Ograniczenia przestrzeni na dane	45
Ograniczenia dostępu do danych	45
Ograniczenia pamięci	46
Ograniczenia interakcji	50
Ograniczenia energetyczne	50
Ograniczenia aplikacji	51
Ograniczenia możliwości działań użytkownika	52

Ograniczenia SDK	52
Korzystanie z serwisu Provisioning Portal	53
Zdefiniowanie zespołu	54
Uzyskiwanie certyfikatów	54
Rejestrowanie urządzeń	55
Rejestrowanie identyfikatorów aplikacji	56
Akredytacja	57
Składanie projektów iPhone w całość	58
Szkielet aplikacji iPhone	60
main.m	61
Delegat aplikacji	63
Kontroler widoku	65
Uwaga na temat przykładowego kodu prezentowanego w tej książce	66
Komponenty aplikacji iOS	67
Hierarchia katalogów aplikacji	67
Plik wykonywalny	67
Plik Info.plist	68
Ikona i obrazy rozruchowe	69
Pliki narzędzia Interface Builder	72
Pliki nienależące do paczki aplikacji	72
Archiwa IPA	73
Piaskownice	74
Paradygmaty programowania	74
Programowanie zorientowane obiektowo	75
Model-widok-kontroler	75
Podsumowanie	83
2 Obóz treningowy Objective-C	85
Język programowania Objective-C	85
Klasy i obiekty	86
Tworzenie obiektów	88
Alokacja pamięci	88
Zwalnianie pamięci	89
Licznik użycia obiektu	90
Metody, wiadomości i selektory	90
Niezadeklarowane metody	91
Wskaźnik do obiektu	92
Dziedziczenie metod	93
Deklarowanie metod	93
Implementacja metody	93
Metody klasy	95
Szybkie wyliczenie	96
Hierarchia klas	97
Wyświetlanie informacji	98
Podstawy zarządzania pamięcią	100
Zarządzanie pamięcią poprzez MRR	100
Zarządzanie pamięcią za pomocą technologii ARC	103

Właściwości	104
Hermetyzacja	105
Zapis z użyciem kropki	105
Właściwości i zarządzanie pamięcią	105
Deklaracja właściwości	106
Tworzenie własnych metod typu getter i setter	108
Kwalifikatory właściwości	110
Funkcja KVC	111
Funkcja KVO	112
Technologia MRR i duża wartość licznika użycia	112
Inne sposoby tworzenia obiektów	113
Usuwanie obiektu z pamięci	114
Używanie bloków	117
Definiowanie bloku w kodzie	118
Przypisywanie odniesień do bloku	119
Blok i zmienne lokalne	120
Blok i typedef	120
Blok i zarządzanie pamięcią w technologii MRR	121
Inne użycie bloków	121
Technologia ARC	121
Kwalifikatory właściwości i zmiennych	122
Licznik odniesień	125
Pula zwalniana automatycznie	127
Stosowanie technologii ARC — za i przeciw	128
Migracja do ARC	128
Wyłączenie ARC w projekcie	129
Wyłączenie ARC dla poszczególnych plików	130
Utworzenie projektu niezgodnego z ARC na bazie szablonu w Xcode	130
Reguły dotyczące ARC	131
Użycie technologii ARC wraz z Core Foundation i rzutowaniem typów danych	132
Rzutowanie pomiędzy Objective-C i Core Foundation	132
Wybór odpowiedniego podejścia	134
Rozwiązania stosowane w trakcie działania aplikacji	134
Wskazówki i podpowiedzi dotyczące pracy z ARC	135
Wzorzec Singleton	136
Kategorie (rozszerzenia klas)	137
Protokoły	138
Definiowanie protokołu	138
Stosowanie protokołu	139
Dodanie wywołań zwrotnych	139
Deklaracja opcjonalnych wywołań zwrotnych	140
Implementacja opcjonalnych wywołań zwrotnych	140
Zgodność z protokołem	141
Klasy Foundation	142
Ciągi tekstowe	142
Liczby i daty	147
Kolekcje	149

I jeszcze jedno: przekazywanie wiadomości	154
Implementacja przekazywania wiadomości	155
Operacje czyszczące	156
Superłatwe przekazywanie	157
Podsumowanie	157

3 Budowanie pierwszego projektu 159

Utworzenie nowego projektu	159
Utworzenie aplikacji na podstawie szablonu	161
Utworzenie nowego projektu	161
Wprowadzenie do przestrzeni roboczej Xcode	164
Przegląd projektu	169
Otwórz plik Storyboard dla iPhone'a	170
Edycja widoku	171
Uruchomienie aplikacji	173
Używanie symulatora	174
Symulator — co się dzieje w tle?	175
Współdzielenie aplikacji symulatora	177
Minimalistyczna wersja aplikacji typu Witaj, świecie	178
Przeglądanie API SDK	180
Konwersja plików modułu Interface Builder na ich odpowiedniki w Objective-C	181
Używanie modułu usuwania błędów	183
Ustawienie punktu kontrolnego	184
Wyświetlenie panelu modułu usuwania błędów	185
Analiza etykiety	186
Ustawienie innego punktu kontrolnego	187
Śledzenie wsteczne	188
Konsola	188
Dodanie prostego śledzenia	188
Zarządzanie pamięcią	189
Sposób: użycie narzędzia Instruments do wykrywania wycieków pamięci	190
Sposób: użycie narzędzia Instruments do monitorowania alokacji buforowanych obiektów	192
Symulacja sytuacji, gdy w systemie pozostało niewiele wolnej pamięci	193
Analiza kodu	195
Od Xcode do urządzenia — interfejs okna Organizer	196
Urządzenia	197
Informacje ogólne	197
Profile akredytacyjne	197
Dzienniki zdarzeń w urządzeniu	197
Aplikacje	199
Konsola	199
Zrzuty ekranu	199
Kompilacja dla urządzenia iOS	200
Używanie profilu akredytacyjnego programisty	200
Dodanie urządzenia, w którym można uruchamiać tworzone aplikacje	201
Przeanalizuj identyfikator aplikacji	201
Ustawienie urządzenia i tożsamości podpisywania kodu	202

Określenie bazowej wersji SDK dla aplikacji	203
Kompilacja i uruchomienie aplikacji Hello World	204
Podpisywanie skompilowanych aplikacji	205
Wykrywanie platformy docelowej podczas kompilacji	205
Sprawdzanie zgodności podczas działania aplikacji	205
Znaczniki pragma mark	207
Ukrywanie treści metod	208
Przygotowanie do dystrybucji	208
Odszukanie i oczyszczenie aplikacji	208
Używanie schematów i akcji	209
Dodanie własnej konfiguracji kompilacji aplikacji	210
Informacje dotyczące dystrybucji tymczasowej	211
Tworzenie pakietu w konfiguracji tymczasowej	212
Dystrybucja tymczasowa OTA	213
Utworzenie pliku manifestu	213
Zgłoszenie aplikacji do iTunes App Store	215
Podsumowanie	217
4 Projektowanie interfejsu użytkownika	219
Klasy UIView i UIWindow	219
Widoki wyświetlające dane	220
Widok przeznaczony do dokonania wyboru	221
Kontrolki	222
Tabele i kontrolki typu Picker	223
Paski	223
Pasek postępu i wskaźnik wykonywania operacji	224
Kontroler widoku	224
Klasa UINavigationController	225
Klasa UINavigationController	225
Klasa UITabBarController	226
Kontroler widoku podzielonego	226
Kontroler widoku strony	227
Kontroler typu Popover	227
Kontroler widoku tabeli	227
Kontroler książki adresowej	228
Klasa UIImagePickerController	228
Tworzenie wiadomości e-mail	228
Klasa UIDocumentInteractionController	229
Klasa GKPeerPickerController	229
Kontrolery odtwarzania treści multimedialnej	229
Geometria widoku	229
Pasek stanu	230
Paski nawigacyjny, narzędziowy i kart	231
Klawiatura i kontrolki Picker	233
Pole tekstowe	233
Klasa UIScreen	234
Tworzenie interfejsu	235

Użycie funkcji Storyboard do utworzenia interfejsu użytkownika	235
Utworzenie nowego projektu	236
Dodanie kolejnych kontrolerów widoku	236
Organizacja widoków	237
Uaktualnienie klas	238
Nazwy scen	238
Edycja atrybutów widoku	239
Dodanie przycisków nawigacyjnych	239
Dodanie kolejnego kontrolera nawigacyjnego	241
Nadawanie nazw kontrolerom	241
Pokolorowanie pasków nawigacyjnych	241
Dodanie przycisku	242
Zmiana początkowo wyświetlanego kontrolera	242
Dodanie kodu usuwającego widok z ekranu	242
Uruchomienie aplikacji	243
Użycie okien typu Popover	243
Dodanie kontrolera nawigacyjnego	244
Zmiana klasy kontrolera widoku	244
Dostosowanie do własnych potrzeb widoku okna Popover	245
Utworzenie połączeń	245
Edycja kodu	245
Użycie modułu Interface Builder do utworzenia konwertera temperatur	247
Utworzenie nowego projektu	247
Dodanie plików	247
Interface Builder	248
Dodanie etykiet i widoków	249
Włączenie zmiany orientacji	249
Test interfejsu	249
Dodanie outletów i akcji	249
Dodanie metody konwersji	251
Uaktualnienie rodzaju klawiatury	251
Połączenie interfejsu iPada	252
Ręczna budowa interfejsu aplikacji konwertera	253
Połączenie wszystkiego razem	255
Tworzenie, wczytywanie i używanie interfejsów hybrydowych	256
Utworzenie nowego pliku .xib przeznaczonego na interfejs użytkownika	256
Dodanie widoku i wypełnienie go treścią	257
Nadanie wartości tag widokom	257
Edycja kodu	257
Projektowanie w celu zapewnienia obrotu interfejsu	258
Włączenie zmiany układu	258
Automatyczna zmiana wielkości	261
Przykład automatycznej zmiany wielkości	262
Omówienie opcji automatycznej zmiany wielkości	264
Przenoszenie widoków	264
Sposób: przenoszenie widoków poprzez naśladowanie szablonów	265
I jeszcze jedno: kilka doskonałych podpowiedzi dotyczących modułu Interface Builder	268
Podsumowanie	270

5	Praca z kontrolerem widoku	271
	Praca z kontrolerem nawigacyjnym i widokiem podzielonym	271
	Używanie stosów i kontrolerów nawigacyjnych	273
	Umieszczanie i usuwanie kontrolera ze stosu	273
	Klasa elementu nawigacyjnego	274
	Kontroler modalny	275
	Sposób: budowa prostego menu składającego się z dwóch elementów	275
	Sposób: dodanie kontrolki segmentowanej	277
	Sposób: nawigacja pomiędzy kontrolerami widoków	279
	Sposób: wyświetlenie własnego widoku modalnego zawierającego informacje	281
	Sposób: kontroler widoku strony	285
	Właściwości książki	285
	Opakowanie implementacji	286
	Omówienie kodu sposobu	290
	Sposób: przewijanie stron w kontrolerze widoku strony	291
	Sposób: pasek kart	292
	Sposób: zapamiętanie stanu paska kart	295
	Sposób: utworzenie kontrolera widoku podzielonego	298
	Sposób: utworzenie aplikacji uniwersalnej używającej widoku podzielonego	302
	Sposób: własny pojemnik i przejście segue	304
	Przejścia pomiędzy kontrolerami widoków	308
	I jeszcze jedno: moduł Interface Builder i kontroler paska kart	309
	Podsumowanie	310
6	Tworzenie widoków i animacji	313
	Hierarchia widoku	313
	Sposób: odkrywanie drzewa hierarchii widoku	315
	Sposób: wykonywanie zapytań do podwidoków	315
	Zarządzanie podwidokami	317
	Dodanie podwidoku	317
	Zmiana kolejności i usuwanie podwidoków	318
	Wywołania zwrotne widoku	318
	Sposób: oznaczanie widoków wartościami tag oraz pobieranie widoków	319
	Używanie wartości tag do wyszukiwania widoków	319
	Sposób: nadawanie nazw widokom	321
	Obiekty powiązane	321
	Użycie słownika nazw	323
	Geometria widoku	326
	Ramka	326
	Przekształcenia	327
	Systemy współrzędnych	328
	Sposób: praca z ramką widoku	328
	Dostosowanie wielkości	329
	Struktura CGRect i punkt środkowy	330
	Inne metody narzędziowe	331
	Sposób: losowe przeniesienie widoku	334
	Sposób: przekształcenie widoku	336
	Wyświetlacz i cechy interakcji	337

Animacje obiektu UIView	338
Transakcyjne tworzenie animacji UIView	338
Tworzenie animacji za pomocą bloku	340
Animacja warunkowa	340
Sposób: pojawianie się i znikanie widoku	341
Sposób: zamiana widoków	342
Sposób: obrót widoku	343
Sposób: używanie przejść Core Animation	343
Sposób: efekt odbicia pojawiającego się widoku	345
Sposób: animacja widoku obrazu	346
I jeszcze jedno: dodanie odbicia do widoku	347
Podsumowanie	349
7 Praca z obrazami	351
Wyszukiwanie i wczytywanie obrazów	351
Odczyt danych obrazu	353
Sposób: uzyskanie dostępu do zdjęć z albumu zdjęć	356
Praca z klasą UIImagePickerController	358
Pobranie informacji z edycji obrazu	360
Sposób: pobieranie obrazu z adresu URL zasobu	360
Sposób: pstrykanie zdjęć i ich zapis w albumie zdjęć	362
Wybór pomiędzy aparatami	365
Zapis obrazów w katalogu Documents	366
Sposób: wysyłanie obrazów za pomocą wiadomości e-mail	367
Utworzenie treści wiadomości	368
Wyświetlenie kontrolera tworzenia wiadomości	369
Automatyzacja wykonywania zdjęć	370
Używanie własnych nakładek na podglądzie aparatu	370
Sposób: uzyskanie dostępu do aparatu z poziomu AVFoundation	371
Aparat jest wymagany	372
Sprawdzanie dostępności i rodzaju aparatu	372
Utworzenie sesji aparatu	373
Przełączanie pomiędzy aparatami	375
Podgląd dostarczany przez aparat	375
Ułożenie podglądu z aparatu	376
EXIF	376
Geometria obrazu	377
Tworzenie klasy pomocniczej do obsługi aparatu	379
Sposób: dodanie filtra Core Image	379
Sposób: wykrywanie twarzy za pomocą Core Image	381
Wyodrębnianie twarzy	386
Sposób: praca z obrazami rastrowymi	387
Rysowanie w kontekście rastrowym	388
Przetwarzanie obrazu	390
Rzeczywistość dotycząca przetwarzania obrazów	391
Sposób: pobieranie próbek na żywo	392
Konwersja na HSB	394

Sposób: utworzenie miniatur z obrazów	395
Wykonywanie zrzutu ekranu widoku	397
Rysowanie w plikach PDF	398
Tworzenie nowego obrazu zupełnie od początku	399
Sposób: wyświetlenie obrazu w przewijanym widoku	400
Tworzenie przewijanej strony zawierającej wiele obrazów	402
Podsumowanie	402
8 Gesty i dotknięcia	405
Dotknięcia	405
Fazy	406
Dotknięcia i metody ich obsługi	407
Dotykane widoki	407
Obsługa wielu dotknięć	408
Procedury rozpoznawania gestów	408
Sposób: dodanie prostej możliwości bezpośredniej manipulacji interfejsem	409
Sposób: dodanie procedury rozpoznawania gestu przesunięcia	410
Sposób: jednoczesne używanie wielu procedur rozpoznawania gestów	412
Usuwanie konfliktów pomiędzy procedurami rozpoznawania gestów	414
Sposób: ograniczenie ruchu	415
Sposób: testowanie miejsca dotknięcia	417
Sposób: testowanie grafiki rastrowej	418
Sposób: zapewnienie trwałości podczas bezpośredniej manipulacji interfejsem	420
Przechowywanie informacji o stanie	421
Przywracanie informacji o stanie	422
Sposób: zapewnienie trwałości poprzez archiwizację	423
Sposób: dodanie obsługi opcji cofnij	425
Utworzenie menedżera opcji cofnij	425
Obsługa operacji cofnięcia dla widoku potomnego	426
Praca z paskiem nawigacyjnym	426
Rejestracja operacji cofnięcia	427
Dodanie obsługi operacji cofnięcia poprzez potrząśnięcie urządzeniem	429
Dodanie nazwy akcji dla operacji cofnięcia i powtórzenia (krok opcjonalny)	430
Zapewnienie obsługi edycji poprzez potrząśnięcie	430
Wymuszenie statusu First Responder	430
Sposób: rysowanie w miejscu dotknięcia ekranu	431
Sposób: płynne rysowanie	432
Sposób: wykrywanie okręgów	435
Utworzenie własnej procedury rozpoznawania gestów	439
Sposób: używanie wielu dotknięć	441
Przytrzymanie ścieżek dotknięć	443
I jeszcze jedno: przeciąganie z widoku przewijanego	444
Podsumowanie	448
9 Tworzenie i używanie kontrolek	449
Klasa UIControl	449
Rodzaje kontrolek	449
Zdarzenia kontrolek	450

Przyciski	452
Dodawanie przycisków w module Interface Builder	453
Grafika	454
Łączenie przycisków z akcjami	455
Przyciski, które nie są przyciskami	455
Tworzenie własnych przycisków w Xcode	456
Tekst przycisku wyświetlany w wielu wierszach	458
Umieszczenie w przycisku animowanych elementów	459
Sposób: animacja reakcji przycisku	459
Sposób: dodanie suwaka wraz z własną gałką	461
Dostosowanie egzemplarza UISlider do własnych potrzeb	461
Poprawa skuteczności	465
Proxy wyglądu	465
Sposób: tworzenie kontrolki segmentowanej umożliwiającej odpowiedź na drugie stuknięcie	466
Sposób: utworzenie podklasy klasy UIControl	468
Utworzenie egzemplarza UIControl	470
Śledzenie dotknięć	470
Przekazywanie zdarzeń	471
Praca z przełącznikami i stepperami	471
Sposób: utworzenie suwaka z gwiazdkami	472
Sposób: budowa tarczy dotykowej	476
Dodanie kontrolki wskaźnika strony	478
Sposób: utworzenie własnego rozwiązania przeznaczonego do przewijania stron	480
Utworzenie paska narzędziowego	484
Utworzenie paska narzędziowego w kodzie	485
Podpowiedzi dotyczące paska narzędziowego w iOS 5	487
Podsumowanie	487
10 Praca z tekstem	489
Sposób: ukrycie klawiatury po zakończeniu edycji tekstu w kontrolce UITextField	489
Właściwości dotyczące cech tekstu	491
Inne właściwości dotyczące pól tekstowych	492
Sposób: dostosowanie widoków wokół klawiatur	493
Sposób: przykrycie widoku tekstu własnym widokiem pomocniczym	495
Sposób: zmiana wielkości widoku po podłączeniu klawiatury sprzętowej	497
Sposób: utworzenie własnego widoku wprowadzania danych	500
Sposób: widoki potrafiące obsłużyć dane wejściowe w postaci tekstu	504
Sposób: dodanie własnego widoku danych wejściowych do widoku nietekstowego	507
Dodanie dźwięku kliknięcia	509
Sposób: zbudowanie lepszego edytora tekstów	509
Sposób: filtrowanie tekstu	511
Sposób: wykrywanie wzorców tekstu	513
Tworzenie własnych wyrażeń	514
Enumeracja wyrażenia regularnego	516
Wykrywanie danych	516
Dodanie wbudowanego detektora danych	517

Sposób: wykrywanie błędów w UITextView	517
Wyszukiwanie ciągów tekstowych	518
Sposób: wyświetlenie dostępnych czcionek	519
Sposób: dodanie własnych czcionek do aplikacji	520
Sposób: podstawy pracy z Core Text i ciągami tekstowymi z atrybutami	521
Użycie pseudo-HTML do utworzenia tekstu z atrybutami	526
Sposób: podział na strony w Core Text	531
Sposób: generowanie tekstu do dokumentu PDF	532
Sposób: wyświetlanie tekstu w nieregularnych kształtach	533
Sposób: wyświetlanie tekstu na ścieżce	536
Wyświetlanie tekstu na ścieżkach Béziera	541
Wyświetlanie proporcjonalne	541
Wyświetlanie liter	542
I jeszcze jedno: duży tekst na ekranie iPhone'a	543
Podsumowanie	546
11 Tworzenie widoków tabel i zarządzanie nimi	547
Wprowadzenie do klas UITableView i UITableViewController	547
Utworzenie tabeli	548
Sposób: implementacja prostej tabeli	550
Wypełnienie tabeli	550
Metody źródła danych	551
Ponowne użycie komórek	552
Udzielanie odpowiedzi na działania użytkownika	552
Kolor zaznaczonej komórki	552
Zmiana koloru tła tabeli	553
Typy komórek	553
Sposób: utworzenie własnej komórki w module Interface Builder	554
Dodanie własnych atrybutów zaznaczonej komórki	556
Zdefiniowanie naprzemiennych kolorów komórek	556
Usunięcie efektu zaznaczenia komórki	557
Tworzenie tabel zgrupowanych	557
Sposób: zapamiętanie stanu kontrolki we własnej komórce	558
Wizualizacja ponownego wykorzystania komórki	560
Utworzenie „wybranej” komórki tabeli	561
Praca z przyciskiem typu disclosure	562
Sposób: edycja tabeli	564
Wyświetlanie kontrolki usuwania komórek	567
Usunięcie z ekranu kontrolki usuwania komórek	567
Obsługa żądań usunięcia komórek	567
Obsługa funkcji cofnięcia operacji	567
Gest machnięcia w komórce	568
Dodawanie komórek	568
Zmiana kolejności komórek	568
Algorytmiczne sortowanie tabeli	569
Sposób: praca z sekcjami	570
Tworzenie sekcji	571
Zliczanie sekcji i rekordów	572

Zwrot komórek	572
Utworzenie nagłówka sekcji	573
Utworzenie indeksu sekcji	573
Mechanizm delegacji w sekcjach	574
Sposób: wyszukiwanie w tabeli	575
Utworzenie kontrolera opcji wyszukiwania	575
Utworzenie metod źródła danych	576
Metody delegata	578
Używanie indeksu wyszukiwania	578
Dostosowanie nagłówka i stopki do własnych potrzeb	579
Sposób: dodanie do tabeli opcji „pociągnij, aby odświeżyć”	581
Utworzenie w kodzie własnej tabeli zgrupowanej	583
Utworzenie zgrupowanej tabeli ustawień	583
Sposób: utworzenie tabeli zawierającej wiele tarcz	585
Utworzenie egzemplarza UIPickerView	586
Sposób: użycie kontrolki UIPickerView z widokami	588
Sposób: użycie egzemplarza UIDatePicker	590
Utworzenie egzemplarza UIDatePicker	592
I jeszcze jedno: formatowanie daty	592
Podsumowanie	595
12 Rozpoczęcie pracy z Core Data	597
Wprowadzenie do Core Data	597
Utworzenie i edycja plików modelu	598
Generowanie plików klasy	600
Utworzenie kontekstu Core Data	600
Dodawanie obiektów	601
Wykonywanie zapytań do bazy danych	603
Wykrywanie zmian	604
Usuwanie obiektów	604
Sposób: użycie Core Data jako źródła danych tabeli	605
Sposób: wyszukiwanie w tabeli a Core Data	608
Sposób: edycja danych na żywo w widoku tabeli Core Data	610
Sposób: implementacja funkcji cofnij i przywróć w Core Data	612
Podsumowanie	615
13 Komunikaty ostrzeżeń wyświetlane użytkownikowi	617
Bezpośrednia komunikacja z użytkownikiem poprzez komunikaty	617
Utworzenie prostego komunikatu	617
Delegat komunikatu	618
Wyświetlenie komunikatu	620
Rodzaje komunikatów	620
„Proszę czekać” — wyświetlenie użytkownikowi paska postępu	621
Używanie klasy UIActivityIndicatorView	621
Używanie klasy UIProgressView	622
Sposób: komunikat bez przycisków	623
Utworzenie pływającego wskaźnika postępu	625
Sposób: utworzenie komunikatu modalnego w pętli działania	625

Sposób: użycie metody o zmiennej liczbie argumentów do utworzenia komunikatu	628
Wyświetlenie prostego menu	629
Przewijanie menu	631
Wyświetlanie tekstu w egzemplarzu UIActionSheet	631
Sposób: utworzenie własnego widoku komunikatu	631
Widok, który można nacisnąć	633
Sposób: proste okno typu Popover	633
Sposób: powiadomienia lokalne	635
Wskaźniki komunikatów	636
Używanie plakierek w aplikacji	636
Sposób: proste komunikaty audio	636
Dźwięki systemowe	637
Wibracja	638
Komunikaty	638
Opóźnienia	639
I jeszcze jedno: wyświetlenie komunikatu i umożliwienie dostosowania poziomu głośności	640
Podsumowanie	641
14 Możliwości urządzenia	643
Uzyskanie dostępu do informacji podstawowych	643
Dodanie ograniczeń związanych z możliwościami urządzeń	644
Sposób: pobieranie informacji dodatkowych o urządzeniu	646
Monitorowanie poziomu naładowania baterii w iPhone	647
Włączanie i wyłączanie czujnika zbliżeniowego	648
Sposób: użycie przyśpieszoniomierza w celu ustalenia „góry” urządzenia	649
Synchroniczne pobieranie bieżącego kąta przyśpieszoniomierza	651
Obliczanie kąta względnego	652
Praca z podstawową orientacją	652
Sposób: użycie przyśpieszoniomierza do poruszania obiektami po ekranie	653
Dodanie nieco blasku	656
Sposób: podstawy Core Motion	656
Sprawdzanie dostępności czujników	657
Obsługa bloku	657
Sposób: pobieranie i używanie informacji o położeniu urządzenia	660
Użycie zdarzeń ruchu do wykrycia wstrząśnięcia urządzeniem	661
Sposób: użycie przyśpieszoniomierza do wykrycia gestu wstrząśnięcia	663
Sposób: używanie ekranów zewnętrznych	666
Wykrywanie ekranu	667
Pobieranie informacji o rozdzielczości ekranu	667
Konfiguracja wyjścia wideo	667
Dodanie obiektu DisplayLink	668
Kompensacja overscanningu	668
VIDEOkit	668
I jeszcze jedno: sprawdzenie dostępnej ilości wolnego miejsca na dysku	671
Podsumowanie	672

15 Sieć	675
Sprawdzenie stanu połączenia z siecią	675
Sposób: rozbudowa klasy UIDevice o możliwości w zakresie sprawdzania dostępności	677
Skanowanie w poszukiwaniu zmian w połączeniu	679
Pobieranie IP i informacji o komputerze	681
Używanie kolejek, aby nie blokować aplikacji	684
Sprawdzanie dostępności witryny	685
Pobieranie synchroniczne	687
Pobieranie asynchroniczne w teorii	690
Sposób: pobieranie asynchroniczne	691
Obsługa wyzwań związanych z uwierzytelnieniem	697
Przechowywanie danych uwierzytelniających	697
Sposób: przechowywanie i pobieranie danych uwierzytelniających z pęku kluczy	700
Sposób: przekazywanie danych do serwera	703
NSOperationQueue	706
Twitter	706
Sposób: konwersja XML na postać drzewa	707
Drzewa	707
Utworzenie drzewa przetwarzania	707
Użycie drzewa wyników	709
Sposób: utworzenie prostego serwera opartego na sieci	711
I jeszcze jedno: użycie serializacji JSON	714
Podsumowanie	715
Skorowidz	717

Możliwości urządzenia

Każde urządzenie iPhone to mieszanka unikatowych, współdzielonych, tymczasowych i trwałych cech. Wspomniane cechy to między innymi aktualna fizyczna orientacja urządzenia, nazwa modelu, stan baterii oraz dostęp do komponentów sprzętowych. W tym rozdziale skoncentrujemy się na urządzeniu, począwszy od konfiguracji kompilacji, aż po aktywne czujniki sprzętowe. Przedstawię przykłady dostarczające różnych informacji na temat urządzenia, w którym została uruchomiona aplikacja. Dowiesz się, jak w trakcie działania aplikacji przetestować sprzęt pod kątem wymaganych cech, a następnie opisać te cechy w pliku *Info.plist*. Poznasz sposoby pobierania informacji z czujników i nasłuchiwanie powiadomień, dzięki którym można wywoływać metody po wykryciu zmian w stanie czujników. W rozdziale koncentruję się na sprzęcie, systemie plików i czujnikach dostępnych w iPhone'ie. Ponadto pomogę Ci w programowym wykorzystaniu możliwości oferowanych przez iPhone'a.

Uzyskanie dostępu do informacji podstawowych

Klasa `UIDevice` udostępnia klucze właściwości charakterystycznych dla urządzeń (np. dla iPhone'a i iPoda touch) pozwalające na sprawdzenie między innymi nazwy urządzenia, nazwy i wersji używanego systemu operacyjnego itd. Wymieniona klasa to proste rozwiązanie, które pozwala na szybkie pobranie pewnych informacji dotyczących systemu. Każda metoda jest metodą egzemplarza wywoływana jako Singleton, np. `[UIDevice currentDevice]`.

Poniżej wymieniono kilka właściwości klasy `UIDevice` pozwalających na pobranie pewnych informacji o urządzeniu.

- **systemName** — przechowuje nazwę aktualnie używanego systemu operacyjnego. W przypadku bieżącej generacji urządzeń iOS istnieje tylko jedna wersja systemu działającego na platformie: iPhone OS. Firma Apple nie uaktualniła jeszcze nazwy systemu, aby ogólnie odnosiła się do urządzeń iOS.
- **systemVersion** — podaje wersję oprogramowania firmware aktualnie zainstalowanego w urządzeniu, np. 4.2, 4.3, 5.0 itd.
- **model** — zawiera ciąg tekstowy opisujący platformę, np. iPhone, iPad lub iPod touch. Jeżeli rodzina urządzeń iOS zostanie powiększona, pojawią się dodatkowe ciągi tekstowe opisujące nowe urządzenia. Właściwość `localizedModel` to zlokalizowana wersja wymienionej właściwości.

- **userInterfaceIdiom** — określa styl interfejsu używanego w bieżącym urządzeniu, czyli iPhone (dla urządzeń iPhone i iPod touch) lub iPad. Mogą pojawić się jeszcze inne style, jeżeli firma Apple wprowadzi kolejne urządzenia iOS.
- **name** — zawiera ciąg tekstowy opisujący nazwę przypisaną urządzeniu w aplikacji iTunes przez użytkownika, np. *iPhone Janka* lub *Mój telefon*. Ta nazwa jest wykorzystywana także do utworzenia nazwy lokalnej dla urządzenia.

Poniżej przedstawiono kilka przykładów użycia wymienionych właściwości:

```
UIDevice *device = [UIDevice currentDevice];
NSLog(@"Nazwa systemu: %@", device.systemName);
NSLog(@"Model: %@", device.model);
NSLog(@"Nazwa urządzenia: %@", device.name);
```

W przypadku bieżących wydań systemu iOS masz możliwość sprawdzenia typu urządzenia dzięki prostemu testowi zwracającego wartość boolowską:

```
#define IS_IPAD (UI_USER_INTERFACE_IDIOM() == UIUserInterfaceIdiomPad)
```

Można przyjąć założenie, że powyższy test zwróci wartość NO, gdy aktualnym urządzeniem nie będzie iPad (czyli w przypadku urządzeń iPhone i iPod touch). Jeśli firma Apple wypuści na rynek nową rodzinę urządzeń, wtedy będziesz musiał odpowiednio uaktualnić kod. Oto przykład rozwiązania:

```
if (IS_IPAD)
{
    // Kod przeznaczony dla iPada.
}
else if (IS_IPHONE)
{
    // Kod przeznaczony dla urządzeń iPhone i iPod touch.
}
else if (IS_OTHER_DEVICE)
{
    // Kod przeznaczony dla innych urządzeń iOS.
}
```

Dodanie ograniczeń związanych z możliwościami urządzeń

Plik *Info.plist* pozwala na określenie wymagań aplikacji podczas jej umieszczania w iTunes. Zdefiniowane ograniczenia informują iTunes, jakie funkcje są wymagane przez daną aplikację.

Każde urządzenie iOS jest wyposażone w pewien zestaw funkcji. Niektóre posiadają wbudowane aparaty i GPS, natomiast inne są ich pozbawione. Jeszcze inne mają żyroskop, autofocus itd. Masz więc możliwość zdefiniowania funkcji wymaganych do działania Twojej aplikacji.

Umieszczając klucz `UIRequiredDeviceCapabilities` w pliku *Info.plist*, powodujesz, że iTunes ogranicza możliwość instalacji Twojej aplikacji jedynie do urządzeń wyposażonych w zdefiniowane funkcje. Listę funkcji podajesz jako tablicę ciągów tekstowych lub w postaci słownika.

Tablica zawiera każdą wymaganą funkcję, a wszystkie elementy tablicy (funkcje) muszą być dostępne w urządzeniu. Z kolei słownik pozwala na wyraźne zdefiniowanie funkcji jako wymaganej lub niedozwolonej w danym urządzeniu. Klucze słownika oznaczają możliwości, natomiast wartości określają, czy funkcja musi być obecna (boolowska wartość YES), czy nie (NO).

Aktualnie dostępne klucze słownika zostały wymienione w tabeli 14.1. Używaj tylko tych funkcji, które są absolutnie wymagane przez aplikację lub nie mogą być w żaden sposób obsługiwane. Jeżeli w aplikacji możesz umieścić pewne rozwiązanie pozwalające na obejście braku pewnej funkcji w urządzeniu, to nie definiuj tej funkcji za pomocą klucza słownika. Wszystkie funkcje wymienione w tabeli 14.1 zostały przedstawione w pozytywnym kontekście. Kiedy stosujesz wykluczenia zamiast wymogu dostępności danej funkcji, wtedy zastosuj jej negatywne znaczenie. Przykładowo zdefiniuj, że urządzenie nie może posiadać autofocusa lub żyroskopu bądź że dostęp do serwisu Game Center jest nieobsługiwany.

Tabela 14.1. *Możliwości oferowane przez urządzenie*

Klucz	Sposób użycia
telephony	Aplikacja wymaga programu Telefon lub używa schematu URL tel://.
wifi	Aplikacja wymaga dostępu do sieci lokalnej 802.11.
sms	Aplikacja wymaga programu Wiadomości lub używa schematu URL sms://.
still-camera	Aplikacja wymaga obecności aparatu w urządzeniu i może używać interfejsu kontrolki UIImagePickerController do wykonywania zdjęć tym aparatem.
auto-focus-camera	Aplikacja wymaga dodatkowego osprzętu (autofocus) w celu wykonywania zdjęć makro lub wyjątkowo ostrych zdjęć na potrzeby wykrywania danych na zdjęciach.
opengles-1	Aplikacja wymaga obsługi OpenGL ES 1.1.
opengles-2	Aplikacja wymaga obsługi OpenGL ES 2.0.
camera-flash	Aplikacja wymaga dostępności flesza w aparacie.
video-camera	Aplikacja wymaga aparatu, który ma możliwość nagrywania filmów.
accelerometer	Aplikacja wymaga obecności przyspieszeniomierza, który pozwala na znacznie więcej niż prostą zmianę orientacji UIViewController.
gyroscope	Aplikacja wymaga obecności żyroskopu w urządzeniu.
location-services	Aplikacja wymaga użycia technologii Core Location.
gps	Aplikacja używa Core Location i wymaga dodatkowej dokładności podczas określania położenia urządzenia za pomocą GPS.
magnetometer	Aplikacja używa Core Location i wymaga zdarzeń związanych z poruszaniem się — to znaczy kierunku podróży. (Ten czujnik jest wbudowany w kompasie).
gamekit	Aplikacja wymaga dostępu do serwisu Game Center (system iOS 4.1 i nowsze).
microphone	Aplikacja używa mikrofonu wbudowanego lub (obsługiwanego) osprzętu oferującego mikrofon.
armv6	Aplikacja jest skompilowana <i>tylko</i> dla zbioru instrukcji armv6 (iOS 3.1 i nowsze).
armv7	Aplikacja jest skompilowana <i>tylko</i> dla zbioru instrukcji armv7 (iOS 3.1 i nowsze).
peer-peer	Aplikacja używa połączeń Bluetooth typu jeden do jednego w serwisie Game Center (iOS 3.1 i nowsze).

Przeanalizujemy przykład aplikacji oferującej możliwość wykonywania zdjęć, jeśli urządzenie jest wyposażone w aparat. Jeżeli taka aplikacja działa w urządzeniach niewyposażonych w aparat, np. wcześniejszych wersjach iPoda touch, nie umieszczaj w słowniku wymagania dostępności aparatu. Zamiast tego dostępność aparatu sprawdź z poziomu kodu w aplikacji i wyświetlaj interfejs aparatu, gdy taki znajduje się w urządzeniu. Dodanie klucza w słowniku spowoduje wyeliminowanie potencjalnych klientów, którzy posiadają urządzenia bez aparatu — wczesne wersje urządzenia iPod touch (od pierwszej do trzeciej generacji) oraz iPada pierwszej generacji.

Sposób: pobieranie informacji dodatkowych o urządzeniu

Funkcje `sysctl()` i `sysctlbyname()` pozwalają na pobieranie informacji dotyczących systemu. To są standardowe funkcje platformy UNIX pozwalające na wykonywanie zapytań do systemu operacyjnego w celu pobrania informacji szczegółowych dotyczących sprzętu i systemu. Jeśli zajrzysz do pliku `/usr/include/sys/sysctl.h` w komputerze Mac, to poznasz ogólny zasięg działania wymienionych funkcji. We wspomnianym pliku znajdziesz obszerną listę stałych, które można wykorzystać w charakterze parametrów wymienionych wcześniej funkcji.

Wspomniane stałe pozwalają na pobranie pewnych informacji, np. częstotliwości taktowania procesora, ilości dostępnej pamięci itd. W sposobie 14.1 zademonstrowałam pobranie tego typu informacji. Kod zawiera kategorię `UIDevice` odpowiedzialną za zbieranie informacji o urządzeniu, a następnie ich przekazywanie poprzez serię wywołań metod.

Sposób 14.1. Rozszerzone zbieranie informacji o urządzeniu

```
@implementation UIDevice (Hardware)
+ (NSString *) getSysInfoByName:(char *)typeSpecifier
{
    // Pobranie informacji dostarczanych przez funkcję sysctl().
    size_t size;
    sysctlbyname(typeSpecifier, NULL, &size, NULL, 0);

    char *answer = malloc(size);
    sysctlbyname(typeSpecifier, answer, &size, NULL, 0);

    NSString *results = [NSString stringWithCString:answer
        encoding: NSUTF8StringEncoding];
    free(answer);

    return results;
}

- (NSString *) platform
{
    return [UIDevice getSysInfoByName:"hw.machine"];
}

- (NSUInteger) getSysInfo: (uint) typeSpecifier
{
    size_t size = sizeof(int);
    int results;
    int mib[2] = {CTL_HW, typeSpecifier};
    sysctl(mib, 2, &results, &size, NULL, 0);
    return (NSUInteger) results;
}

- (NSUInteger) cpuFrequency
{
    return [UIDevice getSysInfo:HW_CPU_FREQ];
}

- (NSUInteger) busFrequency
{
    return [UIDevice getSysInfo:HW_BUS_FREQ];
}
}
```

```

- (NSInteger) totalMemory
{
    return [UIDevice getSysInfo:HW_PHYSMEM];
}

- (NSInteger) userMemory
{
    return [UIDevice getSysInfo:HW_USERMEM];
}

- (NSInteger) maxSocketBufferSize
{
    return [UIDevice getSysInfo:KIPC_MAXSOCKBUF];
}
@end

```

Pobierz kod przedstawionego sposobu

Aby pobrać kod przedstawionego tutaj sposobu, przejdź na stronę <ftp://ftp.helion.pl/przyklady/ios5pp.zip>, lub jeśli pobrałeś obraz dysku zawierający wszystkie przykłady przedstawione w książce, to przejdź do katalogu przeznaczonego dla rozdziału 14. i otwórz projekt dla tego sposobu.

Być może zastanawiasz się, dlaczego przedstawiona kategoria zawiera metodę `platform`, skoro standardowa klasa `UIDevice` na żądanie zwraca nazwę modelu urządzenia. Odpowiedź kryje się w możliwości rozróżniania różnych typów urządzeń.

Standardowo iPhone 3GS jest określany po prostu jako iPhone, podobnie jak iPhone 4. Z kolei przedstawiona kategoria podaje `iPhone2,1` dla modelu 3GS i `iPhone3,1` dla modelu iPhone 4. W ten sposób masz możliwość programowego odróżnienia pierwszej generacji iPhone'a (`iPhone1,1`) od iPhone'a 3G (`iPhone1,2`).

Każdy z wymienionych modeli oferuje różne wbudowane funkcje. Poznanie numeru modelu urządzenia, w którym została uruchomiona aplikacja, pozwala na określenie, czy dane urządzenie będzie potrafiło obsłużyć konkretną funkcję, np. GPS lub kompas.

Monitorowanie poziomu naładowania baterii w iPhone'ie

Masz możliwość programowego śledzenia poziomu naładowania baterii w iPhone'ie i jej aktualnego stanu. Poziom naładowania baterii jest przedstawiany za pomocą wartości zmiennoprzecinkowej z zakresu od 0.0 (w pełni rozładowana) do 1.0 (w pełni naładowana). Wartość ta przedstawia przybliżony poziom naładowania baterii, możesz ją wykorzystać przed rozpoczęciem operacji, której przeprowadzenie wymaga wyjątkowo dużej ilości energii.

Na przykład przed przeprowadzeniem serii długotrwałych obliczeń matematycznych możesz o tym ostrzec użytkownika i zaproponować podłączenie urządzenia do gniazdka sieciowego. Poziom naładowania baterii sprawdzasz poprzez wywołanie klasy `UIDevice`. Wartość zwrótna jest podawana w odstępach co 5%.

```

NSLog(@"Poziom naładowania baterii: %0.2f%",
      [[UIDevice currentDevice] batteryLevel] * 100);

```

Stan baterii ma cztery możliwe wartości. Urządzenie może być ładowane (to znaczy podłączone do źródła prądu), w pełni naładowane, odłączone od źródła prądu lub w stanie nieznanym. Stan baterii można odczytać dzięki właściwości `batteryState` klasy `UIDevice`:

```

NSArray *stateArray = [NSArray arrayWithObjects:
    @"Stan baterii jest nieznan",
    @"Urządzenie nie jest podłączone do źródła zasilania",
    @"Bateria jest ładowana",
    @"Bateria jest w pełni naładowana", nil];

UIDevice *device = [UIDevice currentDevice];
NSLog(@"Stan baterii: %@", [stateArray objectAtIndex:device.batteryState]);

```

Wymienione stany nie mogą być traktowane jako trwałe, a jedynie jako chwilowe, opisujące to, co aktualnie dzieje się z urządzeniem. To nie są opcje i nie są ze sobą łączone w celu przygotowania ogólnych informacji o baterii. Wymienione wartości odzwierciedlają ostatnią zmianę, która dotyczy stanu baterii.

Bardzo łatwo możesz monitorować zmiany stanu baterii poprzez udzielanie odpowiedzi na powiadomienia informujące o zmianie stanu baterii. W ten sposób wychwytyjesz chwilowe zdarzenia, takie jak naładowanie baterii, podłączenie urządzenia do źródła zasilania w celu naładowania baterii oraz odłączenie urządzenia od źródła zasilania.

Aby rozpocząć monitorowanie właściwości `batteryMonitoringEnabled`, przypisz wartość `YES`. Podczas monitorowania klasa `UIDevice` generuje powiadomienia po każdym wystąpieniu zmiany w stanie baterii lub poziomie jej naładowania. Przedstawiona poniżej metoda nasłuchuje obu powiadomień. Zwróć uwagę, że wartości możesz sprawdzać także bezpośrednio bez konieczności oczekiwania na powiadomienie. Firma Apple nie daje żadnej gwarancji dotyczącej częstotliwości generowania uaktualnień informujących o zmianie poziomu naładowania. Jednak na podstawie przedstawionego fragmentu kodu przekonasz się, że tego rodzaju powiadomienia są generowane całkiem regularnie.

```

- (void) viewDidLoad
{
    // Włączenie monitorowania baterii.
    [[UIDevice currentDevice] setBatteryMonitoringEnabled:YES];

    // Dodanie obserwatorów stanu baterii i poziomu jej naładowania.
    [[NSNotificationCenter defaultCenter] addObserver:self
        selector:@selector(checkBattery)
        name:UIDeviceBatteryStateDidChangeNotification
        object:nil];
    [[NSNotificationCenter defaultCenter] addObserver:self
        selector:@selector(checkBattery)
        name:UIDeviceBatteryLevelDidChangeNotification
        object:nil];
}

```

Włączanie i wyłączanie czujnika zbliżeniowego

Obecnie czujnik zbliżeniowy jest komponentem dostępnym jedynie w iPhone'ie, urządzenia iPod touch i iPad nie oferują tego czujnika. O ile nie masz ważnego powodu trzymania smartfona iPhone blisko ciała (lub na odwrót), włączenie czujnika zbliżeniowego niewiele daje.

Po włączeniu czujnika zbliżeniowego ma on jedno podstawowe zadanie — wykrycie dużego obiektu znajdującego się przed urządzeniem. Po wykryciu obiektu następuje wyłączenie ekranu i wygenerowanie powiadomienia. Odsunięcie obiektu od urządzenia powoduje włączenie ekranu z powrotem. W ten sposób unikasz przypadkowego naciśnięcia przycisków lub wybrania numeru uchem, kiedy prowadzisz rozmowę telefoniczną. Niektóre kieszonko zaprojektowane ochrony na urządzenie uniemożliwiają poprawne działanie czujnika zbliżeniowego.

Aplikacja Google Mobile dostępna w sklepie iTunes App Store wykorzystuje czujnik zbliżeniowy do rozpoczęcia sesji nagrywania. Kiedy umieścisz telefon blisko głowy, aplikacja rozpocznie nagrywanie Twojego polecenia, a następnie zinterpretuje je po odsunięciu urządzenia od głowy. Programiści aplikacji nie przejęli się wyłączeniem ekranu, ponieważ do poprawnego nagrywania informacji nie jest wymagany graficzny interfejs użytkownika.

Przedstawiona poniżej metoda demonstruje sposób pracy z czujnikiem zbliżeniowym dostępnym w iPhone’ie. Kod wykorzystuje klasę `UIDevice` w celu włączenia czujnika zbliżeniowego i nasłuchuje powiadomień `UIDeviceProximityStateDidChangeNotification` w celu przechwycenia informacji o zmianie stanu czujnika. Dwa dostępne stany to czujnik włączony lub wyłączony. Kiedy właściwość `proximityState` klasy `UIDevice` zwraca wartość `YES`, oznacza to aktywację czujnika zbliżeniowego.

```
- (void) toggle: (id) sender
{
    UIDevice *device = [UIDevice currentDevice];

    // Określenie, czy czujnik zbliżeniowy jest aktualnie monitorowany, i jego ewentualne włączenie.
    BOOL isEnabled = device.proximityMonitoringEnabled;
    device.proximityMonitoringEnabled = ! isEnabled;
}

- (void) stateChange: (NSNotification *) notification
{
    // Zapis informacji powiadomienia.
    NSLog(@"Czujnik zbliżeniowy %@",
        [UIDevice currentDevice].proximityState ?
        @"spowoduje wyłączenie ekranu" :
        @"spowoduje z powrotem włączenie ekranu");
}

- (void) viewDidLoad
{
    // Dodanie obserwatora czujnika zbliżeniowego.
    [[NSNotificationCenter defaultCenter]
        addObserver:self
        selector:@selector(stateChange)
        name:@"UIDeviceProximityStateDidChangeNotification"
        object:nil];
}
```

Sposób: użycie przyśpieszeniomierza w celu ustalenia „góry” urządzenia

iPhone zawiera trzy czujniki pozwalające na pomiar przyśpieszenia urządzenia względem trzech osi: lewo/prawo (x), góra/dół (y) i do przodu/do tyłu (z). Wymienione wartości wskazują na siły działające na iPhone, zarówno grawitację, jak i ruch urządzeniem przez użytkownika. Możesz otrzymać ciekawe wartości opisujące wspomniane siły podczas obracania iPhone’em wokół głowy (siła dośrodkowa) lub rzucania iPhone’a z wysokiego budynku (spadanie swobodne). Niestety, nie będziesz w stanie odczytać tych wartości, gdy z iPhone’a pozostanie jedynie sterta złomu.

Aby obiekt otrzymywał informacje z przyśpieszeniomierza, należy zdefiniować go jako delegata. Obiekt ustawiony jako delegat musi implementować protokół `UIAccelerometerDelegate`:

```
[[UIAccelerometer sharedAccelerometer] setDelegate:self];
```

Po przypisaniu w obiekcie delegata wywoływana jest metoda `accelerometer:didAccelerate:`, w której możesz udzielić odpowiedzi na dane otrzymane z przyśpieszoniomierza. Struktura `UIAcceleration` przekazuje metodzie delegata trzy wartości zmiennoprzecinkowe dla osi x , y i z . Każda wartość mieści się w zakresie od -1.0 do 1.0 .

```
float x = acceleration.x;
float y = acceleration.y;
float z = acceleration.z;
```

W sposobie 14.2 wspomniane wartości pomagają w określeniu „góry” urządzenia. Kod oblicza arcus tangens pomiędzy wektorami przyśpieszenia X i Y i zwraca kąt. Po otrzymaniu nowych danych z przyśpieszoniomierza kod obraca egzemplarz `UIImageView` zawierający obraz strzałki (zobacz rysunek 14.1) w taki sposób, aby wskazać odpowiedni kierunek. Udzielana użytkownikowi odpowiedź w czasie rzeczywistym gwarantuje, że strzałka zawsze będzie wskazywała górę, niezależnie od sposobu umieszczenia urządzenia przez użytkownika.



Rysunek 14.1. *Odrobina matematyki pozwala na wskazanie kierunku „góra” dzięki użyciu funkcji arcus tangens względem sił działających na osie x i y . W omawianym przykładzie strzałka zawsze wskazuje górę niezależnie od sposobu, w jaki użytkownik umieści urządzenie*

Sposób 14.2. Przechwytywanie zdarzeń przyśpieszoniomierza

```
- (void)accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration
{
    // Określenie kierunku góra na podstawie wartości przyśpieszoniomierza dla osi x i y.
    float xx = -acceleration.x;
    float yy = acceleration.y;
    float angle = atan2(yy, xx);
    [arrow setTransform:CGAffineTransformMakeRotation(angle)];
}
```



```
- (void) viewDidLoad
{
    // Inicjalizacja delegata w celu rozpoczęcia przechwytywania zdarzeń przyśpieszeniomierza.
    [UIAccelerometer sharedAccelerometer].delegate = self;
}
```

Pobierz kod przedstawionego sposobu

Aby pobrać kod przedstawionego tutaj sposobu, przejdź na stronę <ftp://ftp.helion.pl/przyklady/ios5pp.zip>, lub jeśli pobrałeś obraz dysku zawierający wszystkie przykłady przedstawione w książce, to przejdź do katalogu przeznaczonego dla rozdziału 14. i otwórz projekt dla tego sposobu.

Synchroniczne pobieranie bieżącego kąta przyśpieszeniomierza

Czasami możesz chcieć pobrać dane z przyśpieszeniomierza bez konieczności definiowania delegata. Przedstawione poniżej metody przeznaczone do używania w ramach kategorii `UIDevice` ce pozwalają na synchroniczne pobieranie bieżącego kąta urządzenia względem płaszczyzn x i y — płaszczyzny przodu urządzenia iOS. Rozwiązanie polega na przygotowaniu nowej pętli działania, zaczekaniu na zdarzenie przyśpieszeniomierza, pobraniu kąta z wywołania zwrotnego, a następnie opuszczeniu pętli działania i zwróceniu pobranego kąta.

```
- (void) accelerometer:(UIAccelerometer *)accelerometer
    didAccelerate:(UIAcceleration *)acceleration
{
    float xx = acceleration.x;
    float yy = -acceleration.y;
    device_angle = M_PI / 2.0f - atan2(yy, xx);

    if (device_angle > M_PI)
    {
        device_angle -= 2 * M_PI;
    }

    CFRunLoopStop(CFRunLoopGetCurrent());
}

- (float) orientationAngle
{
    // Zdefiniowanie bieżącego delegata.
    id priorDelegate = [UIAccelerometer sharedAccelerometer].delegate;
    [UIAccelerometer sharedAccelerometer].delegate = self;

    // Oczekiwanie na odczyt wartości.
    CFRunLoopRun();

    // Przywrócenie delegata.
    [UIAccelerometer sharedAccelerometer].delegate = priorDelegate;

    return device_angle;
}
```

To podejście nie jest odpowiednie, jeśli chodzi o ciągłe otrzymywanie wartości przyśpieszeniomierza — do tego należy bezpośrednio wykorzystać wywołania zwrotne. Jednak doskonale nadaje się do okazjonalnego sprawdzania wartości kąta. Przedstawione powyżej metody zapewniają prosty i bezpośredni dostęp do bieżącej wartości kąta, pod jakim znajduje się urządzenie.

Obliczanie kąta względnego

Obsługa zmiany orientacji ekranu oznacza, że związki w interfejsie dla danego kąta, pod jakim znajduje się urządzenie, muszą być obsługiwane we wszystkich ćwiartkach, po jednej dla każdej dostępnej orientacji ekranu. Podobnie jak wtedy, gdy `UIViewController` automatycznie obraca widok na ekranie, tak samo tutaj konieczne są pewne obliczenia matematyczne w celu uwzględnienia zmiany orientacji.

Przedstawiona poniżej metoda przeznaczona dla kategorii `UIDevice` oblicza kąt, pod jakim znajduje się urządzenie, dzięki czemu ten kąt pozostaje w zgodzie z orientacją urządzenia. W ten sposób otrzymujesz wartość przesunięcia od pionu, która odpowiada aktualnemu sposobowi wyświetlania graficznego interfejsu użytkownika.

```
- (float) orientationAngleRelativeToOrientation:(UIDeviceOrientation) someOrientation
{
    float dOrientation = 0.0f;
    switch (someOrientation)
    {
        case UIDeviceOrientationPortraitUpsideDown:
            {dOrientation = M_PI; break;}
        case UIDeviceOrientationLandscapeLeft:
            {dOrientation = -(M_PI/2.0f); break;}
        case UIDeviceOrientationLandscapeRight:
            {dOrientation = (M_PI/2.0f); break;}
        default:
            break;
    }

    float adjustedAngle = fmod(self.orientationAngle - dOrientation, 2.0f * M_PI);
    if (adjustedAngle > (M_PI + 0.01f))
    {
        adjustedAngle = (adjustedAngle - 2.0f * M_PI);
    }
    return adjustedAngle;
}
```

Powyższa metoda używa reszty z dzielenia w postaci liczby zmiennoprzecinkowej, aby obliczyć różnicę pomiędzy aktualnym kątem urządzenia i przesunięciem kątowym interfejsu. W ten sposób zostanie obliczona bardzo ważna wartość przesunięcia kąтового względem pionu.

Praca z podstawową orientacją

Klasa `UIDevice` używa wbudowanej właściwości `orientation` w celu pobrania fizycznej orientacji urządzenia. Urządzenie iOS obsługuje siedem możliwych wartości dla wymienionej właściwości:

- `UIDeviceOrientationUnknown` — orientacja jest aktualnie nieznaną;
- `UIDeviceOrientationPortrait` — przycisk *Początek* znajduje się na dole;
- `UIDeviceOrientationPortraitUpsideDown` — przycisk *Początek* znajduje się na górze;
- `UIDeviceOrientationLandscapeLeft` — przycisk *Początek* znajduje się po prawej stronie;
- `UIDeviceOrientationLandscapeRight` — przycisk *Początek* znajduje się po lewej stronie;
- `UIDeviceOrientationFaceUp` — ekran znajduje się na górze;
- `UIDeviceOrientationFaceDown` — ekran znajduje się na dole.

Podczas typowej sesji aplikacji urządzenie może przejść przez dowolne lub wszystkie wymienione orientacje. Choć orientacja jest ustalana w połączeniu z przyspieszeniomierzem, to w żaden sposób nie jest powiązana z wbudowaną wartością kątową.

System iOS oferuje dwa wbudowane makra pomagające w określeniu, czy orientacja urządzenia jest pionowa, czy pozioma: `UIDeviceOrientationIsPortrait()` i `UIDeviceOrientationIsLandscape()`. Uznałam za wygodne rozbudowę klasy `UIDevice` o możliwość sprawdzania orientacji na podstawie właściwości wbudowanych w urządzenie.

```
@property (nonatomic, readonly) BOOL isLandscape;
@property (nonatomic, readonly) BOOL isPortrait;

- (BOOL) isLandscape
{
    return UIDeviceOrientationIsLandscape(self.orientation);
}
- (BOOL) isPortrait
{
    return UIDeviceOrientationIsPortrait(self.orientation);
}
```

Twój kod może bezpośrednio nasłuchiwać powiadomień o zmianie orientacji urządzenia. W tym celu do wzorca Singleton, jakim jest `currentDevice`, należy wysłać wiadomość `beginGeneratingDeviceOrientationNotifications`. Następnie trzeba dodać obserwatora w celu przechwytywania powiadomień `UIDeviceOrientationDidChangeNotification`. Jak możesz oczekiwać, zakończenie nasłuchiwanie następuje po wywołaniu `endGeneratingDeviceOrientationNotification`.

Wskazówka

W trakcie pisania tej książki system iOS niepoprawnie zgłaszał orientację po pierwszym uruchomieniu aplikacji. Uaktualnienie orientacji następuje jedynie po zmianie położenia urządzenia lub wywołaniu metody `UIViewController`. Aplikacja uruchomiona w orientacji pionowej może nie być tak traktowana aż do chwili, gdy użytkownik obróci urządzeniem, a następnie ponownie obróci je do właściwej orientacji. Ten błąd występuje zarówno w symulatorze, jak i w rzeczywistym urządzeniu, a ponadto łatwo go odtworzyć. Rozwiązaniem może być użycie orientacji kątowej przedstawionej w jednym z przykładów zaprezentowanych w rozdziale.

Sposób: użycie przyspieszeniomierza do poruszania obiektami po ekranie

Po zastosowaniu odrobiny kodu przyspieszeniomierz iPhone'a można wykorzystać do poruszania obiektami na ekranie, odpowiadając w czasie rzeczywistym na poruszanie urządzeniem przez użytkownika. W sposobie 14.3 przedstawiłam animowanego motyla, którym użytkownik może poruszać po ekranie.

Sposób 14.3. *Poruszanie obiektami na ekranie na podstawie informacji z przyspieszeniomierza*

```
- (void)accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration
{
    // Wyodrębnienie komponentów przyspieszeniomierza.
    float xx = -acceleration.x;
```

```

float yy = acceleration.y;

// Przechowywanie ostatniej wartości kąta.
mostRecentAngle = atan2(yy, xx);

// Czy nastąpiła zmiana kierunku?
float accelDirX = SIGN(xvelocity) * -1.0f;
float newDirX = SIGN(xx);
float accelDirY = SIGN(yvelocity) * -1.0f;
float newDirY = SIGN(yy);

// Przyspieszenie. Aby zwiększyć „lepkosć”, należy zmniejszyć dodawaną wartość.
if (accelDirX == newDirX)
{
    xaccel = (abs(xaccel) + 0.85f) * SIGN(xaccel);
}
if (accelDirY == newDirY)
{
    yaccel = (abs(yaccel) + 0.85f) * SIGN(yaccel);
}

// Wprowadzenie zmiany przyspieszenia względem bieżącej prędkości.
xvelocity = -xaccel * xx;
yvelocity = -yaccel * yy;
}

- (void) tick
{
    // Wyzerowanie transformacji przed zmianą położenia.
    butterfly.transform = CGAffineTransformIdentity;

    // Przesunięcie motyla zgodnie z bieżącym wektorem prędkości.
    CGRect rect = CGRectOffset(butterfly.frame, xvelocity, 0.0f);
    if (CGRectContainsRect(self.view.bounds, rect))
    {
        butterfly.frame = rect;
    }

    rect = CGRectOffset(butterfly.frame, 0.0f, yvelocity);
    if (CGRectContainsRect(self.view.bounds, rect))
    {
        butterfly.frame = rect;
    }

    // Obrót motyla niezależnie od położenia.
    butterfly.transform = CGAffineTransformMakeRotation(mostRecentAngle + M_PI_2);
}

- (void) initWithButterfly
{
    CGSize size;

    // Wczytanie klatek animacji.
    NSMutableArray *butterflies = [NSMutableArray array];
    for (int i = 1; i <= 17; i++)
    {
        NSString *fileName = [NSString stringWithFormat:@"bf_%d.png", i];
        UIImage *image = [UIImage imageNamed:fileName];
        size = image.size;
        [butterflies addObject:image];
    }
}

```

```
// Rozpoczęcie animacji.
butterfly = [[UIImageView alloc] initWithFrame:(CGRect){.size=size}];
[butterfly setAnimationImages:butterflies];
butterfly.animationDuration = 0.75f;
[butterfly startAnimating];

// Określenie początkowej prędkości i przyspieszenia motyla.
xaccel = 2.0f;
yaccel = 2.0f;
xvelocity = 0.0f;
yvelocity = 0.0f;

// Dodanie motyla.
butterfly.center = RECTCENTER(self.view.bounds);
[self.view addSubview:butterfly];

// Aktywacja przyspieszeniomierza.
[[UIAccelerometer sharedAccelerometer] setDelegate:self];

// Uruchomienie licznika czasu.
[NSTimer scheduledTimerWithTimeInterval: 0.03f
 target: self selector: @selector(tick)
 userInfo: nil repeats: YES];
}
```

Pobierz kod przedstawionego sposobu

Aby pobrać kod przedstawionego tutaj sposobu, przejdź na stronę <ftp://ftp.helion.pl/przyklady/ios5pp.zip>, lub jeśli pobrałeś obraz dysku zawierający wszystkie przykłady przedstawione w książce, to przejdź do katalogu przeznaczony dla rozdziału 14. i otwórz projekt dla tego sposobu.

Sekret rozwiązania kryje się w zastosowaniu licznika czasu. Zamiast bezpośrednio odpowiadać na zmiany w przyspieszeniomierzu jak w przypadku kodu przedstawionego w sposobie 14.2, w omawianym przykładzie wywołania zwrotne przyspieszeniomierza dokonują pomiaru aktualnej siły. Następnie procedura stosuje te siły względem motyla, zmieniając jego ramkę. Poniżej wymieniłam kilka punktów, na które warto zwrócić uwagę.

- Kiedy kierunek siły pozostaje taki sam, motyl przyspiesza. Wzrost prędkości jest skalowany względem stopnia siły przyspieszenia w kierunku X lub Y.
- Metoda `tick` wywoływana przez licznik czasu powoduje przesunięcie motyla poprzez dodanie wektora prędkości do punktu początkowego motyla.
- Zakres ruchu motyla jest ograniczony i po dotarciu do krawędzi ekranu motyl przestaje się poruszać w danym kierunku. W ten sposób motyl cały czas pozostaje widoczny na ekranie. Metoda `tick` sprawdza, czy motyl dotarł do krawędzi ekranu. Na przykład jeśli motyl dotarł do krawędzi pionowej, to nadal może poruszać się poziomo.
- Motyl zmienia orientację i zawsze jest skierowany w dół. Odbywa się to poprzez zastosowanie polecenia obrotu w metodzie `tick`. Zachowaj ostrożność podczas używania transformacji względem ramki lub przesunięcia punktu środkowego. Przed zastosowaniem przesunięcia zawsze zeruj wartości, a następnie ponownie stosuj wszelkie zmiany kąta. W przeciwnym razie mogą wystąpić niepożądane zjawiska, np. powiększenie, zmniejszenie lub zdeformowanie ramki.

Dodanie nieco blasku

Wprawdzie zastosowana w kodzie sposobu 14.3 animacja motyla przyciąga oko, ale efekt można jeszcze poprawić poprzez zastosowanie w warstwie motyla emitera Core Animation. Emiter generuje cząsteczki w czasie rzeczywistym, używając do tego zestawu własnych właściwości. Przedstawiony poniżej fragment kodu powoduje utworzenie cząsteczek przypominających pewnego rodzaju chmurę kurzu poruszającą się wraz z motylem, powstałą jakby na skutek ruchu skrzydeł motyla.

```
// Dodanie emitera cząsteczek.
```

```
float multiplier = 0.25f;
```

```
CAEmitterLayer *emitter = [CAEmitterLayer layer];
emitter.emitterPosition = RECTCENTER(butterfly.bounds);
emitter.emitterMode = kCAEmitterLayerOutline;
emitter.emitterShape = kCAEmitterLayerCircle;
emitter.renderMode = kCAEmitterLayerAdditive;
emitter.emitterSize = CGSizeMake(100 * multiplier, 0);
```

```
// Utworzenie cząsteczki emitera.
```

```
CAEmitterCell* particle = [CAEmitterCell emitterCell];
particle.emissionLongitude = M_PI;
particle.birthRate = multiplier * 100.0;
particle.lifetime = multiplier;
particle.lifetimeRange = multiplier * 0.35;
particle.velocity = 180;
particle.velocityRange = 130;
particle.emissionRange = 1.1;
particle.scaleSpeed = 1.0;
particle.color = [[UIColor orangeColor] colorWithAlphaComponent:0.1f].CGColor;
particle.contents = (__bridge id)[UIImage imageNamed:@"spark.png"].CGImage;
particle.name = @"particle";
```

```
emitter.emitterCells = [NSArray arrayWithObject:particle];
[butterfly.layer addSublayer:emitter];
```

Sposób: podstawy Core Motion

Framework Core Motion centralizuje przetwarzanie danych ruchu. Wprowadzona w iOS 4 SDK technologia Core Motion zastępuje bezpośredni dostęp do przyspieszeniomierza, który wcześniej przedstawiłam. Technologia oferuje punkt centralny monitorowania trzech kluczowych czujników ruchu znajdujących się w urządzeniu. Wspomniane czujniki to żyroskop informujący o obrocie urządzenia, kompas wskazujący kierunek oraz przyspieszeniomierz wykrywający zmiany sił wzdłuż trzech osi. Czwarty punkt nazywany „ruchem urządzenia” łączy wszystkie trzy wymienione czujniki w pojedynczy system monitorowania.

Technologia Core Motion używa niezmodyfikowanych wartości wymienionych czujników w celu utworzenia możliwych do odczytu wartości, najczęściej w postaci wektorów siły. Wspomniane elementy składają się z wymienionych poniżej właściwości.

- **Położenie urządzenia (attitude)** — orientacja urządzenia względem pewnego punktu odniesienia. Położenie jest przedstawione za pomocą trzech kątów o wartościach wyrażanych w radianach.
- **Obrót (rotationRate)** — wartość, o jaką urządzenie jest obrócone względem każdej z trzech osi. Obrót obejmuje wyrażone w radianach wartości kąta dla osi x, y i z na sekundę.

- **Grawitacja (gravity)** — bieżący wektor przyśpieszenia urządzenia będący skutkiem normalnej grawitacji mierzonej w wartościach G względem trzech osi x, y i z. Każda jednostka przedstawia standardową siłę grawitacji powodowaną przez Ziemię (9,8 metra na sekundę).
- **Przyśpieszenie spowodowane przez użytkownika (userAcceleration)** — wektor przyśpieszenia powodowanego przez użytkownika. Podobnie jak w przypadku grawitacji przyśpieszenie jest mierzone w wartościach G względem trzech osi x, y i z. Po dodaniu wektorów grawitacji i przyśpieszenia powodowanego przez użytkownika otrzymujemy całkowite przyśpieszenie, jakie powoduje urządzenie.
- **Pole magnetyczne (magneticField)** — wektor przedstawiający ogólne wartości pola magnetycznego w sąsiedztwie urządzenia. Pole magnetyczne jest mierzone w mikroteslach wzdłuż osi x, y i z. Zapewniana jest także kalibracja dokładności w celu poinformowania aplikacji o jakości pomiaru.

Sprawdzanie dostępności czujników

Jak przeczytałeś we wcześniejszej części rozdziału, plik *Info.plist* możesz wykorzystać do wskazania wymaganych lub niedozwolonych do działania aplikacji czujników urządzenia. Masz więc możliwość sprawdzenia dostępności wszystkich czujników obsługiwanych przez technologię Core Motion.

```
if (motionManager.gyroAvailable)
{
    [motionManager startGyroUpdates];
}

if (motionManager.magnetometerAvailable)
{
    [motionManager startMagnetometerUpdates];
}

if (motionManager.accelerometerAvailable)
{
    [motionManager startAccelerometerUpdates];
}

if (motionManager.deviceMotionAvailable)
{
    [motionManager startDeviceMotionUpdates];
}
```

Rozpoczęcie generowania uaktualnień nie powoduje wywoływania metod delegata, jak miało to miejsce w przypadku klasy `UIAccelerometer`. Zamiast tego to programista jest odpowiedzialny za przechwycenie wartości. Ewentualnie możesz użyć mechanizmu opartego na bloku, którego kod będzie wykonywany podczas każdego uaktualnienia wartości (np. `startAccelerometerUpdatesToQueue:withHandler:`).

Obsługa bloku

Sposób 14.4 to praktycznie kod sposobu 14.3, ale zmodyfikowany do użycia technologii Core Motion. Polecenia z wywołań delegata zostały przeniesione do bloku, a wartości x i y są odczytywane z właściwości przyśpieszeniomierza. Poza tym kod pozostał bez zmian. W tym przykładzie możesz poznać podstawy Core Motion: utworzenie nowego menedżera ruchu. Pierwszym jego zadaniem jest sprawdzenie dostępności przyśpieszeniomierza. Następnie rozpoczyna się przekazywanie uaktualnionych wartości poprzez użycie nowej kolejki operacyjnej istniejącej przez cały czas działania aplikacji.

Sposób 14.4. Podstawowy sposób użycia Core Motion

```

@implementation TestBedViewController
- (void) tick
{
    butterfly.transform = CGAffineTransformIdentity;

    // Przesunięcie motyla zgodnie z bieżącym wektorem prędkości.
    CGRect rect = CGRectOffset(butterfly.frame, xvelocity, 0.0f);
    if (CGRectContainsRect(self.view.bounds, rect))
    {
        butterfly.frame = rect;
    }

    rect = CGRectOffset(butterfly.frame, 0.0f, yvelocity);
    if (CGRectContainsRect(self.view.bounds, rect))
    {
        butterfly.frame = rect;
    }

    butterfly.transform = CGAffineTransformMakeRotation(mostRecentAngle + M_PI_2);
}

- (void) shutDownMotionManager
{
    NSLog(@"Zamknięcie menedżera ruchu");
    [motionManager stopAccelerometerUpdates];
    motionManager = nil;

    [timer invalidate];
    timer = nil;
}

- (void) establishMotionManager
{
    if (motionManager)
    {
        [self shutDownMotionManager];
    }

    NSLog(@"Utworzenie menedżera ruchu.");

    // Utworzenie menedżera ruchu.
    motionManager = [[CMMotionManager alloc] init];
    if (motionManager.accelerometerAvailable)
    {
        [motionManager startAccelerometerUpdatesToQueue:[NSOperationQueue alloc] init]
        withHandler:^(CMAccelerometerData *data, NSError *error)
        {
            // Wyodrębnienie komponentów przyspieszoniomierza.
            float xx = -data.acceleration.x;
            float yy = data.acceleration.y;
            mostRecentAngle = atan2(yy, xx);

            // Czy nastąpiła zmiana kierunku?
            float accelDirX = SIGN(xvelocity) * -1.0f;
            float newDirX = SIGN(xx);
            float accelDirY = SIGN(yvelocity) * -1.0f;
            float newDirY = SIGN(yy);
        }
    }
}

```



```
// Przyspieszenie. Aby zwiększyć „lepkosć”, należy zmniejszyć dodawaną wartość.
if (accelDirX == newDirX)
{
    xaccel = (abs(xaccel) + 0.85f) * SIGN(xaccel);
}
if (accelDirY == newDirY)
{
    yaccel = (abs(yaccel) + 0.85f) * SIGN(yaccel);
}

// Wprowadzenie zmiany przyspieszenia względem bieżącej prędkości.
xvelocity = -xaccel * xx;
yvelocity = -yaccel * yy;
}];
}

// Uruchomienie licznika czasu.
timer = [NSTimer scheduledTimerWithTimeInterval: 0.03f
        target: self selector: @selector(tick)
        userInfo: nil repeats: YES];
}

- (void) initWithButterfly
{
    CGSize size;

    // Uruchomienie licznika czasu.
    NSMutableArray *butterflies = [NSMutableArray array];
    for (int i = 1; i <= 17; i++)
    {
        NSString *fileName = [NSString stringWithFormat:@"bf_%d.png", i];
        UIImage *image = [UIImage imageNamed:fileName];
        size = image.size;
        [butterflies addObject:image];
    }

    // Rozpoczęcie animacji.
    butterfly = [[UIImageView alloc] initWithFrame:(CGRect){.size=size}];
    [butterfly setAnimationImages:butterflies];
    butterfly.animationDuration = 0.75f;
    [butterfly startAnimating];

    // Określenie początkowej prędkości i przyspieszenia motyla.
    xaccel = 2.0f;
    yaccel = 2.0f;
    xvelocity = 0.0f;
    yvelocity = 0.0f;

    // Dodanie motyla.
    butterfly.center = RECTCENTER(self.view.bounds);
    [self.view addSubview:butterfly];
}

- (void) loadView
{
    [super loadView];
    self.view.backgroundColor = [UIColor whiteColor];
    [self initWithButterfly];
}

@end
```

Pobierz kod przedstawionego sposobu

Aby pobrać kod przedstawionego tutaj sposobu, przejdź na stronę <ftp://ftp.helion.pl/przyklady/ios5pp.zip>, lub jeśli pobrałeś obraz dysku zawierający wszystkie przykłady przedstawione w książce, to przejdź do katalogu przeznaczonego dla rozdziału 14. i otwórz projekt dla tego sposobu.

Metody `establishMotionManager` i `shutDownMotionManager` pozwalają aplikacji na uruchamianie i zamykanie menedżera ruchu na żądanie. Wymienione metody są wywoływane z poziomu delegata aplikacji, gdy aplikacja staje się aktywna oraz kiedy jej działanie jest wstrzymywane.

```
- (void) applicationWillResignActive:(UIApplication *)application
{
    [tbvc shutDownMotionManager];
}

- (void) applicationDidBecomeActive:(UIApplication *)application
{
    [tbvc establishMotionManager];
}
```

Wymienione metody zapewniają elegancki sposób zamykania i wznawiania usług ruchu w odpowiedzi na aktualny stan aplikacji.

Sposób: pobieranie i używanie informacji o położeniu urządzenia

Wyobraź sobie iPada postawionego na biurku. Możesz przechylić iPada i spojrzeć na obraz wyświetlany na jego ekranie. A teraz wyobraź sobie obracanie tym iPadem leżącym płasko na biurku, ale podczas obracania urządzeniem obraz pozostaje nieruchomy i zachowuje doskonałą pozycję względem otaczającego go świata. Niezależnie od tego, jak będziesz obracał iPadem, obraz pozostanie nieruchomy, ponieważ widok będzie samodzielnie się uaktualniał w celu zrównoważenia fizycznego ruchu urządzeniem. Na takiej zasadzie działa kod przedstawiony w sposobie 14.5.

Sposób 14.5. Użycie uaktualnień o położeniu urządzenia w celu zmiany położenia obrazu

```
- (void) shutDownMotionManager
{
    NSLog(@"Wyłączenie menedżera ruchu");
    [motionManager stopDeviceMotionUpdates];
    motionManager = nil;
}

- (void) establishMotionManager
{
    if (motionManager)
    {
        [self shutDownMotionManager];
    }

    NSLog(@"Włączenie menedżera ruchu.");

    // Włączenie menedżera ruchu.
```

```
motionManager = [[CMMotionManager alloc] init];
if (motionManager.deviceMotionAvailable)
{
    [motionManager startDeviceMotionUpdatesToQueue:[NSOperationQueue currentQueue]
    withHandler:^(CMDeviceMotion *motion, NSError *error) {
        CATransform3D transform;
        transform = CATransform3DMakeRotation(motion.attitude.pitch, 1, 0, 0);
        transform = CATransform3DRotate(transform, motion.attitude.roll, 0, 1, 0);
        transform = CATransform3DRotate(transform, motion.attitude.yaw, 0, 0, 1);
        imageView.layer.transform = transform;
    }];
}
```

Pobierz kod przedstawionego sposobu

Aby pobrać kod przedstawionego tutaj sposobu, przejdź na stronę <ftp://ftp.helion.pl/przyklady/ios5pp.zip>, lub jeśli pobrałeś obraz dysku zawierający wszystkie przykłady przedstawione w książce, to przejdź do katalogu przeznaczonego dla rozdziału 14. i otwórz projekt dla tego sposobu.

Obraz dostosowuje się niezależnie od sposobu, w jaki będziesz trzymał urządzenie. Oprócz tego możesz podnieść urządzenie i zmieniać jego orientację w przestrzeni. Jeżeli odwrócisz urządzenie i spojrzysz na nie, wtedy zobaczysz odwrócony „dół” obrazu. Urządzeniem możesz obracać w obu osiach: jedna przebiega pionowo od przycisku *Początek* do aparatu, natomiast druga poziomo od lewej do prawej strony iPada. Kolejna oś — pierwsza, którą poznasz — przechodzi jakby „na wylot” przez środek urządzenia, od punktu znajdującego się pod iPadem aż do punktu nad urządzeniem. Kiedy obracasz urządzeniem, reakcją obrazu jest utworzenie wirtualnego świata w tym urządzeniu.

W kodzie sposobu 14.5 pokazano, jak uzyskać powyższy efekt za pomocą kilku prostych przekształceń geometrycznych. Kod tworzy menedżera ruchu, pobiera uaktualnienia dotyczące ruchu urządzenia, a następnie na podstawie otrzymanych danych przeprowadza transformację obrazu.

Użycie zdarzeń ruchu do wykrycia wstrząśnięcia urządzeniem

Kiedy iPhone wykrywa zdarzenie ruchu, przekazuje je do obiektu aktualnie posiadającego przypisany status *First Responder*, czyli do podstawowego obiektu w łańcuchu odpowiedzi. Obiekty, które mogą mieć przypisany wymieniony status, potrafią przetwarzać zdarzenia. Zaliczamy do nich wszystkie widoki i okna, a także obiekt aplikacji.

Łańcuch odpowiedzi definiuje hierarchię obiektów, które mogą odpowiadać na zdarzenia. Kiedy obiekt znajdujący się na początku łańcucha odpowiedzi otrzymuje zdarzenie, to zdarzenie nie jest przekazywane dalej, ale obsłużone przez dany obiekt. Jeśli obiekt nie potrafi obsłużyć zdarzenia, wówczas przekazuje je do kolejnego obiektu w łańcuchu.

Obiekt może odpowiadać na zdarzenia, gdy ma przypisany status *First Responder* za pomocą metody o nazwie *becomeFirstResponder*. W przedstawionym poniżej fragmencie kodu obiekt `UIViewController` będzie odpowiadał na zdarzenia, gdy jego widok pojawi się na ekranie. Po usunięciu widoku z ekranu obiekt przestanie odpowiadać na zdarzenia.

```

- (BOOL)canBecomeFirstResponder
{
    return YES;
}

// Przypisanie statusu First Responder po wyświetleniu widoku na ekranie.
- (void)viewDidAppear:(BOOL)animated
{
    [super viewDidAppear:animated];
    [self becomeFirstResponder];
}

// Rezygnacja ze statusu First Responder po usunięciu widoku z ekranu.
- (void)viewWillDisappear:(BOOL)animated
{
    [super viewWillDisappear:animated];
    [self resignFirstResponder];
}

```

Obiekty z przypisanym statusem *First Responder* otrzymują wszystkie zdarzenia dotyku i ruchu. Wywołania zwrotne ruchu odpowiadają wywołaniom zwrotnym dotknięć, omówionym w rozdziale 8. zatytułowanym „Gesty i dotknięcia”.

- **motionBegan:withEvent:** — ta metoda wskazuje na początek zdarzenia ruchu. W trakcie pisania niniejszej książki istniał tylko jeden rodzaj rozpoznawanego zdarzenia: wstrząśnięcie. To się może zmienić w przyszłości, więc w kodzie warto umieścić procedurę sprawdzania typu ruchu.
- **motionEnded:withEvent:** — ta metoda jest wywoływana na końcu zdarzenia ruchu.
- **motionCancelled:withEvent:** — podobnie jak w przypadku dotknięć, także ruch może zostać przerwany przez połączenie przychodzące lub inne zdarzenie systemowe. Firma Apple zaleca implementację w kodzie produkcyjnym wszystkich trzech wymienionych tutaj metod (i podobnie wszystkich czterech metod dotyczących obsługi dotknięć).

Przedstawiony poniżej fragment kodu pokazuje przykład użycia metod obsługi zdarzeń ruchu. Po przetestowaniu tej aplikacji w rzeczywistym urządzeniu możesz mieć kilka spostrzeżeń. Przede wszystkim z punktu widzenia użytkownika zdarzenia rozpoczynające i kończące ruch występują niemal jednocześnie. Odtwarzanie dźwięku w obu wymienionych rodzajach zdarzeń jest zabójcze. Ponadto istnieje tendencja w kierunku wykrywania gestu wstrząśnięcia urządzeniem na boki. iPhone lepiej sprawdza się podczas wykrywania wstrząśnięcia urządzeniem na boki niż do przodu i do tyłu. Wreszcie zastosowana przez firmę Apple implementacja używa lekkiego podejścia blokowania. Nie ma możliwości wygenerowania nowego zdarzenia ruchu przed upływem sekundy od chwili przetworzenia poprzedniego. Takie samo blokowanie jest używane w gestach „wstrząśnij, aby pomieszać” i „wstrząśnij, aby cofnąć”.

```

- (void)motionBegan:(UIEventSubtype)motion
withEvent:(UIEvent *)event
{
    // Odtwórz dźwięk po rozpoczęciu gestu wstrząśnięcia.
    if (motion != UIEventSubtypeMotionShake)
    {
        return;
    }
    [self playSound:startSound];
}

```

```
- (void)motionEnded:(UIEventSubtype)motion withEvent:(UIEvent *)event
{
    // Odtwórz dźwięk po zakończeniu gestu wstrząśnięcia.
    if (motion != UIEventSubtypeMotionShake)
    {
        return;
    }
    [self playSound:endSound];
}
```

Sposób: użycie przyśpieszoniomierza do wykrycia gestu wstrząśnięcia

W sposobie 14.6 przedstawiłam rozwiązanie powielające przygotowany przez firmę Apple system wykrywania ruchu, ale bez konieczności przekazywania zdarzeń obiektowi z przypisanym statusem *First Responder*. Projekt jest zbudowany w oparciu o dwa kluczowe parametry: poziom czułości określający granicę, która musi być przekroczona w celu rozpoznania wstrząśnięcia, oraz czas blokady określający częstotliwość, z jaką mogą być generowane zdarzenia wstrząśnięcia.

Sposób 14.6. Wykrywanie gestu wstrząśnięcia za pomocą przyśpieszoniomierza

```
@implementation AccelerometerHelper
- (id) init
{
    if (!(self = [super init]))
    {
        return self;
    }

    self.triggerTime = [NSDate date];

    // Bieżący wektor siły.
    cx = UNDEFINED_VALUE;
    cy = UNDEFINED_VALUE;
    cz = UNDEFINED_VALUE;

    // Ostatni wektor siły.
    lx = UNDEFINED_VALUE;
    ly = UNDEFINED_VALUE;
    lz = UNDEFINED_VALUE;

    // Poprzedni wektor siły.
    px = UNDEFINED_VALUE;
    py = UNDEFINED_VALUE;
    pz = UNDEFINED_VALUE;

    self.sensitivity = 0.5f;
    self.lockout = 0.5f;

    // Przypisanie delegata dla przyśpieszoniomierza.
    [[UIAccelerometer sharedAccelerometer] setDelegate:self];

    return self;
}
```

```

- (void) setX: (float) aValue
{
    px = lx;
    lx = cx;
    cx = aValue;
}

- (void) setY: (float) aValue
{
    py = ly;
    ly = cy;
    cy = aValue;
}

- (void) setZ: (float) aValue
{
    pz = lz;
    lz = cz;
    cz = aValue;
}

- (float) dAngle
{
    if (cx == UNDEFINED_VALUE)
    {
        return UNDEFINED_VALUE;
    }
    if (lx == UNDEFINED_VALUE)
    {
        return UNDEFINED_VALUE;
    }
    if (px == UNDEFINED_VALUE)
    {
        return UNDEFINED_VALUE;
    }

    // Obliczenia dla pierwszej pary.
    float dot1 = cx * lx + cy * ly + cz * lz;
    float a = ABS(sqrt(cx * cx + cy * cy + cz * cz));
    float b = ABS(sqrt(lx * lx + ly * ly + lz * lz));
    dot1 /= (a * b);

    // Obliczenia dla drugiej pary.
    float dot2 = lx * px + ly * py + lz * pz;
    a = ABS(sqrt(px * px + py * py + pz * pz));
    dot2 /= a * b;

    // Zwrot różnicy pomiędzy kątami wektorów.
    return acos(dot2) - acos(dot1);
}

- (BOOL) checkTrigger
{
    if (lx == UNDEFINED_VALUE)
    {
        return NO;
    }

    // Sprawdzenie, czy można przetworzyć nowe dane
    if ([[NSDate date] timeIntervalSinceDate:self.triggerTime] < self.lockout)

```

```
{
    return NO;
}

// Pobranie aktualnej zmiany kąta.
float change = [self dAngle];

// Jeżeli dwie próbki nie zostały jeszcze zebrane, wartością zwrotną jest NO.
if (change == UNDEFINED_VALUE)
{
    return NO;
}

// Czy obliczona wartość jest większa od granicznej?
if (change > self.sensitivity)
{
    self.triggerTime = [NSDate date];
    return YES;
}
else
{
    return NO;
}
}

- (void)accelerometer:(UIAccelerometer *)accelerometer
didAccelerate:(UIAcceleration *)acceleration
{
    // Przystosowanie wartości do standardowego systemu współrzędnych.
    [self setX:-acceleration.x];
    [self setY:acceleration.y];
    [self setZ:acceleration.z];

    // Wszystkie zdarzenia przyśpieszeniomierza.
    if (self.delegate && [self.delegate respondsToSelector:@selector(ping)])
    {
        [self.delegate performSelector:@selector(ping)];
    }

    // Wszystkie zdarzenia wstrząśnięcia.
    if ([self checkTrigger] && self.delegate &&
        [self.delegate respondsToSelector:@selector(shake)])
    {
        [self.delegate performSelector:@selector(shake)];
    }
}
@end
```

Pobierz kod przedstawionego sposobu

Aby pobrać kod przedstawionego tutaj sposobu, przejdź na stronę <ftp://ftp.helion.pl/przyklady/ios5pp.zip>, lub jeśli pobrałeś obraz dysku zawierający wszystkie przykłady przedstawione w książce, to przejdź do katalogu przeznaczonego dla rozdziału 14. i otwórz projekt dla tego sposobu.

Klasa `AccelerometerHelper` przechowuje trzy wartości przyśpieszeniomierza. Każda z wartości przedstawia wektor siły w przestrzeni 3D. Każda kolejna para tych trzech wartości może być analizowana w celu określenia kąta pomiędzy dwoma wektorami. W omawianym przykładzie kąt pomiędzy pierwszymi dwoma elementami

i dwoma kolejnymi pomagają w ustaleniu wystąpienia gestu wstrząśnięcia. Kąt wyszukuje parę, w której drugi kąt jest większy od pierwszego. Jeżeli wartość kątowna wzrosła wystarczająco dużo, wówczas wykryty zostanie gest wstrząśnięcia.

Klasa pomocnicza nie wywołuje metod delegata aż do przekazania drugiego zdarzenia. Blokada uniemożliwia przeprowadzanie kolejnych wywołań aż do chwili upłynięcia zdefiniowanej ilości czasu. Takie rozwiązanie jest zaimplementowane poprzez przechowywanie godziny ostatniego zdarzenia wstrząśnięcia. Wszystkie wstrząśnięcia, które nastąpią przed upływem zdefiniowanej ilości czasu, zostaną zignorowane. Następnie mogą być generowane nowe gesty wstrząśnięcia.

Dostarczany przez firmę Apple system wykrywania wstrząśnięcia wykorzystuje znacznie bardziej skomplikowaną analizę danych przyspieszeniomierza. Zgodnie z informacjami technicznymi na ten temat system wyszukuje oscylacji w około ośmiu do dziesięciu kolejnych punktach danych. W kodzie sposobu 14.6 zastosowano mniej skomplikowane podejście i zademonstrowano, jak pracować z nieprzetworzonymi danymi przyspieszeniomierza w celu dostarczenia wyników przygotowanych na podstawie wspomnianych danych.

Sposób: używanie ekranów zewnętrznych

Istnieje wiele sposobów używania ekranów zewnętrznych z iOS 5. Weźmy np. iPada 2, który oferuje wbudowaną funkcję reprodukcji wideo na zewnętrznym ekranie. Dzięki użyciu przewodu VGA lub HDMI treść możesz wyświetlać zarówno w iPadzie, jak i na zewnętrznym ekranie. Począwszy od systemu iOS 5, wybrane urządzenia pozwalają na bezprzewodową reprodukcję wideo do Apple TV za pomocą technologii AirPlay. Wspomniana funkcja reprodukcji jest wyjątkowo użyteczna, ale w iOS nie jesteś ograniczony jedynie do kopiowania treści z jednego ekranu na drugi.

Klasa UIScreen pomaga w wykryciu i niezależnym użyciu zewnętrznego ekranu, który można potraktować jako nowe okno i dla którego można przygotować treść oddzielną od wyświetlanej na ekranie urządzenia iOS. Zewnętrzny ekran można podłączyć zarówno przewodem, jak i bezprzewodowo do Apple TV za pomocą AirPlay (począwszy od iOS 5 i iPada 2).

Geometria ma duże znaczenie i zaraz dowiesz się dlaczego. Obecnie urządzenia iOS korzystają z ekranów następujących urządzeń: stary iPhone o rozdzielczości 320×480 pikseli, Retina w iPhone'ie o rozdzielczości 640×960 pikseli, stary iPad o rozdzielczości 1024×768 pikseli i Retina w iPadzie o rozdzielczości 2048×1536 pikseli. Typowy obraz ma rozdzielczość 720×480 pikseli (480i i 480p), VGA ma rozdzielczości 1024×768 i 1280×720 (720p), a ponadto jest jeszcze dostępna rozdzielczość HD 1920×1080 pikseli (poprzez HDMI).

Dodaj do tego problemy z tak zwanym overscanem oraz inne ograniczenia wyświetlacza docelowego, a wyjście wideo szybko stanie się wyzwaniem pod względem geometrii. Na szczęście firma Apple oferuje odpowiednie rozwiązanie w systemie iOS 5. Zamiast próbować tworzyć związek typu jeden do jednego pomiędzy ekranem wbudowanym w urządzenie i zewnętrznym, po prostu utwórz treść w oparciu o dostępne właściwości ekranu. Możesz więc utworzyć okno, wypełnić je treścią, a następnie wyświetlić.

Ogólnie rzecz biorąc, jeśli zamierzasz tworzyć aplikacje korzystające z zewnętrznego ekranu, upewnij się o posiadaniu co najmniej jednego przewodu każdego z wymienionych typów (kompozytowe, komponentowe, VGA i HDMI), jak również obsługującego technologię AirPlay iPada, co pozwoli Ci na przetestowanie każdej możliwej konfiguracji. Przewody produkowane przez firmy trzecie mogą nie działać, więc upewnij się, że kupujesz przewody produkcji Apple.

Wykrywanie ekranu

Klasa `UIScreen` informuje o liczbie dostępnych ekranów. Jeżeli ich liczba jest większa niż jeden, wtedy wiadomo, że podłączony został ekran zewnętrzny. Pierwszym elementem tablicy `screens` zawsze jest ekran wbudowany w urządzenie iOS.

```
#define SCREEN_CONNECTED ([[UIScreen screens].count > 1])
```

Każdy ekran podaje swoje wymiary (fizyczne, wyrażone w punktach) oraz skalę ekranu (punkty względem pikseli). Dwa standardowe powiadomienia pozwalają na obserwację podłączania i odłączania ekranów od urządzenia.

// Rejestracja otrzymywania powiadomień o włączeniu i wyłączeniu ekranu.

```
[[NSNotificationCenter defaultCenter]
 addObserver:self selector:@selector(screenDidConnect:)
 name:UIScreenDidConnectNotification object:nil];
[[NSNotificationCenter defaultCenter]
 addObserver:self selector:@selector(screenDidDisconnect:)
 name:UIScreenDidDisconnectNotification object:nil];
```

Połączenie oznacza *dowolnego* rodzaju połączenie, zarówno przewodowe, jak i bezprzewodowe w technologii AirPlay. Po otrzymaniu tego rodzaju uaktualnienia upewnij się o policzeniu ekranów i dostosowaniu interfejsu użytkownika do właściwości nowych ekranów.

Do Ciebie należy zadanie konfiguracji okien po podłączeniu i odłączeniu ekranu od urządzenia iOS. Każdy ekran powinien posiadać własne okno przeznaczone do zarządzania treścią wyświetlaną na tym ekranie. Po wykryciu odłączenia ekranu nie zatrzymuj okna w pamięci, usuń je i ponownie je utwórz po wykryciu ponownego podłączenia ekranu.

Pobieranie informacji o rozdzielczości ekranu

Każdy ekran posiada właściwość `availableModes`, to tablica obiektów rozdzielczości uporządkowanych w kolejności od najmniejszej do największej. Z kolei każdy tryb posiada właściwość `size` wskazującą rozdzielczość ekranu wyrażoną w pikselach. Wiele ekranów obsługuje różne rozdzielczości. Na przykład wyświetlacz VGA może obsługiwać kilka różnych rozdzielczości, których liczba zależy od producenta ekranu. Zawsze dostępna jest przynajmniej jedna rozdzielczość, ale jeśli ekran udostępnia ich więcej, to użytkownikowi również powinieneś pozwolić na ich wybór. W tym celu odpowiednio przygotuj kod aplikacji.

Konfiguracja wyjścia wideo

Po pobraniu obiektu zewnętrznego ekranu z tablicy `[UIScreen screens]` sprawdź dostępne tryby i wybierz jeden z nich, który będzie następnie używany. Ogólnie rzecz biorąc, zawsze powinieneś wybierać ostatni element tablicy, udostępniając tym samym użytkownikowi największą możliwą rozdzielczość, ewentualnie możesz wybrać pierwszy — oznaczający najniższą rozdzielczość.

W celu rozpoczęcia przesyłania strumienia wideo musisz utworzyć egzemplarz `UIWindow` i dostosować jego wielkość do używanej rozdzielczości ekranu. W oknie trzeba umieścić nowy widok pozwalający na wyświetlenie treści. Kolejny krok to przypisanie okna zewnętrznemu ekranowi i uczynienie go podstawowym i widocznym. W ten sposób okno zostanie wyświetlone i przygotowane do użycia. Kiedy zakończysz pracę z oknem na ekranie zewnętrznym, ekran w urządzeniu ponownie ustaw jako podstawowy. Nie pomijaj tego kroku! Dla użytkownika nie ma nic bardziej irytującego niż przekonanie się, że drogie urządzenie iOS nie reaguje na dotknięcia ekranu.

```
self.outwindow = [[UIWindow alloc] initWithFrame:theFrame];
outwindow.screen = secondaryScreen;
[outwindow makeKeyAndVisible];
[delegate.view.window makeKeyAndVisible];
```

Dodanie obiektu DisplayLink

Obiekt `DisplayLink` to rodzaj licznika czasu, który jest zsynchronizowany z częstotliwością odświeżania ekranu. Częstotliwość odświeżania ramki możesz zmienić poprzez modyfikację właściwości `frameInterval`. Wartość domyślna tej właściwości wynosi 1. Ustawienie wyższej wartości spowoduje zmniejszenie częstotliwości odświeżania. Tak więc wartość 2 oznacza zmniejszenie o połowę częstotliwości odświeżania. Obiekt `DisplayLink` utwórz po podłączeniu ekranu do urządzenia. Klasa `UIScreen` implementuje metodę, która zwraca obiekt `DisplayLink` dla ekranu. Dla wymienionego obiektu możesz zdefiniować cel oraz wskazać wywoływany selektor.

Obiekt `DisplayLink` regularnie wywołuje zdefiniowaną metodę, pozwalając Ci w ten sposób na uaktualnienie treści wyświetlanej na ekranie. Zmniejszenie częstotliwości powoduje mniejsze obciążenie procesora, ale kosztem utraty niektórych ramek. Trzeba koniecznie pamiętać o tym koszcie, zwłaszcza w przypadku bezpośredniej manipulacji interfejsami wymagającymi wiele pracy ze strony procesora w celu udzielenia odpowiedzi na działania użytkownika.

W dalszej części rozdziału znajdziesz kod sposobu 14.7, w którym użyto najczęściej spotykanych trybów dla pętli działania, zapewniając tym samym najmniejsze opóźnienie. Po zakończeniu pracy z obiektem `DisplayLink` wywołaj metodę `invalidate`, która powoduje usunięcie obiektu z pętli działania.

Kompensacja overscanningu

Poprzez przypisanie wartości właściwości `overscanCompensation` klasa `UIScreen` pozwala na kompensację w przypadku utraty pikseli na krawędziach ekranu. Ta technika została przedstawiona w dokumentacji Apple; w zasadzie odpowiada sytuacji, gdy chcesz obciąć treść lub dopełnić ją czarną przestrzenią.

VIDEOkit

W sposobie 14.7 przedstawiłam `VIDEOkit`, czyli klienta ekranu zewnętrznego. Zawiera on wszystkie funkcje niezbędne do konfiguracji i użycia ekranu zewnętrznego podłączonego zarówno przewodowo, jak i bezprzewodowo. Monitorowanie ekranu włączasz poprzez wywołanie `startupWithDelegate:`. Przekaż podstawowy kontroler widoku, którego zadaniem będzie utworzenie treści na zewnętrznym ekranie.

Sposób 14.7. `VIDEOkit`

```
@interface VIDEOkit : NSObject
{
    UIImageView *baseView;
}
@property (nonatomic, weak) UIViewController *delegate;
@property (nonatomic, strong) UIWindow *outwindow;
@property (nonatomic, strong) CADisplayLink *displayLink;
+ (void) startupWithDelegate: (id) aDelegate;
@end

@implementation VIDEOkit
@synthesize delegate;
```

```
@synthesize outwindow, displayLink;

static VIDEOkit *sharedInstance = nil;

- (void) setupExternalScreen
{
    // Sprawdzenie w poszukiwaniu brakujących ekranów.
    if (!SCREEN_CONNECTED)
    {
        return;
    }

    // Konfiguracja ekranu zewnętrznego.
    UIScreen *secondaryScreen = [[UIScreen screens] objectAtIndex:1];
    UIScreenMode *screenMode = [[secondaryScreen availableModes] lastObject];
    CGRect rect = CGRectMake(0.0f, 0.0f, screenMode.size.width, screenMode.size.height);

    // Utworzenie nowego okna.
    self.outwindow = [[UIWindow alloc] initWithFrame:CGRectZero];
    outwindow.screen = secondaryScreen;
    outwindow.screen.currentMode = screenMode; // Podziękowania dla Scotta Lawrence'a.
    [outwindow makeKeyAndVisible];
    outwindow.frame = rect;

    // Dodanie do okna podstawowego widoku.
    baseView = [[UIImageView alloc] initWithFrame:rect];
    baseView.backgroundColor = [UIColor clearColor];
    [outwindow addSubview:baseView];

    // Ustawienie jako okno główne.
    [delegate.view.window makeKeyAndVisible];
}

- (void) updateScreen
{
    // Przerwanie, jeśli ekran został odłączony.
    if (!SCREEN_CONNECTED && outwindow)
    {
        self.outwindow = nil;
    }

    // (Ponowna) inicjalizacja, jeśli nie ma okna na ekranie zewnętrznym.
    if (SCREEN_CONNECTED && !outwindow)
    {
        [self setupExternalScreen];
    }

    // Przerwanie w przypadku napotkania dziwnego błędu.
    if (!self.outwindow)
    {
        return;
    }

    // Przeprowadzenie uaktualnienia.
    SAFE_PERFORM_WITH_ARG(delegate, @selector(updateExternalView:), baseView);
}

- (void) screenDidConnect: (NSNotification *) notification
{
    NSLog(@"Ekran został podłączony");
}
```

```

UIScreen *screen = [[UIScreen screens] lastObject];
if (displayLink)
{
    [displayLink removeFromRunLoop:[NSRunLoop currentRunLoop]
     forMode:NSRunLoopCommonModes];
    [displayLink invalidate];
    self.displayLink = nil;
}

// Sprawdzenie istnienia obiektu DisplayLink.
if (!displayLink)
{
    self.displayLink = [screen displayLinkWithTarget:self
                      selector:@selector(updateScreen)];
    [displayLink addToRunLoop:[NSRunLoop currentRunLoop]
     forMode:NSRunLoopCommonModes];
}
}

- (void) screenDidDisconnect: (NSNotification *) notification
{
    NSLog(@"Ekran został odłączony.");
    if (displayLink)
    {
        [displayLink removeFromRunLoop:[NSRunLoop currentRunLoop]
         forMode:NSRunLoopCommonModes];
        [displayLink invalidate];
        self.displayLink = nil;
    }
}

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

    // Obsługa utworzenia okna na ekranie zewnętrznym.
    if (SCREEN_CONNECTED)
    {
        [self screenDidConnect:nil];
    }

    // Rejestracja powiadomień informujących o podłączeniu lub odłączeniu ekranu.
    [[NSNotificationCenter defaultCenter]
     addObserver:self selector:@selector(screenDidConnect:)
     name:UIScreenDidConnectNotification object:nil];
    [[NSNotificationCenter defaultCenter]
     addObserver:self selector:@selector(screenDidDisconnect:)
     name:UIScreenDidDisconnectNotification object:nil];

    return self;
}

- (void) dealloc
{
    [self screenDidDisconnect:nil];
    self.outwindow = nil;
}

```

```
+ (VIDEOkit *) sharedInstance
{
    if (!sharedInstance)
    {
        sharedInstance = [[self alloc] init];
    }
    return sharedInstance;
}

+ (void) startupWithDelegate: (id) delegate
{
    [[self sharedInstance] setDelegate:delegate];
}
@end
```

Pobierz kod przedstawionego sposobu

Aby pobrać kod przedstawionego tutaj sposobu, przejdź na stronę <ftp://ftp.helion.pl/przyklady/ios5pp.zip>, lub jeśli pobrałeś obraz dysku zawierający wszystkie przykłady przedstawione w książce, to przejdź do katalogu przeznaczony dla rozdziału 14. i otwórz projekt dla tego sposobu.

Wewnętrzna metoda `init` rozpoczyna nasłuchiwanie zdarzeń informujących o podłączeniu lub odłączeniu ekranu, a następnie tworzy lub usuwa okno. Metoda delegata (`updateExternalView:`) jest wywoływana przez obiekt `DisplayLink`. Parametrem jest widok znajdujący się w zewnętrznym oknie; we wspomnianym widoku delegat może wyświetlić przygotowaną treść.

W omawianym przykładzie kontroler widoku przechowuje wartość koloru lokalnego, a następnie używa jej na ekranie zewnętrznym.

```
- (void) updateExternalView: (UIImageView *) aView
{
    aView.backgroundColor = color;
}

- (void) action: (id) sender
{
    color = [UIColor randomColor];
}
```

Każde naciśnięcie przycisku powoduje wygenerowanie przez kontroler widoku nowego koloru. Kiedy `VIDEOkit` nakazuje kontrolerowi uaktualnienie zewnętrznego widoku, wspomniany kolor jest używany jako kolor tła. Możesz się przekonać, że tło na ekranie zewnętrznym jest natychmiast uaktualniane nowo wygenerowanym kolorem.

I jeszcze jedno: sprawdzenie dostępnej ilości wolnego miejsca na dysku

Klasa `NSFileManager` pozwala na sprawdzenie wolnej ilości miejsca pozostałej w urządzeniu iOS, jak również całkowitej ilości miejsca na dysku dostarczanej przez urządzenie iOS. W listingu 14.1 zademonstrowałam, jak sprawdzić obie wartości i wyświetlić je w postaci łatwego w odczycie ciągu tekstowego. Zwrócone wartości podają ilość miejsca wyrażoną w bajtach.

Listing 14.1. Pobranie informacji o wielkości systemu plików i pozostałej ilości wolnego miejsca

```

- (NSString *) commaFormattedStringWithLongLong: (long long) num
{
    // Wygenerowanie odpowiednio sformatowanego ciągu tekstowego.
    // Alternatywnym rozwiązaniem jest użycie egzemplarza NSNumberFormatter.
    if (num < 1000)
    {
        return [NSString stringWithFormat:@"%d", num];
    }
    return [[self commasForNumber:num/1000]
            stringByAppendingFormat:@"%03d", (num % 1000)];
}

- (void) action: (UIBarButtonItem *) bbi
{
    NSFileManager *fm = [NSFileManager defaultManager];
    NSDictionary *fattributes = [fm fileSystemAttributesAtPath:NSHomeDirectory()];
    NSLog(@"Całkowita ilość miejsca na dysku: %@",
          [self commaFormattedStringWithLongLong:[fattributes
            objectForKey:NSFileSystemSize] longLongValue]);
    NSLog(@"Wolna ilość miejsca na dysku: %@",
          [self commasForNumber:[fattributes
            objectForKey:NSFileSystemFreeSize] longLongValue]);
}

```

Podsumowanie

W tym rozdziale przedstawiłam podstawowe sposoby interakcji z urządzeniem iPhone. Dowiedziałeś się, jak pobrać informacje dotyczące urządzenia, sprawdzić poziom naładowania baterii oraz nasłuchiwać zdarzeń czujnika zbliżeniowego. Poznałeś też sposoby rozróżniania urządzeń iPod touch, iPhone i iPad oraz określania modelu urządzenia, w którym została uruchomiona aplikacja. Przedstawiłam przyspieszeniomierz i jego użycie w kilku przykładach, począwszy od prostego „wyszukania” aż po bardziej skomplikowany algorytm wykrywania gestu wstrząśnięcia. Poznałeś także technologię Core Motion i dowiedziałeś się, jak tworzyć bloki pozwalające na udzielanie odpowiedzi w czasie rzeczywistym na zdarzenia urządzenia. Wreszcie omówiłam temat dodawania obsługi ekranów zewnętrznych w aplikacjach. Zanim przejdziesz dalej, poświęć chwilę i przemyśl kilka kwestii dotyczących sposobów przedstawionych w rozdziale.

- Przyspieszeniomierz w iPhone’ie stanowi uzupełnienie oferowanego przez urządzenie interfejsu dotykowego. Dane przyspieszeniomierza możesz wykorzystać w celu rozbudowy interakcji z użytkownikiem poza „dotknij tutaj” i umożliwić aplikacji reakcję na obrót urządzeniem.
- Niskiego poziomu wywołania mogą być przyjazne dla aplikacji zgłaszanej do sklepu iTunes App Store. Nie zależą od API firmy Apple, które może być zmienione w aktualnym wydaniu oprogramowania typu firmware. Wywołania systemu UNIX mogą być kłopotliwe, ale wiele z nich jest w pełni obsługiwanych przez urządzenia z rodziny iOS.
- Nie zapominaj o ograniczeniach urządzeń. Przed przeprowadzeniem operacji wymagającej intensywnego przetwarzania pliku sprawdź ilość wolnej pamięci w urządzeniu. Podobnie przed operacją wymagającą użycia pełnej mocy procesora sprawdź, czy urządzenie jest podłączone do źródła zasilania.

- Wykorzystaj technologię Core Motion. Dostarczane przez nią dane w czasie rzeczywistym są podstawą dla wielu interesujących aplikacji na platformie iOS.
- Skoro technologia AirPlay pozwala na pozbycie się przewodów łączących urządzenie iOS z zewnętrznym ekranem, masz możliwość użycia wyjścia wideo w celu utworzenia jeszcze bardziej ekscytujących projektów, niż wcześniej sądziłeś. Technologia AirPlay i zewnętrzne ekrany mogą pomóc w przekształceniu urządzenia iOS na zdalnego pilota do gier i narzędzi wyświetlających treść na dużym ekranie i kontrolujących tę treść za pomocą mniejszego urządzenia iOS.
- Podczas zgłaszania aplikacji do sklepu iTunes App Store używaj pliku *Info.plist* w celu określenia funkcji urządzenia wymaganych dla danej aplikacji. iTunes używa wspomnianej listy, ustalając, czy dana aplikacja może zostać pobrana w konkretnym urządzeniu i czy będzie w nim prawidłowo działała.

Skorowidz

A

- adres IP, 681
- akcje, 209
- akredytacja, 57
- album zdjęć, 351, 356
- algorytm Canny, 387, 390
- alokacja
 - obiektu, 114
 - pamięci, 88
- analiza
 - etykiety, 186
 - kodu, 195
 - próbki rastrowej, 392
- analizator statyczny, 195
- animacja, 338
 - reakcji przycisku, 459
 - UIView, 338
 - UIImageView, 346
 - warunkowa, 340
 - widoku, 346
- ANSI C, 24
- API Cocoa, 84
- API SDK, 180
- aplikacja Google Mobile, 649
- aplikacje, 159
- aplikacje iOS, 60
- ARC, Automated Reference Counting, 36, 103, 121, 135
- archiwa IPA, 73
- archiwizacja interfejsu, 423
- atrybuty
 - komórki, 556
 - tekstu, 526, 576

- automatyczna zmiana widoku, 263
- wielkości, 261, 264

B

- bazowa wersja SDK, 203
- biblioteka
 - Cocoa Touch, 37
 - Media, 269
- biblioteki
 - obiektów, 171
 - szablonów plików, 455
- bitmapa ARGB, 389
- blok completion, 340, 346
- bloki, 81, 117, 121, 340
- bloki animacji, 345
- błędne dopasowanie sekcji, 574
- błędy leksykalne, 128
- buforowanie, 192

C

- certyfiikat, 54, 203
- CF, Core Foundation, 134
- ciągi tekstowe, 98, 142
 - konwersja na liczbę, 146
 - konwersja na tablicę, 144
 - modyfikowanie, 146
 - odczyt z pliku, 144
 - porównywanie, 146
 - wielkość znaków, 146
 - wyszukiwanie, 145
 - zakres, 145

- zapis do pliku, 143
- żądanie podciągow, 145
- cofnięcie
 - operacji, 426, 510
 - poprzez potrząśnięcie, 429
- Core Animation, 344
- Core Data, 597
- Core Image, 371
- Core Location, 43
- Core Motion, 43
- CoreMedia, 372
- CoreVideo, 372
- cykl życiowy obiektu, 102
- czcionki, 519
- czujnik zbliżeniowy, 648

D

- dane, 45
 - uwierzytelniające, 697
 - wejściowe, 502
- daty, 147, 593
- definiowanie
 - bloku, 118
 - protokołu, 138
- deklarowanie
 - interfejsu, 86
 - klasy, 87
 - metod, 93
 - odniesienia do bloku, 119
 - właściwości, 106
 - wywołań zwrotnych, 140
- delegat
 - aplikacji, 63
 - komunikatu, 618

- delegowanie, 77
 - detektor danych, 517
 - dodawanie
 - akcji, 249
 - animacji do kontrolek, 459
 - dźwięku kliknięcia, 509
 - elementów paska, 485
 - elementów widoku, 495
 - etykiet, 249
 - filtru, 379
 - komórek, 568
 - kontrolki segmentowanej, 277
 - metod, 98
 - obiektów, 601
 - obsługi klawiatury, 505
 - outletów, 250
 - plików, 247
 - podwidoku, 317
 - przycisków, 453
 - suwaka, 291
 - szablonów, 266
 - widoków, 249, 257
 - własnych czcionek, 520
 - zmiennych egzemplarza, 98
 - odbicia, 347
 - dokumentacja, 180
 - dokumenty PDF, 398
 - dostęp do
 - aparatu, 371
 - zasobów, 362
 - danych, 45
 - danych rastrowych, 392
 - informacji, 104
 - obiektów zbioru, 151
 - pliku, 153
 - podciągu tekstowego, 144
 - sieci, 676
 - słownika, 150
 - ścieżki indeksu, 605
 - tablicy, 149
 - wartości boolowskich, 560
 - właściwości, 111
 - zdjęć, 356
 - dostępność
 - czujników, 657
 - witryny, 685
 - dotknięcia, 405
 - drzewo
 - hierarchii widoku, 315
 - nawigacji, 311
 - przetwarzania, 707, 710
 - wyników, 709
 - duży tekst, 543
 - dyrektywa
 - #import, 87
 - #include, 87
 - @synthesize, 109
 - dystrybucja, 208
 - tymczasowa, 211
 - tymczasowa OTA, 213
 - dziedziczenie
 - metod, 93
 - wielokrotne, 156
 - dzienniki zdarzeń, 197
 - dźwięk, 42
 - dźwięk kliknięcia, 509
 - dźwięki systemowe, 637
- ## E
- edycja
 - danych, 610
 - tabeli, 564, 610
 - widoku, 171
 - edytor
 - interaktywny, 70
 - tekstów, 509
 - Xcode, 599
 - efekt odbicia, 345
 - egzemplarz
 - MKMapView, 221
 - MPMoviePlayerController, 229
 - NSMutableURLRequest, 690
 - NSURLCredential, 698
 - UIActionSheet, 630
 - UIActivityIndicator, 624
 - UIActivityIndicatorView, 621
 - UIAlertView, 319
 - UIButton, 222
 - UIDatePicker, 590
 - UIImage, 457
 - UIImagePickerController, 360, 370, 402
 - UIImageView, 220
 - UILabel, 220
 - UIPi, 585
 - UIPickerView, 586
 - UIScrollView, 221, 402, 478
 - UISegmentedControl, 222
 - UITableView, 548
 - UITableViewSeparatorView, 314
 - UITextView, 517
 - UIViewController, 258, 282
 - UIWebView, 517
 - ekran, 41
 - ekrany zewnętrzne, 666
 - EXIF, 376
- ## F
- fazy dotknięcia, 406
 - filtr Core Image, 379
 - filtrowanie tekstu, 511
 - flesz, 365
 - format EXIF, 376
 - formatowanie daty, 592
 - framework, 59
 - AVFoundation, 371
 - Carbon, 114
 - Cocoa, 25, 115
 - Cocoa Touch, 115
 - Core Data, 597
 - Core Foundation, 114
 - Core Graphics, 487
 - Core Motion, 656
 - Core Text, 521
 - Foundation, 147, 169
 - Graphics, 169
 - MediaPlayer, 229, 637
 - MessageUI, 228
 - Quartz 2D, 487
 - Quartz Core, 53
 - System Configuration, 676
 - Twitter, 706
 - UIKit, 169, 308
 - funkcja
 - CFSHOW, 81
 - CFUUIDCreateString(), 176
 - CGPointApplyAffineTransform(), 419

CGRectContainsRect(), 331
 CGRectMake(), 326
 MPVolumeSettingsAlertIsVisible(), 640
 NSLog(), 81, 98
 NSStringFrom(), 100
 objc_unretainedObject(), 134
 printf(), 81, 88, 99
 SCNetworkReachability
 CreateWithAddress(), 685
 sizeof(), 88
 sprintf(), 107
 Storyboard, 235
 UIApplicationMain(), 62, 179
 funkcje
 cofnij i przywróć, 612
 KVC, 111
 KVO, 112
 wyszukiwania, 578
 funkcjonalność UITableViewCell, 561

G

gałka suwaka, 465
 Gargabe Collector, 114
 generowanie
 plików, 600
 treści, 533
 geometria
 obrazu, 377
 widoku, 326
 gest, 408
 długiego naciśnięcia, 409
 machnięcia, 408, 445
 machnięcia w komórce, 568
 obrotu, 409
 przeciągania, 445
 przesunięcia, 409
 stuknięcia, 408
 uszczyknięcia, 408
 wstrząśnięcia, 663
 git, 31
 GitHub, 31
 GPS, 43
 grafika rastrowa, 418

grawitacja, 657
 grupa
 Editor, 165
 View, 165
 grupy sekcji, 606

H

harmonogram powiadomień
 lokalnych, 635
 hermetyzacja, 105
 hierarchia
 klas, 97
 widoków, 314

I

IB, Interface Builder, 36
 identyfikator aplikacji, 56, 68, 201
 ikona, 69
 implementacja
 metody, 93
 przekazywania wiadomości, 155
 indeks sekcji, 573
 informacje o
 komputerze, 681
 obiekcje, 269
 położeniu, 660
 rozdzielczości ekranu, 667
 stanie, 421
 urządzeniu, 646
 inspektor
 atrybutów, 171
 tożsamości, 238
 wielkości, 261
 interakcja dotknięć, 441
 Interface Builder, 72
 interfejs
 hybrydowy, 256
 użytkownika, 172, 219, 235
 iOS Developer Program, 54
 iOS Developer University
 Program, 24
 iOS SDK, 24
 IPTC, 376

J

język Objective-C, 85

K

kamera, 42
 karta Info, 202
 katalog, 67
 Applications, 176
 Application Support, 73
 Backup, 176
 DerivedData, 176
 Documents, 73, 153, 354
 Library, 73, 354, 424
 lproj, 84
 Preferences, 72
 Provisioning Profiles, 176
 Screenshots, 176
 Snapshots, 176
 Software Images, 176
 tmp, 354
 kategorie, 137
 kąat względny urządzenia, 652
 klasa, 86, 97
 AccelerometerHelper, 665
 AVAudioPlayer, 637
 BigTextView, 544
 BookController, 289
 CIImage, 374
 DetailViewController, 301
 DownloadHelper, 695
 DragView, 424
 GKPeerPickerController, 229
 MergedTableController, 78
 MPMediaPickerController, 229
 NSBundle, 153
 NSCoder, 424
 NSEr, 124
 NSFileManager, 153, 671
 NSJSONSerialization, 714
 NSMutableArray, 92
 NSMutableString, 146
 NSNotificationCenter, 80
 NSNumber, 147
 NSOperation, 706
 NSOperationQueue, 684, 706

- NSRegularExpression, 516
- NSXMLParser, 707
- UIActionSheet, 221
- UIActivityIndicatorView, 622
- UIAlertView, 221
- UIApplication, 136
- UIBarButtonItem, 79
- UIControl, 79, 449
- UIDatePicker, 592
- UIDevice, 136, 509, 649
- UIDocumentInteractionController, 229
- UIGestureRecognizer, 439
- UIImage, 353
- UIImagePicker, 228
- UIImagePickerController, 355, 365
- UILayoutContainerView, 313
- UINavigationController, 225, 271, 311
- UINavigationControllerItem, 274
- UIPageControl, 223, 479
- UIPageViewController, 285
- UIPickerView, 223
- UIPopoverController, 77
- UIProgressView, 622
- UIResponder, 405
- UIScreen, 234, 666
- UISegmentedControl, 466
- UISlider, 222, 461
- UISplitViewController, 77, 272
- UIStepper, 472
- UISwitch, 222
- UITabBarController, 226, 292
- UITableView, 77
- UITableViewController, 548
- UITextChecker, 517
- UITextView, 75, 220
- UIView, 66, 219, 317, 349
- UIViewController, 63, 76, 225
- UIWebView, 220, 400
- UIWindow, 219
- XMLParser, 707
- klasy
 - Foundation, 142
 - widoku, 76
- klawiatura, 233, 252, 493
- klawiatura sprzętowa, 497
- klient ekranu zewnętrznego, 668
- klucz
 - CFBundleIconFiles, 71
 - ścieżki sekcji, 605
 - UIApplicationLaunchOptionsURLKey, 352
 - UIFileSharingEnabled, 73
 - UIInterfaceOrientation, 259
 - UIPrerenderedIcon, 71
 - UIRequiredDeviceCapabilities, 644
- kod
 - konwertera, 253
 - niezgodny z ARC, 131
- kody języków, 517
- kolejki, 684
- kolekcje, 149
 - zapis do pliku, 152
 - zrządzanie pamięcią, 151
- kolor
 - komórki, 552
 - tabeli, 555
 - tła tabeli, 553
- komórki, 549, 560
- kompensacja overscanningu, 668
- kompilacja, 204
- kompilator
 - LLVM 3.0, 128
 - Xcode, 59
- komponenty aplikacji, 67
- komunikat, 617
 - bez przycisków, 623
 - dźwiękowy, 636, 638
 - modalny, 625
- konsola, 188, 199
- kontekst rastrowy, 389
- kontroler, 75
 - BookController, 289
 - DetailViewController, 301
 - GKPeerPickerController, 229
 - MergedTableController, 78
 - MPMediaPickerController, 229
 - PasswordController, 702
 - Root View Controller, 236
 - UIImagePickerController, 355, 365
- UILayoutContainerView, 313
- UINavigationController, 76, 225, 271, 311
- UIPageViewController, 285
- UISplitViewController, 77, 272
- UITabBarController, 76, 226, 292
- UITableViewController, 548
- kontrolery
 - książki adresowej, 228
 - modalne, 275, 283
 - nawigacyjne, 236, 272, 281
 - opcji wyszukiwania, 575
 - paska kart, 293
 - tworzenia wiadomości, 369
 - typu Popover, 227
 - widoku, 65, 77, 171, 224, 241
 - głównego, 301
 - podzielonego, 226, 299
 - strony, 227, 285
 - tabeli, 227
- kontrolka, 222, 449
 - koloru, 468
 - segmentowana, 277, 466
 - suwaka, 475
- UIDatePicker, 592
- UIPageControl, 479
- UIPickerView, 588, 595
- UITextField, 489
 - wskaźnika strony, 478
- kontrolki Picker, 223, 233
- konwersja
 - na CF, 135
 - na HSB, 394
 - plików, 181
- konwerter temperatur, 247, 253
- kopiowanie obiektów, 269
- kropka, 105
- krzywe Catmull-Rom, 433
- KVC, Key-Value Coding, 111
- KVO, Key-Value Observing, 81
- kwalifikator
 - __autoreleasing, 124
 - __block, 120, 127
 - __strong, 124
 - __unsafe_unretained, 124
 - __weak, 124

kwalifikatory

- ARC, 110
- MRR, 110
- niepodzielności, 111
- właściwości, 110

L

leniwe wczytywanie, 108

licencja programisty, 197

liczby, 147

licznik

- czasu, 148, 655
- NSTimer, 381
- odniesień, 125
- użycia, 90, 100, 132

linie

- springs, 261
- struts, 261

lista

- akcji, 210
- Devices, 197
- zainstalowanych aplikacji, 199

lokalizacja, 43

Ł

łączenie przycisków z akcjami, 455

M

magazyn danych, 510

mapowanie współrzędnych, 382

mechanizm

- cel-akcja, 79
- delegowania, 77, 574
- KVO, 81
- usuwania nieużytków, 114
- zarządzania pamięcią, 276

menedżer

- operacji cofnięcia, 430
- plików, 153

menu, 629

- Deployment Target, 204
- Scheme, 209

metadane, 215

metoda

- deklarowanie, 93
- dziedziczenie, 93
- implementacja, 93
- nadpisywanie, 98
- ukrywanie, 208

metoda

- aboveSubview, 317
- accelerometer:didAccelerate:, 650
- alertView:clickedButtonAtIndex:, 619
- allApplicationSubviews(), 317
- alloc, 89
- allSubviews(), 316
- application:didFinishLaunchingWithOptions:, 63
- application:didFinishLaunchingWithOptions:, 352
- applicationDidBecomeActive:, 63
- applicationDidFinishLaunching:, 430
- applicationDidReceiveMemoryWarning:, 63
- applicationWillEnterForeground:, 63
- applicationWillEnterBackground:, 63
- applicationWillResignActive:, 63
- arrayWithObjects:, 106
- belowSubview, 317
- CFRunLoop(), 626
- checkUndoAndUpdateNavBar:, 429
- convert:, 251
- dealloc, 116
- dequeueReusableCellWithIdentifier:, 552
- didAddSubview:, 318
- didMoveToSuperview:, 318
- didMoveToWindow:, 318
- didRotateFromInterfaceOrientation:, 264
- done:, 283

download:withTargetPath:

- withDelegate:, 696
- drawRect:, 431, 525
- forwardInvocation:, 155
- getIPAddressForHost:, 683
- helloController, 179
- hostAvailable:, 685
- hostname, 683
- imageWithBits:withSize, 391
- imageWithContentsOfFile:, 353
- isCameraDeviceAvailable:, 365
- isDescendantOfView:, 317
- isKindOfClass:, 156
- isValidJSONObject, 714
- keyboardDidShow:, 500
- leaveEditMode, 567
- loadView, 65
- localIPAddress, 683
- numberOfComponentsInPickerView, 586
- numberOfSectionsInTableView, 552, 572
- pathToView(), 317
- performFetch, 609
- pickerView:didSelectRow:inComponent, 586
- pickerView:numberOfRowsInComponent:, 586
- pickerView:rowHeightForComponent:, 588
- pickerView:titleForRow:forComponent, 586
- pushViewController:animated:, 280
- release, 101
- respondsToSelector:, 156
- say:, 629
- scanString:, 530
- scrollViewDidEndDecelerating:, 583
- scrollViewDidScroll:, 583
- searchBar:textDidChange:, 609
- segmentAction:, 278
- setAnimationDelegate:, 339
- setDelegate:, 78
- setFontFace:, 79
- setNeedsDisplay, 431

metoda

setObject:forKey:, 297
 setPosition:, 430
 shouldAutorotateToInterfaceOrientation:
 rientation:, 65, 258
 storyboardWithName:bundle:
 256
 stringWithFormat:, 107
 subviews, 316
 tableView:
 tableView:cellForRowAtIndex
 Path:, 552
 tableView:didSelectRow
 AtIndexPath:, 78
 tableView:numberOfRowsIn
 Section, 572
 tableView:numberOfRowsIn
 Section:, 552
 takePicture, 370
 textFieldShouldReturn:, 490
 updateTransformWithOffset:
 414
 useSideBySide:, 290
 viewDidAppear:, 65
 viewDidLoad, 65, 325
 viewWillAppear:, 65
 viewWithTag:, 320
 whatsmyipdotcom, 683
 willMoveToSuperview:, 318
 willMoveToWindow:, 318
 willRemoveSubview, 318

metody

akcesorów, 104, 106, 108
 delegata, 63, 578
 dla podwidoków, 316
 inicjalizacyjne, 88
 klasy, 95
 klasy UIImage, 353
 narzędziowe, 331
 niezadeklarowane, 91
 obsługi dotknięć, 407
 publiczne, 104
 typu getter, 106
 typu setter, 106
 wygodne, 102
 wywoływane przez przyciski,
 243

źródła danych, 82, 584
 miejsce dotknięcia, 417, 419
 migracja do ARC, 128
 modalny kontroler widoku, 281
 model MVC, 406
 modele, 75, 82, 598
 moduł Interface Builder, 248, 268
 modyfikacja kontrolera widoku, 302
 monitorowanie zmian w
 połączeniu, 679
 możliwości urządzenia, 645
 MRC, Manual Reference
 Counting, 89
 MRR, Manual Retain Release, 88,
 101, 106, 112
 MVC, Model-View-Controller, 74,
 406

N

nadpisywanie metod, 98
 nagłówek sekcji, 573
 nakładki, 371
 narzędzie
 Instruments, 190, 191
 Interface Builder, 72
 sqlite3, 602
 nasłuchiwanie powiadomień, 80
 nawias kwadratowy, 91
 nawigacja pomiędzy kontrolerami,
 279
 nazwy widoków, 321
 niepodzielność, 110
 numer UDID, 56

O

obiekt, 86
 DisplayLink, 668, 671
 JSON, 714
 NSData, 153
 NSString, 143
 NSURL, 152
 NSURLResponse, 690
 ObjectCache, 193
 UIAlertView, 617
 UIApplication, 83

UIImage, 366
 UIPickerView, 585
 UISwitch, 471
 UITableView, 562
 obiekty
 Core Foundation, 132
 powiązane, 321
 obracające się kółko, 621
 obrazy, 351
 rastrowe, 387
 rozruchowe, 69
 obrót
 interfejsu, 258
 urządzenia, 656
 widoku, 343
 obsługa
 aparatu, 379
 bloku, 657
 dotknięć, 405
 iPada, 358
 klawiatury, 505
 okna Popover, 245
 opcji cofnij, 425
 plików, 711
 poczty, 367
 połączeń HSDPA, 52
 proxy wyglądu, 466
 układu tabeli, 548
 wibracji, 638
 wielu dotknięć, 408, 441
 oczyszczanie
 pamięci, 53
 projektu, 208
 odbicie, 347
 odtwarzanie dźwięków, 638
 ograniczenia
 aplikacji, 51
 dostępu do danych, 45
 energetyczne, 50
 interakcji, 50
 pamięci, 46
 platformy, 45
 przestrzeni na dane, 45
 ruchu, 415
 SDK, 52
 symulatora, 38

okno

- edytora, 168
 - Interface Builder, 170
 - Organizer, 196
 - typu Popover, 244, 633
- opakowanie implementacji, 286
- opcja cofnij, 425
- OpenGL ES, 44
- operator kolekcji, 111
- opóźnienia, 639
- organizacja widoków, 238
- orientacja urządzenia, 259, 653
- osadzanie obrazu, 401

P

paczka aplikacji, application bundle, 351

pakiet

- Cocoa Touch, 37
- DTrace, 36
- SKD iOS, 37

pamięć, 46, 88, 193

panel

- Debug Navigator, 188
- dzienników zdarzeń, 167
- inspektora, 166
- nawigacyjny, 166
- problemów, 167
- punktów kontrolnych, 167
- symboli, 166
- usuwania błędów, 167, 185
- Utilities, 239
- wyszukiwania, 166

panele narzędziowe, 167

para klucz-wartość, 81

pasek

- kart, 223, 232, 292
- narzędziowy, 484, 487
- nawigacyjny, 223, 232, 239
- postępu, 224, 621
- stanu, 230
- wyszukiwania, 575, 576

pęk kluczy, 702

piaskownica aplikacji, sandbox, 74, 176, 351

plakietki, 636

platforma

- Cocoa, 25
- Mac, 27

plik

- AppDelegate.h, 169
- AppDelegate.m, 169
- Default.png, 69
- Hello_ViewController.h, 245
- HelloViewController.xib, 182
- Info.plist, 68, 84, 169
- main.c, 178
- main.m, 30, 61, 169
- MainStoryboard_iPad.storyboard, 169, 244
- MainStoryboard_iPhone.storyboard, 169, 248
- manifestu, 213
- ViewController.h, 169
- ViewController.m, 169

pliki

- .h, 30
- .ipa, 73
- .m, 30
- .storyboard, 61, 72, 170, 235
- .xcdatamodel, 598
- .xib, 72, 256
- AppDelegate, 255
- dzienników, 198
- graficzne, 69
- implementacji, 61
- interfejsu, 60
- kodu źródłowego, 60
- modelu, 598
- PNG, 46
- wykonywalne, 67

pływający wskaźnik postępu, 625

pobieranie asynchroniczne, 690, 691

pobieranie

- danych, 692
- obrazu, 360
- synchroniczne, 687

podgląd, 375

podłączanie urządzeń, 40

podpisywanie aplikacji, 205

podział na strony, 531

pole magnetyczne, 657

pole tekstowe, 233

polecenie backtrace, 188

połączenie sieciowe, 676

położenie urządzenia, 656

portal akredytacyjny, 53

potrząsanie urządzeniem, 429

powiadomienia, notifications, 80, 493

powiadomienia lokalne, 635

poziom

- głośności, 640
- naładowania baterii, 647

predykaty, 616

prefiks set, 106

procedura rozpoznawania gestów, 439

procesor, 44

profil akredytacyjny programisty, 200

program

- Developer Enterprise Program, 35
- Developer University Program, 35
- Instruments, 36
- Interface Builder, 36
- Online Developer Program, 33
- Shark, 37
- Simulator, 36
- Standard Developer Program, 34
- Xcode, 25, 36, 59

programowanie zorientowane

obiektowo, 74

protokół, 138

delegata, 78

NSURLConnectionDownload

Delegate, 691

UITextField, 506

UITextInputTraits, 491

proxy wyglądu, 465

przechowywanie danych, 698

przeciąganie z widoku, 444

przejsięcie pomiędzy

- kontrolerami widoków, 308
- klawiaturami, 498

przejście segue, 304
 przekazywanie
 danych do serwera, 703
 wiadomości, 154, 157
 zdarzeń, 471
 przekształcanie widoku, 336
 przekształcenia 3D, 327
 przełączanie pomiędzy aparatami, 375
 przełączniki, 471
 przenoszenie
 danych, 616
 obiektów, 269
 widoków, 264, 335
 przestrzeń robocza, 162
 przesuwanie widoku, 411
 przetwarzanie
 obrazów, 388–391
 tekstu, 527
 przewijanie
 menu, 631
 stron, 291, 480
 przezroczystość, 337
 przezroczystość widoku, 341
 przycisk
 Add Contact, 452
 Debugger, 184
 Detail Disclosure, 452
 Info Dark, 452
 Info Light, 452
 Organizer, 165
 Rounded Rect, 452
 wstecz, 281
 przyciski typu disclosure, 303, 562
 przypisywanie właściwości, 110
 przyspieszenie, 657
 przyspieszeniomierz, 649, 653
 przytrzymywanie obiektu, 103
 puła zwalniana automatycznie, 62, 101, 127
 punkt kontrolny, 184, 187
 punkt środkowy, 330

Q

Quartz Core, 371

R

ramki, 326
 refaktoring kodu, 130
 reguły dotyczące ARC, 131
 rejestracja, 35
 rejestrowanie
 identyfikatorów aplikacji, 56
 urządzeń, 55
 repozytoria git, 31
 rodzaje
 gestów, 408
 komunikatów, 620
 kontrolek, 449
 powiadomień, 80
 przytrzymania, 110
 rozdzielczość ekranu, 229, 328, 667
 rozmiar ramki, 329
 rozpoznawanie gestów, 408, 412, 439
 rozszerzenia klas, 137
 rysowanie
 płynne, 432
 po ekranie, 431
 w PDF, 398
 rzutowanie
 bridge_transfer, 133
 obiektów, 320
 typów, 115, 132

S

schematy, 209
 SDK, software development kit, 33
 SDK iOS, 36
 sekcje, 571
 selektory, 87
 serializacja JSON, 714
 serwis Provisioning Portal, 53
 sieć Wi-Fi, 675
 silne wiązanie, 125
 skalowanie obrazu, 330
 składnia metody, 91
 słownik modyfikowalny, 297
 słowniki, 150
 słowo kluczowe, 87
 @optional, 140
 @required, 140

sortowanie tabeli, 569
 specyfikatory formatu, 99
 sprawdzanie
 dostępności, 677
 aparatów, 372
 zasobu, 685
 połączenia sieciowego, 676
 zgodności, 205
 stan
 komórki, 557
 kontrolki, 558
 paska kart, 295
 połączenia z siecią, 675
 przycisku, 454
 urządzenia, 197
 standard 802.11n, 52
 standardy metadanych, 376
 status First Responder, 430
 steppery, 471
 strona przewijana, 402
 struktura CGRect, 330
 styl modalny, 240
 suwak, 291, 461
 suwak z gwiazdkami, 473
 symbol @, 87
 symulator, 38, 173, 175
 szablony aplikacji, 159
 szybkie wyliczenie, 96

Ś

ścieżka dostępu do pliku, 153
 śledzenie
 Core Location, 361
 dotknięć, 431, 443, 470
 komunikatów, 83
 wsteczne, 188

T

tabela, 547, 551
 ustawień, 583
 zgrupowana, 583
 tablica, 149
 kontrolerów, 297, 304
 subviews, 315

tablice

- konwersja na tekst, 150
- sprawdzanie, 150

tarcza dotykowa, 476

technologia

- AirPlay, 220, 673
- ARC, 104, 121, 126, 135
- Core Animation, 343
- Core Motion, 656
- Core Text, 532
- MRR, 106, 112
- Quartz, 127

tekst, 489

tekst przycisku, 458

telefonია, 43

testowanie

- aplikacji, 174, 359
- miejsca dotknięcia, 417

tożsamość profilu, 203

tworzenie

- adresu URL, 152
- animacji, 338, 340
- aplikacji, 115, 161
- ciągu tekstowego, 142, 522
- drzewa przetwarzania, 707
- dynamicznej gałki suwaka, 462
- egzemplarza

- UIAlertView, 628

- UIControl, 470

- UIDatePicker, 592

- UIPickerView, 586

- indeksu sekcji, 573

- interfejsu, 235

- kategorii, 137

- komórki, 554

- komunikatu, 617, 628

- komunikatu dźwiękowego, 638

- kontekstu Core Data, 600

- kontrolki, 449

kontrolera

- danych wejściowych, 507
- paska kart, 293
- widoku podzielonego, 298

- kontrolki koloru, 468

- kontrolki segmentowanej, 466

- konwertera temperatury, 247

- menedżera nazw, 323

- menedżera opcji cofnij, 425

- menu, 276

- metod typu getter i setter, 108

- metod źródła danych, 576

- miniatur, 395

- nagłówka sekcji, 573

- nowego obrazu, 399

- nowego projektu, 159, 162, 236

- obiektów, 88, 100, 113

- obiektu JSON, 714

- odbicia widoku, 347

- opakowania, 286

- outletu, 251

- pakietu, 212

- paska narzędziowego, 484

- pliku manifestu, 213

- plywającego wskaźnika

- postępu, 625

- połączeń, 245

- pól tekstowych, 501

- przejścia, 306

- przewijanej strony, 402

- przycisków, 253, 454

- sekcji, 571

- serwera, 711

- sesji aparatu, 373

- słownika, 150

- suwaka z gwiazdkami, 472

- tabeli, 78, 548, 551

- z sekcjami, 606

- zgrupowanej, 557

- tablicy, 149

- tekstu z atrybutami, 526

- wiadomości, 368

- wiadomości e-mail, 228

- widoków tabel, 547

- widoków tekstowych, 503

- widoku komunikatu, 631

- właściwości, 107

- wygładzonych krzywych, 433

- wyrażeń, 514

- typ unichar, 143

typy

- komórek, 553

- zdarzeń, 451

- zmiennych, 120

- tytuły indeksu, 606

U

- uaktualnienie klas, 238

układ

- Core Text, 531

- obrazu, 378

- urządzenia, 377

- ukrywanie kodu, 208

- uruchamianie aplikacji, 173, 243

- urządzenia, 37, 47, 174

- podłączanie, 40

- rejestrwanie, 55

- zestawienie, 47

- usługa iCloud, 351

usuwanie

- błędów, 183

- komórek, 567

- komunikatu, 623, 632

- kontrolera, 273

- kontrolera modalnego, 283

- kwifikatorów, 124

- obiektów, 114, 604

- podwidoków, 318

- stron, 480

- widoków, 242

- zaznaczenia komórki, 557

- UTI, Uniform Type Identifier, 352

- uwierzytelnienie, 697

- uwierzytelnienie trwałe, 700

- użycie predykatów, 608

V

- VIDEOkit, 668

W

- wartości boolowskie, 560

wartość

- nil, 106, 573

- tag, 257, 319

- wczytywanie obrazu, 355

wiadomość

- description, 99

- release, 101, 116

- wibracje, 44, 638

- widok, 75, 238
 - Core Animation, 343
 - geometria, 326
 - hierarchia, 314
 - nazwy, 321
 - obrót, 343
 - przekształcenia, 336
 - przenoszenie, 334
 - przesuwanie, 411
 - wartości tag, 321
 - wyszukiwanie, 319
 - wywołania zwrotne, 318
 - zamiana, 342
 - zmiany przezroczystości, 341
 - widoki
 - danych wejściowych, 500, 504, 507
 - modalne, 282
 - podzielone, 298
 - przewijane, 400, 445
 - tekstu, 494
 - właściwości, 104
 - assign, 110
 - klasy UIDevice, 643
 - komunikatu, 620
 - książki, 285
 - pól tekstowych, 491
 - widoku, 326
 - właściwość
 - alpha, 337
 - batteryMonitoringEnabled, 648
 - cameraDevice, 365
 - cameraViewTransform, 371
 - countDownDuration, 592
 - date, 592
 - delegate, 490
 - font, 521
 - inputView, 501
 - orientation, 652
 - popoverController, 246
 - sectionIndexTitles, 606
 - setUserInteractionEnabled, 409
 - strong, 122
 - tableViewFooterView, 579
 - tableViewHeaderView, 579
 - titleLabel, 458
 - unsafe_unretained, 110
 - URLConnection, 696
 - urlString, 695
 - userInteractionEnabled, 337
 - weak, 110, 122
 - wprowadzanie danych, 500
 - wskaznik
 - aktywności sieciowej, 685
 - do obiektu, 92, 124
 - do rzeczywistych danych, 153
 - komunikatu, 636
 - strony, 478
 - współczynnik kompresji, 366
 - współrzędne widoku, 328
 - wstrząsanie urządzeniem, 661
 - wybór
 - elementów, 562
 - obrazu, 357
 - wyciek pamięci, 190
 - wyjście wideo, 667
 - wykonywanie zdjęć, 363, 371
 - wykrywanie
 - danych, 516
 - ekranu, 667
 - krawędzi, 387
 - okręgów, 435
 - połączeń, 676
 - twarży, 381–386
 - wyłączenie ARC, 129
 - wyrażenia regularne, 514
 - wyrazenie
 - device, 202
 - signing, 202
 - WYSIWYG, 265
 - wysyłanie
 - obrazów, 367
 - wiadomości, 88, 91
 - wyszukiwanie
 - błędnie zapisanych słów, 517
 - ciągów tekstowych, 518
 - obrazów, 351, 354
 - w tabeli, 575, 608
 - widoków, 319
 - wyświetlacz, 337
 - wyświetlanie
 - danych, 220
 - informacji, 98
 - kluczy, 151
 - komunikatu, 620
 - kontrolkek, 279, 567
 - liter, 542
 - menu, 629
 - obrazu, 400
 - tekstu, 631
 - na ścieżce, 536
 - na ścieżkach Béziera, 541
 - proporcjonalne, 541
 - w okręgu, 533
 - widoku modalnego, 281
 - wywołania zwrotne, 139
 - opcjonalne, 140
 - widoku, 318
 - wzorce, 514
 - wzorzec
 - MVC, 75
 - Singleton, 136, 153
- X**
- Xcode, 25, 162
 - XMP, 376
- Z**
- zamiana widoków, 342
 - zapytania do
 - bazy danych, 603
 - podwidoków, 315
 - zarządzanie
 - pamięcią, 100, 121, 151, 189
 - plikiem, 153
 - podwidokami, 317
 - zaznaczanie kolumn, 587
 - zbiory, 151
 - zbliżenie, 44
 - zdarzenia
 - kontrolkek, 450
 - przyśpieszeniomierza, 650
 - ruchu, 661
 - typu Touch Up, 471
 - zdjęcia, 363
 - zgłoszenie aplikacji, 215
 - zgodność
 - z ARC, 130
 - z protokołem, 141

zmiana

- kolejności komórek, 568
- odcienia obrazu, 461
- orientacji, 249, 258
- poziomu głośności, 640
- wielkości automatyczna, 261
- wielkości widoku, 497

zmienna

- error, 125
- strong, 127

zmiennie

- klasy, 104
- lokalne, 120

znaczniki

- HTML, 527
- pragma mark, 207

znak tyldy (~), 154

zrzut ekranu, 199, 397

zwalnianie pamięci, 89, 115

zwrot

- komórek, 572
- liczby rekordów, 576

Ż

źródło danych, 82

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄŻKI**
- 3. ZBIERAJ PROWIZJĘ**

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>



**ZAOSZCZĘDŹ CZAS
DZIĘKI POMOCNYM PRZYKŁADOM!**

iOS 5. Podręcznik programisty

Oto książka poświęcona programowaniu na platformie iOS na którą czekałeś!

iOS to jeden z najpopularniejszych systemów operacyjnych wykorzystywanych w urządzeniach mobilnych. Znajdziesz go we wszystkich sprzętach Apple – iPhone'ach, iPadach i iPodach. Różnorodność urządzeń gwarantuje Ci dużą liczbę użytkowników, ale z drugiej strony dostarcza wielu nowych wyzwań. Zastanawiasz się, jak poradzić sobie z codziennymi problemami pojawiającymi się w trakcie tworzenia aplikacji dla iOS?

W tej książce Erica Saduna – autorka bestsellerów i guru programowania na platformie iOS – umieścisz wszystkie informacje potrzebne do tego, abyś jak najszybciej zaczął tworzyć znakomite aplikacje dla urządzeń mobilnych na iOS 5. Znajdziesz tu gotowy do natychmiastowego użycia i łatwy w rozbudowie kod źródłowy, co pozwoli Ci uniknąć konieczności szukania własnych rozwiązań metodą prób i błędów. Przykłady przygotowane i przetestowane przez autorkę dotyczą praktycznie każdej dziedziny programowania na platformie iOS, od tworzenia interfejsu użytkownika, poprzez kontrolery widoku, gesty i dotknięcia, aż do obsługi sieci oraz kwestii bezpieczeństwa.

- Opanowanie iOS 5 SDK, podział języka programowania Objective-C oraz cyklu życiowego tworzenia oprogramowania na platformie iOS
- Projektowanie i dostosowanie do własnych potrzeb interfejsu użytkownika przy użyciu modułu Interface Builder i języka Objective-C
- Organizowanie aplikacji za pomocą kontrolerów widoku, widoków i zmiennych, w tym najnowszego kontrolera widoku strony oraz własnych kontenerów
- Maksymalne wykorzystanie dotknięć i gestów oraz tworzenie własnych procedur rozpoznawania gestów
- Praca z technologiami Core Image i Core Text
- Implementacja w pełni wyposażonych widoków tabel, włącznie z edycją komórek, zmianą ich kolejności oraz dostosowaniem własnych komórek
- Tworzenie magazynów zarządzanych danych, dodawanie, usuwanie, pobieranie i wyświetlanie danych
- Wyświetlanie komunikatów dla użytkownika w postaci okien dialogowych, pasków dostępu, powiadomień lokalnych i typu push, okien typu popover oraz polecenia ping
- Zdanie i obsługa informacji pochodzących od użytkowników
- Nawiązywanie połączenia z sieciami i usługami, obsługa uwierzytelniania oraz zarządzanie pobieraniem danych
- Instalacja aplikacji w urządzeniach iOS, przekazywanie aplikacji testerom wersji beta oraz do sklepu iTunes App Store

Erica Sadun jest autorką bestsellerów oraz współautorką licznych książek o programowaniu. Opracowała dziesiątki aplikacji na platformę iOS, oferuje usługi konsultingowe w zakresie szybkiego tworzenia prototypów aplikacji. Jej artykuły są publikowane w wielu serwisach, między innymi Ars Technica, O'Reilly i LifeHacker. Obecnie regularnie publikuje w serwisie TUAW. Uzyskała tytuł doktora nauk informatycznych na Georgia Institute of Technology.

helion.pl
księgarnia
internetowa

Nr katalogowy: **13130**



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900
0 601 339900



Helion

Sprawdź najnowsze promocje:

🔗 <http://helion.pl/promocje>

Książki najchętniej czytane:

🔗 <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

🔗 <http://helion.pl/nowosci>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

sięgnij po **WIĘCEJ**



KOD KORZYŚCI

ISBN 978-83-246-5121-4



Cena 119,00 zł

Informatyka w najlepszym wydaniu

9 788324 651214