Kyle Richter
Joe Keeley

iOS Tajniki SDK

Biblioteka przydatnych narzędzi

Poznaj zaawansowane możliwości platformy iOS!

Tytuł oryginału: iOS Components and Frameworks: Understanding the Advanced Features of the iOS SDK

Tłumaczenie: Krzysztof Rychlicki-Kicior

ISBN: 978-83-246-9182-1

Authorized translation from the English language edition, entitled:
IOS COMPONENTS AND FRAMEWORKS: UNDERSTANDING
THE ADVANCED FEATURES OF THE IOS SDK; ISBN 0321856716;
by Kyle Richter; and by Joe Keeley; published by Pearson Education, Inc,
publishing as Addison Wesley.
Copyright © 2014 Pearson Education, Inc.

All rights reserved. No part of this book may by 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 © 2014.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiejkolwiek 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.

Autor oraz Wydawnictwo HELION dołożyli 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/iostaj.zip

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

Printed in Poland.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » Nasza społeczność

Spis treści

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

Kup książkę Poleć książkę

6

Obsługa bibliotek muzyki

Gdy Steve Jobs po raz pierwszy pokazał światu iPhone'a na konwencji Macworld w 2007 roku, został on ochrzczony mianem połączenia telefonu, iPoda i rewolucyjnego komunikatora internetowego. Wiele lat później, po części dzięki ciężkiej pracy wielu programistów na całym świecie, iPhone stał się czymś więcej. Oryginalne przesłanie nie uległo jednak zmianie — iPhone cały czas pełni przede wszystkim funkcję telefonu, odtwarzacza muzyki i urządzenia do komunikacji w internecie. Użytkownicy nie dodali iPhone'a do kolekcji swoich urządzeń, które noszą każdego dnia — zastąpili swoje stare telefony i iPody jednym urządzeniem.

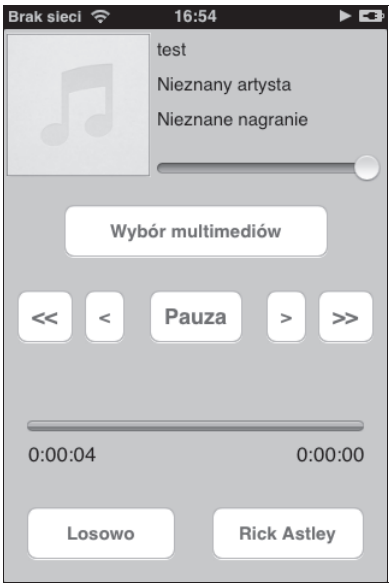
Odtwarzanie muzyki stanowiło istotny aspekt iPhone'a od powstania pierwszych koncepcji tego urządzenia, czyli od 2004 roku. iPhone był zawsze traktowany jako rozszerzenie iPoda. Muzyka stała się również inspiracją do stworzenia iPoda, który okazał się jednym z największych sukcesów firmy Apple. Ludzie kochają muzykę, bo łączy ludzi i pozwala na pełne wyrażanie siebie. Choć wielu użytkowników iPhone'ów może nie traktować swoich urządzeń jako odtwarzaczy, większość z nich bez większego zastanowienia słucha na nich swoich ulubionych piosenek.

W tym rozdziale nauczymy się korzystać z biblioteki muzyki użytkownika wewnątrz aplikacji iOS. Niezależnie od tego, czy chcesz stworzyć w pełni funkcjonalny odtwarzacz muzyki, czy też chcesz odtwarzać wybraną przez użytkownika muzykę w formie ścieżki dźwiękowej w grze, w tym rozdziale dowiesz się, jak to zrobić.

Wprowadzenie do przykładowej aplikacji

Przykładowa aplikacja w tym rozdziale nosi nazwę *Odtwarzacz* (rysunek 6.1). Jest to w pełni funkcjonalny odtwarzacz muzyki na iPhone'a. Użytkownik może wybierać utwory do odtworzenia przy użyciu komponentu Media Picker, może także odtwarzać utwory losowo lub wybrać konkretnych artystów. Co więcej, użytkownik będzie w stanie wstrzymywać utwór i powracać do niego, przechodzić pomiędzy nimi oraz regulować głośność odtwarzania. Ponadto zostanie zaimplementowany licznik, a dwa dodatkowe przyciski pozwolą na trzydziestosekundowe przeskoki w odtwarzanym utworze.

Kup książkę Poleć książkę



Rysunek 6.1. Pierwszy rzut oka na przykładową aplikację — w pełni funkcjonalny odtwarzacz muzyki działający na iPodzie

Co więcej, aplikacja wyświetli okładkę dla aktualnie odtwarzanej ścieżki, jeśli jest ona dostępna.

Emulator (symulator) systemu iOS dostarczony razem z Xcode nie zawiera aplikacji *Muzyka*. Nie możemy też przenieść plików muzycznych do jego systemu plików. Aplikacja zadziała tylko na urządzeniach. W momencie uruchomienia aplikacji w emulatorze zostanie wyświetlonych kilka błędów.

```
2013-03-02 16:04:13.392 player[80633:c07] MPMusicPlayer: Unable to launch iPod music
➥player server: application not found
2013-03-02 16:04:13.893 player[80633:c07] MPMusicPlayer: Unable to launch iPod music
➥player server: application not found
2013-03-02 16:04:14.395 player[80633:c07] MPMusicPlayer: Unable to launch iPod music
➥player server: application not found
```

Dowolna próba dostępu do biblioteki mediów spowoduje zawieszenie się symulatora z następującym komunikatem:

```
*** Terminating app due to uncaught exception 'NSInternalInconsistencyException',
➥reason: 'Unable to load iPodUI.framework'
```

Tworzenie silnika odtwarzania

Zanim zaczniemy korzystać z jakichkolwiek danych audio, musimy zrozumieć, jakie kontrolki odtwarzania będą nam potrzebne. Odtworzenie muzyki z poziomu aplikacji

wymaga utworzenia nowej instancji klasy `MPMusicPlayerController`. Operację tę wykonujemy w pliku nagłówkowym *ICFViewController.h* — nowy obiekt nosi nazwę `player`. Klasa `MPMusicPlayerController` jest wykorzystywana w tym rozdziale do kontroli odtwarzania oraz do pobierania informacji o odtwarzanych elementach.

```
@interface ICFViewController : UIViewController
{
    MPMusicPlayerController *player;
}

@property (nonatomic, retain) MPMusicPlayerController *player;
```

Wewnątrz metody `viewDidLoad` odtwarzacz typu `MPMusicPlayerController` można zainicjalizować, korzystając z jednej z metod klasy `MPMusicPlayerController`. Pierwszy wariant — `applicationMusicPlayer` — odtworzy muzykę wewnątrz aplikacji. Nie wpłynie on na stan iPoda, a odtwarzanie zakończy się w momencie zamknięcia aplikacji. Drugi wariant — metoda `iPodMusicPlayer` — pozwoli na kontrolę samego iPoda. Odtwarzacz powróci do miejsca, w którym użytkownik pozostawił odtwarzanie na iPodzie, i skorzysta z tej samej ścieżki. Odtwarzanie będzie kontynuowane nawet po przejściu aplikacji w tryb pracy w tle. Przykładowa aplikacja wykorzystuje metodę `applicationMusicPlayer`, ale można zmienić to wywołanie, nie wprowadzając dodatkowego kodu ani zachowania.

```
- (void)viewDidLoad
{
    [super viewDidLoad];

    player = [MPMusicPlayerController applicationMusicPlayer];
}
```

Rejestrowanie powiadomień odtwarzania

Wydajne odtwarzanie muzyki wymaga znajomości stanów odtwarzacza. W związku z tym istnieje konieczność obserwacji trzech powiadomień. Są to powiadomienia o zmianie aktualnie odtwarzanego utworu, zmianie głośności i zmianie stanu odtwarzania. Stany te mogą być monitorowane dzięki zapisaniu się na wspomniane zdarzenia przy użyciu klasy `NSNotificationCenter`. Przykładowa aplikacja korzysta z użytecznej metody `registerMediaPlayerNotifications`, dzięki czemu kod aplikacji pozostaje prosty i czytelny. Po dodaniu nowych obserwatorów do klasy `NSNotificationCenter` należy wywołać metodę odtwarzacza `beginGeneratingPlaybackNotifications`.

```
- (void)registerMediaPlayerNotifications
{
    NSNotificationCenter *notificationCenter = [NSNotificationCenter defaultCenter];
    [notificationCenter addObserver: self
        selector: @selector (nowPlayingItemChanged:)
        name: MPMusicPlayerControllerNowPlayingItemDidChangeNotification
        object: player];
```

Kup książkę Poleć książkę

```
    [notificationCenter addObserver: self
        selector: @selector (playbackStateChanged:)
        name: MPMusicPlayerControllerPlaybackStateDidChangeNotification
        object: player];

    [notificationCenter addObserver: self
        selector: @selector (volumeChanged:)
        name: MPMusicPlayerControllerVolumeDidChangeNotification
        object: player];

    [player beginGeneratingPlaybackNotifications];
}
```

Po zarejestrowaniu się na odpowiednie powiadomienia należy się upewnić, że są one prawidłowo wyrejestrowywane podczas czyszczenia pamięci i widoków — w przeciwnym razie aplikacja może się zawiesić lub dojdzie do innego nieoczekiwanego zachowania. Wywołanie metody `endGeneratingPlaybackNotifications` jest wykonywane w metodzie `viewDidUnload`.

```
-(void)viewWillDisappear:(BOOL)animated
{
    [[NSNotificationCenter defaultCenter] removeObserver: self
        name: MPMusicPlayerControllerNowPlayingItemDidChangeNotification
        object: player];

    [[NSNotificationCenter defaultCenter] removeObserver: self
        name: MPMusicPlayerControllerPlaybackStateDidChangeNotification
        object: player];

    [[NSNotificationCenter defaultCenter] removeObserver: self
        name: MPMusicPlayerControllerVolumeDidChangeNotification
        object: player];
    [player endGeneratingPlaybackNotifications];



    [super viewWillDisappear: animated];
}
```

Dodatkowo utworzymy zegar (`NSTimer`), który posłuży nam do odświeżania interfejsu użytkownika w miarę postępu w odtwarzaniu utworu. W przykładowej aplikacji obiekt klasy `NSTimer` nosi nazwę `playbackTimer`. Na razie wywołania zwrotne do obsługi powiadomień i zachowanie zegara pozostaną niedokończone. Omówimy je później w podrozdziale „Obsługa zmiany stanu”.

Kontrolki użytkownika

Przykładowa aplikacji zawiera kilka przycisków zaprojektowanych do obsługi odtwarzanych utworów. Są to przycisk odtwarzania (ang. *play*), pauzy (ang. *pause*), pomiń (ang. *skip*) i poprzedni (ang. *previous*). Użytkownik będzie zmieniać pozycję w aktualnie odtwarzanym

Kup książkę Poleć książkę

utworze o pół minuty. Najpierw musimy zaimplementować metody do odtwarzania i wstrzymywania utworu. Obie funkcje to zwykły przełącznik — jeśli muzyka jest odtwarzana, przycisk wstrzyma odtwarzanie, a jeśli jest wstrzymana — powróci do odtwarzania. Kod odpowiedzialny za odświeżenie treści przycisku jest omówiony jako fragment wywołania zwrotnego dla powiadomienia zmiany stanu w podrozdziale „Obsługa zmiany stanu".

```
- (IBAction)playButtonAction:(id)sender
{
    if ([player playbackState] == MPMusicPlaybackStatePlaying)
    {
        [player pause];
    }

    else
    {
        [player play];
    }
}
```

Użytkownik powinien móc pominąć aktualną ścieżkę lub przejść do poprzedniej. Osiągniemy to, wykonując dwa dodatkowe wywołania za pomocą obiektu `player`.

```
- (IBAction)previousButtonAction:(id)sender
{
    [player skipToPreviousItem];
}
- (IBAction)nextButtonAction:(id)sender
{
    [player skipToNextItem];
}
```

Użytkownicy będą mogli przejść o 30 sekund do przodu lub do tyłu. Jeśli użytkownik osiągnie koniec odtwarzanej ścieżki, poniższy kod spowoduje przejście do następnej. Podobnie jeśli zostanie osiągnięty moment za początkiem utworu, ścieżka zostanie ponownie odtworzona. Obie metody wykorzystują właściwość `currentPlaybackTime` obiektu odtwarzacza. Właściwość ta może być używana do zmiany informacji na temat aktualnie odtwarzanego utworu.

```
- (IBAction)skipBack30Seconds:(id)sender
{
    int newPlayHead = player.currentPlaybackTime - 30;

    if(newPlayHead < 0)
    {
        newPlayHead = 0;
    }

    player.currentPlaybackTime = newPlayHead;
}

- (IBAction)skipForward30Seconds:(id)sender
```

Kup książkę Poleć książkę

```
{
    int newPlayHead = player.currentPlaybackTime + 30;

    if(newPlayHead > currentSongDuration)
    {
        [player skipToNextItem];
    }

    else
    {
        player.currentPlaybackTime = newPlayHead;
    }
}
```

Poza udostępnieniem standardowych kontrolek w przykładowej aplikacji umożliwiamy także zmianę poziomu głośności audio. Obiekt odtwarzacza przyjmuje wartości od `0.0` (wyciszenie) do `1.0` (maksymalna głośność). Skoro korzystamy z kontrolki `UISlider`, wartość może być przekazana bezpośrednio do odtwarzacza.

```
- (IBAction)volumeSliderChanged:(id)sender
{
    [player setVolume:[volumeSlider value]];
}
```

Obsługa zmiany stanu

Wspomniałem już wcześniej, że zostały zarejestrowane trzy powiadomienia, dla których otrzymamy wywołania zwrotne. Powiadomienia te pozwalają aplikacji na określanie stanu i zachowania klasy `MPMusicPlayerController`. Pierwsza metoda będzie wywoływana w momencie zmiany aktualnie odtwarzanego utworu. Metoda ta składa się z dwóch części. Najpierw aktualizujemy okładkę albumu, a następnie modyfikujemy etykiety opisujące artystę, tytuł piosenki i nazwę odtwarzanego albumu.

Każdy plik odtwarzany przy użyciu klasy `MPMusicPlayerController` jest reprezentowany za pomocą obiektu klasy `MPMediaItem`. Dostęp do obiektu jest możliwy przy użyciu metody `nowPlayingItem` tej klasy.

Okładka albumu jest tworzona za pomocą obiektu klasy `UIImage`. Domyślnie otrzymuje ona obrazek zastępczy, który będzie widoczny w sytuacji, gdy użytkownik nie dysponuje okładką albumu dla danego elementu typu `MPMediaItem`. Klasa `MPMediaItem` wykorzystuje właściwości w postaci klucz-wartość w celu przechowywania danych. Pełna lista właściwości została przedstawiona w tabeli 6.1. Następnie tworzymy obiekt typu `MPMediaItemArtwork` i przypisujemy do niego okładkę. Choć według dokumentacji w przypadku braku okładki powinniśmy otrzymać wartość `nil`, w praktyce tak się nie dzieje. Jest na to prosta rada — należy wczytać album do obiektu typu `UIImage` i sprawdzić otrzymaną wartość. Jeśli jest to wartość `nil`, możemy założyć, że nie dysponujemy okładką, w związku z czym zostanie załadowany obrazek zastępczy. Kod zawarty w przykładowej aplikacji będzie działać nawet wtedy, gdy klasa `MPMediaItemArtwork` zacznie zwracać wartość `nil` w przypadku braku okładki.

Kup książkę Poleć książkę

Tabela 6.1. Atrybuty dostępne w trakcie pracy z klasą MPMediaItem

Atrybuty dostępne w klasie MPMediaItem	Dostępne do przeszukiwania predykatowego?
MPMediaItemPropertyPersistentID	TAK
MPMediaItemPropertyAlbumPersistentID	TAK
MPMediaItemPropertyArtistPersistentID	TAK
MPMediaItemPropertyAlbumArtistPersistentID	TAK
MPMediaItemPropertyGenrePersistentID	TAK
MPMediaItemPropertyComposerPersistentID	TAK
MPMediaItemPropertyPodcastPersistentID	TAK
MPMediaItemPropertyMediaType	TAK
MPMediaItemPropertyTitle	TAK
MPMediaItemPropertyAlbumTitle	TAK
MPMediaItemPropertyArtist	TAK
MPMediaItemPropertyAlbumArtist	TAK
MPMediaItemPropertyGenre	TAK
MPMediaItemPropertyComposer	TAK
MPMediaItemPropertyPlaybackDuration	NIE
MPMediaItemPropertyAlbumTrackNumber	NIE
MPMediaItemPropertyAlbumTrackCount	NIE
MPMediaItemPropertyDiscNumber	NIE
MPMediaItemPropertyDiscCount	NIE
MPMediaItemPropertyArtwork	NIE
MPMediaItemPropertyLyrics	NIE
MPMediaItemPropertyIsCompilation	TAK
MPMediaItemPropertyReleaseDate	NIE
MPMediaItemPropertyBeatsPerMinute	NIE
MPMediaItemPropertyComments	NIE
MPMediaItemPropertyAssetURL	NIE
MPMediaItemPropertyIsCloudItem	TAK

Druga część metody `nowPlayingItemChanged:` zajmie się aktualizacją tytułu piosenki, informacji o wykonawcy i nazwy albumu (rysunek 6.1). Jeśli którakolwiek z tych właściwości zwróci `nil`, zostanie ustawiony obrazek zastępczy. Pełna lista dostępnych właściwości została przedstawiona w tabeli 6.1. Pamiętaj, że w przypadku podcastów będą dostępne

Kup książkę Poleć książkę

dodatkowe atrybuty, opisane przez Apple w dokumentacji dla klasy `MPMediaItem`. Dodatkowo w tabeli 6.1 określamy, czy klucz może być wykorzystany do przeszukiwania predykatowego w przypadku wyszukiwania z poziomu kodu obiektów typu `MPMediaItem`.

```
- (void) nowPlayingItemChanged: (id) notification
{
    MPMediaItem *currentItem = [player nowPlayingItem];

    UIImage *artworkImage = [UIImage imageNamed:@"noArt.png"];

    MPMediaItemArtwork *artwork = [currentItem valueForProperty:
    ↪MPMediaItemPropertyArtwork];

    if (artwork)
    {
        artworkImage = [artwork imageWithSize: CGSizeMake (120,120)];

        if(artworkImage == nil)
        {
            artworkImage = [UIImage imageNamed:@"noArt.png"];
        }
    }
    [albumImageView setImage:artworkImage];

    NSString *titleString = [currentItem valueForProperty:MPMediaItemPropertyTitle];

    if (titleString)
    {
        songLabel.text = titleString;
    }

    else
    {
        songLabel.text = @"Nieznany utwór";
    }

    NSString *artistString = [currentItem valueForProperty:MPMediaItemPropertyArtist];

    if (artistString)
    {
        artistLabel.text = artistString;

    }

    else
    {
        artistLabel.text = @"Nieznany artysta";
    }

    NSString *albumString = [currentItem
valueForProperty:MPMediaItemPropertyAlbumTitle];
```

```
    if (albumString)
    {
        recordLabel.text = albumString;
    }

    else
    {
        recordLabel.text = @"Nieznane nagranie";
    }
}
```

Monitorowanie stanu odtwarzacza jest niezwykle ważne, zwłaszcza że wartość ta może być modyfikowana poza aplikacją. W momencie zmiany stanu zostanie wywołana metoda `playbackStateChanged:`. Nowo tworzona zmienna `playbackState` służy do przechowywania stanu odtwarzacza. Metoda ta wykonuje wiele istotnych zadań — przede wszystkim aktualizuje stan przycisku odtwarzania, aby odzwierciedlić obecny stan. Ponadto metoda ta jest odpowiedzialna zarówno za tworzenie, jak i usuwanie obiektu klasy `NSTimer`, o którym wspominałem już wcześniej. W trakcie odtwarzania utworu zadanie zegara jest wykonywane co 0,3 sekundy. Do jego zadań należą aktualizacja etykiet związanych z odtwarzaniem oraz modyfikowanie stanu kontrolki `UIProgressIndicator` informującej użytkownika o momencie odtwarzania utworu, w którym obecnie się znajdujemy. Metodę, którą wykonuje zegar (`updateCurrentPlaybackTime`), omówimy w kolejnym podrozdziale.

Poza stanami przedstawionymi w przykładowej aplikacji możemy skorzystać z trzech dodatkowych. Pierwszy z nich — `MPMusicPlaybackStateInterrupted` — jest używany w momencie przerwania odtwarzania muzyki, np. w wyniku przychodzącego połączenia telefonicznego. Kolejne dwa stany — `MPMusicPlaybackStateSeekingForward` i `MPMusicPlaybackStateSeekingBackward` — są wykorzystywane do zgłoszenia, że stan odtwarzacza został przesunięty w przód lub w tył.

```
- (void) playbackStateChanged: (id) notification
{
    MPMusicPlaybackState playbackState = [player playbackState];

    if (playbackState == MPMusicPlaybackStatePaused)
    {
        [playButton setTitle:@"Odtwarzaj" forState:UIControlStateNormal];

        if([playbackTimer isValid])
        {
            [playbackTimer invalidate];
        }
    }

    else if (playbackState == MPMusicPlaybackStatePlaying)
    {
        [playButton setTitle:@"Pauza" forState: UIControlStateNormal];

        playbackTimer = [NSTimer
```

Kup książkę Poleć książkę

```
        scheduledTimerWithTimeInterval:0.3
        target:self
        selector:@selector(updateCurrentPlaybackTime)
        userInfo:nil
        repeats:YES];
    }
    else if (playbackState == MPMusicPlaybackStateStopped)
    {
        [playButton setTitle:@"Odtwarzaj" forState:UIControlStateNormal];

        [player stop];

        if([playbackTimer isValid])
        {
            [playbackTimer invalidate];
        }
    }
}
```

W sytuacji gdy głośność uległa zmianie, niezwykle ważne jest odzwierciedlenie zmiany również w interfejsie użytkownika. Wystarczy obserwować powiadomienie `volumeChanged:`. Wewnątrz metody możemy pobrać aktualną głośność odtwarzacza i ustawić odpowiednio kontrolkę głośności.

```
- (void) volumeChanged: (id) notification
{
    [volumeSlider setValue:[player volume]];
}
```

Czas trwania a zegary

Większość użytkowników z pewnością zechce wiedzieć, w którym momencie trwania piosenki obecnie się znajdują lub ile czasu pozostało do jej końca. Przykładowa aplikacja zawiera dwie metody do generowania tych danych. Pierwsza z nich — `updateSongDuration` — jest wywoływana w momencie zmiany utworu lub uruchomienia aplikacji. Tworzymy w niej odwołanie do aktywnej ścieżki, a także pobieramy czas trwania przy użyciu klucza `playbackDuration`. Całkowita liczba godzin, minut i sekund jest obliczana bezpośrednio na podstawie tej wartości, a etykieta odpowiedzialna za wyświetlanie tych informacji jest przedstawiona tuż obok kontrolki `UIProgressIndicator`.

```
-(void)updateSongDuration;
{
    currentSongPlaybackTime = 0;

    currentSongDuration = [[[player nowPlayingItem] valueForProperty:
    ↪@"playbackDuration"] floatValue];

    int tHours = (currentSongDuration / 3600);
    int tMins = ((currentSongDuration / 60) - tHours*60);
    int tSecs = (currentSongDuration ) - (tMins*60) - (tHours *3600);
```

Kup książkę Poleć książkę

```
    songDurationLabel.text = [NSString stringWithFormat:@"%i:%02d:%02d",
    ↪tHours, tMins, tSecs];

    currentTimeLabel.text = @"0:00:00";
}
```

Druga metoda, `updateCurrentPlaybackTime`, jest wykonywana co 0,3 sekundy przy użyciu zegara (`NSTimer`), który jest kontrolowany z poziomu poprzednio omówionej metody `playbackStateChanged:`. Do obliczenia godzin, minut oraz sekund korzystamy z tego samego algorytmu co w metodzie `updateSongDuration`. Wartość zmiennej `percentagePlayed` jest obliczana bezpośrednio na podstawie uzyskanego czasu trwania utworu. Korzystamy z niej do aktualizacji kontrolki `playbackProgressIndicator`. Biorąc pod uwagę, że czas przechowywany w zmiennej `currentPlaybackTime` jest podawany z dokładnością do jednej sekundy, teoretycznie nie musimy wywoływać jej zbyt często. Z drugiej strony im częściej będziemy wywoływać tę metodę, tym lepszą dokładność czasową uzyskamy.

```
-(void)updateCurrentPlaybackTime;
{
    currentSongPlaybackTime = player.currentPlaybackTime;

    int tHours = (currentSongPlaybackTime / 3600);
    int tMins = ((currentSongPlaybackTime / 60) - tHours*60);
    int tSecs = (currentSongPlaybackTime) - (tMins*60) - (tHours*3600);

    currentTimeLabel.text = [NSString stringWithFormat:@"%i:%02d:%02d",
    ↪tHours, tMins, tSecs];

    float percentagePlayed = currentSongPlaybackTime/currentSongDuration;

    [playbackProgressIndicator setProgress:percentagePlayed];
}
```

Tryb losowy i powtarzanie

Do kolejnych funkcjonalności udostępnianych przez klasę `MPMusicPlayerController` zaliczamy możliwość powtórzenia odtwarzanego utworu i tryb losowy. Chociaż przykładowa aplikacja nie zawiera implementacji obu właściwości, dodanie jej nie jest zbyt skomplikowane:

```
player.repeatMode = MPMusicRepeatModeAll;
player.shuffleMode = MPMusicShuffleModeSongs;
```

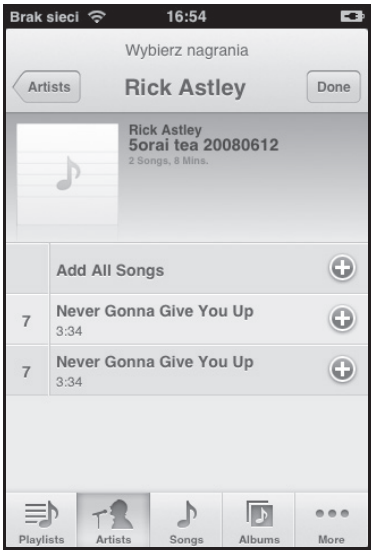
Dostępne tryby powtarzania to: `MPMusicRepeatModeDefault` (domyślny — zgodnie z wcześniejszym wyborem użytkownika), `MPMusicRepeatModeNone` (brak powtarzania), `MPMusicRepeatModeOne` (powtarzaj jedną) i `MPMusicRepeatModeAll` (powtarzaj całość).

Dostępne tryby losowania to: `MPMusicShuffleModeDefault` (domyślny — zgodnie z wcześniejszym wyborem użytkownika), `MPMusicShuffleModeOff` (brak losowania), `MPMusicShuffleModeSongs` (losuj piosenki) i `MPMusicShuffleModeAlbums` (losuj albumy).

Kup książkę Poleć książkę

Media Picker — kontrolka wyboru multimediów

Najprostszym sposobem na umożliwienie użytkownikowi wyboru odtwarzanej piosenki jest udostępnienie mu mechanizmu `MPMediaPickerController` (rysunek 6.2).



Rysunek 6.2. Wybór utworów przy użyciu kontrolera MPMediaPickerController. Elementy wybrane przez użytkownika są zaznaczone (są wyświetlane na szaro)

Klasa `MPMediaPickerController` pozwala użytkownikowi na przeglądanie swojego zbioru muzyki według artystów, piosenek, list odtwarzania i albumów. Zastosowanie klasy `MPMediaPickerController` wymaga zadeklarowania obsługi delegacji `MPMediaPickerControllerDelegate`, która zawiera dwie metody. Pierwsza z nich — `mediaPicker:didPickMediaItems` — jest wywoływana w momencie zakończenia wyboru utworów do odtworzenia. Wybrane utwory są zwracane w postaci obiektu `MPMediaItemCollection`, który możemy przekazać od razu do klasy `MPMusicPlayerController` jako parametr metody `setQueueWithItemCollection:`. Po ustawieniu nowej kolejki dla kontrolera `MPMusicPlayerController` możemy rozpocząć odtwarzanie nowo dodanych multimediów. Kontroler `MPMediaPickerController` nie zostanie automatycznie usunięty po zakończeniu wyboru — konieczne jest jawne wywołanie metody `dismissViewControllerAnimated:completion:`.

```
- (void) mediaPicker: (MPMediaPickerController *) mediaPicker didPickMediaItems:
↪(MPMediaItemCollection *) mediaItemCollection
{
    if (mediaItemCollection)
    {
```

```
        [player setQueueWithItemCollection: mediaItemCollection];
        [player play];
    }

    [self dismissViewControllerAnimated:YES completion:nil];
}
```

W przypadku anulowania lub zamknięcia kontrolera `MPMediaPickerController` bez dokonania wyboru zostanie wykonana metoda `mediaPickerDidCancel:`. Programista musi się pozbyć kontrolera właśnie w ramach tej metody:

```
- (void) mediaPickerDidCancel: (MPMediaPickerController *) mediaPicker
{
    [self dismissViewControllerAnimated:YES completion:nil];
}
```

Po zaimplementowaniu metod delegacji możemy utworzyć instancję klasy `MPMediaPickerController`. W momencie przeprowadzania alokacji i inicjalizacji obiektu tej klasy musimy przekazać parametr określający obsługiwane rodzaje multimediów. Pełna lista dostępnych opcji została przedstawiona w tabeli 6.2. Zauważ, że każdy element może być skojarzony z wieloma rodzajami multimediów. W tym miejscu możemy również skorzystać z kilku dodatkowych parametrów dla klasy `MPMediaPickerController`, np. możliwości zaznaczania wielu elementów czy wyświetlania znaku zachęty w trakcie zaznaczania (rysunek 6.2). Dodatkowy parametr typu logicznego pozwala określić, czy powinny być uwzględniane również elementy z usługi iCloud (domyślnie tak).

Tabela 6.2. Rodzaje mediów możliwe do zastosowania w momencie tworzenia kontrolera MPMediaPickerController

Stała	Opis
`MPMediaTypeMusic`	Dowolny element muzyczny
`MPMediaTypePodcast`	Podcast audio
`MPMediaTypeAudioBook`	Audiobook
`MPMediaTypeITunesU`	Element muzyczny skojarzony z usługą iTunes U
`MPMediaTypeAnyAudio`	Dowolny element dźwiękowy
`MPMediaTypeMovie`	Dowolny element zawierający film
`MPMediaTypeTVShow`	Dowolny element zawierający audycję telewizyjną
`MPMediaTypeVideoPodcast`	Dowolny element zawierający podcast wideo (nie należy go mylić z elementem typu `MPMediaTypePodcast`)
`MPMediaTypeMusicVideo`	Dowolny film zawierający nagranie muzyczne
`MPMediaTypeVideoITunesU`	Dowolny element wideo skojarzony z usługą iTunes U (nie należy go mylić z elementem typu `MPMediaTypeAudioITunesU`)
`MPMediaTypeAnyVideo`	Dowolny element wideo
`MPMediaTypeAny`	Dowolny element zawierający zarówno audio, jak i wideo

Kup książkę Poleć książkę

```
- (IBAction)mediaPickerButtonAction:(id)sender
{
    MPMediaPickerController *mediaPicker = [[MPMediaPickerController alloc]
    ↪initWithMediaTypes:MPMediaTypeAny];

    mediaPicker.delegate = self;
    mediaPicker.allowsPickingMultipleItems = YES;
    mediaPicker.prompt = @"Wybierz utwory do odtworzenia";
    [self presentViewController:mediaPicker animated:YES completion:nil];

    [mediaPicker release];
}
```

Na tym kończymy opis czynności wymaganych do wybrania muzyki do odtwarzania przy użyciu klasy `MPMediaPickerController`. Czasami może się jednak zdarzyć, że konieczne będzie stworzenie własnego interfejsu użytkownika do tego celu, a nawet umożliwienie wyboru piosenek bez pomocy interfejsu! Tymi kwestiami zajmujemy się w kolejnym podrozdziale.

Ręczny wybór piosenek

Mogą zdarzyć się sytuacje, w których konieczne będzie opracowanie bardziej szczegółowego interfejsu do wyboru muzyki przez użytkownika. Może się to wiązać np. z utworzeniem innego zestawu kontrolek do wyboru pojedynczych utworów lub automatycznym wyszukiwaniem artysty czy albumu. W tym podrozdziale omówimy czynności niezbędne do zapewnienia możliwości wyboru piosenek w dowolny sposób.

Pobranie piosenek bez użycia klasy `MPMediaPickerController` wymaga utworzenia obiektu klasy `MPMediaQuery`. Obiekt taki funkcjonuje jako magazyn przechowujący określoną liczbę elementów typu `MPMediaItem`, z których każdy reprezentuje pojedynczy utwór lub ścieżkę audio.

Przykładowa aplikacja udostępnia dwie metody, które korzystają z klasy `MPMediaQuery`. Pierwsza z nich — `playRandomSongAction:` — wybierze losowo piosenkę z biblioteki użytkownika i odtworzy ją, korzystając z istniejącego kontrolera `MPMusicPlayerController`. Znalezienie muzyki z poziomu kodu wiąże się z utworzeniem nowego obiektu klasy `MPMediaQuery`.

Odtwarzanie losowo wybranej piosenki

Jeśli nie określimy żadnego predykatu (warunku przeszukiwania), obiekt klasy `MPMediaQuery` znajdzie wszystkie elementy zawarte w ramach biblioteki muzyki. Dane znajdą się w nowej tablicy (`NSArray`), z której skorzystamy za pomocą metody elementu w klasie `MPMediaQuery`. Każdy element jest reprezentowany oczywiście przy użyciu obiektu klasy `MPMediaItem`. Funkcjonalność losowego wybierania piosenki pozwoli na odtworzenie jednej losowo wybranej piosenki. Jeśli nie uda się znaleźć żadnych utworów, obiekt typu `UIAlert` zostanie zaprezentowany użytkownikowi. W przeciwnym razie wybór nastąpi losowo spośród wszystkich znalezionych utworów.

Kup książkę Poleć książkę

Po znalezieniu jednego lub większej liczby elementów typu `MPMediaItem` tworzymy obiekt typu `MPMediaItemCollection`, przekazując do niego tablicę elementów typu `MPMediaItem`. Kolekcja posłuży jako lista odtwarzania dla kontrolera `MPMusicPlayerController`. Po utworzeniu kolekcji obiekt jest przekazywany do odtwarzacza przy użyciu metody `setQueueWithItemCollection`. W tym momencie odtwarzacz wie, które piosenki zostaną odtworzone przez użytkownika, dlatego też zostanie rozpoczęte odtwarzanie piosenek jedna po drugiej, w takiej kolejności, w jakiej znajdują się one w tablicy przekazanej w momencie tworzenia kolekcji typu `MPMediaItemCollection`.

```
- (IBAction)playRandomSongAction:(id)sender
{
    MPMediaItem *itemToPlay = nil;
    MPMediaQuery *allSongQuery = [MPMediaQuery songsQuery];
    NSArray *allTracks = [allSongQuery items];

    if([allTracks count] == 0)
    {
        UIAlertView *alert = [[UIAlertView alloc]
            initWithTitle:@"Błąd"
            message:@"Nie znaleziono utworów!"
            delegate:nil
            cancelButtonTitle:@"Anuluj"
            otherButtonTitles:nil];

        [alert show];
        [alert release];
        return;
    }

    if ([allTracks count] < 2)
    {
        itemToPlay = [allTracks lastObject];
    }

    int trackNumber = arc4random() % [allTracks count];
    itemToPlay = [allTracks objectAtIndex:trackNumber];

    MPMediaItemCollection * collection = [[MPMediaItemCollection alloc]
    ➥initWithItems:[NSArray arrayWithObject:itemToPlay]];

    [player setQueueWithItemCollection:collection];
    [collection release];

    [player play];

    [self updateSongDuration];
    [self updateCurrentPlaybackTime];
}
```

Kup książkę Poleć książkę

> **Uwaga**
>
> Funkcja `arc4random()` należy do standardowej biblioteki języka C. Można z niej korzystać do generowania liczb pseudolosowych w projektach tworzonych w języku Objective-C. W przeciwieństwie do większości generatorów liczb pseudolosowych funkcja ta otrzymuje ziarno automatycznie w momencie pierwszego użycia.

Wyszukiwanie piosenek w oparciu o predykaty

Odtwarzanie losowo wybranych piosenek może się znudzić. W takiej sytuacji przydałaby się możliwość przeszukiwania biblioteki muzyki. Funkcjonalność tę możemy osiągnąć przy użyciu predykatów. W poniższym przykładzie korzystamy z predykatu w celu znalezienia elementów, w których atrybut `artist` ma wartość `"Rick Astley"` (rysunek 6.3).

Rysunek 6.3. Odtwarzamy tylko piosenki Ricka Astleya dzięki przeszukiwaniu opartemu na predykatach

Przedstawiona metoda działa podobnie do losowego wybierania piosenek, z tą różnicą, że korzystamy z metody `addFilterPredicate` w celu dodania filtru do klasy `MPMediaQuery`. Co więcej, wyniki nie są ograniczane do jednego elementu — odtwarzacz otrzyma tablicę wszystkich pasujących utworów. Pełna lista dostępnych predykatów została przedstawiona w tabeli 6.1 w podrozdziale „Obsługa zmiany stanu”. Aby skorzystać z wielu predykatów jednocześnie, wystarczy wykonać kolejne wywołania metody `addFilterPredicate` w klasie `MPMediaQuery`.

Kup książkę Poleć książkę

```
- (IBAction)playDylan:(id)sender
{
    MPMediaPropertyPredicate *artistNamePredicate = [MPMediaPropertyPredicate
    ↪predicateWithValue: @"Rick Astley" forProperty: MPMediaItemPropertyArtist];

    MPMediaQuery *artistQuery = [[MPMediaQuery alloc] init];

    [artistQuery addFilterPredicate: artistNamePredicate];

    NSArray *tracks = [artistQuery items];

    if([tracks count] == 0)
    {
        UIAlertView *alert = [[UIAlertView alloc]
            initWithTitle:@"Błąd"
            message:@"Nie znaleziono utworów!"
            delegate:nil
            cancelButtonTitle:@"Anuluj"
            otherButtonTitles:nil];

        [alert show];
        [alert release];

        return;
    }

    MPMediaItemCollection * collection = [[MPMediaItemCollection alloc]
    ↪initWithItems:tracks];

    [player setQueueWithItemCollection:collection];
    [collection release];

    [player play];

    [self updateSongDuration];
    [self updateCurrentPlaybackTime];
}
```

Podsumowanie

W tym rozdziale omówiliśmy mechanizmy związane z obsługą biblioteki muzyki użytkownika. Najpierw stworzyliśmy silnik odtwarzania, który pozwala użytkownikowi kontrolować proces odtwarzania, np. wstrzymywanie, powracanie, kontrolę głośności i przechodzenie pomiędzy kolejnymi utworami. W kolejnych dwóch podrozdziałach zajęliśmy się dostępem do samej biblioteki i wyborem z niej konkretnych utworów. Zademonstrowaliśmy możliwości kontrolki wyboru multimediów za pomocą wbudowanego interfejsu. Omówiliśmy także możliwości ręcznego przeprowadzenia procesu wyboru utworów z wykorzystaniem predykatów.

Kup książkę Poleć książkę

Przykładowa aplikacja stanowi przykład w pełni funkcjonalnego, ale jednocześnie stosunkowo prostego odtwarzacza muzyki dla systemu iOS. Wiedza przedstawiona w tym rozdziale może się stać podstawą do stworzenia jeszcze bardziej rozbudowanych odtwarzaczy, a także do opracowania mechanizmów odtwarzania muzyki w tle w dowolnej aplikacji.

Ćwiczenia

1. Dodaj znacznik do kontrolki postępu odtwarzania, który umożliwi zmianę aktualnego momentu odtwarzania utworu, pozwalając na pełną kontrolę użytkownika nad odtwarzaniem.
2. Dodaj funkcjonalność, która pozwoli użytkownikowi na wprowadzenie nazwy piosenki lub artysty, a następnie przedstawi listę pasujących elementów, po czym umożliwi ich odtworzenie.

Kup książkę Poleć książkę

Skorowidz

A

B

Kup książkę Poleć książkę

C

D

Kup książkę Poleć książkę

E

F

G

H

I

Kup książkę Poleć książkę

J

K

Kup książkę Poleć książkę

Kup książkę Poleć książkę

L

Ł

M

Kup książkę Poleć książkę

N

Kup książkę Poleć książkę

O

P

Kup książkę Poleć książkę

R

Kup książkę Poleć książkę

S

Ś

T

Kup książkę Poleć książkę

U

W

Kup książkę Poleć książkę

X

Z

Kup książkę Poleć książkę

Ż

Kup książkę Poleć książkę

Kup książkę Poleć książkę

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

GRUPA WYDAWNICZA
Helion SA

iOS dla ambitnych!

Systemy spod znaku jabłka od zawsze wyznaczały trendy w wyglądzie interfejsu użytkownika. Nie inaczej jest w przypadku urządzeń mobilnych. Platforma iOS zdobyła uznanie użytkowników dzięki niezwykłej wygodzie użytkowania, atrakcyjnemu wyglądowi oraz ogromnym możliwościom. Jeżeli chcesz stworzyć aplikację, w której wykorzystasz zaawansowane narzędzia i mechanizmy SDK, to trafiłeś na doskonałą publikację!

W trakcie lektury będziesz mieć niepowtarzalną okazję dogłębnego poznania systemu iOS — wraz z jego licznymi bibliotekami, ułatwiającymi pracę programistom. Jaką wiedzę zdobędziesz dzięki tej książce? Nauczysz się tworzyć animacje z wykorzystaniem UIKit Dynamics oraz stosować biblioteki Core Location, MapKit i Geofencing. Ponadto dowiesz się, do czego służą Game Center oraz iCloud. W kolejnych rozdziałach znajdziesz informacje poświęcone komunikacji między systemami za pośrednictwem formatu JSON, lokalnym repozytoriom danych oraz zaawansowanym operacjom na tekście przy użyciu biblioteki TextKit. Dodatkowo poznasz tajniki SDK i sprytne techniki pracy z IDE. To obowiązkowa lektura każdego programisty platformy iOS!

Dzięki tej książce:

- poznasz najbardziej przydatne biblioteki platformy iOS,
- wykorzystasz potencjał środowiska Xcode,
- zintegrujesz swoją aplikację z iCloud i Game Center,
- zaznajomisz się z zaawansowanymi funkcjami języka Objective-C,
- stworzysz lepszą aplikację w krótszym czasie.

helion.pl księgarnia internetowa
Nr katalogowy: 24483
Księgarnia internetowa http://helion.pl
Zamówienia telefoniczne: 0 801 339900 0 601 339900
Informatyka w najlepszym wydaniu

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

Addison Wesley

cena: 99,00 zł
ISBN 978-83-246-9182-1
9 788324 691821

ALWAYS LEARNING
PEARSON