

Wejdź do świata twórców oprogramowania dla urządzeń mobilnych!



Android Flash

Zaawansowane programowanie
aplikacji mobilnych

Stephen Chin • Dean Iverson • Oswald Campesato • Paul Trani

Apress®



Tytuł oryginału: Pro Android Flash

Tłumaczenie: Łukasz Schmidt

ISBN: 978-83-246-3920-5

Original edition copyright © 2011 by Stephen Chin, Dean Iverson, Oswald Campesato, and Paul Trani.
All rights reserved.

Polish edition copyright © 2012 by HELION SA.
All rights reserved.

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 the Publisher.

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.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą 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)

Drogi Czytelniku!

Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres

<http://helion.pl/user/opinie/andfzp>

Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzje.

Printed in Poland.

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

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

Spis treści

	O autorach	11
	O recenzencie	12
	Podziękowania	13
	Przedmowa	14
	Wprowadzenie	15
Rozdział 1	Wprowadzenie do Flasha mobilnego	17
	Dlaczego Android?	17
	Flash na innych platformach	18
	Poznanie Androida	18
	Platforma Flash	20
	Środowisko wykonawcze Flash	20
	Narzędzia Flash	21
	Uruchamianie aplikacji w programie Flash Professional	23
	Uruchamianie aplikacji w programie Flash Builder	26
	Uruchamianie aplikacji z wiersza poleceń	36
	Podsumowanie	37
Rozdział 2	Dostosowywanie aplikacji do profilu urządzenia mobilnego	39
	Rozmiar ekranu	39
	Rozdzielczość ekranu a jego gęstość	41
	Symulowanie pikseli niezależnych od gęstości wyświetlacza urządzenia we Flashu	42
	Gęstość ekranu w aplikacjach Flex	42
	Aplikacja Density Explorer	43
	Obsługa gęstości w CSS	47

Orientacja ekranu	50
Przełączanie się pomiędzy trybem portretowym i krajobrazowym we Fleksie	51
Automatyczna reorientacja we Flashu	55
Reorientacja we Flashu — obracająca się twarz	56
Wielodotykowość i gesty	58
Gesty w aplikacji mobilnej	58
Aplikacja Flash Scrapbook	60
API punktu dotykowego	66
Podsumowanie	71
Rozdział 3 Tworzenie aplikacji Flash i Flex dla Androida	73
Konstruowanie mobilnych UI za pomocą Fleksa	73
ViewNavigatorApplication	74
Ważne zdarzenia w życiu widoku	76
TabbedViewNavigatorApplication	78
Po prostu aplikacja	82
Komponenty ViewNavigator i View	84
Przekazywanie danych pomiędzy komponentami View	92
Zachowywanie komponentów View i danych sesji	92
Kontrolki wizualne	93
Kontrolki tekstowe	94
Obsługa klawiatury ekranowej	99
Kontrolki przycisków	101
Listy Flex	106
Kontrolki Slider, Scroller i BusyIndicator	108
Podsumowanie	112
Rozdział 4 Grafika i animacja	113
Wykorzystywanie prostych obiektów Spark do rysowania grafiki 2D	113
Rysowanie prostokątów i elips	113
Stosowanie gradientów liniowych i radialnych	114
Renderowanie sześciennych krzywych Béziera	117
Inny przykład elementu Path	120
Stosowanie filtrów Spark	121
Stosowanie przekształceń obiektów geometrycznych	122
Dodawanie efektu zmiany rozmiaru	123
Dodawanie animacji za pomocą biblioteki Spark	125
Stosowanie elementu Animate	126
Animacje odtwarzane równolegle i sekwencyjnie	128
Tworzenie animacji 3D	130
Tworzenie skórek Spark	133
Generowanie wykresów 2D w Spark	136
Tworzenie wykresów słupkowych 2D	137
Tworzenie wykresów kołowych 2D	138
Wykorzystywanie FXG i Spark	140

Program do szkicowania	143
Podsumowanie	147
Rozdział 5 Wdrażanie i publikowanie aplikacji	149
Przygotowywanie emulatora Androida	149
Instalowanie Android SDK	150
Tworzenie urządzenia wirtualnego Android	151
Instalowanie AIR na emulatorze	153
Powiązania klawiatury z emulatorem	154
Wdrażanie aplikacji AIR	155
Przygotowanie ADT	156
Uprawnienia aplikacji	157
Ikony i zasoby	160
Certyfikaty podpisywania kodu	162
Tworzenie certyfikatów za pomocą ADT	163
Publikowanie z Flash Professional	165
Eksportowanie wersji finalnej z Flash Builder	166
Uruchamianie aplikacji Flex w emulatorze Androida	167
Wdrażanie aplikacji z wiersza poleceń	167
Publikowanie aplikacji AIR w Google Play	170
Krok 1. Utwórz konto programisty w Google Play	170
Krok 2. Opakowywanie aplikacji	170
Krok 3. Wysyłanie aplikacji Adobe AIR	171
Podsumowanie	171
Rozdział 6 Adobe AIR i aplikacje natywne Androida	173
Wywoływanie funkcji URI z Adobe AIR	174
Wywoływanie własnych stron HTML z Adobe AIR	176
Otwieranie stron HTML w Adobe AIR	177
Korzystanie z SQLite w Adobe AIR	179
Wprowadzenie do podstawowych koncepcji Androida	183
Podstawowe funkcje Androida 3.0	184
Pobieranie i instalowanie Androida SDK	185
Podstawowe koncepcje Androida	185
Tworzenie aplikacji dla Androida	191
Struktura aplikacji dla Androida	191
Najważniejsze pliki aplikacji dla Androida	192
Wysyłanie powiadomień w aplikacjach dla Androida	195
Integracja aplikacji Adobe AIR z natywną aplikacją Androida	205
Podsumowanie	207
Rozdział 7 Wykorzystywanie wejść sprzętowych	209
Mikrofon	209
Aparat — klasy Camera i CameraUI	212
Klasa Camera	212
Wykonywanie operacji na strumieniu wideo aparatu	215

	Klasa CameraRoll	223
	CameraUI	227
	Akcelerometr	228
	Klasy Accelerometer i AccelerometerEvent	230
	Geolokalizacja	232
	Podsumowanie	235
Rozdział 8	Integracja z multimediami	237
	Odtwarzanie efektów dźwiękowych	237
	Klasa SoundEffect	238
	Przykład osadzonego SoundEffect	239
	Zaawansowane rozwiązanie dźwiękowe	240
	Odtwarzanie nagranego dźwięku	242
	Dynamiczne generowanie danych dźwiękowych	243
	Obsługa przechodzenia pomiędzy stanami	245
	Odtwarzacz muzyki Flash	250
	Odtwarzanie plików MP3	250
	Od prototypu do aplikacji	254
	Odtwarzanie wideo	275
	Optymalizowanie wideo dla urządzeń mobilnych	275
	Komponent Spark VideoPlayer	276
	Wideo za pomocą klasy NetStream	277
	Odtwarzanie wideo za pomocą OSMF	282
	Przykład aplikacji rejestrującej wideo	285
	Podsumowanie	287
Rozdział 9	Obieg pracy projektant – programista	289
	Rola projektanta wizualnego	290
	Początek w Adobe Device Central	290
	Korzystanie z Device Central	290
	Adobe Photoshop	295
	Formaty plików graficznych	297
	Adobe Illustrator	299
	Adobe Fireworks	300
	Rola programisty	300
	Narzędzia programisty	302
	Podsumowanie	304
Rozdział 10	Optymalizowanie wydajności	305
	Podstawy optymalizowania wydajności aplikacji mobilnej	305
	Postrzegana a faktyczna wydajność aplikacji	306
	Optymalizowanie wydajności grafiki	307
	Elastyczny tor wyścigowy	308
	Redukowanie czasu wykonywania kodu	309
	Przyspieszanie renderowania	309

Buforowanie bitmap na scenie	311
Renderowanie za pomocą GPU	317
Wydajne renderery elementów	319
Klasy obrazów Flex	321
Wydajność komponentów tekstowych	322
Wbudowane renderery elementów	324
API i narzędzia monitorujące wydajność	326
Hi-ReS! Stats	326
PerformanceTest v2 Beta	327
Program profilujący we Flash Builder	328
Wydajność Flasha w przyszłości	329
Szybsze odświeżanie	329
Lepsza wydajność kodu ActionScript	330
Współbieżność	331
Wielowątkowy potok renderowania	332
Stage3D	333
Podsumowanie	334
Rozdział 11 Inne urządzenia: tablety i TV	335
Skalowanie ekranu	335
Dostosowywanie oparte na stanach	337
Wykorzystywanie grup stanów	338
Dostosowywanie oparte na projekcie	344
Implementowanie interfejsu dla tabletu	348
Przenoszenie aplikacji do odbiorników TV	352
Przenoszenie aplikacji do urządzeń PlayBook	353
System iOS	356
Podsumowanie	356
Skorowidz	359

ROZDZIAŁ 4



Grafika i animacja

Grafiki są atrakcyjne dla osób w każdym wieku. Jeśli lubisz grafiki tak bardzo jak my, to ucieszysz się z tego, że grafiki oparte na Fleksie, które działają w przeglądarce komputera biurkowego, będą także poprawnie wyświetlane na urządzeniu mobilnym. Poza tym przy tworzeniu aplikacji związanych z grafiką dla urządzeń mobilnych można również wykorzystać zdarzenia dotykowe i gesty (które opisaliśmy w rozdziale 2.).

W pierwszej części niniejszego rozdziału pokażemy, w jaki sposób renderować różne dwuwymiarowe kształty, takie jak prostokąty, elipsy, krzywe Béziera i ścieżki. Druga część rozdziału zawiera przykłady kodu renderującego obiekty geometryczne wypełnione gradientami liniowymi i radialnymi. W trzeciej części rozdziału przedstawimy przykłady wykorzystywania filtrów, takich jak `Blur`, `DropShadow` i `Glou`.

Zobaczysz także wycinki kodu ilustrujące, jak wykonywać przekształcenia (przesunięcia, zmiana rozmiaru, obracanie i przekształcenia poprzeczne) obiektów graficznych omówionych w pierwszej części rozdziału. Następnie dowiesz się, jak renderować wykresy i grafy za pomocą komponentów `MX`. Ostatnim przykładem w tym rozdziale będzie aplikacja do szkicowania, w której wykorzystamy omówione wcześniej elementy. Program ten będzie obsługiwał także zdarzenia dotykowe, zapewni również możliwość rysowania na obrazie w formacie `JPG` i opcję zapisu rysunków w postaci plików `JPG` na urządzeniu mobilnym.

Dzięki lekturze tego rozdziału dobrze zrozumiesz możliwości, jakie są związane z wykorzystaniem grafiki w aplikacjach dla urządzeń mobilnych. Być może niektóre przykłady w tym rozdziale zainspirują Cię do napisania własnych aplikacji z efektownymi grafikami!

Wykorzystywanie prostych obiektów Spark do rysowania grafiki 2D

W tym podrozdziale pokażemy, jak renderować różne obiekty 2D, takie jak prostokąty, elipsy, krzywe Béziera, wielokąty i ścieżki. Dodatkowo w niektórych przykładowych aplikacjach będziesz mógł obejrzeć utworzone przez nas obrazy — w ten sposób porównasz wyniki działania prezentowanego kodu.

Rysowanie prostokątów i elips

Zacznijmy od narysowania dwóch prostokątów i elipsy — dwóch znanych każdemu obiektów geometrycznych. Utwórz nowy projekt mobilny `Flex`, nadaj mu nazwę `RectEclipse1` — skorzystaj z szablonu aplikacji mobilnej i dodaj kod pokazany w listingu 4.1.

Listing 4.1. Renderowanie dwóch prostokątów oraz elipsy

```

<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="HomeView">
  <s:Rect id="rect1" x="10" y="10" width="250" height="200">
    <s:fill>
      <s:SolidColor color="0xFF0000"/>
    </s:fill>
    <s:stroke>
      <s:SolidColorStroke color="0xFFFF00" weight="4"/>
    </s:stroke>
  </s:Rect>

  <s:Ellipse id="ellipse1" x="10" y="220" width="250" height="200">
    <s:fill>
      <s:SolidColor color="0x0000FF"/>
    </s:fill>
    <s:stroke>
      <s:SolidColorStroke color="0xFF0000" weight="4"/>
    </s:stroke>
  </s:Ellipse>

  <s:Rect id="rect2" x="10" y="460" width="250" height="100">
    <s:fill>
      <s:SolidColor color="0xFFFF00"/>
    </s:fill>
    <s:stroke>
      <s:SolidColorStroke color="0x0000FF" weight="8"/>
    </s:stroke>
  </s:Rect>

  <fx:Declarations>
    <!-- Tu umieść elementy niewizualne -->
  </fx:Declarations>
</s:View>

```

Kod w listingu 4.1 zaczyna się od elementu XML `Rect`, który określa wartości atrybutów `id`, `x`, `y`, `width` i `height`. Warto zauważyć, że element XML `Rect` zawiera elementy XML `fill` i `stroke` zamiast atrybutów `fill` i `stroke`, co odróżnia go od SVG, gdzie wartości `fill` i `stroke` są ustalane za pomocą atrybutów. Jednak element XML `stroke` zawiera element potomny XML `SolidColorStroke`, który wskazuje wartości `color` i `weight` jako atrybuty, a nie wartości elementów XML. Zauważ też, że w SVG istnieją atrybuty `stroke` i `stroke-width` zamiast atrybutów `color` i `weight`.

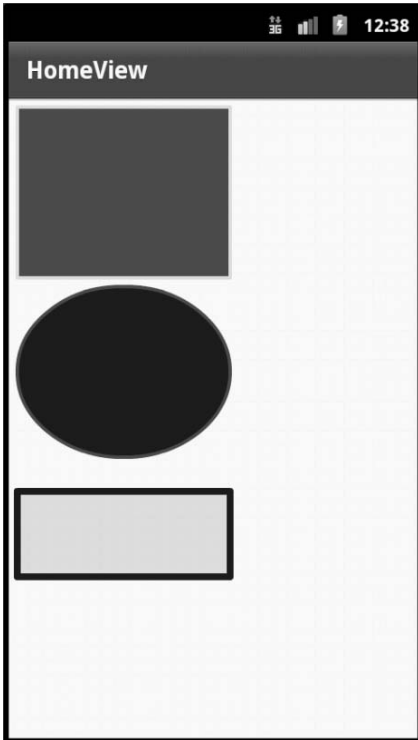
Kod w listingu 4.1 zawiera także element XML `Ellipse` definiujący elipsę z niemal identycznymi atrybutami i wartościami jak element XML `Rect`, ale generujący elipsę zamiast prostokąta.

Drugi element XML `Rect` jest podobny do pierwszego elementu `Rect`, ale ma inne kolory i położenie na ekranie.

Na rysunku 4.1 pokazano dwa prostokąty i elipsę, narysowane w wyniku działania kodu z listingu 4.1.

Stosowanie gradientów liniowych i radialnych

Aplikacje mobilne Flex obsługują gradienty liniowe i radialne. Jak sugeruje nazwa, gradient liniowy powstaje przez obliczenie barw pośrednich i nałożenie ich w sposób liniowy pomiędzy kolorem początkowym a końcowym. Na przykład: jeśli gradient zmienia się od koloru czarnego do czerwonego, początkowym kolorem jest czarny, a końcowym czerwony z liniowym przechodzeniem odcieni kolorów pomiędzy czarnym a czerwonym.



Rysunek 4.1. Dwa prostokąty i elipsa

Gradient radialny różni się od liniowego tym, że przechodzenie odcieni następuje w sposób radialny. Wyobraź sobie kroplę upadającą na powierzchnię wody: powstają kolejne okręgi o coraz większej średnicy — w podobny sposób są renderowane gradienty radialne.

Jako ilustrację powyższego napisaliśmy kod, który renderuje prostokąt wypełniony gradientem liniowym i elipsę wypełnioną gradientem radialnym. Utwórz zatem nowy projekt mobilny Flex, nadaj mu nazwę `LinearRadial1` — skorzystaj z szablonu aplikacji mobilnej i dodaj kod pokazany w listingu 4.2.

Listing 4.2. Stosowanie gradientów liniowych i radialnych

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  title="HomeView">

  <s:Panel width="100%" height="100%"
    title="Gradienty liniowy i radialny">
    <s:Group>
      <s:Rect id="rect1" x="10" y="10"
        height="250" width="300">
        <s:fill>
          <s:LinearGradient>
            <s:GradientEntry color="0xFF0000"
              ratio="0" alpha=".5"/>
            <s:GradientEntry color="0xFFFF00"
              ratio=".33" alpha=".5"/>
            <s:GradientEntry color="0x0000FF"
              ratio=".66" alpha=".5"/>
          </s:LinearGradient>
        </s:fill>
      </s:Rect>
    </s:Group>
  </s:Panel>
</s:View>
```

```

        </s:LinearGradient>
    </s:fill>

    <s:stroke>
        <s:SolidColorStroke color="0x000000" weight="2"/>
    </s:stroke>
</s:Rect>

<s:Ellipse id="ellipse1" x="10" y="270"
    height="250" width="300">
    <s:fill>
        <s:RadialGradient>
            <s:GradientEntry color="0xFF0000"
                ratio="0" alpha="1"/>
            <s:GradientEntry color="0xFFFF00"
                ratio=".9" alpha="1"/>
        </s:RadialGradient>
    </s:fill>

    <s:stroke>
        <s:SolidColorStroke color="0x000000" weight="2"/>
    </s:stroke>
</s:Ellipse>
</s:Group>
</s:Panel>

<fx:Declarations>
    <!-- Tu umieść elementy niewizualne -->
</fx:Declarations>
</s:View>

```

Kod w listingu 4.2 zawiera jeden element XML `Panel`. W tym elemencie znalazł się jeden element XML `Group`, którego atrybuty określają układ panelu. Element XML `Group` zawiera dwa elementy potomne: `Rect` i `Ellipse`. Element XML `Rect` definiuje prostokąt wypełniony gradientem liniowym:

```

<s:Rect id="rect1" x="10" y="10" height="250" width="300">
    <s:fill>
        <s:LinearGradient>
            <s:GradientEntry color="0xFF0000" ratio="0" alpha=".5"/>
            <s:GradientEntry color="0xFFFF00" ratio=".33" alpha=".5"/>
            <s:GradientEntry color="0x0000FF" ratio=".66" alpha=".5"/>
        </s:LinearGradient>
    </s:fill>

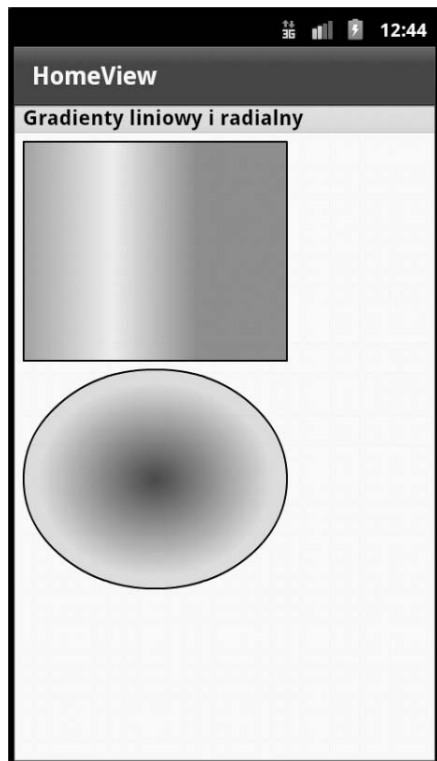
    <s:stroke>
        <s:SolidColorStroke color="0x000000" weight="2"/>
    </s:stroke>
</s:Rect>

```

Powyzszy element XML `Rect` określa wartości atrybutów `id`, `x`, `y`, `width` i `height`. Element ten zawiera również element XML `Fill` (jak widziałeś już w poprzednim przykładzie), w którym z kolei znajduje się element XML `LinearGradient` określający trzy elementy XML `GradientEntry`. Każdy z nich wskazuje wartość dziesiętną (pomiędzy 0 i 1) dla atrybutów `ratio` i `alpha`. Ostatnia część elementu XML `Rect` zawiera element XML `Stroke`, w którym z kolei znajduje się element XML `SolidStrokeElement` określający wartości atrybutów `color` i `weight`.

W kodzie z listingu 4.2 znajduje się także element XML `Ellipse`, który definiuje elipsę z gradientem radialnym. Ten kod zawiera niemal te same atrybuty i wartości, co element XML `Rect`, ale reprezentuje elipsę zamiast prostokąta.

Na rysunku 4.2 pokazano prostokąt wypełniony gradientem liniowym i elipsę wypełnioną gradientem radialnym.



Rysunek 4.2. Prostokąt wypełniony gradientem liniowym i elipsa wypełniona gradientem radialnym

Renderowanie sześciennych krzywych Béziera

Flex obsługuje sześcienną krzywą Béziera (krzywe trzeciego stopnia o dwóch punktach końcowych i dwóch punktach kontrolnych) i kwadratowe krzywe Béziera (krzywe drugiego stopnia o dwóch punktach końcowych i jednym punkcie kontrolnym). Sześcienną krzywą Béziera łatwo odróżnić, ponieważ jej definicja zaczyna się od litery *C* (lub *c*), podczas gdy definicja kwadratowej krzywej Béziera zaczyna się od litery *Q* (lub *q*). Wielkie litery *C* i *Q* wskazują bezwzględne położenie, a małe litery *c* i *q* położenie względem poprzedzającego punktu w elemencie XML `Path`.

Pierwszy z punktów wskazanych dla sześcienną lub kwadratowej krzywej Béziera jest pierwszym punktem kontrolnym, za nim następuje drugi punkt kontrolny w przypadku sześcienną krzywej Béziera, a potem drugi punkt końcowy. Pierwszy punkt końcowy w kwadratowej i sześcienną krzywej Béziera jest poprzedzającym punktem wskazanym w elemencie XML `Path`. Jeśli punkt taki nie zostanie wskazany, początek układu współrzędnych $(0, 0)$ służy za pierwszy punkt końcowy.

Sekwencję dla krzywych Béziera można także wskazać za pomocą litery *S* (dla krzywej sześcienną) lub *T* (dla krzywej kwadratowej).

Teraz utwórz nowy projekt mobilny Flex, nadaj mu nazwę `BezierCurves1` — skorzystaj z szablonu aplikacji mobilnej i dodaj kod pokazany w listingu 4.3. Kod ten tworzy cztery krzywe Béziera: sześcienną, kwadratową, dwie połączone sześcienną oraz sześcienną połączoną z kwadratową.

Listing 4.3. Renderowanie sześciennych i kwadratowych krzywych Béziera

```

<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  title="HomeView">

  <s:Panel width="100%" height="100%"
    title="Sześciennie/kwadratowe krzywe Béziera">
    <!-- sześcienna krzywa Béziera -->
    <s:Path data="C 100 150 200 20 300 100">
      <s:fill>
        <s:LinearGradient rotation="90">
          <s:GradientEntry color="#FFFFFF" alpha="0.5"/>
          <s:GradientEntry color="#FF0000" alpha="0.5"/>
        </s:LinearGradient>
      </s:fill>
      <s:stroke>
        <s:SolidColorStroke color="0x0000FF" weight="4"/>
      </s:stroke>
    </s:Path>

    <!-- kwadratowa krzywa Béziera -->
    <s:Path data="Q 250 200 100 300">
      <s:fill>
        <s:RadialGradient rotation="90">
          <s:GradientEntry color="#000000" alpha="0.8"/>
          <s:GradientEntry color="#0000FF" alpha="0.8"/>
        </s:RadialGradient>
      </s:fill>

      <s:stroke>
        <s:SolidColorStroke color="0xFF0000" weight="8"/>
      </s:stroke>
    </s:Path>

    <!-- dwie połączone sześciennie krzywe Béziera -->
    <s:Path data="C 100 300 200 20 300 100 S 250 200 300 250">
      <s:fill>
        <s:LinearGradient rotation="90">
          <s:GradientEntry color="#FF0000" alpha="0.5"/>
          <s:GradientEntry color="#FFFF00" alpha="0.5"/>
        </s:LinearGradient>
      </s:fill>

      <s:stroke>
        <s:SolidColorStroke color="0x00FF00" weight="2"/>
      </s:stroke>
    </s:Path>

    <!-- dwie połączone krzywe Béziera: sześcienna i kwadratowa -->
    <s:Path data="C 250 400 200 150 350 100 T 250 250 400 280">
      <s:fill>
        <s:LinearGradient rotation="90">
          <s:GradientEntry color="#FFFF00" alpha="0.5"/>
          <s:GradientEntry color="#FF0000" alpha="0.5"/>
        </s:LinearGradient>
      </s:fill>

```

```

    <s:stroke>
      <s:SolidColorStroke color="0x000000" weight="4"/>
    </s:stroke>
  </s:Path>
</s:Panel>
</s:View>

```

Kod w listingu 4.3 zawiera element XML Panel, w którym z kolei znajdują się cztery elementy XML Path tworzące krzywe Béziera z różnymi rodzajami cieniowania. Pierwszy element XML Path definiuje sześcienną krzywą Béziera w następujący sposób:

```

<s:Path data="C 100 300 200 20 300 100 S 250 200 300 250">
  [inne elementy pominięte]
</s:Path>

```

Pierwszym punktem końcowym tej krzywej jest (0, 0), ponieważ nie zostały wskazane inne współrzędne; punkty kontrolne to (100, 300) i (200, 20), docelowym punktem końcowym jest (300, 100).

Element XML Path zawiera element XML LinearGradient definiujący przejście barw od białej do czerwonej, z poziomem krycia 0.5 oraz z niebieskim obrysem o szerokości linii równej 4, tak jak poniżej:

```

<s:LinearGradient rotation="90">
  <s:GradientEntry color="#FFFFFF" alpha="0.5"/>
  <s:GradientEntry color="#FF0000" alpha="0.5"/>
</s:LinearGradient>
</s:fill>
<s:stroke>
  <s:SolidColorStroke color="0x0000FF" weight="4"/>
</s:stroke>

```

Drugi element XML Path definiuje kwadratową krzywą Béziera, której pierwszym punktem końcowym jest (0, 0), ponieważ nie zostały wskazane inne współrzędne; punkt kontrolny to (250, 200), a docelowym punktem końcowym jest (100, 300). Element XML Path zawiera element XML LinearGradient z przejściem barw od czarnej do niebieskiej i poziomem krycia 0.8.

Trzeci element XML Path definiuje sześcienną krzywą Béziera, która jest „powiązana” z drugą sześcienną krzywą Béziera, tak jak poniżej:

```

<s:Path data="C 100 300 200 20 300 100 S 250 200 300 250">
  [inne elementy pominięte]
</s:Path>

```

Dwoma punktami kontrolnymi dla tej sześcienną krzywej Béziera są (100, 300) i (20, 300), punktem docelowym jest (300, 100). Druga część elementu XML Path definiuje kwadratową krzywą Béziera z punktem kontrolnym (250, 200) i z docelowym punktem końcowym (300, 250).

Element XML Path zawiera element XML LinearGradient tworzący gradient liniowy z przejściem od barwy żółtej do czerwonej, a następnie element XML Stroke z kolorem czarnym i o szerokości linii równej 4 jednostkom.

Ostatni element XML Path definiuje sześcienną krzywą Béziera, a następnie drugą sześcienną krzywą Béziera tak jak poniżej:

```

<s:Path data="C 250 300 200 150 350 100 T 250 250 400 280">
  [inne elementy pominięte]
</s:Path>

```

Dwa punkty kontrolne dla tej sześcienną krzywej Béziera to (250, 300) i (200, 150), punktem docelowym jest (350, 100). Druga część elementu XML Path definiuje kwadratową krzywą Béziera, której punktem kontrolnym jest (250, 250), a docelowym punktem końcowym (400, 280).

Element XML Path zawiera element XML LinearGradient tworzący gradient liniowy — przejście od barwy żółtej do czerwonej, z poziomem krycia 0.5, a następnie element XML Stroke z kolorem czarnym i o szerokości linii równej 4 jednostkom.

Na rysunku 4.3 pokazano sześciennie, kwadratowe oraz połączone krzywe Béziera.



Rysunek 4.3. Sześciennie, kwadratowe oraz połączone krzywe Béziera

Inny przykład elementu Path

W poprzednim przykładowym fragmencie kodu pokazaliśmy, jak używać elementu Path do renderowania zestawu krzywych Béziera. Element Path umożliwia także łączenie innych obiektów 2D, takich jak segmenty linii i krzywe Béziera, z gradientami liniowymi i radialnymi. Utwórz teraz nowy projekt mobilny Flex, nadaj mu nazwę Path1 — skorzystaj z szablonu aplikacji mobilnej i dodaj kod pokazany w listingu 4.4.

Listing 4.4. Łączenie segmentów linii i krzywych Béziera

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="HomeView">

    <s:Panel width="100%" height="100%"
            title="Linie/krzywe Béziera oparte na ścieżce">
        <s:Path data="M 50 50 L150 50 350 150 50 150z
                C 250 300 200 150 350 100 T 250 250 400 500">
            <s:fill>
                <s:LinearGradient rotation="90">
                    <s:GradientEntry color="#FF0000" alpha="1"/>
                    <s:GradientEntry color="#0000FF" alpha="1"/>
                </s:LinearGradient>
            </s:fill>

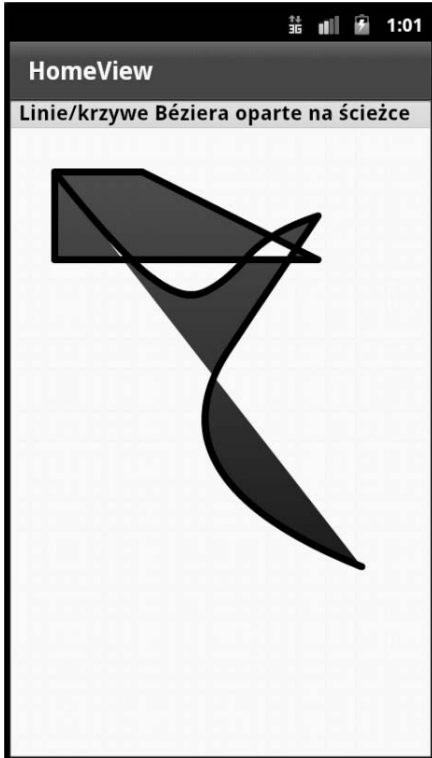
            <s:stroke>
                <s:SolidColorStroke color="0x000000" weight="8"/>
            </s:stroke>
        </s:Path>
    </s:Panel>
</s:View>
```

Element XML Panel z kodu w listingu 4.4 zawiera jeden element XML Path, w którym fragmenty linii posłużyły do wyrenderowania trapezoidu, a następnie pary sześciennych krzywych Béziera. Atrybut data elementu XML Path jest następujący:


```
<s:Path data="M 50 50 L150 50 350 150 50 150z
  C 250 300 200 150 350 100 T 250 250 400 280">
```

Część pierwsza atrybutu data (zaczynająca się literą M) definiuje trapezoid, część druga (zaczynająca się literą C) renderuje sześcienną krzywą Béziera, część trzecia (zaczynająca się literą T) definiuje kolejną taką krzywą.

Na rysunku 4.4 pokazano narysowane elementy: trapezoid i dwie sześcienne krzywe Béziera.



Rysunek 4.4. Elementy oparte na ścieżkach: trapezoid i krzywe Béziera

Stosowanie filtrów Spark

Aby znacząco poprawić wygląd aplikacji pisanych we Fleksie, warto wykorzystać filtry Flex — są bardzo przydatne przy tworzeniu zaawansowanych efektów wizualnych. Proste obiekty Spark (z pakietu `primitives`) obsługują różnorodne filtry, w tym `Blur`, `DropShadow` i `Glow`. Wszystkie filtry należą do pakietu `spark.filters`.

A zatem utwórz nowy projekt mobilny Flex, nadaj mu nazwę `RectLGradFilters3` — skorzystaj z szablonu aplikacji mobilnej i dodaj kod pokazany w listingu 4.5.

Listing 4.5. Rysowanie prostokątów z wykorzystaniem filtrów Spark

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  title="Prostokąt: gradient i filtry">

  <s:Rect id="rect1" x="50" y="50" height="300" width="250">
```

```

<stroke>
  <s:LinearGradient>
    <s:GradientEntry color="0xFF0000"
      ratio="0" alpha=".5"/>
    <s:GradientEntry color="0xFFFF00"
      ratio=".33" alpha=".5"/>
    <s:GradientEntry color="0x0000FF"
      ratio=".66" alpha=".5"/>
  </s:LinearGradient>
</stroke>
<stroke>
  <s:SolidColorStroke color="0xFF0000" weight="2"/>
</stroke>
<filters>
  <s:DropShadowFilter distance="80" color="#0000FF"/>
  <s:BlurFilter/>
  <s:GlowFilter/>
</filters>
</s:Rect>
</s:View>

```

W kodzie z listingu 4.5 znalazł się element XML `Rect` definiujący prostokąt wyrenderowany z gradientem liniowym. Atrybut `ratio` to liczba dziesiętna z przedziału od 0 do 1, określająca ułamek odległości pomiędzy punktem startowym a docelowym, w której zostanie utworzone przejście koloru. W listingu 4.5 element `GradientEntry` posiada atrybut `ratio` o wartości 0, co oznacza, że prostokąt jest renderowany w kolorze `0xFF0000` (szesnastkowa wartość koloru czerwonego). Drugi element `GradientEntry` posiada atrybut `ratio` o wartości 0.33, co oznacza, że prostokąt jest renderowany w kolorze `0xFFFF00` (szesnastkowa wartość oznaczająca żółtą barwę) od punktu, który znajduje się w 33% odległości od punktu początkowego do docelowego. Trzeci element `GradientEntry` posiada atrybut `ratio` o wartości 0.66, co oznacza, że prostokąt jest renderowany w kolorze `0x0000FF` (szesnastkowa wartość oznaczająca barwę niebieską) od punktu znajdującego się w 66% odległości od punktu początkowego do docelowego.

Atrybut `alpha` oznacza wartość krycia. Jest to wartość dziesiętna z przedziału od 0 (element niewidoczny) do 1 (element całkowicie widoczny). Trzy elementy `GradientEntry` w listingu 4.5 mają wartość atrybutu `alpha` równą 0.5, a więc prostokąt jest częściowo widoczny. Wypróbuj różne wartości atrybutów `ratio` i `alpha` — sprawdź, które z nich dają atrakcyjny wizualnie efekt.

Ostatnia część elementu XML `Rect` zawiera element XML `stroke`, który definiuje kolor jako czerwony i szerokość obrysu 2. Za nim zdefiniowano trzy następujące filtry Spark:

```

<filters>
  <s:DropShadowFilter distance="80" color="#0000FF"/>
  <s:BlurFilter/>
  <s:GlowFilter/>
</filters>

```

Trzy filtry Spark wykorzystane w tym przykładzie noszą nazwy wskazujące na uzyskiwany za ich pomocą efekt. Pierwszy filtr, `DropShadowFilter`, dodaje efekt cienia do prostokąta określonego w elemencie XML `Rect`. Drugi filtr, `BlurFilter`, dodaje efekt rozmycia. Ostatni z filtrów, `GlowFilter`, tworzy efekt poświaty.

Na rysunku 4.5 pokazano prostokąt wypełniony gradientem liniowym, dla którego zastosowano trzy filtry Spark.

Stosowanie przekształceń obiektów geometrycznych

W tym podrozdziale pokażemy, jak stosować przekształcenia obiektów geometrycznych, w tym tych, o których powiedzieliśmy we wcześniejszej części tego rozdziału. Proste obiekty geometryczne Spark obsługują następujące efekty i przekształcenia:



Rysunek 4.5. Prostokąt z gradientem liniowym, dla którego zastosowano trzy filtry Spark

- Animate
- AnimateColor
- AnimateFilter
- AnimateShaderTransition
- AnimateTransform
- Fade
- Move
- Resize
- Rotate
- Scale
- Wipe
- CrossFade

Proste obiekty geometryczne Spark są dostępne w pakiecie `spark.effects` i mogą być zastosowane do komponentów Spark oraz komponentów MX. Pakiet `mx.effects` (dołączony do Flex 4 SDK) udostępnia efekty odpowiadające funkcjonalnie komponentom Spark, które można zastosować dla komponentów MX.

W dalszej części rozdziału zamieściliśmy przykłady kodu Flex ilustrujące, jak animować zmiany rozmiaru obiektu.

Dodawanie efektu zmiany rozmiaru

Efekty zmian rozmiaru (czyli powiększanie lub zmniejszanie odpowiednich wymiarów obiektu) mogą być przydatne w grach i są bardzo łatwe w stosowaniu w aplikacjach Flex. Utwórz więc nowy projekt mobilny Flex, nadaj mu nazwę `ScaleEffect1` — skorzystaj z szablonu aplikacji mobilnej i dodaj kod pokazany w listingu 4.6.

Listing 4.6. Dodawanie efektów zmian rozmiaru za pomocą gradientów liniowych

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
```

```

    title="Skalowany prostokąt i elipsa">
<fx:Library>
  <fx:Definition name="MyRect1">
    <s:Rect x="50" y="50" height="40" width="20">
      <s:fill>
        <s:LinearGradient>
          <s:GradientEntry color="0xFF0000"
            ratio="0" alpha=".5"/>
          <s:GradientEntry color="0xFFFF00"
            ratio=".33" alpha=".5"/>
          <s:GradientEntry color="0x0000FF"
            ratio=".66" alpha=".5"/>
        </s:LinearGradient>
      </s:fill>
      <s:stroke>
        <s:SolidColorStroke color="0xFF0000" weight="1"/>
      </s:stroke>
      <s:filters>
        <s:BlurFilter/>
        <s:GlowFilter/>
      </s:filters>
    </s:Rect>
  </fx:Definition>

  <fx:Definition name="MyEllipse1">
    <s:Ellipse x="200" y="200" height="40" width="80">
      <s:fill>
        <s:LinearGradient>
          <s:GradientEntry color="0xFF0000"
            ratio="0" alpha=".5"/>
          <s:GradientEntry color="0xFFFF00"
            ratio=".33" alpha=".5"/>
          <s:GradientEntry color="0x0000FF"
            ratio=".66" alpha=".5"/>
        </s:LinearGradient>
      </s:fill>
      <s:stroke>
        <s:SolidColorStroke color="0xFF0000" weight="1"/>
      </s:stroke>
      <s:filters>
        <s:DropShadowFilter distance="20" color="#FF0000"/>
      </s:filters>
    </s:Ellipse>
  </fx:Definition>
</fx:Library>

<s:Group>
  <fx:MyRect1 scaleX="6" scaleY="4"/>
  <fx:MyEllipse1 scaleX="3" scaleY="8"/>
  <fx:MyRect1 scaleX="2" scaleY="2"/>
  <fx:MyEllipse1 scaleX="2" scaleY="2"/>
</s:Group>
</s:View>

```

Listing 4.6 zawiera dwa elementy XML Definition. Jeden z nich określa element XML Rect zawierający definicję prostokąta, a drugi — element XML Ellipse z definicją elipsy. Element XML Group zawiera dwie referencje do prostokąta i dwie referencje do elipsy, jak pokazano poniżej:

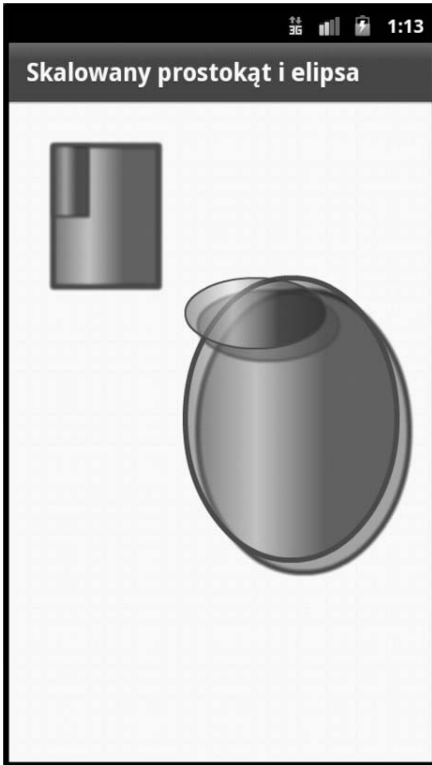
```

<s:Group>
  <fx:MyRect1 scaleX="6" scaleY="4"/>
  <fx:MyEllipse1 scaleX="3" scaleY="8"/>
  <fx:MyRect1 scaleX="2" scaleY="2"/>
  <fx:MyEllipse1 scaleX="2" scaleY="2"/>
</s:Group>

```

Pierwszy element XML zapewnia przeskalowanie zdefiniowanego wcześniej prostokąta `MyRect1` poprzez ustawienie wartości 6 i 4 dla atrybutów `scaleX` i `scaleY`. Drugi element XML zapewnia przeskalowanie zdefiniowanej wcześniej elipsy `MyEllipse1` poprzez ustawienie wartości 3 i 8 dla atrybutów `scaleX` i `scaleY`.

Na rysunku 4.6 pokazano efekt zmiany rozmiaru prostokąta i elipsy.



Rysunek 4.6. Efekt zmiany rozmiaru prostokąta i elipsy

Dodawanie animacji za pomocą biblioteki Spark

W tym podrozdziale przedstawimy aplikację, na przykładzie której pokażemy, jak animować obiekty geometryczne, o których mówiliśmy we wcześniejszej części rozdziału. Efektami animacji z biblioteki Spark `primitives` są:

- `Animate`
- `AnimateColor`
- `AnimateFilter`
- `AnimateShaderTransition`
- `AnimateTransform`

- CrossFade
- Fade
- Move
- Resize
- Rotate
- Scale
- Wipe

W kolejnych punktach, wykorzystując sekwencje kodu, pokażemy, w jaki sposób używać elementu XML Animate i jak definiować animacje odtwarzane równoległe i sekwencyjnie.

Stosowanie elementu Animate

Animacje są oczywiście bardzo popularne w grach, ale mogą być efektywnie wykorzystywane także w innych rodzajach aplikacji. Jednocześnie powinniśmy pamiętać, że w aplikacjach biznesowych prawdopodobnie lepiej będzie nie nadużywać elementów animowanych.

Utwórz nowy projekt mobilny Flex, nadaj mu nazwę AnimPropertyWidth — skorzystaj z szablonu aplikacji mobilnej i dodaj kod pokazany w listingu 4.7.

Listing 4.7. Animowanie szerokości prostokąta

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Animowanie szerokości prostokąta">

    <fx:Declarations>
        <s:Animate id="MyAnimate1">
            <s:motionPaths>
                <s:MotionPath property="width">
                    <s:keyframes>
                        <s:Keyframe time="0" value="200"/>
                        <s:Keyframe time="2000" value="400"/>
                    </s:keyframes>
                </s:MotionPath>
            </s:motionPaths>
        </s:Animate>
    </fx:Declarations>

    <s:VGroup>
        <s:Rect id="rect1" height="300" width="200">
            <s:fill>
                <s:LinearGradient>
                    <s:GradientEntry color="0xFF0000"
                                    ratio="0" alpha=".5"/>
                    <s:GradientEntry color="0xFFFF00"
                                    ratio=".33" alpha=".5"/>
                    <s:GradientEntry color="0x0000FF"
                                    ratio=".66" alpha=".5"/>
                </s:LinearGradient>
            </s:fill>
            <s:stroke>
                <s:SolidColorStroke color="0xFF0000" weight="2"/>
            </s:stroke>
        </s:Rect>
```

```

<s:Button id="MyButton1" label="Animuj szerokość"
        click="MyAnimate1.play([rect1])"
        bottom="150" right="50">
</s:Button>
</s:VGroup>
</s:View>

```

Kod w listingu 4.7 zawiera element XML `Declarations`, w którym z kolei znajduje się element XML `Animate` definiujący szczegóły animacji. Element XML `Animate` posiada atrybut `id` o wartości `MyAnimate1`, do którego tworzymy referencję w procedurze obsługi zdarzenia `click`. Procedurę tę opisujemy w dalszej części tego podrozdziału.

W listingu 4.7 znajduje się także element XML `VGroup`, w którym z kolei znajduje się element XML `Rect`. Jego zawartość przypomina przykładowe elementy XML, jakie już widziałeś w tym rozdziale. W tym samym listingu znalazł się również element XML `Button` uruchamiający efekt animacji. Kiedy użytkownik kliknie lub stuknie ten przycisk, zostanie wykonana procedura obsługi zdarzenia o wartości atrybutu `id` `MyAnimate1`, która została zdefiniowana we wcześniejszej części kodu. Efekt animacji jest prosty: szerokość prostokąta zwiększa się z 200 do 400 jednostek w czasie dwóch sekund (2000 milisekund).

Na rysunkach 4.7 i 4.8 pokazano dwa ujęcia prostokąta, który przesuwa się poziomo po ekranie, kiedy użytkownik kliknie przycisk.



Rysunek 4.7. Animowany prostokąt (położenie początkowe)



Rysunek 4.8. Animowany prostokąt (położenie końcowe)

Animacje odtwarzane równoległe i sekwencyjnie

Flex obsługuje dwa sposoby odtwarzania animacji. Równoległe odtwarzanie animacji polega na wykonywaniu dwóch lub większej liczby animacji w tym samym czasie. Z kolei odtwarzanie animacji sekwencyjnie polega na wykonywaniu dwóch lub większej liczby animacji jedna po drugiej, co oznacza, że w danej chwili wykonywana jest tylko jedna z nich. Mając to na uwadze, utwórz nowy projekt mobilny Flex, nadaj mu nazwę `SequentialAnimation1` — skorzystaj z szablonu aplikacji mobilnej i dodaj kod pokazany w listingu 4.8.

Listing 4.8. Tworzenie animacji odtwarzanej sekwencyjnie

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Animacja sekwencyjna">

    <fx:Declarations>
        <s:Sequence id="transformer1" target="{button1}">
            <s:Move xFrom="50" xTo="150"
                autoCenterTransform="true"/>
            <s:Rotate angleFrom="0" angleTo="360"
                autoCenterTransform="true"/>
            <s:Scale scaleXFrom="1" scaleXTo="2"
                autoCenterTransform="true"/>
        </s:Sequence>

        <s:Sequence id="transformer2" target="{button2}">
            <s:Move xFrom="50" xTo="150"
                autoCenterTransform="true"/>
            <s:Scale scaleXFrom="1" scaleXTo="2"
                autoCenterTransform="true"/>
            <s:Rotate angleFrom="0" angleTo="720"
                autoCenterTransform="true"/>
        </s:Sequence>
    </fx:Declarations>

    <s:Rect id="rect1" x="10" y="10" width="400" height="400">
        <s:fill>
            <s:SolidColor color="0xFF0000"/>
        </s:fill>
        <s:stroke>
            <s:SolidColorStroke color="0x0000FF" weight="4"/>
        </s:stroke>
    </s:Rect>

    <s:Button id="button1" x="50" y="100" label="Przekształć mnie"
        click="transformer1.play()"/>

    <s:Button id="button2" x="50" y="200" label="Przekształć także mnie"
        click="transformer2.play()"/>
</s:View>
```

Listing 4.8 zawiera element XML `Declarations`, a ten z kolei trzy elementy XML `Sequence` definiujące trzy efekty przekształcenia. Wykonywanie animacji rozpoczyna się od elementu XML `Move` (efekt przesunięcia), następnym jest element XML `Rotate` (efekt obrotu) i wreszcie element XML `Scale` (efekt zmiany rozmiaru). Kiedy użytkownik stuknie pierwszy element XML `Button`, zostaną wywołane animacje zdefiniowane w elemencie XML `Sequence` o wartości atrybutu `id` równej `transformer1`.

Podobny opis mógłby dotyczyć drugiego elementu XML `Sequence` i drugiego przycisku, poza różnicą polegającą na tym, że animacja obejmuje dwa obroty zamiast jednego.

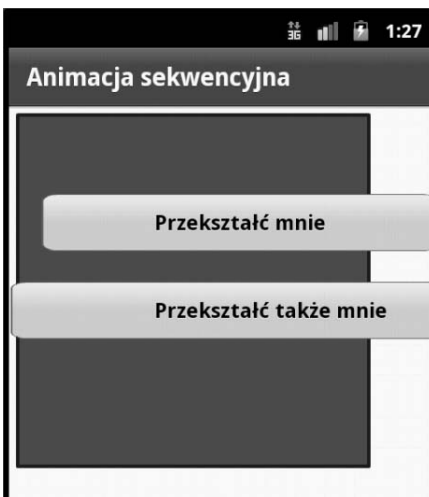
Warto zauważyć, że można łatwo zmienić efekt animacji z sekwencyjnego na równoległy poprzez zmianę elementu XML `Sequence` na element XML `Parallel`, tak jak poniżej:

```
<s:Parallel id="transformer" target="{button}">
  <s:Move xFrom="50" xTo="150"
    autoCenterTransform="true"/>
  <s:Rotate angleFrom="0" angleTo="360"
    autoCenterTransform="true"/>
  <s:Scale scaleXFrom="1" scaleXTo="2"
    autoCenterTransform="true"/>
</s:Parallel>
```

Na zrzutach ekranu przedstawionych na rysunkach 4.9 i 4.10 pokazano dwa przyciski animowane w sposób sekwencyjny. Na zrzutach ekranu pokazano tylko początkowy i końcowy stan animacji, a więc uruchom tę aplikację mobilną i sam zobacz, jak wyglądają animacje przesunięcia i obrotu.



Rysunek 4.9. Przycisk animowany sekwencyjnie (stan początkowy)



Rysunek 4.10. Przycisk animowany sekwencyjnie (stan końcowy)

Tworzenie animacji 3D

Flex obsługuje kilka rodzajów animacji 3D, w tym przesuwanie, obracanie i zmiany rozmiaru obrazu (pliku JPG). Efekt „ruchu” 3D polega na przesunięciu obrazu JPG oraz zmianie jego rozmiaru, a efekt zmiany rozmiaru 3D obejmuje zwiększenie (lub zmniejszenie) szerokości i wysokości obrazu JPG od wartości początkowej (zwykle 1) do ostatecznej (która może być większa lub mniejsza od 1). Efekt „obracania” 3D polega na przekształcaniu obrazu w taki sposób, że wydaje się on obracać w trzech wymiarach.

Przykładowy kod w listingu 4.9 pokazuje, jak dodawać efekty animacji 3D przesuwania, obracania i zmian rozmiaru pliku JPG w aplikacji mobilnej.

Listing 4.9. Tworzenie animacji 3D

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark"
        title="Tworzenie efektów 3D">

    <fx:Declarations>
        <s:Move3D id="moveEffect" target="{targetImg}" xBy="100" zBy="100"
            repeatCount="2" repeatBehavior="reverse"
            effectStart="playMoveButton.enabled=false"
            effectEnd="playMoveButton.enabled=true;"/>

        <s:Rotate3D id="rotateEffect" target="{targetImg}"
            angleYFrom="0" angleYTo="360"
            repeatCount="4" repeatBehavior="reverse"
            effectStart="playRotateButton.enabled=false;"
            effectEnd="playRotateButton.enabled=true;"/>

        <s:Scale3D id="atScale" target="{targetImg}"
            scaleXBy="-.45" repeatCount="2"
            repeatBehavior="reverse"
            effectStart="playScaleButton.enabled=false"
            effectEnd="playScaleButton.enabled=true;"/>
    </fx:Declarations>

    <s:VGroup width="100%" height="100%" >
        <s:Image id="targetImg"
            horizontalCenter="0"
            verticalCenter="0"
            source="@Embed(source='images/Cassandra4.jpg')"/>

        <s:HGroup>
            <s:Button id="playMoveButton"
                left="10" bottom="25"
                label="Przesuń"
                click="moveEffect.play();"/>

            <s:Button id="playRotateButton"
                left="110" bottom="25"
                label="Obróć"
                click="rotateEffect.play();"/>

            <s:Button id="playScaleButton"
                left="222" bottom="25"
                label="Skaluj" click="atScale.play();"/>
        </s:HGroup>
    </s:VGroup>
</s:View>
```

```
</s:HGroup>
</s:VGroup>
```

```
</s:View>
```

Na rysunku 4.11 pokazano obraz JPG *Cassandra4.jpg* — zdjęcie Cassandry Chin (córci Stephena China). Obraz ten wykorzystaliśmy w kodzie, za pomocą którego zademonstrowaliśmy efekty animacji 3D.



Rysunek 4.11. Obraz JPG do prezentowania efektów animacji 3D

Kod z listingu 4.9 zawiera element XML `Declarations`, w którym znalazły się trzy elementy określające animacje 3D wraz z trzema elementami XML `Button`. Użytkownik może klikać te przyciski, aby zobaczyć efekt przekształcenia 3D. Elementami tymi są `Move3D`, który wskazuje położenie docelowe za pomocą atrybutów `xBy` i `zBy`, `repeatCount` o wartości 2 (który powoduje dwukrotne wykonanie efektu animacji) oraz `repeatBehavior` o wartości `reverse` (który za każdym razem przekazuje informację o pierwotnym położeniu). Odpowiadający animacji element XML `Button` zawiera atrybut `label` o wartości `Przesuń` i atrybut `click` o wartości `moveEffect.play()` — ta funkcja wywołuje animację ruchu zdefiniowaną za pomocą elementu XML `MoveEffect`, który znajduje się w elemencie XML `Declarations`.

Efekt obrotu jest obsługiwany za pomocą elementu XML `Rotate3D`, którego atrybuty `angleYFrom` i `angleYTo` określają kąty początkowy i końcowy o wartościach odpowiednio: 0 i 360 (oznacza to, że obiekt wykona pełny obrót). Animacja obrotu jest wykonywana czterokrotnie. Element XML `Button` zawiera atrybut `label` o wartości `Obróć` i atrybut `click` o wartości `rotateEffect.play()` — ta funkcja wywołuje animację obracania zdefiniowaną za pomocą elementu XML `Rotate3D`, który znajduje się w elemencie XML `Declarations`.

Efekt zmiany rozmiaru (trzeci i jednocześnie ostatni) jest obsługiwany za pomocą elementu XML `Scale3D`, który zawiera kilka atrybutów o wartościach określających szczegóły animowania obrazu JPG. Atrybut `id` ma wartość `atScale` i służy do tworzenia referencji do tego elementu w innych miejscach kodu. Atrybut `target` jest referencją do elementu XML, którego atrybut `id` ma wartość `targetImg`, co wskazuje na obraz JPG. Atrybut `scaleXBy` ma wartość `-0.25`, co powoduje zmniejszenie obrazu JPG o 25%. Atrybut `RepeatCount` ma wartość 4, a `repeatBehavior` wartość `reverse`, co oznacza, że efekt animacji będzie odtwarzany czterokrotnie, na przemian od lewej do prawej i odwrotnie. Dwa pozostałe atrybuty to `effectStart` i `effectEnd` wskazujące zachowania na początku i na końcu animacji, w tym przypadku polegają one na wyłączeniu i włączeniu przycisku `playButton`.

Zauważ, że element XML Image wskazuje położenie obrazu *Cassandra4.jpg* w folderze, który znajduje się w katalogu najwyższego poziomu projektu. Aby mógł powstać układ, element XML Image znalazł się wewnątrz elementu XML VGroup, który zawiera element XML HGroup, gdzie z kolei znalazły się trzy elementy XML Button.

Na rysunku 4.12 pokazano obraz JPG po wykonaniu animacji 3D.



Rysunek 4.12. Obraz JPG po wykonaniu animacji 3D

Na rysunku 4.13 pokazano obraz JPG po wykonaniu animacji obrotu 3D.



Rysunek 4.13. Obraz JPG po wykonaniu animacji obrotu 3D

Na rysunku 4.14 pokazano obraz JPG po wykonaniu animacji zmiany rozmiaru 3D.



Rysunek 4.14. Obraz JPG po wykonaniu animacji zmiany rozmiaru 3D

Tworzenie skórek Spark

Własne skórki będą przydatne za każdym razem, kiedy zechcesz zapewnić użytkownikowi ciekawsze efekty wizualne dla określonych części aplikacji mobilnych. Na przykład możesz utworzyć kilka własnych skórek, za pomocą których wzbogacisz przyciski o efekty graficzne (w tym te, które poznałeś wcześniej w tym rozdziale). Poniżej zamieściliśmy przykładowy fragment kodu, na podstawie którego omówimy proces tworzenia własnych skórek Spark.

W listingach od 4.10 do 4.12 pokazaliśmy zawartość plików: *CustomSkinHomeView.xml*, *ButtonSkin1.xml* oraz *ButtonSkin2.xml*.

Przed omówieniem plików MXML pokazanych w tym podrozdziale przyjrzyjmy się czynnościom, jakie należy wykonać, aby dodać do projektu plik *ButtonSkin1.xml* (w pakiecie skins).

1. Dodaj folder skins do projektu.
2. Kliknij nazwę projektu prawym przyciskiem myszy i wybierz *New/MXML Skin* (nowy/skórka MXML).
3. Wskaż skins jako nazwę pakietu nowej skórki.
4. Wpisz *ButtonSkin1* jako nazwę skórki.
5. Wskaż *spark.components.Button* jako nazwę komponentu.
6. Usuń zaznaczenie obok pola *Create as a copy of*: (utwórz jako kopię).

Powtórz powyższe czynności dla skórki *ButtonSkin2.xml*, a także dla wszelkich innych własnych skórek, które będziesz chciał dodać do tego projektu. Przyjrzyjmy się teraz zawartości pliku *CustomSkin.xml* (listing 4.10).

Listing 4.10. Tworzenie własnych skórek Spark

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark"
  title="Skórki własne">

<s:VGroup>
  <s:Label text="To jest zwykły przycisk:" x="10" y="0"/>
```

```

<s:Button label="Button1" x="10" y="25"/>

<s:Label text="Pierwszy przycisk ze skórką:" x="10" y="60"/>
<s:Button skinClass="skins.ButtonSkin1" x="10" y="85"/>

<s:Label text="Drugi przycisk ze skórką:" x="10" y="100"/>
<s:Button skinClass="skins.ButtonSkin2" x="10" y="125"/>

<s:Label text="Trzeci przycisk ze skórką:" x="10" y="140"/>
<s:Button skinClass="skins.ButtonSkin1" x="10" y="165"/>

<s:Label text="Czwarty przycisk ze skórką:" x="10" y="180"/>
<s:Button skinClass="skins.ButtonSkin2" x="10" y="205"/>
</s:VGroup>
</s:View>

```

Kod w listingu 4.10 zawiera element XML VGroup z dziesięcioma ułożonymi w pary elementami XML, które renderują standardowe elementy XML Label i Button. Pierwszy przycisk jest zwykłym przyciskiem, definiowanym przez poniższy wycinek kodu:

```

<s:Label text="To jest zwykły przycisk:" x="10" y="0"/>
<s:Button label="Button1" x="10" y="25"/>

```

Działanie powyższych elementów XML jest proste: pierwszy to etykieta (*To jest zwykły przycisk*), a drugi renderuje przycisk.

Omówimy teraz pary elementów XML związane z przyciskami, na które nałożyliśmy skórki. Pierwszy element z pary wyświetla etykietę z tekstem *Pierwszy przycisk ze skórką.*, podczas gdy drugi renderuje element XML Button na podstawie zawartości skórki ButtonSkin1 w pakiecie skins. Podobnie działa druga para elementów XML: pierwszy element wyświetla etykietę z tekstem *Drugi przycisk ze skórką.*, a drugi renderuje element XML Button na podstawie zawartości skórki ButtonSkin2 w pakiecie skins. Podobna sytuacja zachodzi w przypadku dwóch pozostałych przycisków.

Przyjrzymy się teraz zawartości pliku *ButtonSkin1.mxml*, w którym znalazły się dane służące do renderowania drugiego przycisku (pierwszego ze skórką). Kod pokazano w listingu 4.11.

Listing 4.11. Tworzenie skórki przycisku z grafiką

```

<?xml version="1.0" encoding="utf-8"?>
<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:s="library://ns.adobe.com/flex/spark">

  <fx:Metadata>
    [HostComponent("spark.components.Button")]
  </fx:Metadata>

  <s:states>
    <s:State name="disabled" />
    <s:State name="down" />
    <s:State name="over" />
    <s:State name="up" />
  </s:states>

  <s:Rect id="rect1" x="0" y="0" height="40" width="100">
    <s:fill>
      <s:LinearGradient>
        <s:GradientEntry color="0xFF0000"
          ratio="0" alpha=".5"/>
        <s:GradientEntry color="0xFFFF00"
          ratio=".33" alpha=".5"/>

```

```

        <s:GradientEntry color="0x0000FF"
            ratio=".66" alpha=".5"/>
    </s:LinearGradient>
</s:fill>

    <s:stroke>
        <s:SolidColorStroke color="0x000000" weight="2"/>
    </s:stroke>
</s:Rect>
</s:Skin>

```

Listing 4.11 zawiera węzeł podstawowy XML Skin z trzema elementami potomnymi XML definiującymi zachowanie skórki. Pierwszym z tych elementów jest element XML Metadata:

```

<fx:Metadata>
    [HostComponent("spark.components.Button")]
</fx:Metadata>

```

Powyższy element XML wskazuje nazwę pakietu klasy Button. Nazwę tę podałeś podczas dodawania pliku ButtonSkin1.mxml do projektu.

Drugi element potomny XML to element states:

```

<s:states>
    <s:State name="disabled" />
    <s:State name="down" />
    <s:State name="over" />
    <s:State name="up" />
</s:states>

```

Element XML states zawiera cztery elementy potomne odpowiadające stanom przycisku i trzem zdarzeniom związanym z zachowaniem myszy. Jeśli chcesz zapewnić obsługę tych stanów, możesz dodać odpowiedni kod. Trzecim elementem potomnym jest element XML Rect definiujący gradient liniowy, co zapewni efekt cieniowania oraz czarne obramowanie. Listing 4.12 zawiera kod dla drugiej skórki przycisku.

Listing 4.12. Tworzenie drugiej skórki przycisku

```

<s:Skin xmlns:fx="http://ns.adobe.com/mxml/2009"
    xmlns:s="library://ns.adobe.com/flex/spark" >

    <fx:Metadata>
        [HostComponent("spark.components.Button")]
    </fx:Metadata>
    <s:states>
        <s:State name="disabled" />
        <s:State name="down" />
        <s:State name="over" />
        <s:State name="up" />
    </s:states>

    <s:Path data="M 0 0 L 100 0 L 100 40 L 0 40 Z ">
        <s:fill>
            <s:SolidColor color="#FF0000" alpha="1"/>
        </s:fill>
        <s:stroke>
            <s:SolidColorStroke color="#0000FF" weight="4"/>
        </s:stroke>
    </s:Path>
</s:Skin>

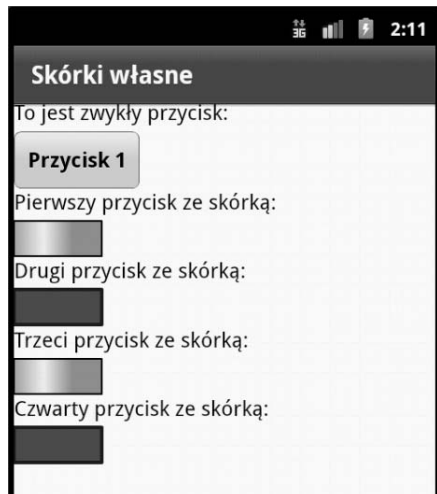
```

Zapewne zauważyłeś, że jedyną różnicą pomiędzy listingami 4.12 i 4.11 jest użycie elementu XML Path zamiast Rect.

Element XML Path jest prosty: zawiera atrybut data, którego wartością jest zbiór segmentów linii definiujących prostokąt w kolorze #FF0000 (czyli czerwonym), z obramowaniem w kolorze #0000FF (w niebieskim) i o szerokości linii 4.

Jak możesz zobaczyć, Flex znacznie ułatwia definiowanie własnych skórek. Jednak w przypadku bardziej złożonych (i bardziej interesujących) skórek często określa się zachowanie po wywołaniu zdarzeń myszy (kliknięcia, zwolnienia przycisku itp.) oraz odpowiadających im zdarzeń dotykowych jako zmiany stanów. Te zmiany stanów możesz powiązać z funkcjami ActionScript (które możesz napisać samodzielnie), tak aby były wykonywane podczas tych zdarzeń, co zwiększy wizualną atrakcyjność aplikacji.

Na rysunku 4.15 pokazano standardowy przycisk Flex i cztery przyciski z własnymi skórkami.



Rysunek 4.15. Standardowy przycisk i cztery przyciski z własnymi skórkami Spark

Generowanie wykresów 2D w Spark

Flex 4 umożliwia eleganckie tworzenie następujących rodzajów wykresów 2D:

- wykresy warstwowe,
- wykresy kolumnowe,
- wykresy słupkowe,
- wykresy liniowe,
- wykresy kołowe,
- wykresy punktowe.

Za pomocą przedstawionych niżej przykładów pokażemy Ci, jak pisać kod aplikacji mobilnych renderujący wykresy słupkowe 2D i wykresy kołowe 2D. Dowiesz się także, jak można dodawać do wykresów animacje oraz obsługę zdarzeń myszy i zdarzeń dotykowych. Zwróć uwagę, że we Fleksie używamy pojęcia „wykres słupkowy” na określenie wykresu słupkowego narysowanego poziomo (każdy słupek jest renderowany poziomo od lewej do prawej) i pojęcia „wykres kolumnowy” na określenie wykresu, gdzie poszczególne „słupki” są narysowane pionowo — jak kolumny.

Tworzenie wykresów słupkowych 2D

Wykresy słupkowe są bardzo popularne, szczególnie w aplikacjach biznesowych. Ułatwiają dostrzeganie trendów zmian, które trudno byłoby zauważyć, analizując dane przedstawione w zwykłej tabeli. W następnym przykładzie pokażemy Ci, jak utworzyć aplikację mobilną, która odczyta dane zapisane w formacie XML z dokumentu XML, a następnie wyrenderuje te dane w postaci wykresu słupkowego 2D. Te dane służą tylko celom demonstracyjnym i na pewno zechcesz wykorzystać własne. Pamiętaj tylko, że pełny kod źródłowy przykładów prezentowanych na kartach tej książki możesz pobrać z jej strony internetowej, www.helion.pl/ksiazki/andfzp.htm.

Utwórz nowy projekt mobilny Flex, nadaj mu nazwę BarChart1 — skorzystaj z szablonu aplikacji mobilnej. Do projektu dodaj nowy folder najwyższego poziomu o nazwie chartdata, a następnie umieść w nim nowy dokument XML o nazwie ChartData.xml, zawierający dane pokazane w listingu 4.13.

Listing 4.13. Definiowanie danych XML dla wykresu

```
<?xml version="1.0"?>
<chartdata>
  <data>
    <month>Styczeń</month>
    <revenue>1500</revenue>
  </data>
  <data>
    <month>Luty</month>
    <revenue>1400</revenue>
  </data>
  [część danych pominięto]
  <data>
    <month>Listopad</month>
    <revenue>1900</revenue>
  </data>
  <data>
    <month>Grudzień</month>
    <revenue>1800</revenue>
  </data>
</chartdata>
```

Listing 4.13 zawiera element XML chartdata z dwunastoma elementami XML data; każdy z nich przechowuje dane dotyczące jednego miesiąca. Każdy element XML data w listingu 4.13 zawiera elementy XML month oraz revenue. Przykładowo: pierwszy element XML data zawiera element revenue o wartości 1500 oraz element month o wartości Styczeń (nie wskazujemy jednostki waluty).

Przyjrzyj się teraz listingowi 4.14, który zawiera kod renderujący wykres słupkowy z wykorzystaniem danych z dokumentu XML pokazanego w listingu 4.13.

Listing 4.14. Tworzenie wykresu słupkowego

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/mx"
  xmlns:s="library://ns.adobe.com/flex/spark"
  title="Wykres słupkowy">

  <!-- dane XML dla wykresu -->
  <fx:Declarations>
    <fx:Model id="chartModel" source="chartdata/ChartData.xml"/>
    <s:ArrayCollection id="chartData" source="{chartModel.data}"/>
    <mx:NumberFormatter id="nf" precision="1" rounding="nearest"/>
  </fx:Declarations>
```

```

<fx:Style>
  @namespace s "library://ns.adobe.com/flex/spark";
  @namespace mx "library://ns.adobe.com/flex/mx";
  mx|ColumnChart
  {
    fontSize:12;
    fontWeight:bold;
  }
</fx:Style>

<!-- wskazanie słupka wykresu z odpowiednimi atrybutami -->
<mx:ColumnChart dataProvider="{chartData}"
  height="70%" width="100%">
  <mx:horizontalAxis>
    <mx:CategoryAxis dataProvider="{chartData}"
      categoryField="month"/>
  </mx:horizontalAxis>
  <mx:series>
    <mx:ColumnSeries xField="month" yField="revenue"/>
  </mx:series>
</mx:ColumnChart>
</s:View>

```

Kod w listingu 4.14 definiuje położenie dokumentu XML `ChartData.xml` za pomocą elementu XML `Model`, wraz z obiektem `ArrayCollection` zawierającym dane z XML. Tworzy także prosty formater danych. Ten sam listing zawiera także element XML `Style` definiujący wartości dwóch atrybutów CSS: `fontSize` oraz `fontWeight` — odpowiednio 12 i `bold`. Służą one do formatowania tekstu w wykresie słupkowym.

Element XML `ColumnChart` definiuje wykres słupkowy wraz z potrzebnymi wartościami atrybutów `dataProvider`, `height` i `width` — odpowiednio `chartData`, 75% i 80%. Wartość `chartData` jest zmienną typu `ArrayCollection`, która została zdefiniowana w elemencie XML `Declarations`. Zmienna `chartData` jest wypełniana wartościami danych wskazanych w dokumencie XML `ChartData.xml`.

Wartości atrybutów `height` i `width` zostały określone jako wartości procentowe wymiarów ekranu, na którym jest renderowany wykres słupkowy. Możesz je dopasować odpowiednio do obszaru, jaki ma zajmować wykres (50% dla połowy szerokości lub wysokości, 25% dla jednej czwartej itp.).

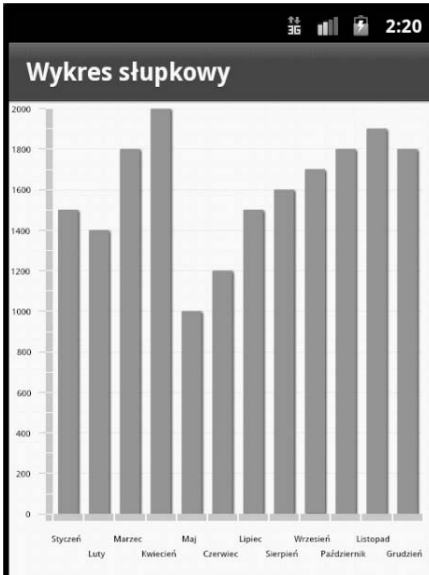
Element XML `ColumnChart` zawiera dwa istotne elementy. Pierwszy z nich to element XML `horizontalAxis` określający wartości zmiennych `month` (wskazane w pliku `ChartData.xml`) dla osi poziomej. Drugi to element XML `series`, który odnosi się do wartości zmiennych `month` dla osi poziomej i jednocześnie zmiennych `revenue` dla osi pionowej wykresu słupkowego.

Na rysunku 4.16 pokazano wykres słupkowy, który powstał na podstawie danych XML z pliku `CharData.xml` z listingu 4.13.

Warto pamiętać, że na zrzucie ekranu z rysunku 4.16 brakuje pewnych istotnych informacji, takich jak waluta, w której wyrażone są wielkości dochodu, bieżący rok, nazwa i adres firmy, region (lub państwo). Jeśli będziesz chciał dodać informacje tego rodzaju, wprowadź potrzebne zmiany w kodzie z listingu 4.14. W ten sposób upewnisz się, że kod wskazuje właściwą ścieżkę dostępu do danych.

Tworzenie wykresów kołowych 2D

Wykresy kołowe są bardzo popularnym sposobem przedstawiania danych — na takim wykresie łatwo dostrzec relacje pomiędzy danymi. Utworzymy więc wykres kołowy, korzystając z danych XML z dokumentu `CharData.xml`, który pokazaliśmy w listingu 4.13. Są to więc te same dane, którymi się posłużyliśmy, rysując wykres słupkowy opisany w poprzednim punkcie. Utwórz nowy projekt mobilny Flex, nadaj mu nazwę `PieChart1` — skorzystaj z szablonu aplikacji mobilnej. Do projektu dodaj kod pokazany w listingu 4.15.



Rysunek 4.16. Wykres słupkowy 2D

Listing 4.15. Tworzenie wykresu kołowego

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
  xmlns:mx="library://ns.adobe.com/flex/mx"
  xmlns:s="library://ns.adobe.com/flex/spark"
  title="Wykres kołowy">

  <!-- dane XML dla wykresu -->
  <fx:Declarations>
    <fx:Model id="chartModel" source="chartdata/ChartData.xml"/>
    <s:ArrayCollection id="chartData" source="{chartModel.data}"/>
    <mx:NumberFormatter id="nf" precision="1" rounding="nearest"/>
  </fx:Declarations>

  <fx:Style>
    @namespace s "library://ns.adobe.com/flex/spark";
    @namespace mx "library://ns.adobe.com/flex/mx";
    mx|PieChart
    {
      fontSize:12;
      fontWeight:bold;
    }
  </fx:Style>

  <!-- dane dla poszczególnych części - pary nazwa:wartość -->
  <fx:Script>
    <![CDATA[
      private function getWedgeLabel (item:Object,
        field:String,
        index:Number,
        percentValue:Number):String {
        return item.month+": "+item.revenue;
      }
    ]]>
  </fx:Script>
</s:View>
```

```

    ]]>
</fx:Script>

<!-- definiowanie wykresu kołowego z odpowiednimi atrybutami -->
<mx:PieChart dataProvider="{chartData}"
    height="100%" width="100%"
    horizontalCenter="0" verticalCenter="-150">
    <mx:series>
        <mx:PieSeries field="revenue"
            labelFunction="getWedgeLabel"
            labelPosition="callout"
            explodeRadius="0.05"/>
    </mx:series>
</mx:PieChart>
</s:View>

```

W kodzie z listingu 4.15 znalazł się element XML `Declarations` oraz element XML `Style` — takie same jak w listingu 4.14. Element XML `Script` definiuje funkcję prywatną `getWedgeLabel()`. Funkcja ta przekazuje ciąg znaków zawierający parę *nazwa:wartość* dla każdej części wykresu:

```

<fx:Script>
    <![CDATA[
        private function getWedgeLabel (item:Object,
            field:String,
            index:Number,
            percentValue:Number):String {
            return item.month+": "+item.revenue;
        }
    ]]>
</fx:Script>

```

Element XML `PieChart` definiuje wykres kołowy wraz z atrybutami, których wartości wskazują, w jaki sposób ten wykres będzie renderowany. Na przykład: oba atrybuty `height` i `width` mają wartość 80%, co oznacza, że wykres będzie renderowany z wysokością i szerokością równą 80% wymiarów ekranu. Możesz je dopasować odpowiednio do obszaru, jaki ma zajmować wykres (tak jak dla wykresu słupkowego).

Element XML `PieChart` zawiera także element XML `PieSeries`, w którym z kolei znajdują się cztery atrybuty umożliwiające określenie, w jaki sposób będą renderowane dane wykresu i jego poszczególne części. Atrybut `field` ma wartość `revenue`, co oznacza, że wartości danych elementu XML `revenue` będą renderowane na wykresie kołowym.

Atrybut `labelFunction` ma wartość `getWedgeLabel()`, która jest nazwą funkcji `ActionScript` (zdefiniowanej wcześniej w elemencie `fx:Script`). Funkcja ta tworzy etykiety dla każdego elementu wykresu.

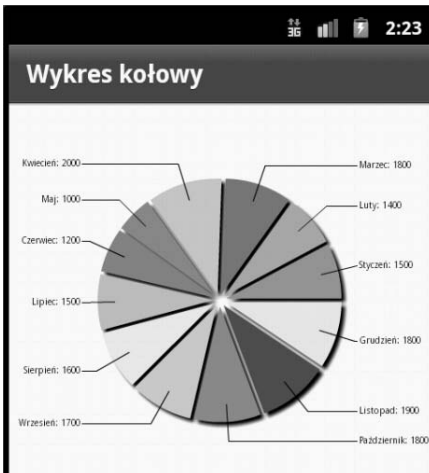
Atrybut `labelPosition` ma wartość `callout`, co oznacza, że etykieta dla każdego elementu wykresu jest renderowana na zewnątrz elementu z łamaną linią łączącą element z etykietą. Warto zauważyć, że atrybut `labelPosition` może mieć trzy inne wartości: `inside`, `outside` i `insideWithCallout`. Przetestuj je i zobacz, jaki mają wpływ na wygląd wykresu.

Na zakończenie: atrybut `explodeRadius` ma wartość 0.5, dzięki czemu poszczególne elementy wykresu rozdzielone są odstępami, co daje efekt „eksplozji”.

Na rysunku 4.17 pokazano wykres kołowy 2D.

Wykorzystywanie FXG i Spark

W rozdziale 3. zawarliśmy bardzo krótkie wprowadzenie do FXG. W tym podrozdziale zademonstrujemy fragment kodu, dzięki któremu dowiesz się, jak przekształcić kod z listingu 4.1 (renderujący prostokąt i elipsę) do projektu Flex wykorzystującego FXG.



Rysunek 4.17. Wykres kołowy 2D

Utwórz nowy projekt mobilny Flex, nadaj mu nazwę FXG1 — skorzystaj z szablonu aplikacji mobilnej, utwórz folder na najwyższym poziomie projektu i nadaj mu nazwę components, następnie utwórz wewnątrz tego folderu plik RectEllipse1.fwg z zawartością pokazaną w listingu 4.16.

Listing 4.16. Wykorzystanie FXG do zdefiniowania elementów graficznych

```
<?xml version="1.0" encoding="utf-8"?>
<Graphic xmlns="http://ns.adobe.com/fwg/2008" version="2">
  <Rect id="rect1" x="10" y="10" width="250" height="200">
    <fill>
      <SolidColor color="#FF0000"/>
    </fill>
    <stroke>
      <SolidColorStroke color="#FFFF00" weight="4"/>
    </stroke>
  </Rect>

  <Ellipse id="ellipse1" x="10" y="220" width="250" height="200">
    <fill>
      <SolidColor color="#0000FF"/>
    </fill>
    <stroke>
      <SolidColorStroke color="#FF0000" weight="4"/>
    </stroke>
  </Ellipse>

  <Rect id="rect2" x="10" y="460" width="250" height="100">
    <fill>
      <SolidColor color="#FFFF00"/>
    </fill>
    <stroke>
      <SolidColorStroke color="#0000FF" weight="8"/>
    </stroke>
  </Rect>
</Graphic>
```

Element XML `Graphic` zawiera dwa elementy XML, których wartości danych są takie same jak wartości elementu XML `Rect` i elementu XML `Ellipse` z listingu 4.1. Różnice między tymi elementami są następujące:

- Nazwy elementów nie posiadają przedrostków określających przestrzeń nazw.
- Nazwy elementów należą do domyślnej przestrzeni nazw.
- W atrybucie `color` znalazł się symbol `#` zamiast przedrostka `0x`.

Kod w listingu 4.17 stanowi przykład, w jaki sposób można tworzyć referencje do elementu zdefiniowanego w listingu 4.16.

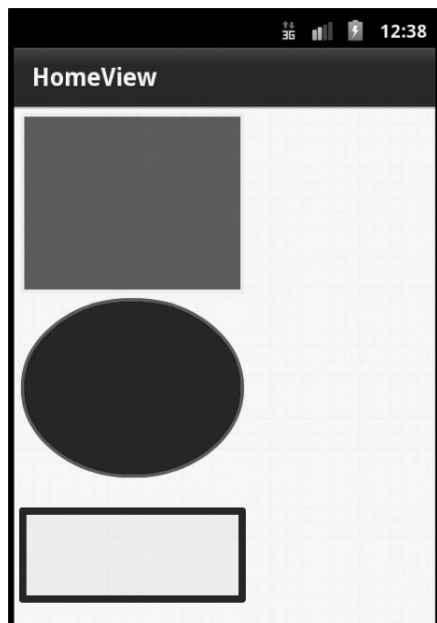
Listing 4.17. Tworzenie referencji do komponentów FXG

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:mx="library://ns.adobe.com/flex/mx"
        xmlns:s="library://ns.adobe.com/flex/spark"
        xmlns:comps="components.*">

    <s:VGroup>
        <comps:RectEllipse1 id="rect1"/>
    </s:VGroup>
</s:View>
```

Kod z listingu 4.17 określa przestrzeń nazw, za pomocą której można tworzyć referencje do pliku FXG *RectEllipse1.fgx* znajdującego się w podkatalogu `components`. Element XML `VGroup` zawiera element `RectEllipse1` w przestrzeni nazw `comp`. Element ten tworzy referencję do elementu XML, którego atrybut `id` ma wartość `rect1`, zdefiniowanego w pliku *FXG RectEllipse1.fgx* (listing 4.16).

Na rysunku 4.18 pokazano efekt wykonania kodu z listingu 4.16. Wyrenderowana elipsa i dwa prostokąty są takie same jak te na rysunku 4.1.



Rysunek 4.18. Prostokąty i elipsa

Jak mogłeś przekonać się na podstawie tego przykładu, FXG pozwala na tworzenie modularnego kodu w projektach Flex. Ponadto warto wiedzieć, że za pomocą następujących produktów Adobe możesz eksportować projekty jako pliki FXG, które następnie możesz zaimportować do projektów Flex:

- Adobe Photoshop
- Adobe Illustrator
- Adobe Fireworks

Bardziej zaawansowane przykłady zastosowania plików FXG znajdziesz w rozdziale 9.

Program do szkicowania

W tym podrozdziale pokażemy Ci, jak utworzyć kod aplikacji mobilnej — programu do szkicowania. Wykorzystamy w nim łącznie różne techniki związane z grafiką, które opisaliśmy we wcześniejszej części tego rozdziału: obsługę zdarzeń dotykowych, rysowanie na pliku JPG i zapewnienie możliwości zapisywania szkicu na urządzeniu mobilnym jako pliku JPG.

Utwórz zatem nowy projekt mobilny Flex, nadaj mu nazwę Sketch1 — skorzystaj z szablonu aplikacji mobilnej i dodaj do niego kod pokazany w listingu 4.18. Aby ułatwić omawianie aplikacji, zaprezentujemy kod w postaci mniejszych bloków. Pełny kod można pobrać z witryny tej książki.

Listing 4.18. Renderowanie i zapisywanie szkiców

```
<?xml version="1.0" encoding="utf-8"?>
<s:View xmlns:fx="http://ns.adobe.com/mxml/2009"
        xmlns:s="library://ns.adobe.com/flex/spark" title="HomeView">
<fx:Script>
    <![CDATA[
        import flash.ui.Multitouch;
        import flash.ui.MultitouchInputMode;
        import flash.events.TouchEvent;

        import mx.graphics.ImageSnapshot;
        import mx.graphics.SolidColor;
        import mx.graphics.codec.JPEGEncoder;

        private var colors:Array = [0xFF0000, 0x00FF00, 0xFFFF00, 0x0000FF];
        private var singleTapCount:int = 0;
        private var touchMoveCount:int = 0;
        private var widthFactor:int = 0;
        private var heightFactor:int = 0;
        private var currentColor:int = 0;
        private var rectWidth:int = 20;
        private var rectHeight:int = 20;

        Multitouch.inputMode = MultitouchInputMode.TOUCH_POINT;

        function touchMove(event:TouchEvent):void {
            ++touchMoveCount;

            if (event.isPrimaryTouchPoint) {
                currentColor = colors[touchMoveCount%colors.length];
            } else {
                currentColor = colors[(touchMoveCount+2)%colors.length];
            }
        }
    ]]>
</fx:Script>
</s:View>
</?xml>
```

```

var myRect:Rect = new Rect();
myRect.x = event.localX;
myRect.y = event.localY;
myRect.width = rectWidth;
myRect.height = rectHeight;
myRect.fill = new SolidColor(currentColor);

var myGroup1:Group = event.target as Group;
myGroup1.addElement(myRect);
}

```

Kod z listingu 4.18 rozpoczyna się od elementu XML Script, który zawiera instrukcje importu i definicje odpowiednio nazwanych zmiennych (np. służących do śledzenia zdarzeń dotykowych) wykorzystywanych w metodach ActionScript 3.

Wartość `MultiTouch.inputMode` ustawiana jest odpowiednio dla trybu wielodotykowego, dzięki czemu renderowany jest więcej niż jeden prostokąt, kiedy użytkownik przeciągnie po ekranie więcej niż jednym palcem. Jeśli musisz odświeżyć swoje informacje na temat wielodotykowości, zajrzyj do odpowiedniego podrozdziału w rozdziale 2.

Funkcja `touchMove()` zawiera kod obsługujący zdarzenia ruchu. Funkcja na początku inkrementuje zmienną `touchMoveCount`, a następnie wykorzystuje ją jako indeks w tablicy `colors`. Dzięki temu renderowany jest ciąg prostokątów, których kolory zmieniają się odpowiednio do iterowanych kolorów w tablicy. Pozostała część kodu funkcji tworzy mały prostokąt w tym położeniu, gdzie zostało zarejestrowane zdarzenie dotykowe. Tę funkcję właściwie można nazwać sercem kodu renderującego grafikę, podczas gdy pozostałe funkcje obsługują inne zdarzenia.

Kolejnym blokiem kodu jest definicja funkcji `touchEnd()`, która jest co prawda opcjonalna, jednak stanowi dobry przykład procedury obsługi zdarzenia.

```

function touchEnd(event:TouchEvent):void {
    ++touchMoveCount;

    if (event.isPrimaryTouchPoint) {
        currentColor = colors[touchMoveCount%colors.length];
    } else {
        currentColor = colors[0];
    }

    widthFactor = (touchMoveCount%3)+1;
    heightFactor = (touchMoveCount%3)+2;

    var myRect:Rect = new Rect();
    myRect.x = event.localX;
    myRect.y = event.localY;
    myRect.width = rectWidth*widthFactor;
    myRect.height = rectHeight*heightFactor;
    myRect.fill = new SolidColor(currentColor);

    var myGroup1:Group = event.target as Group;
    myGroup1.addElement(myRect);
}

```

Kod obsługujący zdarzenia „zwolnienia” palca z wyświetlacza w funkcji `touchEnd()` inkrementuje zmienną `touchMoveCount` i służy ona jako indeks w tablicy `colors`, ale tym razem wykonywana jest prosta operacja arytmetyczna mająca na celu wyrenderowanie prostokąta innej wielkości.

```

function touchSingleTap(event:TouchEvent):void {
    var myRect:Rect = new Rect();
    myRect.x = event.localX;
    myRect.y = event.localY;
}

```



```

++singleTapCount;
if (event.isPrimaryTouchPoint) {
    currentColor = colors[singleTapCount%colors.length];
    myRect.width = rectWidth*3;
    myRect.height = rectHeight*2;
} else {
    currentColor = colors[(singleTapCount+1)%colors.length];
    myRect.width = rectWidth*2;
    myRect.height = rectHeight*3;
}

myRect.fill = new SolidColor(currentColor);

var myGroup1:Group = event.target as Group;
myGroup1.addElement(myRect);
}

```

Logika obsługi zdarzeń pojedynczego stuknięcia znajduje się w funkcji `touchSingleTap()`. Ta funkcja inkrementuje zmienną `touchSingleTapCount`, a następnie stosuje prostą logikę w celu ustalenia wymiarów prostokąta, renderowanego w miejscu, gdzie wystąpiło zdarzenie pojedynczego stuknięcia.

```

function touchMoveHandlerImage(event:TouchEvent):void {
    touchMove(event);
}

function touchTapHandlerImage(event:TouchEvent):void {
    touchSingleTap(event);
}

private function saveImageToFileSystem():void {
    var jpegEncoder:JPEGEncoder = new JPEGEncoder(500);
    var imageSnapshot:ImageSnapshot = ImageSnapshot.captureImage(imgPanel, 0, jpegEncoder);
    var fileReference:FileReference = new FileReference();
    fileReference.save(imageSnapshot.data, "fingersketch.jpg");
}
]]>
</fx:Script>

```

Funkcje `touchMoveHandlerImage()` i `touchTapHandlerImage()` (jak sugerują ich nazwy) obsługują zdarzenia ruchu i pojedynczego stuknięcia dla pliku JPG, który jest przechowywany w podkatalogu `images` aplikacji Flex. Definicje tych dwóch funkcji zawierają pojedynczą linię kodu wywołującą odpowiednio funkcje: `touchMove()` i `touchTapHandler()`, omówione we wcześniejszej części tego podrozdziału.

Funkcja `saveImageToFileSystem()` jest wywoływana za każdym razem, kiedy użytkownik kliknie przycisk *Zapisz szkic*, i zawiera kod zapisujący edytowany szkic w systemie plików urządzenia mobilnego. Zostanie także wyświetlone wyskakujące okno z domyślnym położeniem i nazwą pliku JPG. Oba te elementy użytkownik może zmienić przed zapisaniem pliku.

```

<s:Panel id="imgPanel" title="Szkicowanie palcem dla zabawy!" width="100%" height="100%" >
    <s:Button id="saveImage"
        left="150" bottom="5"
        label="Save Sketch"
        click="saveImageToFileSystem();"/>

    <s:Group name="myGroup1" width="500" height="500"
        touchMove="touchMove(event)"
        touchEnd="touchEnd(event)"
        touchTap="touchSingleTap(event)">
    <s:Ellipse id="ellipse1" x="10" y="10" width="100" height="50">

```

```

<s:fill> <s:SolidColor color="0xFFFF00"/> </s:fill>
<s:stroke> <s:SolidColorStroke color="red" weight="5"/> </s:stroke>
</s:Ellipse>
<s:Rect id="rect1" x="110" y="10" width="100" height="50">
  <s:fill> <s:SolidColor color="0xFF0000"/> </s:fill>
  <s:stroke> <s:SolidColorStroke color="blue" weight="5"/> </s:stroke>
</s:Rect>
<s:Ellipse id="ellipse2" x="210" y="10" width="100" height="50">
  <s:fill> <s:SolidColor color="0xFFFF00"/> </s:fill>
  <s:stroke> <s:SolidColorStroke color="red" weight="5"/> </s:stroke>
</s:Ellipse>
<s:Rect id="rect2" x="310" y="10" width="100" height="50">
  <s:fill> <s:SolidColor color="0xFF0000"/> </s:fill>
  <s:stroke> <s:SolidColorStroke color="blue" weight="5"/> </s:stroke>
</s:Rect>

<s:Path data="C100 300 200 20 300 100 S 250 200 300 250">
  <s:fill>
    <s:LinearGradient rotation="90">
      <s:GradientEntry color="#FF0000" alpha="0.8"/>
      <s:GradientEntry color="#0000FF" alpha="0.8"/>
    </s:LinearGradient>
  </s:fill>

  <s:stroke>
    <s:SolidColorStroke color="0x00FF00" weight="2"/>
  </s:stroke>
</s:Path>

<s:Path data="C 350 300 200 150 350 100 T 250 250 400 280">
  <s:fill>
    <s:LinearGradient rotation="90">
      <s:GradientEntry color="#FFFF00" alpha="0.5"/>
      <s:GradientEntry color="#FF0000" alpha="0.5"/>
    </s:LinearGradient>
  </s:fill>

  <s:stroke>
    <s:SolidColorStroke color="0x000000" weight="4"/>
  </s:stroke>
</s:Path>
</s:Group>

<s:Image id="img" width="480" height="320" source="images/fingersketch.jpg"
  touchMove="touchMoveHandlerImage(event)"
  touchTap="touchTapHandlerImage(event)"
  horizontalCenter="-10" verticalCenter="60"/>
</s:Panel>
</s:View>

```

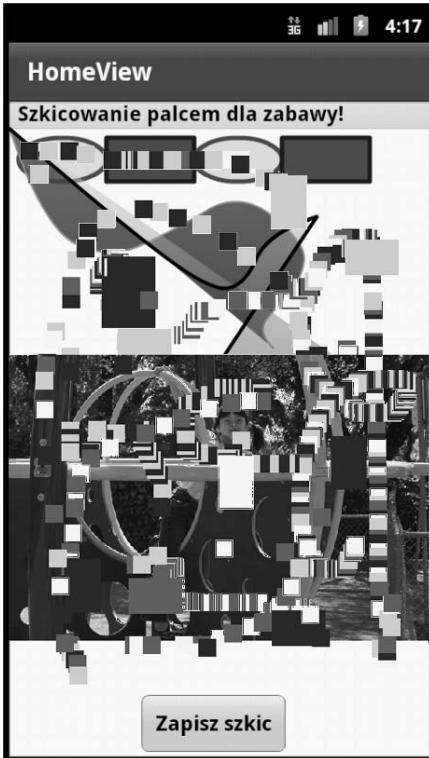
Kolejny duży blok kodu składa się z elementu XML Panel zawierającego element XML Button, który służy do zapisywania bieżącego szkicu, a następnie elementu XML Group definiującego procedury obsługi zdarzeń dotykowych touchMove, touchEnd i touchTap. Zdarzenia te omówiliśmy już wcześniej.

Element XML Group zawiera także definicje elementów różnych obiektów graficznych, w tym elips, prostokątów i krzywych Béziera — opisywaliśmy je we wcześniejszej części tego rozdziału.

Obiekty graficzne są oczywiście opcjonalne i mają tylko na celu pokazanie Ci, jak utworzyć program do szkicowania — program, który będzie atrakcyjny i spodoba się użytkownikom.

Element XML Image wskazuje plik JPG *fingersketch.jpg* w podkatalogu *images* aplikacji Flex. Element XML Image wskazuje funkcję `touchMoveHandlerImage()` dla dotykowych zdarzeń ruchu i funkcję `touchTapHandlerImage()` dla zdarzeń stuknięć. Wypróbuj różne wartości atrybutów `horizontalCenter` i `verticalCenter`, które zmieniają poziome i pionowe położenie układu obrazu JPG.

Na rysunku 4.19 pokazano przykładowy szkic po uruchomieniu programu na urządzeniu mobilnym.



Rysunek 4.19. Przykładowy szkic

Podsumowanie

Podczas lektury tego rozdziału dowiedziałeś się, jak używać komponentów Spark do renderowania różnych obiektów graficznych 2D na potrzeby aplikacji mobilnych zorientowanych graficznie. Jeśli już znasz możliwości renderowania grafiki we Fleksie, możesz łatwo poszerzyć swoją wiedzę, aby tworzyć aplikacje mobilne wykorzystujące grafikę.

Obrazy i efekty graficzne, które możesz wykorzystać, zależą od wymagań aplikacji. Oto niektóre z efektów, jakie możesz zastosować w aplikacjach mobilnych:

- Renderowanie podstawowych obiektów, takich jak prostokąty, elipsy i segmenty linii.
- Renderowanie kwadratowych i sześciennych krzywych Béziera dla aplikacji mobilnych wymagających bardziej wymyślnych, nieliniowych efektów wizualnych.
- Stosowanie gradientów liniowych, radialnych i efektów filtrów w celu uzyskania bardziej złożonych i atrakcyjnych efektów.
- Stosowanie przekształceń (przesuwanie, zmiany rozmiarów, obracanie i zniekształcanie).

- Tworzenie własnych skórek w celu zmiany standardowego wyglądu przycisków.
- Wizualizowanie danych za pomocą wykresów słupkowych i kołowych.
- Używanie animacji równoległych i sekwencyjnych w połączeniu ze zdarzeniami dotykowymi.

Skorowidz

A

- ADB, Android Debug Bridge, 153, 205
- ADL, AIR Debug Launcher, 25, 36, 149
 - obsługa argumentów, 36
 - składnia, 36
- Adobe
 - AIR, 20, 173, 179
 - Device Central, 290
 - Fireworks, 23, 300
 - Flash Builder, 22–34, 149, 297
 - kreator eksportu, 166
 - uprawnienia, 157
 - Flash Builder 4.5, 303
 - Flash Catalyst, 23
 - Flash CS5.5, 56
 - Flash Player, 19, 237, 323
 - Flash Professional, 22, 36, 149, 157, 296
 - publikowanie aplikacji, 165
 - Flash Professional CS3, 296
 - Flash Professional CS5.5, 302
 - Flex, 21, 26, 101, 167
 - Flex 4.5 SDK, 36
 - Illustrator, 23, 299
 - InDesign, 23
 - Photoshop, 23, 161, 295
 - Photoshop CS5, 295
- ADT, AIR Debug Tool, 154
- ADT, AIR Developer Tool, 156, 167
- ADT, Android Development Tools, 151
- ADV, Android Virtual Device, 151
- AIR, 153, 170, 232
- AIR for TV, 352
- AIR SDK, 153
- akcelerometr, 228
- aktualizowanie percentComplete, 268
- aktywność Activity, 186
- algorytm odświeżania, 329
- Android, 18
- Android Emulator, 149
- Android SDK, 150, 183, 185
- animacja, 125, 128
 - obrotu 3D, 130, 132
 - Rect i Ellipse, 316
 - ruchu, 131
 - prostokąta, 126
 - przycisku, 129
 - twarzy, 56
 - zmiany rozmiaru 3D, 132
- animacje
 - równoległe, 128
 - sekwencyjne, 128
- aparatus fotograficzny, 19, 212
- API punktu dotykowego, 66
- APK, Android Package File, 166
- aplikacja
 - AccelerometerBasic, 231
 - ADB, 153, 205
 - ADT, 154, 205
 - Adobe User Group, 320
 - Caterpillar Generator, 68–71
 - Density Explorer, 43, 47, 54
 - Flash Capability Reporter, 24
 - Flash Mobile Bench, 313
 - Flash Scrapbook, 60, 63, 66
 - GeolocationBasic, 234
 - Gesture Check, 26, 32, 167
 - Hi-ReS! Stats, 326
 - MediaPlayer, 283
 - MicrophoneBasic, 211
 - MediaPlayer, 336, 345
 - MediaPlayerWithStates, 344
 - PerformanceTest, 327
 - PhotoCollage, 223, 225
 - PlayBook, 353
 - SoundEffectBasic, 239
 - SoundRecorder, 245, 285
 - TabbedViewNavigatorApplication, 78
 - VideoRecorder, 285
 - ViewNavigatorApplication, 75, 255
- aplikacje
 - AIR, 19, 170, 353
 - Flash, 18
 - Flex, 167

aplikacje
 mobilne, 38
 mobilne Flex, 74, 83
 na Androida, 185
 natywne, 19
 połączone, 205
 sprawdzające możliwości
 urządzenia, 24
 uruchamiane w emulatorze,
 168
 architektura
 systemu Android, 19
 zorientowana na usługi, 188
 argumenty certyfikatu, 164
 arkusz stylów, 47, 49
 atrybut labelFunction, 140
 automatyczna
 reorientacja ekranu, 52
 reorientacja we Flashu, 55
 automatyczne
 pozycjonowanie, 55
 skalowanie, 55
 wdrożenie, 166
 zmienianie trybów, 55
 automatyczny zapis stanu, 93
 AVD Manager, 151

B

balans dźwięku, 253
 baza danych SQLite, 173, 179
 biblioteka
 Metaphile, 269
 OSMF, 276
 Spark, 75, 125
 Bluetooth, 20
 błędy, 182
 buforowanie bitmap, 311

C

certyfikat, 162, 163, 207
 argumenty, 164
 dla aplikacji Flex, 207
 charakterystyki wyświetlaczy, 40
 CheckBox, 28
 CSS, 47, 49

cykl życia aplikacji, 190
 czas
 przetwarzania, 308
 wykonywania kodu, 306, 309
 częstotliwości odtwarzania, 248

D

Dalvik, 19
 dane lokalizacyjne, 232
 debugger Flash Builder, 31
 debugowanie, 225
 debugowanie USB, 35
 definiowanie elementów
 graficznych, 141
 deklaracja
 interfejsu SongView, 263
 klasy SongViewModel, 266
 ProgressBarSkin, 273
 ViewMenu, 90
 deklaracje komponentów, 78
 delegowanie SWF, 331
 deskryptor aplikacji, 28
 detektor zdarzenia, 60
 Panorama, 64
 Stuknięcie palcami, 64
 Device Central, 22, 293
 dips, device independent pixels, 42
 dodawanie detektorów zdarzeń, 64
 dostęp do
 akceleracji sprzętowej, 275
 bazy danych, 181
 strumienia wideo, 215
 dpi, dots per inch, 40, 47
 dwukierunkowe wiązanie, 44
 dynamika przejść, 87
 dyrektywa @Embed, 239

E

Eclipse, 185
 efekt
 animacji, 125
 obrotu, 131
 przekształcenia 3D, 131
 zmiany rozmiaru, 123, 131
 efekty dźwiękowe, 237

ekran
 gęstość, 40
 reorientacja automatyczna, 52
 rozdzielczość, 40
 rozmiar, 40
 specyfikacje, 336
 eksportowanie projektu, 143
 element
 action, 188
 Animate, 126
 application, 194
 Button, 146, 197
 category, 188
 chartdata, 137
 ColumnChart, 138
 Declarations, 128, 131, 140
 Ellipse, 116, 124
 Graphic, 142
 Group, 124, 146
 HGroup, 132
 Image, 132, 147
 intent-filter, 188, 190
 ItemRenderer, 64
 LinearGradient, 119
 LinearLayout, 195
 manifest, 194
 Metadata, 135
 Move, 128
 MoveEffect, 131
 MultitouchImage, 64
 Parallel, 129
 Path, 136
 PieChart, 140
 receiver, 190
 Rect, 116, 135
 Rotate, 128
 Rotate3D, 131
 Scale, 128
 ScrapbookPage, 64
 Script, 140
 Sequence, 128
 states, 135
 Stroke, 119
 TextView, 195
 VGroup, 132, 142
 elementy
 GradientEntry, 122
 Move3D, 131

UI, 29
 XML, 125
 emulator Androida, 149, 154, 155
 enkapsulacja zdarzeń, 62
 etykieta, 28

F

filtr

ColorMatrixFilter, 217
 ConvolutionFilter, 217, 220
 DisplacementFilter, 218
 intencji, 188

filtry

Flex, 121
 obrazów, 216
 Spark, 121, 123
 wideo, 215

Flash, 18, 305

Flex, 21, 26, 101, 167

folder

bibliotek, 28
 bin-debug, 28
 libs, 28
 platform-tools, 153
 src, 28
 wyjściowy, 28
 źródłowy, 28

format

FXG, 298
 GIF, 297
 JPEG, 298
 PNG-24, 297
 dźwięku, 244
 M.264, 275
 On2 VP6, 275
 wideo, 275

formaty plików graficznych, 297

FPS, 214, 220, 308

framework WebKit, 19

FTE, Flash Text Engine, 322

funkcja

ActionScript(), 42
 CameraRoll(), 228
 cloneModelForEntry(), 261
 createMusicEntryForfile(), 259
 createSongViewModel(), 261

getCurrentViewState(), 341
 getDPIScale(), 44
 getMusicEntries(), 259
 getProperty(), 93
 getRuntimeDPI(), 44
 getWedgeLabel(), 140
 incrementProgramState(), 247
 init(), 30
 ioErrorHandler(), 201
 Math.max(), 70
 Microphone.getMicrophone(),
 210
 navigator.pushView(), 262
 nextSong(), 350
 onBrowse(), 225
 onCameraRollError(), 225
 onCaptureImage(), 222
 onChange(), 89
 onClick(), 252
 onCreateComplete(), 225,
 252
 onID3(), 251
 onLoaded(), 225
 onMetaData(), 281
 onPanChange(), 254
 onPlayPause(), 281
 onResize(), 342
 onSaveComplete(), 225
 onSelect(), 225
 onSelectCanceled(), 225
 onTimer(), 281
 onTouchBegin(), 249
 onTouchMove(), 249
 onTouchTap(), 247
 onViewActivate(), 339
 onVolumeChange(), 254
 playSongAtCurrentIndex(), 350
 popAll(), 87
 popToFirstView(), 87
 popView(), 87
 previousSong(), 350
 pushView(), 80, 87
 replaceView(), 87
 saveImageToFileSystem(), 145
 setProperty(), 93
 setSilenceLevel(), 211
 showMessage(), 210, 225

stageInit(), 53
 sqlError(), 201
 touchEnd(), 144
 touchMove(), 144
 touchMoveHandlerImage(),
 145
 touchSingleTap(), 145
 touchTapHandler(), 145
 touchTapHandlerImage(), 145
 trace(), 225
 transitionToRecordingState(),
 248
 transitionToStoppedState(),
 248
 writeFloat(), 244
 funkcje
 Androida, 19
 Androida 3.0, 183
 specjalne emulatora, 156
 URI, 174
 FXG, 140

G

geolokalizacja, 232
 gest, 61
 pan, 32
 rotate, 32
 swipe, 32
 two-finger tap, 32
 zoom, 32
 gesty mobilne, 59
 gęstość ekranu, 40, 42
 głośność, 253
 główny widok aplikacji, 28
 Google Play, 149
 GPS, 19, 232
 GPU, 317
 gradient
 liniowy, 114, 123
 radialny, 115
 grafika
 warstwy klipu, 25
 we Flashu, 310
 grafiki rastrowe, 43
 grupy stanów, 338

I

ikony, 160
 implementacja
 klasy ItemRenderer, 320
 usługi LocalMusicService, 258
 wiadomości, 82
 importowanie pliku Photoshopa, 296
 informacja
 o błędzie, 32
 o pierwszym geście, 29
 inicjalizowanie
 aparatu, 214
 mikrofonu, 210
 instalowanie
 AIR, 153, 154
 Android SDK, 150, 185
 aplikacji, 157, 170
 instrukcje importu, 177
 intencja
 domniemana, implicit Intent, 188
 rozgłaszana, broadcast Intent, 188
 skierowana, directed Intent, 188
 intencje Androida, 187
 interfejs
 kuchenki, 106
 metadanych, 342
 MediaPlayer, 345
 MusicService, 256, 257
 odtwarzacza muzyki, 265
 SongView, 263, 348
 użytkownika, UI, 30, 187

J

jakość obrazu, 313
 Java SDK, 150
 język Java, 185
 JNI, Java Native Interface, 186

K

katalog
 extdir, 36
 src, 28
 root-dir, 37
 klasa
 Accelerometer, 230
 Activity, 187
 Application, 28
 Button, 135
 Camera, 212
 CameraRoll, 212, 222
 CameraUI, 212, 227
 Capabilities, 24, 276
 ContentCache, 322
 DisplayObject, 212
 DraggableGroup, 62
 EventDispatcher, 267
 Geolocation, 232
 GeolocationEvent, 232
 Group, 62
 HGroup, 28
 ID3Info, 269
 ID3Reader, 269
 Image, 28
 ImageView, 65
 InteractiveObject, 60
 ItemRenderer, 320
 Java MainApp, 202
 LocalMusicService, 256
 MediaPlayer, 285
 Microphone, 209, 243
 Multitouch, 24, 60
 MultitouchImage, 60
 NetConnection, 277
 NetStream, 277, 282
 Notification, 205
 persistenceManager, 93
 PressAndTapGestureEvent, 61
 Skin, 273
 Sound, 240, 242
 SoundChannel, 242, 251
 SoundEffect, 238, 240
 SoundTransform, 242
 StageOrientationEvent, 52
 Timer, 181
 TouchEvent, 69
 TransformGestureEvent, 61
 VGroup, 28
 VideoElement, 283
 VideoPlayer, 277
 View, 85, 87, 213
 ViewNavigator, 73, 85
 ViewNavigatorApplication, 85
 klasy
 główne aktywności, 201
 główne aplikacje, 28
 obrazów Flex, 321
 typu entrypoint, 331
 usług Androida, 202, 203
 klawiatura ekranowa, 99, 102
 klikanie, 62
 kod
 ActionScript, 53, 57, 246, 315–317
 ActionScript 3, 178
 Adobe AIR, 205
 aplikacji korzystającej z IconItemRenderer, 107
 aplikacji renderującej akapity, 97
 Caterpillar Generator, 69
 Gesture Check, 32
 demonstrujący użycie kontrolek, 110
 DensityExploer, 44, 49
 DraggableGroup, 62
 FirstView, 75
 Flash Mobile Bench, 314
 Flash Scrapbook, 63
 GeolocationBasicHome, 233
 inicjalizujący
 CameraUI, 285
 MediaPlayer, 283
 mikrofon, 210
 przeglądanie, 224
 interfejsu kuchenki mikrofalowej, 105
 ItemRenderer, 322, 323
 kontrolujący klawiaturę ekranową, 100
 MXML, 46, 53, 256
 natywny Androida, 205
 NetStreamVideoView, 278
 OSMFVideoView, 282

- SongListView, 261
 - SongViewModel, 350
 - tworzący kontrolkę List, 107
 - wykonywalny Dalvik, 185
 - wykrywający obsługę gestów, 30
 - wymuszający odśmiecianie, 330
 - zamkający aplikację, 78
 - kodowanie wideo, 275
 - kolejność zdarzeń, 77
 - kompas, 19
 - komponent
 - ActionBar, 73, 83–85
 - FPSComponent, 319
 - navigator, 75
 - NetStreamVideoView, 279
 - ProgressButton, 272
 - SongListView, 262
 - Spark VideoPlayer, 276
 - TextArea, 100
 - TextInput, 99
 - VideoPlayer, 277, 278
 - View, 75, 84, 213, 245
 - ViewNavigator, 78, 83, 92
 - komponenty
 - Flex, 73
 - FXG, 142
 - tekstowe, 324
 - UI, 246
 - View, 73–84, 92
 - komunikat o błędzie, 31, 167
 - konto programisty, 170
 - kontrolka
 - BusyIndicator, 110
 - Button, 102
 - ButtonBar, 102
 - CheckBox, 102
 - DataGrid, 182
 - Group, 265
 - HSlider, 109
 - List, 87, 106
 - RadioButton, 102
 - Scroller, 110, 111
 - SkinnablePopupContainer, 83
 - VGroup, 344
 - własna ProgressButton, 271
 - kontrolki
 - Density Explorer, 43
 - Flex 4.5, 95
 - formularza, 28
 - odtworzenia, 343
 - przycisków, 101
 - tekstowe, 94
 - wizualne, 93
 - kreator projektu Flex, 51
 - krzywe Béziera, 117, 120, 121
- ## L
- liczba pikseli niezależnych, 42
 - liczba punktów dotyku, 58
 - licznik FPS, 315
 - lista Adobe User Group, 321
 - lista kontrolek Spark, 94
 - listy Flex, 106
 - logo Androida, 25
- ## Ł
- ładowanie danych audio, 241
 - ładowanie pliku MP3, 269
 - łączenie aplikacji, 207
 - łączenie segmentów, 120
- ## M
- mapowanie przycisków, 155
 - maszyna wirtualna, 19
 - menu komponentów View, 89, 91
 - metadane ID3, 250
 - metoda
 - addEventListener(), 52, 61
 - addPerson(), 181
 - addWebsiteInfo(), 199
 - browseForImage(), 225
 - Capabilities.screenDPI(), 44
 - completeHandler(), 200
 - connect(), 279
 - db_opened(), 182, 198
 - displayNotification(), 205
 - doNotification(), 204, 205
 - event.preventDefault(), 216
 - event.stopImmediatePropagation(), 70
 - getCurrentViewState, 338
 - getStyle, 49
 - initFilters(), 216, 218
 - invokeMontastic(), 200
 - loadCurrentSong(), 267
 - onBind(), 203
 - onCreate(), 187, 190, 204
 - onCreationComplete(), 216, 228, 231
 - onDestroy(), 190
 - onPause(), 190
 - onRestart(), 190
 - onResume(), 190
 - onStart(), 190
 - onStop(), 190
 - refreshDataGrid(), 182, 200
 - remove(), 200
 - removeAll(), 200
 - removePerson(), 181
 - result(), 183, 200
 - run(), 204
 - setMode(), 214
 - setRequestedUpdateInterval(), 230
 - start(), 182, 198
 - stop(), 57
 - swipe, 64
- metody klasy ViewNavigator, 84
 - migające światelka, 25
 - mikrofon, 209
 - moc przetwarzania, 276
 - model
 - danych MusicEntry, 257
 - prezentacji, 254, 260
 - SongListViewModel, 260
 - współbieżności, 331
 - modyfikowanie UI, 54
- ## N
- nakładka ADT Eclipse, 151
 - narzędzia
 - programisty, 302
 - programistyczne Adobe, 22
 - projektanckie Adobe, 21
 - nasłuchiwanie intencji Androida, 190
 - NDK, Native Development Kit, 186

O

obieg pracy, 296–303

obiekt

- activityLevel, 243
- Application, 42
- ArrayCollection, 138
- ByteArray, 242
- Camera, 213
- Capabilities, 45
- context, 92
- data, 92
- DataGroup, 64
- displayObject, 286
- DisplayObject, 270
- GeolocationEvent, 232
- ItemRenderer, 63
- MediaElement, 282
- MediaEvent, 225
- MediaPlayer, 283
- MediaPromise, 286
- messageLabel, 225, 227
- Multitouch, 66
- MusicEntry, 257, 269
- navigator, 75, 92
- NetStream, 281
- parentApplication, 45
- PendingIntent, 205
- Sound, 241, 244, 245
- SoundChannel, 252
- SoundEffect, 238
- SoundTransform, 253
- Stage, 52, 70
- timer, 268
- UIComponent, 279
- VGroup, 64
- Video, 213
- videoContainer, 286

obiekty

- 2D, 113
- geometryczne Spark, 122
- UIComponent, 213, 311

obliczanie szybkości odtwarzania

klatek, 315

obracanie twarzy, 56

obrót, 32, 60

obsługa

- aparatu, 213
- audio i wideo, 20

błędów, 182

gestów, 30, 34, 60

gestu Panorama, 64

gestu Pociągnięcie, 63, 64

gestu Stuknięcie dwoma
palcami, 65

gęstości, 47

grafiki 3D, 334

klawiatury ekranowej, 99

oddzielnych stanów, 337

Powiększenia i Obrotu, 60

projektu interfejsu, 351

przechwytywania obrazów, 221

wiązania dwukierunkowego,
267

zapytań o media, 50

zdarzenia żądania danych, 244

zdarzeń klasy NetStream, 280

zdarzeń MediaPlayer, 286

zdarzeń widoku, 279

zmian orientacji, 57

zmian układu aplikacji, 53

odbieranie i przesyłanie danych,
196

odbiorcy wiadomości, 190

odbiorniki TV, 352

odczytywanie

metadanych, 251

rozdzielczości, 40

odrysowywanie, 312

odśmiecianie, 329

odtworacz

Flash Player, 19, 237, 323

wideo, 282

odtworzenie

muzyki, 263

plików MP3, 250

wideo, 275

okno

Mobile Settings, 27

wyboru konfiguracji, 29

opakowywanie aplikacji, 157, 168,
170

opcja Automatically reorient, 51

operator @, 44

optymalizowanie

kompilatora JIT, 330

wideo, 275

wydajności, 305

wydajności grafiki, 307

orientacja

automatyczna, 55

ekranu, 50

urządzenia, 53

osadzanie pliku dźwiękowego, 237

osie akcelerometru, 229

OSMF, Open Source Media

Framework, 276

otrzymywanie i wysyłanie danych,
197

otwieranie

HTML, 176–178

URL, 178, 180

P

pakiet

com.proandroidflash, 202

Creative Suite 5.5, 21

flash.filesystem, 259

flash.filters, 216

mx.effects, 123

spark.effects, 123

views, 28

pakiety

Android SDK, 151

projektu-biblioteki, 347

panorama, 32, 60

parametr

app-desc, 37

nodebug, 36

pubid, 36

runtime, 36

screenize, 36

pasek

ActionBar Androida, 86

suwaka, 109

pauza, 251

perspektywa Flash Debug, 32

piksele niezależne, 42

platforma

Adobe Flash, 20

TV, 352

plik

.apk, 185

android.jar, 192

- AndroidManifest.xml, 185–193, 201
 - ButtonSkin1.mxml, 134
 - CameraBasic-app.xml, 214
 - CharData.xml, 138
 - default.properties, 192
 - DensityExplorerHomeView.mxml, 43
 - HelloWorld.java, 186, 192
 - icon.png, 192
 - main.xml, 192–194
 - Metaphile.swc, 346
 - R.java, 192, 193
 - RectEllipse1.fgx, 142
 - SimpleService.java, 189
 - SQLite1HomeView.mxml, 181
 - SQLiteAccess.as, 181
 - strings.xml, 192–194
 - ustawień Flash Builder, 28
 - VideoFilterView.mxml, 215
 - pliki
 - .actionScriptProperties, 28
 - .flexProperties, 28
 - .project, 28
 - .settings, 28
 - APK, 153, 166
 - aplikacji dla Androida, 192
 - FLA, 302
 - FXG, 143, 274
 - MP3, 237
 - projektu-biblioteki, 347
 - PSD, 297
 - SWF, 237
 - pociągnięcie, 32, 60
 - podgląd
 - aplikacji, 293
 - projektu w Device Central, 293
 - podpis cyfrowy, Digital signature, 163, 166
 - pojemniki na aplikacje, 73
 - pole wyboru, 28
 - polecenia
 - DOS, 206
 - Linux, 206
 - połączenie asynchroniczne, 182
 - połączenie USB, 36
 - pomiar wydajności, 326
 - powiadomienia, 195–197
 - powiększenie, 60
 - ppi, pixels per inch, 40
 - prędkość odtwarzania klatek, 275, 307
 - procedura
 - onCaptureHandler(), 286
 - onCreationComplete(), 243
 - onDurationChange(), 284
 - onLoadComplete(), 270
 - onMetaData(), 270
 - onSize(), 286
 - onTimeChange(), 284
 - procedury obsługi zdarzeń OSMF, 284
 - procesor graficzny GPU, 317
 - program
 - apktool, 205
 - curl, 201
 - do szkicowania, 143
 - profilujący, 328
 - programista, 300
 - projekt
 - aplikacji, 295
 - CameraBasic, 214
 - CameraFunHouse, 221
 - GestureCheck, 167
 - mobilny Flex, 26
 - MusicPlayerPhone, 347
 - projektant, 290
 - próbki dźwiękowe, 245
 - próbkowanie dźwięku, 275
 - przeciąganie myszą, 62
 - przeglądanie obrazów, 224
 - przeglądarka mobilna, 19
 - przeglądarka obrazów, 67
 - przejścia pomiędzy komponentami, 87
 - przejście flip, 89, 90
 - przekształcenia, 113, 147
 - obiektów geometrycznych, 122
 - przełączanie trybów, 54
 - przełącznik odtwarzanie – pauza, 251
 - przenoszenie aplikacji, 352, 353
 - przepływ pracy Flash, 23
 - przetwarzanie
 - obrazów, 216
 - tekstu, 323
 - w wątkach, 331
 - przezroczystość, 46, 298, 310
 - przybliżenie, 32
 - przyciski, 101, 136
 - Androida, 155
 - modyfikujące klawiatury, 155
 - przyspieszanie renderowania, 309
 - publiczne właściwości przycisków, 104
 - publikowanie aplikacji, 165, 170
- ## R
- rejestrowanie
 - dźwięku, 211
 - obrazów, 213, 221, 227, 286
 - renderer
 - IconItemRenderer, 325
 - LabelItemRenderer, 325
 - renderery
 - elementów, 319
 - programowe, 309
 - wbudowane, 324
 - renderowanie, 308
 - elipsy, 114
 - krzywych Béziera, 118
 - obrazów, 63
 - prostokątów, 114, 144
 - szkiców, 143
 - za pomocą GPU, 317
 - reorientacja automatyczna, 51, 55
 - reorientacja we Flashu, 56
 - rozdzielczość ekranu, 40
 - rozmiar
 - aplikacji, 306
 - ekranu, 40
 - elementów UI, 41
 - rozpoznawanie
 - gestów, 19, 64, 67
 - gęstości, 42
 - rysowanie prostokątów, 113
- ## S
- SAO, service-oriented architecture, 188
 - schodkowanie, 313
 - SDK Setup, 150
 - selektor arkusza stylów, 50

selektor dpi, 49
 selektory mediów CSS, 48
 silnik renderowania Flasha, 311
 skalowanie
 ekranu, 335
 grafiki, 42, 46
 sklep
 Amazon Appstore, 166
 Google Android Market, 166
 Google Play, 170
 skórka
 emulatora Androida, 154
 MobileVideoPlayerSkin, 277
 Nexus S, 154
 smartfon Samsung Galaxy S II,
 173
 SongViewModel, 339, 349
 specyfikacje ekranów, 336
 SQLite, 173, 179
 stała DEFAULT_DIR, 259
 stan zarejestrowanych witryn, 206
 stany
 aplikacji SoundRecorder, 245
 komponentu View, 338
 stos komponentów View, 76
 strony HTML, 176
 struktura
 ArrayCollection, 80
 katalogów, 191
 projektu, 28
 strumień wideo, 214, 215, 221
 stuknięcie dwoma palcami, 32, 60
 style
 IconItemRenderer, 108
 przycisków, 103, 104
 tekstu, 96
 symulator PlayBook BlackBerry,
 355
 system operacyjny
 Android, 17
 Apple iOS, 17, 356
 BlackBerry RIM, 17
 Linux, 18
 webOS, 18
 Windows 7, 18
 system przepływu pracy, 23
 szablon
 Flash Professional, 24

ikony, 161
 View-Based Application, 28
 szablony Photoshopa, 160

Ś

ścieżka do ikon, 161
 śledzenie kodu aplikacji, 31
 środowisko
 Adobe Flash, 352
 AIR, 154, 209
 Android SDK, 149
 Eclipse, 151, 185, 191
 Java, 150

T

tablet
 BlackBerry, 353
 Motorola Xoom, 173
 Samsung Galaxy Tab 10.1, 173
 takt silnika Flash, 307
 technologia Flash, 18, 305
 telefonia GSM, 20
 test wydajności, 71
 TLF, Text Layout Framework, 322
 token debugowania, 356
 trapezoid, 121
 triangulacja, 232
 tryb
 debugowania, 36
 krajobrazowy, 52
 ładowania, 36
 obsługi danych, 67
 portretowy, 51
 renderowania, 318
 wielodotkowy, 144
 tworzenie
 animacji, 128
 animacji 3D, 130
 aplikacji, 35, 290
 aplikacji dla Androida, 187,
 191
 aplikacji natywnych, 183
 aplikacji połączonej, 205
 arkusza stylów, 49
 certyfikatów, 162–165, 169

certyfikatów podpisujących,
 157
 dokumentu Fireworks, 291
 edytowalnej kopii profilu, 294
 filtrów, 216
 grafik, 300
 ikon, 160
 instancji filtrów obrazów, 217
 obiektów GraphicElement, 316
 obiektu NetStream, 280
 obiektu SoundEffect, 238
 odtwarzaczy wysokiej jakości,
 282
 oprogramowania, 302
 pakietów ADT, 169
 pliku APK, 166–169
 projektów dla tabletów, 347
 projektów dla telefonów, 347
 projektu AIR, 24
 projektu Flex, 179
 projektu mobilnego Flex, 26
 projektu w Device Central,
 290
 projektu-biblioteki, 345
 referencji do komponentów
 FXG, 142
 skórek Spark, 133
 skórki przycisku, 134, 135
 UI, 39
 urządzenia wirtualnego, 152
 własnego komponentu, 270
 własnych profili, 294
 wykresów, 136
 wykresu kołowego, 139
 wykresu słupkowego, 137
 typ debugowanej aplikacji, 36
 typ zdarzenia, 52
 typy obiektów Flex, 311
 typy selektorów, 48

U

UI, interfejs użytkownika, 187
 uprawnienia
 Androida, 159
 aplikacji, 157
 w deskrypcji aplikacji, 158

uprawnienie
 ACCESS_COARSE_LOCATION, 235
 ACCESS_FINE_LOCATION, 235
 android.permission.RECORD_AUDIO, 211
 CAMERA, 215
 INTERNET, 157

uruchamianie
 aplikacji, 34, 157, 167, 170
 w emulatorze, 150
 w urządzeniu, 150
 emulatora Androida, 155

urządzenia
 mobilne, 39
 testowe, 292
 wirtualne, 151

urządzenie PlayBook, 353, 356

usługa
 LocalMusicService, 258
 MusicService, 256

usługi Androida, 189

ustawianie
 map przemieszczenia, 218
 uprawnień, 157

ustawienia
 applicationDPI, 47
 cacheAsBitmap, 317
 deviceDPI, 44
 Eclipse, 28
 Flash Builder, 28
 Flex, 28

utrwalanie danych, 93

W

W3C, 47

wartość
 FPS, 220
 null, 286
 progowa threshold, 46

wdrażanie aplikacji, 165
 AIR, 155
 z wiersza poleceń, 167

wdrożenie, Deployment, 166

węzeł Skin, 135

wiązanie dwukierunkowe, 267

widok HomeView, 28

widoki aplikacji, 28

widżet, 190

wielodotykowość, 19, 58, 71

wielowątkowy potok
 renderowania, 332

wielozadaniowość, 20

wiersz poleceń, 36, 156, 162, 167

Wi-Fi, 20, 232

właściwości
 komponentu ActionBar, 85
 obiektu NetStream, 281
 publiczne kontrolki, 97

właściwość
 applicationDPI, 42, 49, 71
 autoOrients, 51
 cacheAsBitmap, 311
 cacheAsBitmapMatrix, 312
 renderMode, 317
 titleContent, 82

włączanie
 cacheAsBitmap, 312
 obsługi gestów, 60
 obsługi zdarzeń, 67

wprowadzanie danych, 154

współbieżność, 331

współczynnik skalowania, 44

wybór obiegu pracy, 304

wydajność, 276, 305
 aplikacji, 325
 CPU i GPU, 319
 Flasha, 329
 kodu ActionScript, 330
 komponentów tekstowych, 322
 renderowania, 310

wykres
 kołowy, 138, 141
 słupkowy, 137
 2D, 136

wykrywanie
 mikrofonu, 243
 zdarzenia aktywacji, 100

wyłączanie
 rozpoznawania gestów, 67
 skalowania, 55

wysyłanie aplikacji, 171

wyświetlacze, 40

wyświetlanie wskaźnika
 wydajności, 220

wywoływanie
 funkcji C/C++, 186
 funkcji URI, 174
 stron HTML, 176

wzorzec Presentation Model, 255

Z

zakładka Mobile Settings, 27

zapytania SQL, 182

zawartość pakietu, Package Contents, 166

zdarzenia
 API, 68
 dotykowe, 67, 68
 zmiany orientacji, 57

zdarzenie
 AccelerometerEvent, 230
 activate, 77, 100
 click, 221
 creationComplete, 76, 77
 deactivate, 77
 ENTER_FRAME, 308
 ErrorEvent.ERROR, 225
 Event.COMPLETE, 225
 initialize, 76, 77
 KeyboardEvent, 216, 224
 kliknięcie, 62
 metaData, 280
 MouseEvent, 273
 onChange(), 262
 onDeactivate, 93
 onFilterChange, 216
 onInitialize, 93, 262
 onMetaData, 269
 Panorama, 64
 pojedynczego stuknięcia, 145
 przeciągnięcie myszą, 62
 reorientacji, 52
 softKeyboardActivate, 100
 softKeyboardActivating, 100
 softKeyboardDeactivate, 100
 stuknięcie dwoma palcami, 64
 touchBegin, 68
 touchEnd, 68
 touchMove, 68

zdarzenie

- touchOut, 67, 68
- touchOver, 67, 68
- touchRollOut, 67, 68
- touchRollOver, 67, 68
- sampleData, 210, 243, 249
- SONG_ENDED, 267
- Timer, 308
- touchTap, 68
- viewActivate, 76, 77
- viewActive, 77

zmiana

- głośności, 253
- uprawnień, 157, 158

zmienna

- Application.applicationDPI, 44
- APP_HOME, 206
- JAVA_HOME, 185
- percentComplete, 268
- typu Bindable, 181

zmiennie

- obserwowane, bindable, 43, 198

znacznik

- ID3, 250
- metadanych Embed, 238

- znak kropki (.), 187

- znaki kresek (--), 37

- zużycie pamięci, 306

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**

Platforma Android z każdym dniem zdobywa tysiące kolejnych użytkowników. Andy Rubin — odpowiedzialny za nią w Google — pochwalił się aktywacją 700 tysięcy nowych urządzeń z tym systemem każdego dnia. Robi wrażenie? Pomyśl, jak wykorzystać ten rynek i stworzyć aplikację, która podbije serca użytkowników. Dzięki możliwości wykorzystania technologii Flash teraz jest to jeszcze łatwiejsze!

Android Flash. Zaawansowane programowanie aplikacji mobilnych to kompletny przewodnik po budowaniu skomplikowanych, zajmujących aplikacji dla Androida. Dzięki tej książce nauczysz się programować smartfony oraz tablety z wykorzystaniem najpopularniejszych frameworków służących do tworzenia bogatych aplikacji internetowych (RIA) dla Androida — Flash i Flex. Gdy wraz z autorami przejrysz już zestaw narzędzi programistycznych Flasha, dowiesz się, jak dodawać multimedia, animacje i efekty specjalne do aplikacji. Zrozumiesz, jak działa optymalizowanie dla różnorodnych wyświetlaczy oraz jak wykorzystać dane wejściowe pochodzące z aparatu, GPS, mikrofonu i akcelerometru. Poznasz metody tworzenia certyfikatów podpisywania kodu oraz uzyskiwania jak najlepszej wydajności aplikacji. Zapoznasz się z dużą ilością praktycznego, dobrze skomentowanego kodu i zbudujesz trzy kompletne projekty: program do rysowania, odtwarzacz muzyczny Flash oraz aplikację AIR przechowującą informacje o stanach witryn w bazie danych SQLite.

Sprawdź możliwości technologii Flash dla platformy Android.

- Korzystaj z usług geolokalizacyjnych
- Sprawdź położenie telefonu za pomocą wbudowanych sensorów
- Wykorzystaj narzędzia przeznaczone dla technologii Flash
- Opublikuj aplikację w Android Market

Twórz atrakcyjne aplikacje mobilne za pomocą Flasha, Fleksa i Androida!

helion.pl
księgarnia
internetowa

Nr katalogowy: 8628

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/nowości>

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 978-83-246-3920-5



Cena: 69,00 zł

Informatyka w najlepszym wydaniu