O'REILLY®

HTML5

Strony mobilne

WYKORZYSTAJ POTENCJAŁ HTML5 NA URZĄDZENIACH MOBILNYCH!

HELION

Estelle Weyl

Tytuł oryginału: Mobile HTML5

Tłumaczenie: Tomasz Walczak

ISBN: 978-83-246-8912-5

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/htm5sm
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 5.

Elementy svg, canvas, audio i video

Wcześniej omówiłam większość nowych elementów z HTML5. Wyjątkiem są elementy powiązane bezpośrednio z tworzonymi dopiero interfejsami API, a także dobrze obsługiwane elementy multimedialne `svg`, `canvas`, `audio` i `video`. Elementy z pierwszej z tych grup mogą jeszcze zostać zmodyfikowane, dlatego nie omawiam ich w tej książce. Elementy z drugiej grupy opisuję w tym rozdziale.

Przedstawiam tu najważniejsze funkcje, które programiści frontonów aplikacji sieciowych prawdopodobnie będą stosować w codziennej pracy. Dzięki temu przy tworzeniu kodu dla przeglądarek mobilnych będziesz mógł korzystać z nowych mechanizmów. Wszystkie nowe przeglądarki mobilne (z wyjątkiem Opery Mini) obsługują elementy `<canvas>`, `<video>` i `<audio>`, a także sieciowe interfejsy API związane z geolokalizacją, pamięcią lokalną, aplikacjami sieciowymi działającymi w trybie offline itd.

Na każdy z tematów omawianych w tym rozdziale można napisać całą książkę — i dla wielu opisywanych zagadnień takie książki już są dostępne. Przestawiam tu wystarczająco dużo informacji, abyś mógł uznać, że chcesz przeczytać książkę na dany temat, lub stwierdzić, że danym zagadnieniem nie jesteś zainteresowany. Choć nie opisuję szczegółowo poszczególnych kwestii, ten rozdział zapewni Ci wiedzę potrzebną do rozpoczęcia pracy. Co ważniejsze, zrozumiesz zalety i wady omawianych tu technologii w kontekście urządzeń przenośnych.

Multimedialne interfejsy API w HTML5

Pierwotna specyfikacja HTML-a dotyczyła wyłącznie materiałów tekstowych. Nie obejmowała nawet elementu `<img>`. Od tamtego czasu dużo się zmieniło. HTML5 umożliwia tworzenie skalowalnej grafiki wektorowej za pomocą formatu SVG i pustego obszaru, po którym można rysować (odpowiada mu element `<canvas>`). Oprócz grafiki w HTML5 obsługiwane są elementy `<video>` i `<audio>`, które działają bez konieczności stosowania niezależnych wtyczek.

SVG

Format SVG umożliwia tworzenie złożonej *skalowalnej grafiki wektorowej*. Ten format został wprowadzony w 2001 r. i jest otwartym standardem definiowania dwuwymiarowej grafiki wektorowej. Skalowalność w SVG oznacza, że ta sama grafika będzie równie ostra na dużych monitorach, jak i na małych wyświetlaczach urządzeń przenośnych. Nie wymaga to wprowadzania żadnych modyfikacji.

W specyfikacji SVG jest zdefiniowana oparta na XML-u gramatyka tworzenia kształtów, linii, krzywych, rysunków i tekstu. Obsługiwane są też różne funkcje, takie jak: przezroczystość, dowolne kształty geometryczne, filtry (cienie, efekty świetlne itd.), skrypty oraz animacje.

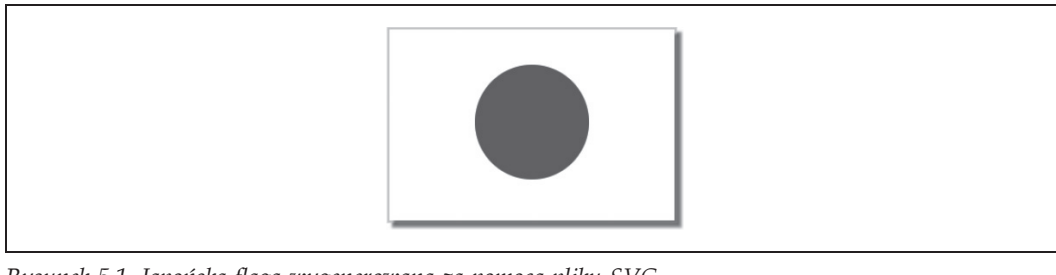
Ponieważ SVG to format graficzny oparty na teście, pliki mogą być bardzo małe. Ponieważ w tym formacie używany jest model obiektowy, grafiki można modyfikować za pomocą skryptów. Ponieważ jest to format wektorowy, można skalować go bez negatywnych efektów takich jak pikselizacja lub nierówne krawędzie. Ten format jest zrozumiały (dzięki temu, że jest deklaratywny) i obsługuje animacje.

Istnieje kilka wersji formatu SVG. Są one obsługiwane w przeglądarkach w różnym stopniu. Podstawowa obsługa samodzielnych plików *.svg* jest dostępna we wszystkich urządzeniach przenośnych i nowych przeglądarkach. W Androidzie pliki tego typu są obsługiwane od wersji 3. Grafikę w formacie SVG można stosować jako źródło elementów `<img>` od systemów iOS 3.2 i Android 3.0 oraz od mobilnej wersji Internet Explorera 8.

Pliki w formacie SVG jako wartość właściwości `background-image` w stylach CSS są obsługiwane od systemów Android 3 i iOS 3.2. Ponadto od dawna można ich używać w Operze Mobile. Element `<svg>` na stronach HTML5 można stosować od systemów iOS 5 i Android 3 oraz od przeglądarki Internet Explorer 9 (a także we wszystkich innych nowych przeglądarkach). Tylko w systemach Android 2.3.3 i starszych, Amazon Silk i nieużywanym już WebOS firmy HP przeglądarki mobilne nie zapewniają obsługi formatu SVG. Statyczne pliki SVG są obsługiwane nawet w Operze Mini (podobnie jak element `<canvas>`, który jest obsługiwany we wszystkich przeglądarkach mobilnych), przy czym nie można ich animować, ponieważ poziom obsługi JavaScriptu w tej przeglądarce to uniemożliwia.

Ponieważ SVG jest oparty na XML-u, głównym elementem plików w tym formacie jest nie `<html>`, a `<svg>`. Pliki SVG, podobnie jak wszystkie dokumenty XML, rozpoczynają się od prologu XML-a i deklaracji DTD dla formatu SVG. Element główny `<svg>` zawiera całą treść dokumentu. Pliki w formacie SVG nie obejmują elementów `<head>` i `<body>`. Cała treść, w tym zagnieżdżone elementy `<svg>`, znajduje się w głównym elemencie `<svg>`.

Dobrym wprowadzeniem do formatu SVG jest japońska flaga. Jest to biały prostokątny obszar z czerwonym kołem (słońcem) w środkowej części, co przedstawia rysunek 5.1.



Rysunek 5.1. Japońska flaga wygenerowana za pomocą pliku SVG

```
1 <!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
2   "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
3 <svg xmlns="http://www.w3.org/2000/svg" height="220" width="320" version="1.0">
4   <title>Japońska flaga</title>
5   <desc>Czerwone koło na białym prostokącie</desc>
6   <rect x="10" y="10" width="300" height="200"
7     style="fill: #ffffff; stroke: #e7e7e7;"/>
8   <circle cx="160px" cy="107px" r="60px" fill="#d60818"/>
9 </svg>
```

Cóż ten kod oznacza? Wiersze od 1 do 3 to deklaracja DTD dla plików SVG i główny element <svg>. Warto zauważyć, że w głównym elemencie zadeklarowana jest wielkość danej grafiki wektorowej. Aby w stylach CSS móc używać grafiki w formacie SVG we właściwości background-position, trzeba zadeklarować wymiary tej grafiki. Jest to ważne, gdy tworzysz sprite'y w formacie SVG.

Można dodać element <title> (wiersz 4), jeśli plik SVG jest używany niezależnie od innych zasobów. Element <desc> (wiersz 5) zawiera tekstowy opis, który normalnie się nie wyświetla, gdy pokazywany jest plik SVG. Umieszczenie odpowiednich informacji w elementach <desc> i <title> zwiększa dostępność stron. Ponieważ nie wszystkie czytniki ekranu obsługują format SVG, dodanie atrybutu aria-label pozwala poprawić dostępność stron.

Element <rect> (wiersz 6) oznacza prostokąt. Dostępne kształty i linie to: <path>, <rect>, <circle>, <ellipse>, <line>, <polyline> i <polygon>. Tu podane są cztery atrybuty: x, y, width i height. Określają one współrzędne x i y oraz szerokość i wysokość prostokąta. Używany jest też atrybut style.

W dokumentach SVG, podobnie jak w dokumentach HTML, można używać stylów CSS do określania stylu elementów. Możesz zadeklarować style wewnątrzwierszowo za pomocą atrybutu style, jak to zrobiłam w przedstawionym przykładzie, albo dołączyć osadzony lub zewnętrzny arkusz stylów i wskazywać w nim elementy przy użyciu selektorów (ta technika jest stosowana w plikach HTML).

Nazwy właściwości są tu nieco odmienne niż w stylach CSS, do których jesteś przyzwyczajony, jednak są zrozumiałe. Właściwość fill działa podobnie jak właściwość background — tu określa kolor tła. Właściwość stroke pełni funkcję podobną do właściwości border w stylach CSS. Można też określić gradient lub wzór.

Choć w plikach SVG można stosować większość właściwości i wartości ze stylów CSS, ze względu na bezpieczeństwo niektórzy producenci przeglądarek[1] blokują importowanie grafiki rastrowej i skryptów w plikach SVG zapisanych jako grafika tła w elemencie <img>.

Element <circle> (wiersz 8) tworzy koło o czerwonym tle. Zamiast wysokości i szerokości w tym elemencie podawany jest atrybut r, który oznacza promień. Pozycja nie jest tu podawana na podstawie górnego lewego rogu (jak to się odbywa w elemencie <rect>), ale za pomocą środka koła. Atrybut cx określa współrzędną x środka koła, natomiast cy to współrzędna y środka.

Jeśli przyjrzysz się atrybutom elementu <circle>, zauważysz, że jednym z nich jest fill, natomiast w elemencie <rect> fill jest podany jako właściwość stylów CSS.

Dołączanie grafiki SVG do dokumentów

Możesz dodać plik SVG bezpośrednio do dokumentu. Umożliwiają to znaczniki <img>, <object> i <embed>:

```
<img src="flag.svg" alt="Japońska flaga"/>
```

lub:

```
<embed type="image/svg+xml" src="flag.svg" width="320" height="220"/>
```

lub:

```
<object data="flag.svg" type="image/svg+xml" width="320" height="220"></object>
```

[1] Obecnie silniki WebKit i Mozilla blokują importowanie skryptów i grafiki rastrowej do plików SVG ze znaczników <img>. Dzieje się tak nawet wtedy, gdy grafika rastrowa pochodzi z tego samego źródła.

Warto zauważyć, że choć elementy `<embed>` i `<object>` nie mają atrybutu *`alt`*, grafika SVG może być dostępna. Aby zwiększyć dostępność strony, należy opisać grafikę za pomocą elementów `<desc>` lub `<title>` i dodać atrybut `aria-label` o wartości odpowiadającej tytułowi grafiki. Gdy podasz wysokość i szerokość w pliku `<svg>`, nie musisz tego robić w elementach `<img>`, `<embed>` i `<object>`, natomiast te wartości są potrzebne w stylach CSS.

Technika Clown Car — SVG i dynamicznie dopasowywana grafika pierwszego planu

Format SVG można wykorzystać do tworzenia i udostępniania dynamicznie dopasowywanej grafiki. Możesz wykorzystać obsługę formatu SVG w przeglądarkach oraz obsługę zapytań `media` i grafiki wektorowej w formacie SVG, aby tworzyć dynamicznie dopasowywane rysunki. Aby pobierać odpowiednią grafikę, należy zastosować zapytania `media` w pliku SVG.

Wiadomo, że dzięki grafice tła ze stylów CSS możliwe jest pobieranie tylko potrzebnych rysunków. Podobnie aby zapobiec pobieraniu w plikach SVG wszystkich dołączanych obrazów, można wykorzystać grafikę tła ze stylów CSS zamiast grafiki pierwszego planu. W dynamicznie dopasowywanych plikach SVG należy dołączać wszystkie potrzebne rysunki i na podstawie zapytań `media` wyświetlać tylko odpowiedni z nich (szczegółowe omówienie zapytań `media` znajdziesz w rozdziale 7.).

```
<svg xmlns="http://www.w3.org/2000/svg"
  viewBox="0 0 300 329" preserveAspectRatio="xMidYMid meet">

<title>Tu wpisz wartość atrybutu alt</title>

<style>
  svg {
    background-size: 100% 100%;
    background-repeat: no-repeat;
  }

  @media screen and (max-width: 400px) {
    svg {
      background-image: url(images/small.png");
    }
  }

  @media screen and (min-width: 401px) and (max-width: 700px) {
    svg {
      background-image: url(images/medium.png);
    }
  }

  @media screen and (min-width: 701px) and (max-width: 1000px) {
    svg {
      background-image: url(images/big.png);
    }
  }

  @media screen and (min-width: 1001px) {
    svg {
      background-image: url(images/huge.png);
    }
  }
</style>
</svg>
```

Aby zachować proporcje elementów i zapewnić ich jednolite skalowanie, zastosowałam atrybuty `viewbox` i `preserveAspectRatio`. Wartość atrybutu `viewbox` to lista czterech rozdzielonych odstępami lub przecinkami liczb: minimalnej wartości x, minimalnej wartości y, szerokości i wysokości. Zdefiniowanie szerokości i wysokości okna roboczego pozwala określić proporcje grafiki SVG.

Z uwagi na związane z bezpieczeństwem problemy z importowaniem grafiki rastrowej za pomocą elementu `<img>` i plików SVG należy używać elementu `<object>`, aby dodawać do witryny dynamicznie dopasowywane rysunki. Element `<object>` umożliwia traktowanie zewnętrznych zasobów jak grafiki.

```
<object data="awesomefile.svg" type="image/svg+xml"></object>
```

Element `<object>` domyślnie jest tak szeroki jak element nadrzędny. Jednak, podobnie jak przy dodawaniu grafiki, można określić szerokość i wysokość elementu za pomocą atrybutów `width` i `height` albo przy użyciu właściwości `width` i `height` w stylach CSS. Z uwagi na umieszczone w przykładowym pliku SVG deklaracje `viewbox` i `preserveAspectRatio` element `<object>` domyślnie zachowa zadeklarowane proporcje, jeśli podany zostanie tylko jeden wymiar (szerokość lub wysokość).

Ponieważ w opisanej technice używany jest element `<object>` zamiast `<img>`, nie występuje tu atrybut `alt`. Aby ta technika była dostępna dla czytników ekranu z obsługą formatu SVG[2], należy umieścić zawartość atrybutu `alt` w elemencie `<title>` w pliku SVG.

W elemencie `<object>` osadzany jest plik SVG. Ten plik pobiera grafikę tła, która pasuje do zapytania `media`. Robi to na podstawie wymiarów elementu `<object>`, a nie okna roboczego. Dla pokazanego wcześniej kodu zgłaszane są dwa żądania HTTP. Jedno dotyczy pliku SVG, a drugie odpowiedniej grafiki. Aby wystarczyło jedno żądanie HTTP, w atrybucie `data` elementu `<object>` zapisz identyfikator URI danych (musisz zastosować w nim znaki ucieczki[3]).

Omawianą technikę nazywam Clown Car (czyli: samochód klauna), ponieważ polega na umieszczaniu wielu dużych rysunków (klaunów) w jednym małym pliku graficznym SVG (w samochodzie).

Nauka SVG

Opisałam tu tylko podstawy formatu SVG. Pliki SVG mogą być dostępne, skalują się do ekranów o dowolnej rozdzielczości, a także obsługują animacje oparte na składni SVG lub skryptach JavaScriptu. Pełną kontrolę nad każdym elementem animacji można uzyskać za pomocą interfejsu API modelu SVG DOM. Za pomocą formatu SVG można zrobić bardzo wiele rzeczy, jednak ich omawianie wykracza poza zakres tej książki. W specyfikacji tego formatu w witrynie W3C (*http://www.w3.org/TR/SVG/*) znajdziesz więcej informacji na temat wszystkich elementów, atrybutów i interfejsu API do obsługi animacji.

[2] Zob. *http://www.iheni.com/just-how-accessible-is-svg/*.

[3] Znaki ucieczki w identyfikatorze URI danych są potrzebne w Internet Explorerze 9 i nowszych wersjach tej przeglądarki. Tu tylko pokrótce omawiam technikę Clown Car. Więcej szczegółów i przykładów wraz z rozwiązaniami rezerwowymi dla przeglądarek, które nie obsługują formatu SVG, znajdziesz pod adresem *https://github.com/estelle/clowncar*.

Plik z japońską flagą to bardzo prosty dokument SVG. Pliki SVG mogą być bardzo skomplikowane. Jeśli znasz program Adobe Illustrator, może zauważyłeś, że rysunki można wyeksportować do formatu SVG. Choć jest to bardzo dobry sposób tworzenia skomplikowanych plików SVG, powstaje w ten sposób bardzo długi kod, a wspomniany program jest drogi.

Amaya to bezpłatny program obsługujący podzbiór formatu SVG. Obsługiwane są m.in.: proste kształty, tekst, grafika, zewnętrzne obiekty, przezroczystość oparta na kanale alfa, transformacje oraz animacje. Program Amaya (*http://www.w3.org/Amaya/*) możesz pobrać bezpośrednio z witryny W3C. Amaya jest przydatna w nauce formatu SVG, ponieważ wygenerowany kod źródłowy można badać i edytować. Możesz też wypróbować program Inkscape (*http://inkscape.org/en/*). Jest to edytor grafiki wektorowej. Ten edytor jest programem o otwartym dostępie do kodu źródłowego i oferuje możliwości podobne do aplikacji Illustrator, CorelDraw i Xara. Inkscape używa plików SVG w formacie zgodnym ze standardem W3C.

Format SVG w grze CubeeDoo

W CubeeDoo używamy formatu SVG w dwóch miejscach. Grafiką tła w motywie z kształtami jest sprite w formacie SVG, a dla ikony wyciszenia używamy identyfikatora URI danych wskazującego plik SVG.

W grze dostępnych jest kilka motywów, m.in.: liczby, kolory i kształty. Do tworzenia kształtów służą proste sprite'y w formacie SVG. Poniżej znajduje się kod do tworzenia sprite'u SVG używanego na przednich stronach kart (rysunek 5.2).

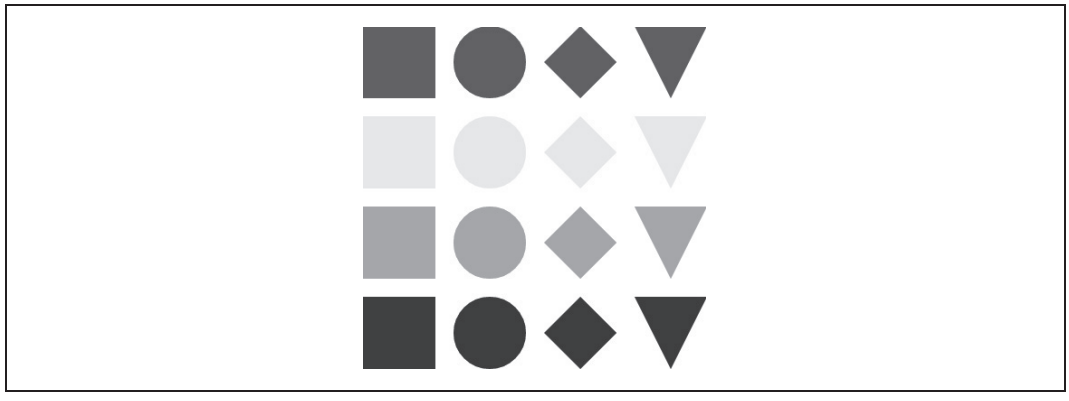
```
<!DOCTYPE svg PUBLIC "-//W3C//DTD SVG 1.0//EN"
  "http://www.w3.org/TR/2001/REC-SVG-20010904/DTD/svg10.dtd">
<svg xmlns="http://www.w3.org/2000/svg" height="400" width="400" version="1.0">
 <desc>Sprite z kwadratami, kółkami, rombami i trójkątami</desc>

 <!-- Kolorowe kwadraty -->
 <rect x="10" y="10" width="80" height="80" style="fill: #d60818;"/>
 <rect x="10" y="110" width="80" height="80" style="fill: #ffff33;"/>
 <rect x="10" y="210" width="80" height="80" style="fill: #00ff00;"/>
 <rect x="10" y="310" width="80" height="80" style="fill: #0000ff;"/>

 <!-- Kolorowe kółka -->
 <circle cx="150" cy="50" r="40" style="fill: #d60818;"/>
 <circle cx="150" cy="150" r="40" style="fill: #ffff33;"/>
 <circle cx="150" cy="250" r="40" style="fill: #00ff00;"/>
 <circle cx="150" cy="350" r="40" style="fill: #0000ff;"/>

 <!-- Romby -->
 <polygon points="250,10 210,50 250,90 290,50" style="fill: #d60818;"/>
 <polygon points="250,110 210,150 250,190 290,150" style="fill: #FFFF33;"/>
 <polygon points="250,210 210,250 250,290 290,250" style="fill: #00ff00;"/>
 <polygon points="250,310 210,350 250,390 290,350" style="fill: #0000ff;"/>

 <!-- Trójkąty -->
 <polygon points="310,10 350,90 390,10" style="fill: #d60818;"/>
 <polygon points="310,110 350,190 390,110" style="fill: #FFFF33;"/>
 <polygon points="310,210 350,290 390,210" style="fill: #00ff00;"/>
 <polygon points="310,310 350,390 390,310" style="fill: #0000ff;"/>
</svg>
```



Rysunek 5.2. Sprite SVG z kształtami

Wiersz 1 zawiera deklarację DTD. W wierszu 3 deklarowany jest element główny. Podana jest tu też wysokość i szerokość grafiki SVG. Choć zgodnie ze specyfikacją nie trzeba podawać tych atrybutów, są one niezbędne, jeśli chcesz używać rysunku SVG jako grafiki tła. W wierszu 4 znajduje się opis, co zwiększa dostępność stron i pomaga w ich optymalizacji ze względu na wyszukiwarki.

W wierszach od 7 do 10 znajdują się deklaracje czterech kwadratów. Wiersz 9 oznacza: utwórz prostokąt w odległości 10 pikseli od lewej krawędzi i 210 pikseli od górnej krawędzi; prostokąt ma być szeroki na 80 pikseli i wysoki na 80 pikseli; wypełnij utworzony kształt kolorem #00FF00.

```
<rect x="10" y="210" width="80" height="80" style="fill: #00FF00;"/>
```

W wierszach od 13 do 16 zdefiniowane są cztery kółka. Wiersz 16 oznacza: znajdź punkt oddalony o 150 pikseli od lewej krawędzi i 350 pikseli od górnej krawędzi; ustaw ten punkt jako środek kółka o promieniu 40 pikseli wypełnionego kolorem #0000FF.

```
<circle cx="150" cy="350" r="40" style="fill: #0000FF;"/>
```

Wiersz od 18 do 28 zawierają osiem wielokątów: cztery romby i trzy trójkąty. Wielokąty deklaruje się przez podanie ich narożników. Wiersz 19 oznacza: ten kształt ma cztery narożniki; pierwszy punkt jest oddalony o 250 pikseli od lewej krawędzi i 10 pikseli od górnej krawędzi; drugi punkt jest oddalony o 210 pikseli od lewej krawędzi i 50 pikseli od górnej krawędzi; trzeci punkt jest oddalony o 250 pikseli od lewej krawędzi i 90 pikseli od górnej krawędzi; czwarty punkt jest oddalony o 290 pikseli od lewej krawędzi i 50 pikseli od górnej krawędzi; obszar między tymi punktami należy wypełnić kolorem #d60818 (jest to odcień czerwonego).

```
<polygon points="250,10 210,50 250,90 290,50" style="fill: #d60818;"/>
```

Pokazany kod tworzy kwadraty, kółka, romby i trójkąty obrócone podstawą do góry.

Te małe rysunki można też zapisać jako identyfikatory URI danych bezpośrednio w pliku CSS lub udostępnić jako grafikę pierwszego planu. Możesz np. dodać zakodowany plik SVG jako identyfikator URI danych:

```
background-image: url(data:image/svg+xml,%3Csvg%20xmlns%3D%22http%3A%2F%2F
www.w3.org%2F2000%2Fsvg%22%20version%3D%221.0%22%3E%3Crect%20x%3D%2220%22%20y
%3D%2220%22%20fill%3D%22%23abcdef%22%20width%3D%22100%25%22%20height
%3D%22100%25%22%20%2F%3E%3C%2Fsvg%3E);
```

W CubeeDoo pojawia się także ikona wyciszenia dźwięku. Identyfikator URI danych dla tej ikony wygląda tak:

```
background-image:
  background-image:
  url("data:image/svg+xml;utf8,%3Csvg%20xmlns=
'http://www.w3.org/2000/svg'%20width='100'%20height='100'%3E
%3Cpolygon%20points='39,13%2022,28%206,28%206,47%2022,48%2039,63
%2039,14'%20style='stroke:#111111;stroke-width:5;stroke-linejoin:round;
fill:#111111;'%20/%3E%3Cpath%20d='M%2048,50%2069,26'%20%20style='fill:none;
stroke:#111111;stroke-width:5;stroke-linecap:round'%20/%3E%3Cpath%20
%20d='M%2069,50%2048,26'%20style='fill:none;stroke:#111111;stroke-width:5;
stroke-linecap:round'%20/%3E%3C/svg%3E");
```

Dane w tym przykładzie są mało czytelne dla ludzi. Utworzono jest za pomocą programu Amaya. Składnia powinna jednak wyglądać znajomo. Używamy tu właściwości `background-image` ze stylów CSS. Zamiast wywołania `url(path/mute.jpg)` lub nawet `url(path/mute.svg)` stosujemy instrukcję `url("data:image/svg+xml;utf8,<svg... /></svg>");`. Cały plik SVG ze znakami ucieczki jest ujęty w cudzysłów.

Zgodnie ze specyfikacją dla wersji Internet Explorera obsługujących format SVG (obecnie są to wersje 9 i 10) w identyfikatorach URI danych należy stosować znaki ucieczki.

Element canvas

Specyfikacja HTML5 Canvas dotyczy przeznaczonego do tworzenia rysunków interfejsu API JavaScriptu. Ten interfejs umożliwia zdefiniowanie na stronie HTML obiektu płótna w postaci elementu `<canvas>`, po którym można rysować. Rysunki utworzone w ten sposób można nawet wstawiać w stylach CSS jako grafikę tła.

Rysować można w przestrzeni dwu- lub trójwymiarowej (do tworzenia rysunków trójwymiarowych służy technologia WebGL). Rysunki dwuwymiarowe są obsługiwane we wszystkich nowych przeglądarkach internetowych. Technologia WebGL dopiero zyskuje popularność w świecie mobilnym. Ze względu na wydajność należy ją stosować tylko w urządzeniach ze sprzętową akceleracją grafiki.

W wersji dwuwymiarowej dostępny jest prosty, ale dający duże możliwości interfejs API do szybkiego rysowania na dwuwymiarowych bitmapach. Nie występuje tu konkretny format plików, a rysować można tylko za pomocą skryptów. Nie są dostępne węzły modelu DOM odpowiadające poszczególnym rysowanym kształtom. Element `<canvas>` służy do rysowania pikseli, a nie wektorów. Używanie tylko jednego węzła sprawia, że element `<canvas>` wydaje się wygodny w obsłudze dla przeglądarek mobilnych, jednak wysokie zapotrzebowanie na cykle procesora ze strony animacji JavaScriptu może szybko wyczerpywać baterie urządzeń przenośnych (choć dzięki akceleracji sprzętowej zużycie baterii jest niższe).

Pierwszy element <canvas>

Ponieważ jest to bardzo podstawowe wprowadzenie do elementu `<canvas>`, omawiam tu tylko proste kształty i linie. Jeśli nie znasz JavaScriptu, składnia może być początkowo mało zrozumiała. Jeżeli jednak znasz ten język, powinieneś rozumieć przedstawiony kod.

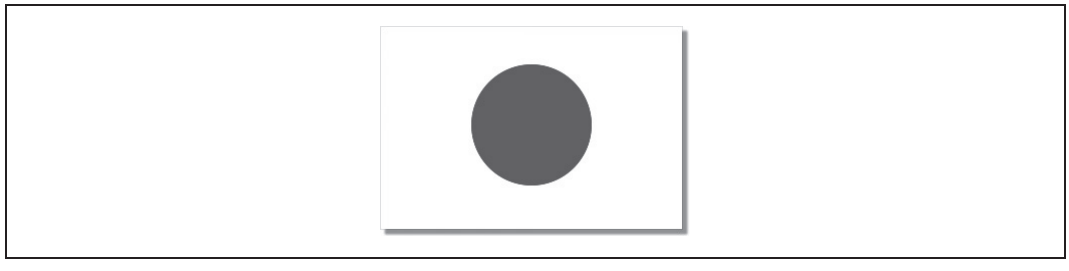
Pierwszy krok polega na dodaniu do dokumentu elementu `<canvas>`. W HTML-u jest to jedyna potrzebna operacja.

```
<canvas id="flag" width="320" height="220">
  Przeglądarka nie obsługuje elementu <canvas>. Gdyby obsługiwała, zobaczyłbyś flagę.
</canvas>
```

To już wszystko, co trzeba zrobić w HTML-u, aby dodać element `<canvas>`. Można napisać tylko `<canvas></canvas>`. Atrybut `id` jest używany, aby ułatwić wskazywanie elementu w JavaScripcie, choć można też go wskazać za pomocą modelu DOM. Dodałam również zastępczy tekst dla użytkowników, których przeglądarka nie obsługuje elementu `<canvas>` lub którzy z innych przyczyn nie mogą zobaczyć tego elementu.

Element `<canvas>` w obecnej wersji to interfejs API, który nie zapewnia dostępności stron. Jedyna możliwość poprawienia dostępności to dodanie atrybutu `aria-label`.

Ten kod tworzy pustą tablicę do rysowania — płótno. Wszystkie pozostałe zadania wykonuje się w JavaScripcie. Tu kod ponownie tworzy japońską flagę. Przedstawia ją rysunek 5.3.



Rysunek 5.3. Japońska flaga utworzona jako element <canvas>

Następny krok polega na utworzeniu rysunku na płótnie. Od tego momentu wszystko dzieje się w JavaScripcie. Węzeł `<canvas>` można wskazać w prostym JavaScripcie na trzy sposoby:

```
document.getElementById('flag')
document.getElementsByTagName('canvas')[0]
document.querySelector('#flag')
```

Potem należy zainicjować kontekst dwuwymiarowy i rozpocząć rysowanie za pomocą poleceń interfejsu API tego kontekstu. Kod rysuje tu japońską flagę.

```
1  <script>
2  var el= document.getElementById("flag");
3
4  if (el && el.getContext) {
5    var context = el.getContext('2d');
6      if (context) {
7        context.fillStyle = "#ffffff";
8        context.strokeStyle = "#cccccc";
9        context.lineWidth = 1;
10       context.shadowOffsetX = 5;
11       context.shadowOffsetY = 5;
12       context.shadowBlur = 4;
13       context.shadowColor = 'rgba(0, 0, 0, 0.4)';
14       context.strokeRect(10, 10, 300, 200);
15       context.fillRect(10, 10, 300, 200);
16       context.shadowColor='rgba(0,0,0,0)';
17       context.beginPath();
18       context.fillStyle = "#d60818";
19       context.arc(160, 107, 60, 0, Math.PI*2, false);
```

```
20        context.closePath();
21        context.fill();
22    }
23 }
24 </script>
```

Kod w wierszu 2 znajduje odpowiedni element `<canvas>` na podstawie atrybutu `id` tego elementu. Przed utworzeniem kontekstu dwuwymiarowego należy sprawdzić, że element `<canvas>` został odnaleziony *oraz* że przeglądarka obsługuje takie elementy. W tym celu w wierszu 4 kod sprawdza, czy dostępna jest metoda `getContext`.

Można wykorzystać skrypty do wykrywania funkcji (np. Modernizr — *http://modernizr.com/*) do sprawdzania, czy przeglądarka obsługuje element `<canvas>` i inne nowe mechanizmy. Modernizr pozwala wykrywać wszystkie funkcje lub tylko potrzebny w danym momencie mechanizm. Nie używam tu tego narzędzia, ponieważ chcę pokazać, jak bezpośrednio sprawdzać dostępność funkcji. Zwykle jednak warto stosować Modernizr, chyba że chcesz zminimalizować liczbę używanych zewnętrznych skryptów i żądań HTTP.

W wierszu 5 kod tworzy referencję do kontekstu, wywołując metodę `getContext(contextId)` elementu `<canvas>`. Właściwym kontekstem dla tego elementu jest `2d`. Jeśli tworzenie kontekstu kończy się powodzeniem (co kod sprawdza w wierszu 6), można wreszcie przejść do rysowania, za co odpowiada reszta skryptu.

Wprawdzie jest to technika eksperymentalna, ale jeśli chcesz w przeglądarce z silnikiem WebKit umieścić rysunek z elementu `<canvas>` jako tło za pomocą stylów CSS (*http://updates.html5rocks.com/2012/12/Canvas-driven-background-images*)[4], to zamiast wywoływać element `<canvas>` w modelu DOM, możesz dodać rysunek jako grafikę tła. W stylach CSS należy zastosować następujący kod:

```
background: -webkit-canvas(theCanvas);
```

W JavaScripcie umieść poniższe wywołanie:

```
var context = document.getCSSCanvasContext("2d", "theCanvas", 320, 220);
```

Drugi parametr jest tu nazwą używanego elementu `<canvas>`, którą bez cudzysłowu podano w stylach CSS.

Do wiersza 6, a nawet do wiersza 13 kod nic nie rysuje. Do wiersza 6 definiowany jest jedynie kontekst elementu `<canvas>`, który umożliwia rysowanie i modyfikowanie pikseli.

Przed narysowaniem kształtu trzeba zdefiniować jego wygląd i styl. W tym celu należy ustawić właściwości w obiekcie `context`. Tu kod definiuje wygląd obramowania (właściwości `stroke` i `linewidth`), kolor tła (właściwość `fill`) i cień (właściwości `shadowOffsetX`, `shadowOffsetY`, `shadowBlur` i `shadowColor`) dla pierwszego prostokąta, który jest rysowany za pomocą metody `strokeRect()` w wierszu 14. Przekazywane są tu te same parametry co w przykładowym pliku SVG (10, 10, 300, 200). Te cztery wartości określają współrzędne x i y oraz szerokość i wysokość.

[4] Firefox 4 i nowsze wersje tej przeglądarki także obsługują element `<canvas>` w stylach CSS. Pozwala to dynamicznie tworzyć wirtualny element `<canvas>` za pomocą JavaScriptu i wywołania `-moz-element('#elementCanvas')` w stylach CSS.

Po wykonaniu polecenia skrypt „zapomina" o tym, co zrobił wcześniej, i przechodzi do następnego wiersza. Inaczej niż w przykładowym pliku SVG z wcześniejszego punktu, tu prostokąt jest rysowany w elemencie `<canvas>` i nie jest częścią modelu DOM. Właściwości `stroke`, `fill`, `linewidth` i `border` są zapamiętywane, jednak przeglądarka i skrypt nie potrafią określić, co zostało narysowane. Jeśli chcesz rejestrować, co i w którym miejscu jest rysowane w elemencie `<canvas>`, wykorzystaj metodę `getImageData()` kontekstu do zapisania wartości składowych czerwonej, zielonej i niebieskiej oraz kanału alfa pikseli.

W wierszu 15 kod wywołuje metodę `fillRect`, co prowadzi do narysowania drugiego prostokąta za pomocą ustawionej wcześniej właściwości `fillStyle`. Trzeba tu ponownie podać współrzędne, ponieważ pierwszy prostokąt nie jest zapamiętywany w modelu DOM (choć można uzyskać dostęp do informacji o pikselach).

W obu wywołaniach metod tworzących prostokąty (wiersze 14 i 15) używane są identyczne parametry — 10, 10, 300 i 200. Powoduje to narysowanie prostokąta z wypełnieniem bezpośrednio na prostokącie reprezentującym cień. Można utworzyć obiekt ze współrzędnymi i przekazać go do obu metod, jednak nie da się za pomocą elementu `<canvas>` uzyskać dostępu do współrzędnych pierwszego prostokąta i skopiować ich do drugiego.

Kod najpierw rysuje prostokąt, a potem go wypełnia. Gdyby zastąpić kolejność tych operacji, cień znajdowałby się nad kolorem tła. Ponieważ współrzędne są identyczne, a obramowanie jest ustawione na 1 piksel szerokości, ostatecznie ramka ma 0,5 piksela, ponieważ wypełnienie zajmuje wewnętrzną połowę jej szerokości.

Jak wcześniej wspomniałam, w momencie rozpoczęcia rysowania kółka (słońca) na fladze, w modelu DOM nie ma informacji o narysowanych wcześniej elementach. To prawda, JavaScript zapamiętuje ustawione wartości właściwości, np. `shadowColor`. Zapamiętuje też ostatnie operacje niezależnie od tego, czy dana figura została narysowana. Jednak piksele w elemencie `<canvas>` to tylko punkty o określonym kolorze. Ponieważ czerwone kółko nie ma mieć cienia, przed narysowaniem kółka trzeba ustawić właściwość `shadowColor` tak, aby była przezroczysta, co robi kod w wierszu 16.

Instrukcje dotyczące rysowania kółka rozpoczynają się od wywołania `beginPath()` (wiersz 17), a kończą wywołaniem `closePath()` (wiersz 20). Skrypt zapamiętuje operacje rysowania niezależnie od tego, czy element został już narysowany. Jeśli rysujesz kółko, a następnie bez otwierania i zamykania kontekstu dodasz kilka linii, to w momencie rysowania linii kółko nadal będzie w pamięci. Nowa linia może wtedy przechodzić przez kółko i przecinać je na pół. Tu unikam tego dzięki otwarciu i zamknięciu ścieżki za pomocą wywoływań `beginPath()` i `closePath()`.

Kółko jest definiowane za pomocą wywołania `context.arc(współrzędna_x, współrzędna_y, promień, kąt_początkowy, kąt_końcowy, niezgodnie_z_ruchem_wskazówek_zegara)`. Powoduje ono dodanie punktów do krzywej, w wyniku czego powstaje wirtualny okrąg opisany za pomocą argumentów wywołania. Tu tym wywołaniem jest `context.arc(160, 107, 60, 0, Math.PI*2, false);`. Rysowanie zaczyna się od określonego kąta początkowego (tu jest to 0, czyli pozioma linia biegnąca w prawo) i kończy się podanym kątem końcowym. Rysowanie odbywa się zgodnie z określonym kierunkiem (tu zgodnie z ruchem wskazówek zegara). Gdyby *kąt końcowy* był mniejszy niż 2π, „kółko" byłoby przycięte — punkty początkowy i końcowy byłyby połączone prostą linią. Wartość π powoduje utworzenie półokręgu.

Ponadto w wierszu 18. zmieniany jest kolor wypełnienia (z białego na czerwony). Następnie kod wypełnia utworzone kółko za pomocą metody `fill()` (wiersz 21) według koloru ustawionego we właściwości `fillStyle`.

Nie opisałam tu nawet podstaw możliwości, jakie daje element `<canvas>`. Pod adresem *http://ie.microsoft.com/testdrive/Graphics/CanvasPad/Default.html* znajdziesz ciekawą stronę, która pomaga w nauce używania prostych kształtów, kolorów, cieni, tekstu, rysunków, transformacji, animacji i przesuwania kursora w elemencie `<canvas>`.

Przykładowy kod wykorzystujący element <canvas>

W przykładowej grze w celu zwiększenia trudności gry na wyższych poziomach zmieniamy tło planszy. W animacji pojawiają się kształty z przednich stron kart, przez co najwyższe poziomy są niezwykle trudne.

W internetowych materiałach do tego rozdziału (*http://www.standardista.com/mobile/*) znajduje się kilka przykładowych fragmentów kodu wykorzystujących element `<canvas>`. Spróbuj narysować japońską flagę, a następnie przekształcić ją w statyczny rysunek Pac-Mana (czerwone kółko zmień w żółte, a następnie dodaj buźkę, czarne kółka jako oczy, a obok kółka trzy małe kropki do zjedzenia). W przykładach z materiałów znajdziesz też tekst z kodem wykraczającym poza zakres tej książki. Opisana jest w nim funkcja zmieniająca kolory Pac-Mana. Pomoże Ci to nauczyć się, jak uzyskać dostęp do pikseli narysowanych już w elemencie `<canvas>`. Poznasz też inne metody rysowania na stronie.

Choć w tej książce podałam przykład wykorzystania elementu `<canvas>` w grze CubeeDoo, obecnie nie stosowałabym dynamicznie generowanych elementów `<canvas>` w produkcyjnej wersji mobilnej aplikacji sieciowej. Przy dzisiejszym stanie technologii zużycie baterii w wyniku wykonywania animacji w elemencie `<canvas>` za pomocą JavaScriptu byłoby dla użytkowników bardzo nieprzyjemne, jednak akceleracja sprzętowa dla elementu `<canvas>` jest coraz lepsza.

Element <canvas> a element <svg>

Elementy `<canvas>` i `<svg>` mają w HTML5 kilka podobnych cech. Często porównuje się ze sobą te elementy. Oba związane są z technologiami sieciowymi, które pozwalają tworzyć zaawansowane grafiki w przeglądarkach. Jednak między tymi technologiami występują poważne różnice.

W formacie SVG rysowanie odbywa się za pomocą XML-a. W elementach `<canvas>` używany jest do tego JavaScript. Przy używaniu tego elementu piksele są rysowane na płótnie, a po ich dodaniu każdy piksel jest zapominany. W SVG powstają węzły modelu DOM. Można do nich uzyskać dostęp do momentu ich usunięcia lub opuszczenia strony przez użytkownika. Obie te technologie mają wady i zalety.

Rysunki w formacie SVG dostosowują się do dowolnej rozdzielczości, dzięki czemu doskonale nadają się do interfejsów o dowolnych wymiarach (można skalować je pod kątem rozdzielczości każdego ekranu). Rysunki w tym formacie są zapisywane w plikach XML, co pozwala łatwo zwiększyć dostępność stron. Rysunki SVG można animować albo za pomocą składni deklaratywnej, albo przy użyciu JavaScriptu. Każdy narysowany element jest częścią modelu SVG DOM i można uzyskać dostęp do niego w JavaScripcie. Jednak korzystanie z modelu DOM spowalnia działanie strony, co jest ważne i odczuwalne zwłaszcza w środowisku mobilnym.

W elementach <canvas> wszystko jest rysowane za pomocą pikseli. Zwiększenie rysunku może prowadzić do powstania efektu pikselizacji. Element <canvas> zmniejsza dostępność stron. Można jedynie dodać rezerwowy tekst pokazywany, gdy nie da się wyświetlić tego elementu. Aby umożliwić interakcje, trzeba ponownie narysować każdy piksel. Narysowane elementy nie są dostępne jako węzły DOM. Nie ma też interfejsu API do tworzenia animacji. Zamiast tego zwykle używa się zegarów lub wywołań `requestAnimationFrame`, aby aktualizować element <canvas> w krótkich odstępach czasu. Element <canvas> zapewnia powierzchnię, po której można rysować za pomocą interfejsu API wybranego kontekstu. Ten element doskonale nadaje się do edytowania grafiki, generowania grafiki rastrowej na potrzeby gier lub tworzenia fraktali, a także do wykonywania operacji wymagających manipulowania danymi na poziomie pikseli. Rysunki utworzone za pomocą interfejsu API elementu <canvas> można też eksportować.

Dwuwymiarowy kontekst elementu <canvas> jest dobrze obsługiwany we wszystkich przeglądarkach (od Internet Explorera 9). Także grafika SVG jest poprawnie interpretowana, ale w różnych formatach (od Internet Explorera 9 i Androida 3). Choć obie te technologie są poprawnie obsługiwane, obie mają też pewne wady.

Wydajność grafiki SVG bywa niska. Przeglądarki mobilne mają trudności z obsługiwaniem dużej liczby elementów modelu DOM. Każdy dodatkowy węzeł modelu DOM zajmuje pamięć i wymaga ponownego przetworzenia w momencie zmiany układu strony. Dlatego przy tworzeniu aplikacji na przeglądarki mobilne należy ograniczać liczbę dodawanych węzłów modelu DOM. Rysunki SVG składają się z węzłów modelu DOM. Większa liczba takich węzłów może prowadzić do spadku wydajności, a w skrajnych przypadkach nawet do zawieszenia się przeglądarki mobilnej. Z kolei element <canvas> przy stosowaniu animacji (w odróżnieniu od generowania pojedynczych rysunków) może szybko zużyć baterię urządzenia. Jednak we wszystkich popularnych przeglądarkach dostępna jest akceleracja sprzętowa dla tego elementu, dzięki czemu czas rysowania i aktualizowania jest znacznie krótszy, a zużycie baterii — mniejsze.

Sam uwzględnij wady i zalety obu technologii, zanim zdecydujesz się na zastosowanie jednej z nich (lub zrezygnowanie z obu).

WebGL

Obsługa grafiki trójwymiarowej i technologii WebGL jest dopiero wprowadzana. Obecnie występują poważne problemy z wydajnością, znacznym zużyciem baterii i ograniczoną obsługą w urządzeniach przenośnych. W czasie, gdy powstawała ta książka, najlepszą obsługę technologii WebGL zapewniał system BlackBerry 10. Niedawno obsługę tej technologii dodano także do systemu Firefox OS. Przy korzystaniu ze starszych urządzeń przenośnych warto dobrze się zastanowić nad stosowaniem tej technologii. Duże obciążenie procesora prowadzi do szybkiego zużycia baterii, a JavaScript mocno obciąża procesor. Nie chcesz przecież wyczerpać baterii. W urządzeniach obsługujących technologię WebGL (np. w urządzeniach z systemem BlackBerry 10) do przetwarzania grafiki w tym formacie używany jest procesor graficzny. Pozwala to zwiększyć wydajność i ograniczyć zużycie baterii w porównaniu z używaniem zwykłego procesora, jednak i tak waham się z zachęcaniem do korzystania z tej technologii. Jeśli zdecydujesz się użyć WebGL, zawsze pamiętaj o wydajności (np. o ilości zajmowanej pamięci i zużyciu baterii).

Elementy <audio> i <video>

Przed pojawieniem się HTML5 nie istniał standardowy sposób osadzania filmów wideo na stronach internetowych. Zamiast tego filmy z internetu wyświetlane były za pomocą niezależnych wtyczek, takich jak Flash lub QuickTime. Ponadto nie było łatwego sposobu na tworzenie dostępnych multimediów. Dlatego dołączone pliki wideo były często niedostępne dla osób z wadami wzroku i słuchu.

W HTML5 zdefiniowany jest standardowy sposób osadzania plików dźwiękowych i wideo na stronach internetowych. Służą do tego elementy `<video>` i `<audio>`. Oba są obsługiwane we wszystkich popularnych przeglądarkach mobilnych (z wyjątkiem Opery Mini), jednak na razie nie wszystkie przeglądarki obsługuję te same formaty filmów. Przed omówieniem dołączania nagrań dźwiękowych i wideo do dokumentu trzeba opisać kodeki plików multimedialnych oraz obsługę multimediów w przeglądarkach. Jest to konieczne, ponieważ w różnych przeglądarkach należy udostępniać odmienne pliki multimedialne. Ponadto trzeba zapewniać rozwiązania rezerwowe dla przeglądarek, które nie obsługują multimediów określonego typu.

Typy plików multimedialnych

Gdy przeglądarki będą obsługiwać elementy `<video>` i `<audio>` z HTML5 oraz standardowe typy plików multimedialnych, nie trzeba będzie stosować niezależnych wtyczek do odtwarzania multimediów. Jednak na razie poszczególne przeglądarki obsługują różne kodeki audio i wideo. Jak prawdopodobnie wiesz, na iPadach i iPhone'ach nie działa Flash. Działają za to elementy `<video>` i `<audio>` oraz formaty H.264 (dla wideo) i AAC (dla dźwięku). Te formaty omawiam w następnym punkcie. Wszystkie nowe przeglądarki obsługują element `<video>` z HTML5, przy czym robią to dla różnych formatów. Internet Explorer 9, Safari, Chrome, Android oraz iOS obsługują format MPEG4/H.264 (pliki *.mp4*), natomiast Firefox 4 i jego nowsze wersje, Chrome, Opera oraz Android 2.3 i nowsze wersje poprawnie interpretują format WebM/VP8 (pliki *.webm*). Ten ostatni format działa także także w Internet Explorerze 9, jeśli w systemie zainstalowane są potrzebne kodeki. Listę obsługiwanych formatów znajdziesz w tabeli 5.1.

Tabela 5.1. Obsługa kodeków wideo w przeglądarkach (w przeglądarce Internet Explorer 9 można odrębnie zainstalować kodeki Ogg i WebM)

	iPhone i iPad	Android	BlackBerry	Opera Mobile	Opera Mini	Windows (Internet Explorer)	Chrome w Androidzie	Firefox w Androidzie
<video>	Tak	Tak	7	11		9	Tak	Tak
Ogg		2		11		(9*)		Tak
H.264	Tak	3.0[a]	7			9	Tak	Tak*
WebM		2.3		14		(9*)	Tak	Tak

[a] Zobacz stronę *http://www.broken-links.com/2010/07/08/making-html5-video-work-on-android-phones/*.

Istnieje kilka kodeków wideo. Najważniejsze z nich to: Theora/Ogg, WP8 i H.264. Theora/Ogg (pliki *.ogv*) to otwarty standard z wbudowaną obsługą w przeglądarkach Firefox 3.5, Chrome 4 i Opera 10.5 oraz nowszych. W Internet Explorerze ten format jest obsługiwany po zainstalowaniu wtyczki. WebM to nowszy format używany razem z kodekiem wideo VP8. Wbudowana obsługa tego formatu jest dostępna w najnowszych wersjach przeglądarek Chrome i Mozilla Firefox oraz w Operze 10.6 i nowszych.

Obecnie kodek VP8 jest dostępny bez opłat licencyjnych. Kodek ten jest opatentowany, jednak właściciel patentu (firma Google) udostępnia ten kodek bezpłatnie. Niestety, choć jest on dobrze obsługiwany w nowych przeglądarkach, prawa do patentów związanych z formatem WebM i kodekiem VP8 rości sobie także Nokia. Dlatego w najbliższej przyszłości ten format prawdopodobnie nie stanie się standardem sieciowym.

Format H.264 jest przeznaczony dla urządzeń o niskiej, umiarkowanej i wysokiej przepustowości. Można go odtwarzać za pomocą technologii Adobe Flash oraz w różnych urządzeniach przenośnych (m.in. w systemie Android i w iPhone'ach). Nie jest to jednak otwarty standard. Wykupienie licencji może być kosztowne. Producent przeglądarki Chrome zapowiedział, że zrezygnuje z obsługi tego formatu, jednak na razie tego nie zrobił. W Firefoksie ten format jest obsługiwany, jeśli kodek jest zainstalowany w używanym systemie operacyjnym. Do samo dotyczy Opery w urządzeniach przenośnych.

Na razie, co widać w tabeli 5.1, nie istnieje format działający we wszystkich przeglądarkach. Aby film można było odtwarzać wszędzie, trzeba zakodować go w więcej niż jednym formacie.

Obecnie przy tworzeniu witryn na telefony komórkowe w Stanach Zjednoczonych najlepiej jest używać formatu H.264. Trzeba jednak pamiętać, że poziom obsługi tego formatu może się zmienić. Jeśli docelowe urządzenia przenośne obejmują systemy GPS, konsole do gier itd., warto pamiętać, że najpopularniejszą przeglądarką mobilną na świecie jest Opera. Także w Stanach Zjednoczonych jest to najczęściej używana przeglądarka w urządzeniach innych niż telefony i komputery.

Dodawanie elementu <video> do witryny

Wprawdzie jeśli film docelowo ma być wyświetlany na smartfonach w Stanach Zjednoczonych, wystarczy wykorzystać format H.264, to aby uzyskać maksymalną zgodność z różnymi urządzeniami, trzeba przygotować dwie wersje nagrania. Utwórz wersje w formatach WebM (WP8 dla wideo i Vorbis dla audio) i MP4 (H.264 dla wideo i AAC dla audio). Odnośniki do obu plików wideo podaj za pomocą elementu `<video>` z HTML5 i podrzędnych znaczników `<source>`. Jako domyślny ustaw odtwarzacz oparty na Flashu.

Atrybuty elementów <video> i <audio>

Jest kilka atrybutów elementów `<video>` i `<audio>`, które służą do kontrolowania wyglądu i działania osadzanych multimediów.

Oto atrybuty obsługiwane w elementach `<video>` i `<audio>`:

`src`
: Atrybut `src` przyjmuje jako wartość adres URL pliku wideo lub audio. Można go zastąpić kilkoma podrzędnymi znacznikami `<source>`.

`autoplay`
: Atrybut logiczny `autoplay` informuje przeglądarkę, że odtwarzanie nagrania rozpoczyna się automatycznie (bez oczekiwania na wciśnięcie przycisku *Play* przez użytkownika). Ten atrybut warto stosować tylko na tych stronach, których główną zawartością jest odtwarzany film.

`loop`
: Atrybut logiczny `loop` powoduje, że nagranie filmowe lub dźwiękowe jest odtwarzane w pętli. Gdy zostanie uruchomione, będzie odtwarzane w kółko do czasu wstrzymania lub zatrzymania. Jeśli ten atrybut jest używany, nagranie po dotarciu do końca jest uruchamiane ponownie.

`controls`
: Jeśli używany jest ten atrybut logiczny, przeglądarka powinna wyświetlać kontrolki do sterowania multimediami (przyciski do przeskakiwania do różnych miejsc nagrania, włączania go, wstrzymywania itd.).

`preload`
: Atrybut `preload` informuje przeglądarkę, jak dużą część filmu powinna pobrać przed rozpoczęciem jego odtwarzania. Jeśli nie podasz tego atrybutu lub ustawisz jego wartość na `none`, przeglądarka nie będzie wstępnie pobierać danych. Jeżeli ten atrybut jest używany i ma wartość `auto`, przeglądarka wstępnie pobiera plik. Ustawienie `metadata` powoduje, że przeglądarka pobiera wymiary, długość i inne metadane dotyczące nagrania, natomiast nie ściąga samego pliku multimedialnego.

Poniższe atrybuty dotyczą tylko elementu `<video>` (nie są obsługiwane w elemencie `<audio>`):

`poster`
: Atrybut `poster` przyjmuje adres URL rysunku używanego jako graficzny wypełniacz do momentu rozpoczęcia odtwarzania filmu. Jeśli nie podasz tego atrybutu, odtwarzacz wyświetli pierwszą klatkę z filmu (zwykle jest to czarny prostokąt).

`width`
: Wartość atrybutu `width` to szerokość (w pikselach) pola z filmem.

`height`
: Wartość atrybutu `height` to wysokość (w pikselach) pola z filmem.

Oto przykładowa deklaracja elementu `<video>` (opis poszczególnych komponentów przedstawia tabela 5.2):

```
<video autoplay controls loop poster="poster.jpg" preload="metadata"
src="video.mp4" height="360" width="480">Tekst zastępczy</video>
```

Tabela 5.2. Komponenty przykładowej deklaracji elementu <video>

Komponent	Opis
`<video>`	Znacznik `<video>`
`autoplay`	Jeśli jest ustawiony, odtwarzanie filmu rozpoczyna się w momencie wczytania strony
`controls`	Jeśi jest ustawiony, dostępny jest pasek sterowania
`loop`	Jeśli jest ustawiony, wideo jest odtwarzane w pętli
`poster="/img/poster.jpg"`	Jeśli jest ustawiony, wyświetlany jest podgląd nagrania
`preload="metadata"`	Przyjmuje wartości `none`, `metadata` i `auto`
`src="/video/video.mp4"`	Odnośnik do pliku wideo
`height="360"`	Wysokość obrazu w wideo
`width="480">`	Szerokość obrazu w wideo
`Tekst zastępczy`	Dowolny poprawny kod w HTML-u (standardowo podawany jest odnośnik do danego filmu)
`</video>`	Wymagany zamykający znacznik `</video>`

Elementy <audio> i <video> z HTML5 umożliwiają powiązanie etykiet z osadzanymi multimediami. W HTML5 te elementy są częścią modelu DOM, co umożliwia określanie dla nich stylów CSS i zapewnia rozbudowany interfejs API, który daje programistom kontrolę nad odtwarzaniem filmu przy użyciu licznych nowych metod i właściwości JavaScriptu (takich jak: play(), pause(), muted i ended).

Gdy element <video> z HTML5 będzie w pełni obsługiwany, a wszystkie przeglądarki będą udostępniać ten sam kodek, wystarczy następujący kod:

```
<video src="myVideo.mp4" width="400" height="300"
  controls poster="myImage.jpg">
  Przeglądarka nie obsługuje HTML5, ale możesz pobrać plik
  <a href="myVideo.ogv"> pod tym adresem</a>.
</video>
```

Niestety, na razie ten kod zadziała tylko w niektórych przeglądarkach. Jak wcześniej wspomniałam, nie wszystkie przeglądarki obsługują ten sam kodek, dlatego dla poszczególnych przeglądarek trzeba udostępnić różne pliki.

Aby było to możliwe, w HTML5 dostępny jest element <source>. Ten element pozwala wskazać więcej niż jeden plik multimedialny. Element <source> ma trzy atrybuty (oprócz atrybutów globalnych): src, type i media.

Aby dynamicznie zmieniać odtwarzane multimedia, zmodyfikuj atrybut src w znaczniku <video> lub <audio>. Nie wystarczy zmienić wartości atrybutu src w elemencie <source>. Aby wybrać typ plik obsługiwany przez przeglądarkę, wykorzystaj metodę canPlayType().

Atrybut type określa typ zasobów multimedialnych, dzięki czemu przeglądarka przed pobraniem piku może ustalić, czy obsługuje dany typ. Wartością atrybutu type musi być poprawny typ MIME.

Nawet w sytuacji, gdy nie wszystkie przeglądarki obsługują ten sam kodek, kod nie jest zanadto skomplikowany. W przykładowej grze mogliśmy dodać film z instrukcjami do gry. Nie zrobiliśmy tego, ale było to możliwe. Gdybyśmy to zrobili, kod zapewniający obsługę wideo we wszystkich przeglądarkach powinien wyglądać tak:

```
<video width="400" height="300" preload="none" poster="posterImg.jpg"
  controls>
  <source src="myVideo.mp4" type="video/mp4; codecs=avc1.42E01E, mp4a.40.2"/>
  <source src="myVideo.webm" type="video/webm; codecs=vp8, vorbis"/>
  <source src="myVideo.ogg" type="video/ogg; codecs=dirac, speex"/>
  <object width="400" height="324" type="application/x-shockwave-flash"
    data="myVideo.swf"/>
  <param name="movie" value="myVideo.swf"/>
  <param name="flashvars"
    value="image=posterImg.jpg&amp;file=myVideo.mp4"/>
  <!-- Rozwiązanie rezerwowe -->
  <a href="linktovideo">
    <img src="posterImg.jpg" width="400" height="300"
      alt="Fantastyczne wideo"/>
  </a>
  </object>
</video>
```

Jeśli przeglądarka obsługuje element <video> z HTML5, zostanie on użyty. Jeżeli przeglądarka nie obsługuje pierwszego typu multimediów, sprawdzi następny. Gdy element <video> z HTML5 nie jest obsługiwany, przeglądarka używa technologii Adobe Flash. Jeśli brakuje obsługi zarówno Flasha, jak i elementu <video>, wyświetlana jest zastępcza grafika. Możesz też dodać odnośnik umożliwiający pobranie nagrania.

Film z pliku we Flashu jest zadeklarowany jako wyższy o 24 piksele od innych wersji. Wynika to z tego, że kontrolki Flasha zajmują 24 piksele w pionie pod oknem z filmem, natomiast w formatach wideo w HTML5 kontrolki są nakładane na obraz.

Jeśli aplikacja jest przeznaczona tylko na nowe urządzenia przenośne, można pominąć Flasha i dodać elementy <track> (opis tej techniki znajdziesz w podpunkcie „Element <track>".

```
<video width="400" height="300" preload="none" poster=
  "posterImg.jpg" controls>
  <source src="myVideo.mp4" type="video/mp4;
    codecs=avc1.42E01E, mp4a.40.2"/>
  <source src="myVideo.webm" type="video/webm;
    codecs=vp8, vorbis"/>
  <source src="myVideo.ogg" type="video/ogg;
    codecs=theora, vorbis"/>
    <img src="posterImg.jpg" width="400" height="300"
      alt="Tu tytuł filmu" title="Przeglądarka nie obsługuje
      odtwarzania filmów"/>
  <track kind="subtitles" label="English" src="en.vtt"
    srclang="en" default></track>
  <track kind="subtitles" label="Deutsche" src="de.vtt"
    srclang="de"></track>
</video>
```

Pliki wideo zwykle obejmują ścieżki audio i wideo. Ścieżki audio zawierają znaczniki umożliwiające synchronizację dźwięku z obrazem. Poszczególne ścieżki mogą zawierać metadane określające np. proporcje obrazu lub język ścieżki dźwiękowej. Także w kontenerach mogą znajdować się metadane (np. tytuł filmu, okładka płyty z filmem, numer odcinka serialu itd.).

Do dokumentu można też dodać element <audio>:

```
<audio id="sound">
  <source src="music.mp3" type="audio/mp3"/>
  <source src="music.ogg" type="audio/ogg"/>
  <!-- Można też dodać wersję we Flashu na potrzeby przeglądarek bez obsługi elementu <audio> -->
</audio>
```

W artykule w serwisie Dev.Opera (*http://dev.opera.com/articles/view/everything-you-need-to-know-↪about-html5-video-and-audio/*) znajdziesz bardzo szczegółowe instrukcje na temat wykrywania obsługi elementów.

Element <track>

Aby pliki wideo i audio były dostępne dla użytkowników z wadami słuchu, a nawet dla osób, które nie rozumieją języka z nagrania, można dodać do filmu podpisy, wskazując plik z nimi w elemencie <track>.

Atrybut src w elemencie <track> (umieszcza się go w elementach <video> i <audio>) prowadzi do ścieżki z informacjami o czasie. Atrybut kind określa, jakiego rodzaju dane są wskazywane w atrybucie src. Wartości atrybutu kind to: subtitles, captions, descriptions, chapters i metadata.

W elemencie multimedialnym można zapisać wiele ścieżek, jednak każda z nich musi być unikatową kombinacją typu i języka. Oto typy ścieżek:

`subtitles`
: Jest to wartość domyślna atrybutu `kind`. Określa, że ścieżka zawiera tłumaczenie dialogów, które domyślnie jest wyświetlane razem z filmem lub nagraniem dźwiękowym. Ten typ jest najprzydatniejszy, gdy rozmowy są niesłyszalne lub prowadzone w obcym języku.

`captions`
: Określa ścieżkę z transkrypcją lub tłumaczeniem dialogów. Przypomina typ `subtitles`, ale obejmuje transkrypcję efektów dźwiękowych, uwagi na temat muzyki i inne informacje na temat dźwięku, które pozwalają w pełni zastąpić ścieżkę dźwiękową, jeśli jest niedostępna. Ten typ jest najprzydatniejszy, gdy dźwięk jest wyłączony lub gdy użytkownik ma wadę słuchu.

`descriptions`
: Ścieżki tego typu to opis obrazu z odtwarzanych multimediów. Ten typ jest przeznaczony dla syntezytatorów dźwięku, gdy obraz jest niedostępny. Przydaje się dla osób z wadami wzroku i innych użytkowników, którzy nie mogą zobaczyć filmu lub odczytać tekstu ścieżki.

`chapters`
: Określa ścieżkę z tytułami rozdziałów przeznaczoną do poruszania się po nagraniu.

`metadata`
: Określa ścieżkę przeznaczoną dla skryptów (niewyświetlaną użytkownikom).

Plik ze ścieżką podaje się w atrybucie `src`. Atrybut `srclang` określa język tekstu danych z elementu `<track>`. Atrybut `label` to czytelny dla użytkownika tytuł elementu `<track>`. Przeglądarka używa tego tytułu do wyświetlania w interfejsie użytkownika ścieżek z napisami, transkrypcją i opisem obrazu.

Atrybut `default` informuje, że przeglądarka ma domyślnie zastosować dany element `<track>`, jeśli preferencje nie określają, iż bardziej odpowiedni jest inny element tego rodzaju. Jako domyślny można ustawić tylko jeden element `<track>`.

Elementy `<audio>` i `<video>` z HTML5 umożliwiają powiązanie napisów z osadzonymi multimediami. Te elementy są częścią modelu DOM dla HTML5, co pozwala dodawać do nich style CSS i zapewnia rozbudowany interfejs API zapewniający programistom kontrolę nad odtwarzaniem filmów. Dostępne są tu liczne nowe metody i właściwości JavaScriptu (np.: `play()`, `pause()`, `muted` i `ended`).

Elementy <video> i <audio> a JavaScript

Jeśli zamierzasz używać JavaScriptu do kontrolowania elementów `<audio>` i `<video>`, powinieneś wykrywać dostępne funkcje. W ten sposób upewnisz się, że potrzebne elementy są obsługiwane, oraz unikniesz błędów JavaScriptu.

```
if (createElement('audio').canPlayType) { /* Element audio jest obsługiwany */}
```

Możesz dodać wbudowane kontrolki lub utworzyć własne. Elementy `<audio>` i `<video>` obsługują metody `play()` i `pause()`. Aby utworzyć własne kontrolki, możesz dodać odpowiadający im kod w HTML-u i użyć JavaScriptu do uruchamiania i wstrzymywania nagrań. Potrzebny kod może wyglądać tak:

```
<div id="controls" style="display: none">
  <button id="playButton">Start</button>
  <button id="pauseButton">Pauza</button>
</div>
<script>
  if (document.createElement('audio').canPlayType) {
    if (document.createElement('audio').canPlayType('audio/mp3') ||
      (document.createElement('audio').canPlayType('audio/ogg')) {
      // Element <audio> z HTML5 i podany typ audio są obsługiwane
      document.getElementById('player').style.display = 'block';
    } else {
      ... Tu możesz umieścić nagrania we Flashu lub inne ...
  }
</script>
```

W celu utworzenia własnych kontrolek możesz zastosować następujący kod:

```
var videoClip = document.querySelector('#clip');
var playButton = document.querySelector('#playButton');
var pauseButton = document.querySelector('#pauseButton');

playButton.addEventListener('touchEnd', function() {
  playVideo();
});
pauseButton.addEventListener('touchEnd', function() {
  pauseVideo();
});

function playVideo() {
  // Odtwarzanie filmu
  videoClip.play();
  // Aktualizowanie kontrolek
  playButton.disabled = true;
  pauseButton.disabled = false;
}

function pauseVideo() {
  // Wstrzymywanie filmu
  videoClip.pause();
  // Aktualizowanie kontrolek
  playButton.disabled = false;
  pauseButton.disabled = true;
}

function MuteUnMute() {
  // Zmiana wartości przycisku
  document.getElementById('mute').value =
    videoClip.muted ? 'Wyłącz dźwięk' : 'Włącz dźwięk';
  // Zmiana stanu filmu
  videoClip.muted = videoClip.muted ? false : true;
}
```

CubeeDoo

W omawianej tu przykładowej grze użyliśmy kilku dźwięków. Oprócz odtwarzania w tle opcjonalnej irytującej muzyki gra generuje dźwięki informujące o powodzeniu lub porażce (np. po przejściu na następny poziom, dopasowaniu kart, nieudanym dopasowaniu itd.).

Muzykę odtwarzaną w tle należy umieścić w elemencie `<audio>`, ponieważ jest kontrolowana przez użytkownika. Dźwięki informacyjne zależą od poczynań użytkownika i powodzenia lub porażki, dlatego są dynamicznie generowane za pomocą JavaScriptu.

Oto dwie techniki, które można stosować do dodawania dźwięku. Dźwięk można dołączyć bezpośrednio w HTML-u:

```
<audio id="nonmatchsound" preload src="notmatch.mp3"></audio>
<audio id="matchsound" preload src="match.mp3"></audio>
```

Program wstępnie wczytuje nagranie, ale nie odtwarza plików dźwiękowych automatycznie ani nie odgrywa ich w pętli. Zamiast tego kod w JavaScripcie uruchamia odtwarzanie dźwięków informujących o dopasowanej lub niedopasowanej karcie:

```
playSound: function(matched) {
  if (qbdoo.mute) {
    return false;
  }
  if (matched) {
    qbdoo.matchfound.play();
  } else {
    qbdoo.failedmatch.play();
  }
},
```

Dźwięku nie trzeba dołączać w HTML-u. Zamiast tego można dodać go do modelu DOM za pomocą JavaScriptu bez dołączania plików dźwiękowych na stronie:

```
playSound: function(matched) {
  // Jeśli dźwięk jest wyłączony, należy przejść dalej
  if (qbdoo.mute) {
    return false;
  }
  // Jeśli węzeł audio nie istnieje, należy go utworzyć
  if (!qbdoo.audio) {
    qbdoo.audio = document.createElement('audio')
  }
  if (matched) {
    qbdoo.audio.src = qbdoo.matchedSound;
  }
  else {
    qbdoo.audio.src = qbdoo.failedMatchSound;
  }
  qbdoo.audio.play();
},
```

W grze dodaliśmy dźwięk tylko w celu zademonstrowania posługiwania się elementem `<audio>`. Nigdy nie należy automatycznie odtwarzać muzyki — irytuje to użytkowników. Zauważ, że gra zawiera przycisk wyciszania. Jeśli udostępniasz dźwięk i domyślnie jest on włączony, to gdy użytkownik zdecyduje się go wyłączyć, zapamiętaj to ustawienie w pamięci lokalnej (pamięć lokalną omawiam w rozdziale 6.).

Określanie stylu elementu <video>

Element `<video>` jest elementem HTML-a. Można więc określić jego styl (podobnie jak wszystkich elementów). Możesz wykorzystać style CSS do zdefiniowania wysokości i szerokości obrazu. Możesz też ukryć fragment nagrania, dodać zaokrąglone narożniki, a nawet powielić obraz. Za pomocą elementu `<canvas>` można pobrać piksele i zmienić ich kolor (ten sam efekt można osiągnąć za pomocą filtrów CSS).

Dynamiczne dopasowywanie wielkości obrazu

Co ważniejsze, można zmieniać wymiary obrazu na podstawie wielkości urządzenia i proporcji ekranu. Thierry Koblentz zaproponował skuteczną metodę (*http://alistapart.com/article/creating-intrinsic-ratios-for-video*) umożliwiającą przeglądarkom określanie wymiarów obrazu na podstawie szerokości bloku zawierającego film (lub szerokości strony) i pierwotnej wielkości. Zmiana szerokości, np. w wyniku zmiany orientacji urządzenia, prowadzi do ponownego wyznaczenia wysokości. To umożliwia zmianę wielkości obrazu i pozwala skalować filmy w taki sam sposób jak rysunki.

Aby utworzyć film pozwalający na zmianę wymiarów, należy przygotować pole o zmiennej wielkości i odpowiednich proporcjach (4:3, 16:9 itd.). Następnie film trzeba rozciągnąć, aby dopasować go do wymiarów pola. Można przy tym wykorzystać marginesy wewnętrzne (właściwości z rodziny `padding`), procentowe wartości wymiarów i bezwzględne określanie pozycji. Marginesy wewnętrzne są zwykle ustawiane na 56,25% lub 75% wielkości okna (w zależności od proporcji). W tym modelu można bezwzględnie określić pozycję elementu `<video>` w taki sposób, aby zajmował całą wysokość i szerokość obszaru wyznaczanego przez wewnętrzne marginesy.

Jeśli chcesz dodać filmy o zmiennej wielkości do witryny z dynamicznie dopasowywanym rozmiarem, możesz zastosować następujący kod:

```
.wrapper {
  position: relative;
  height: 0;
  width: 100%;
  padding-bottom: 56.25%;
  / * Lub */
  padding-bottom: 75%;
}
video {
  position: absolute;
  width: 100%;
  height: 100%;
  left: 0;
  top: 0;
}
```

Co warto wiedzieć o stosowaniu elementu <video>?

W odróżnieniu od Flasha, który jest obsługiwany za pomocą wtyczki kontrolowanej przez jedną firmę, dlatego wszędzie działa w ten sam sposób, element `<video>` funkcjonuje nieco inaczej w poszczególnych przeglądarkach i systemach operacyjnych. Na iPhone'ach oraz w systemach Android i Windows Phone 8 filmy zawsze są odtwarzane w trybie pełnoekranowym. Na iPadach dostępna jest kontrolka trybu pełnoekranowego, a wielkość obrazu można zmieniać także za pomocą gestu „szczypania". W systemach iOS i Windows do odtwarzania filmów używany jest procesor graficzny, natomiast w Androidzie do wersji 4 filmy były wyświetlane przy użyciu głównego procesora.

- Upewnij się, że używany serwer obsługuje potrzebne typy MIME wideo. Brak takiej obsługi może prowadzić do problemów w Firefoksie. Dodaj wiersz `AddType video/ogg.ogv` i podobne wpisy do pliku *.htaccess*, jeśli ich w nim nie ma.
- Urządzenia iPhone i iPad nie odtwarzają automatycznie filmów, nawet jeśli dodasz atrybut `autoplay`.

- Wygląd kontrolek zależy od natywnych kontrolek przeglądarki. Jak wcześniej wspomniałam, wygląd i styl kontrolek można zmienić za pomocą JavaScriptu. Jeśli chcesz dodać skórkę dla kontrolek, zajrzyj na stronę *http://www.videojs.com/*.
- Jeżeli chcesz zacząć udostępniać własne filmy, istnieje otwarty, wieloplatformowy transkoder wideo z obsługą wielowątkowości. Jego nazwa to Handbrake (*http://handbrake.fr/*) i jest dostępny na licencji GPL w systemach Mac OS X, Linux i Windows.

Pamiętaj, że odtwarzanie nagrań audio i wideo szybko wyczerpuje baterie. Choć multimedia tego typu są obsługiwane we wszystkich smartfonach, większe możliwości wymagają więcej odpowiedzialności. Powinieneś zadbać o to, aby tworzone przez Ciebie aplikacje nie zużywały baterii w urządzeniach użytkowników. Zachowaj ostrożność przy stosowaniu funkcji wyczerpujących baterie.

Skorowidz

A

B

C

D

E

F

G

H

I

J

K

L

M

N

O

P

R

S

Ś

T

U

V

W

X

Y

Z

Ż

O'REILLY®

HTML5. Strony mobilne

HTML5 to prawdziwa rewolucja w świecie stron WWW i aplikacji internetowych. Jego potencjał oraz możliwości zaskoczyły niejedną osobę. Wykorzystywany jest do tworzenia atrakcyjnych, intuicyjnych i dynamicznych stron WWW. To nie wszystko! Dzięki HTML5 i CSS3 możesz także stworzyć aplikację działającą na urządzeniach z systemami Android lub iOS. Chcesz zobaczyć, jak to zrobić?

Jeśli tak, to trafiłeś na świetną książkę, która wprowadzi Cię w świat nowoczesnych stron i aplikacji. W trakcie lektury będziesz poznawać składnię oraz możliwości HTML, przygotujesz środowisko pracy oraz zdobędziesz potrzebne narzędzia. Kolejne rozdziały to obowiązkowa dawka informacji na temat tworzenia formularzy oraz korzystania z elementów svg, canvas, audio i video. HTML5 dostarcza wielu nowych narzędzi, pozwalających m.in. sprawdzić stan połączenia z siecią, przechowywać dane na komputerze użytkownika lub uzyskać informację o jego lokalizacji. Dzięki tej książce opanujesz je w mig. W tym podręczniku znajdziesz również obszerny opis kaskadowych arkuszy stylów w wersji 3. Pozwolą Ci one błyskawicznie wprowadzać atrakcyjne dla oka efekty. Książka ta jest genialnym źródłem informacji dla wszystkich pasjonatów tworzenia stron i aplikacji internetowych.

Dzięki tej książce:

- poznasz składnię HTML5 i CSS3
- zaznajomisz się z możliwościami API
- zbudujesz zaawansowaną stronę lub mobilną aplikację internetową

Sprawdź, jak stworzyć aplikację internetową działającą na platformie Android lub iOS!

helion.pl
księgarnia internetowa

Nr katalogowy: 19756

Księgarnia internetowa:
http://helion.pl

Zamówienia telefoniczne:
0 801 339900
0 601 339900

Helion

Sprawdź najnowsze promocje:
http://helion.pl/promocje
Książki najchętniej czytane:
http://helion.pl/bestsellery
Zamów informacje o nowościach:
http://helion.pl/nowosci

Helion SA
ul. Kościuszki 1c, 44-100 Gliwice
tel.: 32 230 98 63
e-mail: helion@helion.pl
http://helion.pl

Informatyka w najlepszym wydaniu

sięgnij po WIĘCEJ
KOD KORZYŚCI

ISBN 978-83-246-8912-5

Cena 69,00 zł

9 788324 689125