

TWÓRZ ZAAWANSOWANE APLIKACJE!

Apress®

HTML5

Zaawansowane programowanie



Peter Lubbers, Brian Albers, Frank Salim

Helion 

Tytuł oryginału: Pro HTML5 Programming, Second Edition

Tłumaczenie: Jakub Hubisz

ISBN: 978-83-246-4809-2

Original edition copyright © 2011 by Peter Lubbers, Brian Albers, and Frank Salim.
All rights reserved.

Polish edition copyright © 2013 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.

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

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

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
<ftp://ftp.helion.pl/przyklady/htm5zp.zip>

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
<http://helion.pl/user/opinie/htm5zp>
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

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

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

Spis treści

Przedmowa	13
O autorach	14
O korektorze merytorycznym	15
Podziękowania	16
Wprowadzenie	17
Dla kogo jest ta książka	17
Przegląd zawartości książki	18
Przykładowe kody i strona książki	18
Rozdział 1. Przegląd HTML5	19
Do tej pory — historia HTML5	19
Mit roku 2022 i dlaczego nie ma on znaczenia	20
Kto tworzy HTML5?	21
Nowa wizja	21
Kompatybilność i wytyczanie ścieżek	21
Użyteczność i priorytet użytkownika	21
Uproszczenie współpracy	22
Uniwersalny dostęp	23
Paradygmat braku wtyczek	23
Co przyszło, a co odeszło?	24
Co nowego w HTML5?	25
Nowy DOCTYPE i deklaracja zestawu znaków	25
Nowe i wycofane elementy	26
Znaczniki semantyczne	27
Uprozczone wybieranie przy wykorzystaniu API selektorów	32

Logowanie i debugowanie JavaScript	34
window.JSON	35
DOM poziom 3	36
Wzrost wydajności	36
Podsumowanie	37
Rozdział 2. Wykorzystanie Canvas API	39
Wprowadzenie do canvas w HTML5	39
Historia	39
Czym jest canvas	40
Koordynaty	40
Kiedy nie używać elementu canvas	40
Zawartość zastępcza	41
CSS i canvas	42
Wsparcie przeglądarek dla elementu canvas	42
Wykorzystanie Canvas API	42
Sprawdzenie wsparcia w przeglądarce	42
Dodanie elementu na stronę	43
Wykonywanie transformacji na rysunkach	45
Praca ze ścieżkami	47
Praca ze stylami obramowania	49
Praca ze stylami wypełnienia	49
Wypełnienie prostokątnego obszaru	50
Rysowanie krzywych	51
Wstawianie obrazów na elemencie canvas	53
Wykorzystanie gradientów	53
Wykorzystanie deseni tła	55
Zmiana rozmiaru obiektów na elemencie canvas	56
Wykorzystanie operacji transformacji	58
Renderowanie tekstu na elemencie canvas	60
Tworzenie cieni	61
Praca z pikselami	62
Implementacja zasad bezpieczeństwa	64
Budowanie aplikacji za pomocą elementu canvas HTML5	65
Porady praktyczne: szklany panel na całą stronę	68
Porady praktyczne: synchronizowanie animacji	68
Podsumowanie	70
Rozdział 3. Praca ze skalowalną grafiką wektorową	73
Przegląd SVG	73
Historia	73
Zrozumieć SVG	74
Grafika skalowalna	75
Tworzenie grafiki 2D za pomocą SVG	76

Umieszczenie SVG na stronie	76
Proste kształty	77
Transformacje elementów SVG	78
Powtórne wykorzystanie zawartości	78
Desenie i gradienty	79
Ścieżki SVG	80
Tekst w SVG	81
Budowanie sceny	81
Budowanie interaktywnej aplikacji z wykorzystaniem SVG	83
Dodawanie drzew	83
Dodanie funkcji updateTrees	83
Dodanie funkcji removeTree	84
Dodanie stylu CSS	84
Pełen kod	84
Podsumowanie	87
Rozdział 4. Praca z elementami audiowizualnymi	89
Przegląd informacji o elementach audio i video	89
Kontenery wideo	89
Kodeki audio i video	90
Restrykcje audio i video	91
Wsparcie przeglądarek dla dźwięku i wideo	91
Wykorzystanie API dla dźwięku i wideo	92
Sprawdzanie wsparcia przeglądarek	92
Dostępność	93
Zrozumieć elementy audio i video	94
Praca z dźwiękiem	98
Praca z wideo	99
Praktyczne dodatki	106
Podsumowanie	108
Rozdział 5. Wykorzystanie geolokalizacji	109
Informacje o lokalizacji	109
Szerokość i długość geograficzna	110
Skąd pochodzą informacje	110
Dane wynikające z adresu IP	111
Dane GPS	111
Dane pobierane w oparciu o Wi-Fi	111
Dane pobierane z telefonu komórkowego	112
Dane podawane przez użytkownika	112
Wsparcie geolokalizacji przez przeglądarki	112
Prywatność	113
Uruchomienie mechanizmu ochrony prywatności	113
Obchodzenie się z danymi geolokalizacyjnymi	114

Wykorzystanie Geolocation API	115
Sprawdzenie wsparcia w przeglądarce	115
Prośby o podanie lokalizacji	115
Budowa aplikacji wykorzystującej geolokalizację	120
Stworzenie kodu HTML	121
Przetwarzanie danych geolokalizacyjnych	122
Kompletny kod	124
Praktyczne dodatki	127
Jaki jest mój status?	127
Pokaż mnie na mapie	128
Podsumowanie	129
Rozdział 6. Wykorzystanie API komunikacji	131
Cross Document Messaging	131
Zrozumieć bezpieczeństwo pochodzenia	133
Wsparcie w przeglądarce dla API komunikacji	134
Wykorzystanie API postMessage	134
Budowa aplikacji za pomocą API postMessage	135
XMLHttpRequest Level 2	140
Przesyłanie żądań XMLHttpRequest pomiędzy różnymi domenami	140
Zdarzenia postępu	142
Wsparcie w przeglądarkach dla XMLHttpRequest Level 2	142
Wykorzystanie XMLHttpRequest API	142
Budowanie aplikacji wykorzystującej XMLHttpRequest	144
Praktyczne dodatki	147
Struktury danych	147
Framebusting	148
Podsumowanie	148
Rozdział 7. Wykorzystanie mechanizmu WebSocket	149
Informacje ogólne o mechanizmie WebSocket	149
Komunikacja w czasie rzeczywistym i HTTP	149
Zrozumieć mechanizm WebSocket	151
Prosty serwer WebSocket	156
Ramki WebSocket	157
Wykorzystanie WebSocket API	163
Sprawdzenie wsparcia w przeglądarce	163
Podstawowe wykorzystanie API	164
Tworzenie aplikacji WebSocket	168
Utworzenie pliku HTML	169
Kod WebSocket	170
Kod geolokalizacji	171
Połączenie komponentów	171
Kompletny kod	173
Podsumowanie	175

Rozdział 8. Wykorzystanie Forms API	177
Przegląd formularzy HTML5	177
Formularze HTML kontra XForms	177
Formularze funkcjonalne	178
Wsparcie przeglądarek dla formularzy HTML5	178
Katalog elementów input	178
Wykorzystanie API formularzy	182
Nowe atrybuty i funkcje	182
Sprawdzanie formularzy za pomocą walidacji	185
Pola i funkcje walidujące	188
Informacja o poprawności	189
Tworzenie aplikacji z formularzami HTML5	190
Praktyczne dodatki	193
Podsumowanie	195
Rozdział 9. Praca z techniką „przeciągnij i upuść”	197
Internetowe „przeciągnij i upuść” — historia	197
Przegląd elementów „przeciągnij i upuść” w HTML5	198
Obraz całości	198
Zdarzenia, o których należy pamiętać	200
Udział w przeciąganiu	202
Przenoszenie i kontrola	203
Budowa aplikacji „przeciągnij i upuść”	203
Dotrzeć do strefy rzutu	211
Obsługa „przeciągnij i upuść” dla plików	211
Praktyczne dodatki	215
Modyfikacja wyglądu przeciąganego obiektu	215
Podsumowanie	215
Rozdział 10. Wykorzystanie wątków roboczych	217
Wsparcie w przeglądarkach	218
Wykorzystanie Web Workers API	218
Sprawdzenie wsparcia w przeglądarce	218
Tworzenie wątków roboczych	219
Ładowanie i wykonywanie dodatkowego kodu JavaScript	219
Komunikacja z wątkami roboczymi	219
Tworzenie strony głównej	220
Obsługa błędów	221
Zatrzymywanie wątków roboczych	221
Wykorzystanie wątków wewnątrz innych wątków	221
Wykorzystanie liczników czasu	223
Kod przykładu	223
Budowa aplikacji wykorzystującej wątki robocze	224
Tworzenie pliku pomocniczego blur.js	225
Tworzenie strony aplikacji blur.html	226

Tworzenie pliku blurWorker.js dla wątku	227
Komunikacja z wątkami	228
Aplikacja w akcji	229
Kod przykładowy	229
Podsumowanie	234
Rozdział 11. Wykorzystanie API Web Storage	235
Informacje ogólne o Web Storage	235
Wsparcie w przeglądarkach	236
Wykorzystanie API Web Storage	236
Sprawdzanie wsparcia w przeglądarce	236
Zapisywanie i pobieranie danych	237
Zatykanie wycieków danych	238
Magazyn lokalny kontra magazyn sesji	240
Inne atrybuty i funkcje API Web Storage	240
Aktualizacje magazynu	242
Przeglądanie magazynu	243
Budowa aplikacji przy wykorzystaniu magazynu	245
Przyszłość baz danych magazynu przeglądarki	254
Baza Web SQL	254
API Indexed Database	256
Praktyczne dodatki	258
Przechowywanie obiektów JSON	258
Współdzielenie danych między oknami	258
Podsumowanie	260
Rozdział 12. Tworzenie aplikacji lokalnych	261
Informacje ogólne o aplikacjach lokalnych HTML5	261
Wsparcie w przeglądarkach	263
Wykorzystanie bufora aplikacji HTML5	263
Sprawdzanie wsparcia w przeglądarce	263
Prosta aplikacja lokalna	263
Praca w trybie offline	264
Pliki manifestu	264
API aplikacji lokalnych	266
Pamięć podręczna w akcji	267
Budowa aplikacji przy wykorzystaniu API aplikacji lokalnych	268
Tworzenie pliku manifestu dla aplikacji	270
Tworzenie struktury HTML i arkuszy stylów CSS	270
Tworzenie pliku JavaScript dla trybu offline	271
Sprawdzanie wsparcia w przeglądarce	273
Dodanie obsługi dla przycisku	273
Dodanie kodu geolokalizacyjnego	274
Dodanie kodu obsługującego magazyn lokalny	274
Dodanie obsługi dla zdarzeń trybu offline	275
Podsumowanie	275

Rozdział 13. Przyszłość HTML5	277
Wsparcie w przeglądarkach dla HTML5	277
HTML ewoluje	277
WebGL	278
Urządzenia	280
API dźwięku	280
Zdarzenia urządzeń z ekranem dotykowym	280
Sieci peer-to-peer	282
Kierunek rozwoju	283
Podsumowanie	283
Skorowidz	285

ROZDZIAŁ 5



Wykorzystanie geolokalizacji

Załóżmy, że chcesz stworzyć aplikację oferującą zniżki i promocje na buty biegowe w sklepach znajdujących się w niewielkiej odległości od użytkownika. Korzystając z Geolocation API, możesz poprosić użytkownika, aby podał swoją aktualną pozycję. Jeżeli wyrazi zgodę, możesz przekazać mu instrukcje na temat tego, w jaki sposób dostać się do pobliskiego sklepu po nową parę butów w obniżonej cenie.

Innym przykładem wykorzystania geolokalizacji może być aplikacja mierząca odległość, jaką przebiegłeś (lub przeszedłeś). Wyobraź sobie aplikację działającą w przeglądarce Twojego telefonu komórkowego, którą włączasz, kiedy zaczynasz bieg. Aplikacja śledzi trasę, jaką przebyłeś. Koordynaty trasy mogą nawet zostać nałożone na mapę, być może wraz z profilem wysokościowym. Jeżeli ścigasz się z innymi uczestnikami biegu, aplikacja może nawet pokazywać pozycje Twoich przeciwników.

Pomysły na skorzystanie z możliwości geolokalizacji to na przykład aplikacja nawigująca przez kolejne etapy trasy, aplikacja społecznościowa pokazująca, gdzie dokładnie znajdują się Twoi znajomi, tak abyś mógł wybrać dogodną dla wszystkich kawiarnię, a także wiele innych, bardziej niecodziennych.

W tym rozdziale pokażemy, co można osiągnąć dzięki Geolocation API — ekscytującemu interfejsowi pozwalającemu użytkownikom na dzielenie się swoimi lokalizacjami w aplikacjach webowych, tak aby możliwe było wykorzystywanie uzależnionej od lokalizacji usługi. Najpierw zapoznamy się ze źródłem informacji geograficznych — szerokości, długości i innych atrybutów — i dowiemy się, skąd mogą one pochodzić (GPS, Wi-Fi, triangulacja itd.). Następnie omówimy problemy związane z prywatnością podczas korzystania z takich usług oraz opiszemy, jak przeglądarki obchodzą się z danymi geolokalizacyjnymi.

Potem zagłębimy się w praktyczną dyskusję na temat dwóch różnych funkcji (metod) żądania pozycji udostępnianych przez Geolocation API: jednorazowego żądania pozycji oraz ciągłej aktualizacji. Powiemy, kiedy i w jaki sposób z nich korzystać. W końcu pokażemy, jak zbudować praktyczną aplikację przy wykorzystaniu Geolocation API. Zakończymy opisem kilku ciekawych przypadków użycia oraz praktycznymi wskazówkami.

Informacje o lokalizacji

Interfejs Geolocation jest przejrzysty. Żądasz podania pozycji i jeżeli użytkownik wyrazi zgodę, przeglądarka zwraca informacje o lokalizacji. Pozycja podawana jest do wspierającej geolokalizację przeglądarki przez urządzenie (np. laptop lub telefon komórkowy), na którym została ona uruchomiona. Informacje obejmują zestaw koordynatów geograficznych (długość i szerokość geograficzna) wraz z towarzyszącymi metadanymi. Mając dostęp do tych informacji, możesz stworzyć aplikację zależną od lokalizacji użytkownika.

Szerokość i długość geograficzna

Informacje o lokalizacji składają się przede wszystkim z długości i szerokości geograficznej — podobnych do tych pokazanych w kolejnym przykładzie, które składają się na koordynaty pięknego Tahoe City, położonego na brzegu jeziora Tahoe, najpiękniejszego jeziora górskiego Ameryki:

Szerokość: 39.17222, Długość: -120.13778

W tym przykładzie szerokość geograficzna (wartość liczbową reprezentująca odległość na północ lub południe od równika) wynosi 39.17222, natomiast długość (wartość liczbową reprezentująca odległość na wschód lub zachód od południka) ma wartość -120.13778.

Wartości te mogą być wyrażone w różny sposób:

- w formacie liczbowym — np. 39.17222,
- jako „stopnie-minuty-sekundy” (DMS — *Degree Minute Second*) — np. 39° 10' 20",

■ **Uwaga.** W Geolocation API koordynaty podawane są zawsze w formacie liczbowym.

Oprócz długości i szerokości geograficznej udostępniana jest także *dokładność* koordynatów. Zależnie od urządzenia, na jakim uruchamiana jest przeglądarka, dostępne mogą być także dodatkowe metadane — mogą one zawierać *wysokość nad poziomem morza* wraz z *dokładnością*, *kąt odchylenia* i *prędkość*. Jeżeli dodatkowe informacje nie będą dostępne, przyjmą wartość null.

Skąd pochodzą informacje

Geolocation API nie definiuje, z jakiej technologii musi skorzystać urządzenie w celu określenia lokalizacji użytkownika — udostępnia jedynie możliwość pobrania tych informacji oraz poziomu dokładności, z jaką urządzenie zostało zlokalizowane. Nie ma żadnej gwarancji, że współrzędne zwrócone przez urządzenie są poprawne.

Lokalizacja, lokalizacja

Peter mówi: Oto ciekawy przykład. W domu korzystam z sieci bezprzewodowej. Otwarłem przykładową aplikację pokazaną w tym rozdziale w przeglądarce Firefox i dowiedziałem się, że jestem w Sacramento (ok. 120 km od mojego rzeczywistego położenia). Wynik jest błędny, ale nie zaskakujący — dostawca internetu, z którego usług korzystam, ma siedzibę w Sacramento.

Poprosiłem wtedy moich synów — Seana i Rocky'ego — aby otwarli stronę na iPhone (przy wykorzystaniu tego samego połączenia bezprzewodowego). Safari pokazało, że znajdują się w Marysville w Kalifornii — mieście znajdującym się 50 km od Sacramento.

Dane o lokalizacji mogą pochodzić z następujących źródeł:

- adresu IP,
- triangulacji:
 - systemu globalnego pozycjonowania (GPS — *Global Positioning System*),
 - Wi-Fi z adresami MAC z sieci RFID, Wi-Fi lub Bluetooth,
 - identyfikatorów GSM lub CDMA,
- danych podanych przez użytkownika.

Wiele z urzędzeń wykorzystuje kombinację różnych źródeł w celu zapewnienia większej dokładności. Każda z tych metod ma wady i zalety, które opiszemy w kolejnym podrozdziale.

Dane wynikające z adresu IP

Dane wynikające z adresu IP były dawniej jedyną możliwością uzyskania informacji o lokalizacji użytkownika, często jednak były niepoprawne. W tej metodzie automatycznie zwracany jest adres fizyczny rejestratora danego adresu IP. W związku z tym jeżeli adres IP uzyskujesz od dostawcy internetu, Twoja lokalizacja będzie często równoznaczna z jego fizycznym adresem, który może się znajdować wiele kilometrów od Ciebie. Tabela 5.1 przedstawia wady i zalety stosowania geolokalizacji opartej na adresie IP.

Tabela 5.1. Wady i zalety stosowania geolokalizacji opartej na adresie IP

Zalety	Wady
Dostępna wszędzie.	Niezbyt dokładna (często albo błędna, albo z dokładnością ograniczoną do miejscowości).
Przetwarzanie po stronie serwera.	Może być kosztowną operacją.

Wiele stron wyświetla reklamy w oparciu o lokalizację adresu IP. Możesz to zaobserwować podczas podróży do innego kraju, kiedy nagle zobaczysz reklamy lokalnych firm (zależnie od adresu IP regionu lub kraju, w którym się znalazłeś).

Dane GPS

Dopóki widać niebo, GPS może być źródłem bardzo dokładnych danych geolokalizacyjnych. Pozycja wyliczana jest na podstawie sygnału z wielu satelitów okrążających Ziemię. Sporo czasu może jednak zająć uzyskanie sygnału z odpowiedniej ich liczby, co nie jest dobre w przypadku aplikacji, które muszą rozpocząć działanie natychmiast.

Z powodu czasu potrzebnego na uzyskanie informacji można próbować wykonywać tę operację asynchronicznie. Aby pokazać użytkownikom, że pobierana jest ich lokalizacja, można wyświetlić pasek postępu. Tabela 5.2 pokazuje wady i zalety stosowania geolokalizacji opartej na GPS-ie.

Tabela 5.2. Wady i zalety stosowania geolokalizacji opartej na GPS-ie

Zalety	Wady
Bardzo dokładna.	Uzyskanie informacji o pozycji może zająć dużo czasu, co może doprowadzić do szybszego wyczerpania baterii urządzenia. Niezbyt dobrze działa wewnątrz budynków. Może wymagać dodatkowego sprzętu.

Dane pobierane w oparciu o Wi-Fi

Pobierane informacji w oparciu o sieć bezprzewodową działa na zasadzie triangulacji — sprawdzana jest odległość użytkownika od znanych punktów dostępowych (głównie w miastach). W przeciwieństwie do GPS-u dane uzyskiwane tą metodą są równie dokładne wewnątrz budynków. Tabela 5.3 przedstawia wady i zalety stosowania geolokalizacji opartej na Wi-Fi.

Tabela 5.3. Wady i zalety stosowania geolokalizacji opartej na Wi-Fi

Zalety	Wady
Dokładna. Działa wewnątrz budynków. Namierzanie jest szybkie i tanie.	Niezbyt dobrze działa na terenach z małą liczbą punktów dostępu.

Dane pobierane z telefonu komórkowego

Te dane także są pobierane w oparciu o triangulację — w tym przypadku jednak mierzona jest odległość użytkownika od nadajników telekomunikacyjnych. Dzięki temu otrzymujemy wynik o umiarkowanej dokładności. Metoda ta często wykorzystywana jest w połączeniu z metodą opartą na GPS-ie lub Wi-Fi. Tabela 5.4 przedstawia wady i zalety stosowania geolokalizacji działającej na podstawie telefonu komórkowego.

Tabela 5.4. Wady i zalety stosowania geolokalizacji działającej na podstawie telefonu komórkowego

Zalety	Wady
Dość dokładna. Działa wewnątrz budynków. Namierzanie jest szybkie i tanie.	Wymaga urządzenia z dostępem do telefonu lub modemu komórkowego. Niezbyt dobrze działa na terenach z małą liczbą nadajników telekomunikacyjnych.

Dane podawane przez użytkownika

Zamiast próbować uzyskać lokalizację użytkownika programistycznie, możesz pozwolić mu na jej samodzielne wprowadzenie. Aplikacja może pozwalać na wprowadzenie adresu, kodu pocztowego lub innych informacji; te dane możesz wykorzystać do zaserwowania użytkownikowi usług zależnych od lokalizacji. Tabela 5.5 przedstawia wady i zalety danych podawanych przez użytkownika.

Tabela 5.5. Wady i zalety danych podawanych przez użytkownika

Zalety	Wady
Użytkownik może dysponować danymi dokładniejszymi niż dostępne programistycznie. Pozwala na wprowadzanie alternatywnych lokalizacji. Wprowadzenie danych przez użytkownika może być szybsze niż detekcja lokalizacji.	Dane mogą być bardzo niedokładne, zwłaszcza gdy lokalizacja ulega zmianie.

Wsparcie geolokalizacji przez przeglądarki

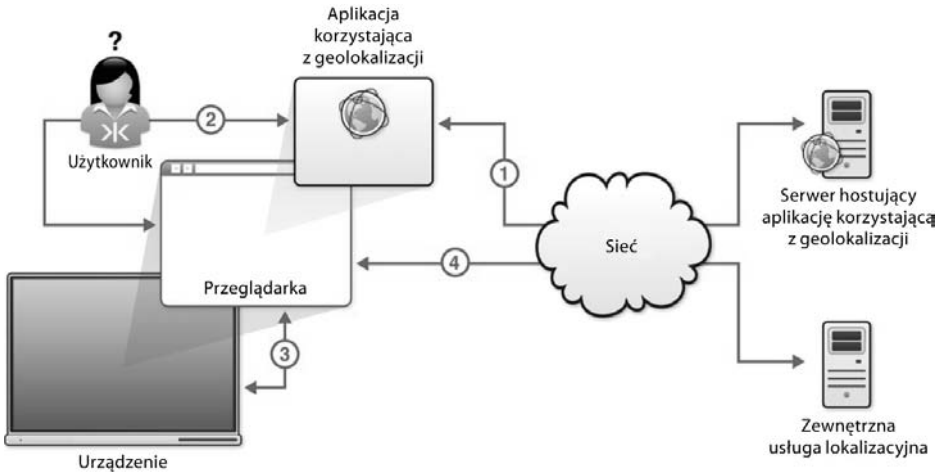
Interfejs Geolocation był jednym z pierwszych elementów HTML5, który został w pełni zaimplementowany, i wspierany jest obecnie przez wszystkie istotne przeglądarki. Jeżeli interesują Cię dokładniejsze informacje na temat wsparcia geolokalizacji, wejdź na stronę <http://caniuse.com/> i poszukaj hasła *Geolocation*.

Jeśli musisz zapewnić wsparcie dla starszych przeglądarek, przed skorzystaniem z API warto sprawdzić, czy interfejs Geolocation jest obsługiwany. W podrozdziale „Sprawdzenie wsparcia przeglądarki” pokażemy, w jaki sposób to zrobić.

Prywatność

Specyfika geolokalizacji wymaga, aby zapewniony został mechanizm ochrony prywatności użytkownika. Co więcej, informacje nie powinny być udostępniane, jeżeli użytkownik nie wyrazi na to zgody.

Jest to odpowiedź na problem „wielkiego brata”, często poruszany przez użytkowników aplikacji wykorzystujących geolokalizację. Jak jednak widać z niektórych przypadków użycia Geolocation API w aplikacjach, użytkownicy zachęceni są do dzielenia się swoimi lokalizacjami. Na przykład użytkownik może chętniej podzielić się danymi geolokalizacyjnymi, jeżeli w zamian może uzyskać informację o rabacie na buty biegowe, dostępnym w sklepie oddalonym parę przecznic od miejsca, w którym właśnie pije kawę. Przyjrzyjmy się bliżej architekturze prywatności pokazanej na rysunku 5.1.



Rysunek 5.1. Architektura prywatności przeglądarki i urządzenia

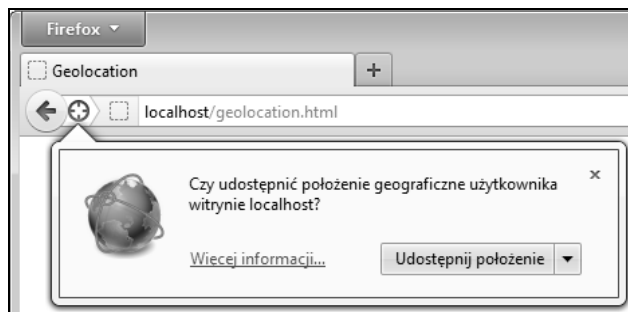
Na diagramie pokazano następujące kroki:

1. Użytkownik uruchamia w przeglądarce aplikację korzystającą z geolokalizacji.
2. Strona aplikacji ładuje się i przesyła prośbę o koordynaty poprzez wywołanie funkcji z Geolocation API. Przeglądarka przechwytyje zapytania i pyta użytkownika o pozwolenie. Załóżmy, że użytkownik udziela pozwolenia.
3. Przeglądarka pobiera koordynaty z urządzenia, na którym została uruchomiona. W tym przypadku jest to kombinacja danych pochodzących z adresu IP, Wi-Fi i GPS-u. Jest to wewnętrzna funkcjonalność przeglądarki.
4. Przeglądarka przesyła dane do zaufanej, zewnętrznej usługi lokalizacyjnej — zwracane koordynaty mogą być przekazane do aplikacji.

■ **Ważne.** Aplikacja *nie* ma bezpośredniego dostępu do urządzenia; może jedynie wysłać do przeglądarki prośbę o podanie informacji.

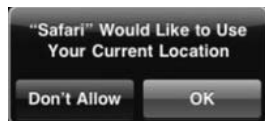
Uruchomienie mechanizmu ochrony prywatności

Kiedy uruchamiasz stronę korzystającą z Geolocation API, powinien zostać uruchomiony mechanizm ochrony prywatności. Rysunek 5.2 pokazuje, jak wygląda to w przeglądarce Firefox.



Rysunek 5.2. Jeżeli w przeglądarce Firefox wykorzystywany jest interfejs Geolocation, pojawia się panel informacyjny

Mechanizm uruchamiany jest w momencie uruchomienia kodu Geolocation. Dodanie kodu, który nie jest wywoływany (np. w zdarzeniu onload), nie spowoduje reakcji. Jeżeli jednak kod Geolocation zostanie uruchomiony, na przykład wywołaniem `navigator.geolocation.getCurrentPosition` (w dalszej części metoda zostanie omówiona dokładnie), użytkownik zostanie poproszony o podzielenie się informacjami o lokalizacji. Rysunek 5.3 pokazuje, co stanie się w przeglądarce Safari działającej na iPhone'ie.



Rysunek 5.3. Jeżeli w przeglądarce Safari wykorzystywany jest interfejs Geolocation, pojawia się okno dialogowe z powiadomieniem

W niektórych przypadkach (np. w Firefoksie) oprócz mechanizmu pytającego użytkownika o zezwolenie na pobranie lokalizacji możliwe jest także zapamiętanie decyzji użytkownika i uwzględnienie jej podczas kolejnych wizyt na stronie. Działa to podobnie do zapamiętywania haseł w przeglądarce.

-
- **Uwaga.** Jeżeli w Firefoksie ustawiliś, aby dla strony zawsze było udzielane pozwolenie, a później zmienisz zdanie, możesz łatwo odwołać pozwolenie. Musisz wejść na stronę, wybrać *Informacje o stronie* z menu *Narzędzia*, a następnie zmienić ustawienie dla opcji *Udostępnij położenie* na zakładce *Uprawnienia*.
-

Obchodzenie się z danymi geolokalizacyjnymi

Są to dane wrażliwe, więc jeżeli je otrzymasz, musisz być ostrożny podczas ich obsługi, przechowywania i powtórnej transmisji. Jeżeli użytkownik nie wyrazi zgody na przechowywanie danych, powinieneś się ich pozbywać zawsze po zakończeniu pracy z nimi.

Jeżeli retransmitujesz pozyskane dane, pamiętaj, że powinieneś je najpierw zaszyfrować. Twoja aplikacja powinna też zawsze w jasny sposób informować użytkownika:

- że zbierasz dane geolokalizacyjne,
- dlaczego je zbierasz,
- jak długo są przechowywane,
- w jaki sposób są zabezpieczone,
- z kim i jak dzielisz się tymi danymi,
- w jaki sposób użytkownik może sprawdzić i zaktualizować swoje dane.

Wykorzystanie Geolocation API

W tym podrozdziale bardziej szczegółowo omówimy wykorzystanie Geolocation API. Stworzymy prostą stronę — *geolocation.html*. Pamiętaj, że cały kod możesz pobrać pod adresem: <ftp://ftp.helion.pl/przyklady/htm5zp.zip>.

Sprawdzenie wsparcia w przeglądarce

Zanim wywołasz funkcje z Geolocation API, dobrze jest sprawdzić, czy to, co zamierzasz zrobić, wspierane jest przez przeglądarkę. Dzięki temu, jeżeli przeglądarka nie wspiera odpowiednich funkcji, możesz przekazać użytkownikowi odpowiednią informację lub poprosić go o zainstalowanie zewnętrznej wtyczki (np. Gears), która poprawi funkcjonalność przeglądarki. Na listingu 5.1 przedstawiony został jeden ze sposobów sprawdzenia wsparcia w przeglądarce.

Listing 5.1. Sprawdzenie wsparcia w przeglądarce

```
<script>
function loadDemo() {
    if(navigator.geolocation) {
        document.getElementById("support").innerHTML = "Geolokalizacja jest wspierana.";
    } else {
        document.getElementById("support").innerHTML = "Twoja przeglądarka nie wspiera
        geolokalizacji.";
    }
}
</script>
```

Funkcja `loadDemo` sprawdza wsparcie dla geolokalizacji w przeglądarce — może zostać wywołana po załadowaniu strony. Jeżeli będzie to możliwe, wywołanie `navigator.geolocation` (można także wykorzystać `Modernizr`) zwróci obiekt `Geolocation` — w przeciwnym razie wykonany zostanie wariant braku wsparcia. Skrypt aktualizuje zdefiniowany wcześniej element `support` tekstem informującym o możliwości wykorzystania geolokalizacji w przeglądarce.

Prośby o podanie lokalizacji

Dostępne są dwa typy prośby o podanie lokalizacji:

- jednorazowa prośba,
- ciągła aktualizacja.

Jednorazowa prośba

W wielu przypadkach akceptowalne będzie jednorazowe pobranie lokalizacji użytkownika. Na przykład jeżeli ktoś szuka najbliższego kina pokazującego najnowszy film w ciągu najbliższej godziny, może zostać wykorzystana najprostsza forma geolokalizacji, pokazana na listingu 5.2.

Listing 5.2. Jednorazowe pobranie lokalizacji

```
void getCurrentPosition(in PositionCallback successCallback,
    in optional PositionErrorCallback errorCallback,
    in optional PositionOptions options);
```

Przyjrzyjmy się dokładniej wywołaniu tej funkcji.

Funkcja ta udostępniana jest przez obiekt `navigator.geolocation`, więc aby z niej skorzystać, należy najpierw pobrać sam obiekt. Jak już wcześniej wspominaliśmy, przed pobraniem obiektu warto sprawdzić, czy przeglądarka wspiera geolokalizację — jeżeli nie, należy zapewnić jakąś formę treści zastępczej.

Funkcja przyjmuje jeden obowiązkowy i dwa opcjonalne argumenty:

- Parametr `successCallback` mówi przeglądarce, która funkcja ma zostać wywołana po udostępnieniu danych geolokalizacyjnych. Jest to ważne, ponieważ operacja uzyskania informacji o lokalizacji może być długotrwała. Użytkownik nie będzie chciał, aby przeglądarka była zablokowana podczas pobierania danych. Programista także nie będzie chciał, aby jego program był zatrzymany na nieznaną okres — zwłaszcza że przedłużenie tego czasu może być spowodowane oczekiwaniem na udzielenie przez użytkownika pozwolenia. Funkcja `successCallback` jest miejscem, w którym dostaniesz właściwe dane i będziesz mógł z nich skorzystać.
- Jak w przypadku większości działań programistycznych, dobrze jest zabezpieczyć się przed wystąpieniem błędów. Możliwe jest, że proces pobierania lokalizacji zakończy się niepowodzeniem z przyczyn będących poza Twoją kontrolą — w takich przypadkach wykorzystywana jest funkcja `errorCallback`, która może przekazać użytkownikowi wyjaśnienia lub spróbować повторно pobrać dane. Mimo że parametr ten jest opcjonalny, warto z niego skorzystać.
- Można także przekazać obiekt opcji, aby zmodyfikować sposób pobierania informacji. Jest to parametr opcjonalny, który bardziej szczegółowo omówimy w dalszej części rozdziału.

Załóżmy, że stworzyłeś funkcję JavaScript o nazwie `updateLocation()`, która aktualizuje dane na stronie zgodnie z nową pozycją geograficzną, oraz funkcję `handleLocationError()` do obsługi ewentualnych błędów (szczegóły tych funkcji pokażemy w dalszej części). Oznacza to, że wywołanie żądania podania pozycji będzie wyglądało następująco:

```
navigator.geolocation.getCurrentPosition(updateLocation, handleLocationError);
```

Funkcja `updateLocation`

Co dzieje się wewnątrz funkcji `updateLocation()`? Funkcja zostanie wywołana, gdy tylko przeglądarka uzyska dostęp do danych geolokalizacyjnych. Zostanie do niej wówczas przekazany pojedynczy parametr: obiekt lokalizacji. Obiekt będzie zawierał koordynaty (atrybut `coords`) oraz stempel czasowy oznaczający moment pobrania danych. O ile stempel czasowy może być dla Ciebie nieprzydatny, o tyle atrybut `coords` zawiera najważniejsze informacje.

Koordynaty zawsze zawierają kilka atrybutów, jednak od przeglądarki i urządzenia, na którym została uruchomiona, zależy, czy atrybuty zostaną wypełnione wartościami. Pierwsze trzy atrybuty są następujące:

- `latitude` — szerokość geograficzna,
- `longitude` — długość geograficzna,
- `accuracy` — dokładność.

Te atrybuty muszą posiadać wartość, a ich znaczenie jest jasne. Atrybuty `latitude` i `longitude` będą przechowywać określoną przez usługę Geolocation lokalizację, wyrażoną w stopniach dziesiętnych. Atrybut `accuracy` informuje, w jakiej odległości (w metrach) od podanej lokalizacji znajduje się lokalizacja właściwa — informacja ta jest podawana z prawdopodobieństwem 95%, może więc zostać wykorzystana do zaprezentowania promienia odległości, który zobrazuje użytkownikowi poziom dokładności. Ze względu na specyfikę Geolocation API dane często będą podawane w przybliżeniu. Zawsze sprawdzaj dokładność uzyskanych danych, zanim zaprezentujesz wyniki. Polecenie użytkownikowi „pobliskiego” sklepu, do którego musiałby jechać wiele godzin, może mieć nieprzewidziane konsekwencje.

Pozostałe atrybuty nie muszą być podawane, a jeżeli nie zostaną uzupełnione, będą zawierały wartość `null` (np. jest mało prawdopodobne, że dostęp do tych informacji uzyskasz na komputerze stacjonarnym):

- `altitude` — wysokość nad poziomem morza wyrażona w metrach,
- `altitudeAccuracy` — dokładność wysokości w metrach lub `null`, jeżeli wysokość (`altitude`) nie zostanie podana,

- heading — wyprzedzenie, czyli kierunek w stopniach relatywny do kierunku północnego,
- speed — prędkość w metrach na sekundę.

Jeżeli nie jesteś pewien, czy użytkownicy dysponują urządzeniami podającymi takie informacje, nie powinieneś polegać na nich w swojej aplikacji. Urządzenia GPS prawdopodobnie będą udostępniać te dane, jednak prosta triangulacja już na to nie pozwoli.

Przyjrzyjmy się teraz implementacji funkcji `updateLocation` — wykonuje ona proste aktualizacje na podstawie pobranych informacji (listing 5.3).

Listing 5.3. Przykład wykorzystania funkcji `updateLocation`

```
function updateLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var accuracy = position.coords.accuracy;
    var timestamp = position.timestamp;

    document.getElementById("latitude").innerHTML = latitude;
    document.getElementById("longitude").innerHTML = longitude;
    document.getElementById("accuracy").innerHTML = accuracy;
    document.getElementById("timestamp").innerHTML = timestamp;
}
```

W przykładzie funkcja `updateLocation` aktualizuje tekst elementów na stronie. Długość, szerokość, dokładność i stempel czasowy wstawiamy w odpowiednie pola.

Funkcja `handleLocationError`

Bardzo istotnym elementem Geolocation API jest obsługa błędów; wiele jej części jest zmiennych i istnieje spore prawdopodobieństwo niedostępności usługi geolokalizacyjnej. Na szczęście API definiuje kody błędów dla wszystkich przypadków, z jakimi możesz mieć do czynienia. Do funkcji obsługi błędów przekazywany jest obiekt, którego atrybut `code` zawiera odpowiedni kod. Oto dostępne kody:

- `PERMISSION_DENIED` (kod 1) — użytkownik nie zezwolił na pobranie danych geolokalizacyjnych,
- `POSITION_UNAVAILABLE` (kod 2) — podjęta została próba pobrania danych, jednak zakończyła się niepowodzeniem,
- `TIMEOUT` (kod 3) — przekroczony został, zdefiniowany jako opcja, maksymalny czas oczekiwania na dane.

W przypadku wystąpienia któregoś z błędów będziesz najprawdopodobniej chciał dać użytkownikowi znać, że coś poszło nie tak. W dwóch ostatnich przypadkach możesz także spróbować ponownie pobrać dane.

Listing 5.4 zawiera przykład funkcji obsługującej błędy.

Listing 5.4. Funkcja obsługująca błędy

```
function handleLocationError(error) {
    switch(error.code){
        case 0:
            updateStatus("Podczas pobierania lokalizacji wystąpił błąd: " + error.message);
            break;
        case 1:
            updateStatus("Użytkownik nie zezwolił na dostęp do danych geolokalizacyjnych.");
            break;
        case 2:
            updateStatus("Przeglądarka nie była w stanie uzyskać informacji o lokalizacji: " +
                blad.message);
            break;
        case 3:
            updateStatus("Maksymalny dozwolony czas pobierania danych został przekroczony.");
    }
}
```

```

        break;
    }
}

```

Kody błędów udostępniane są w atrybucie `code` obiektu `error`, atrybut `message` zawiera bardziej szczegółowy opis napotkanego błędu. We wszystkich przypadkach wywołujemy stworzoną przez nas metodę aktualizującą status na stronie.

Opcjonalne atrybuty geolokalizacji

Obsłużyliśmy już zarówno poprawne, jak i niepoprawne pobranie lokalizacji. Możemy teraz przyjrzeć się możliwości wpływania na sposób pobierania danych przez usługę geolokalizacyjną poprzez przekazanie jednego z trzech atrybutów opcjonalnych. Mogą być one przekazane przy wykorzystaniu skróconej notacji obiektowej, dzięki czemu ich wykorzystanie jest bardzo proste.

- `enableHighAccuracy` — jest to informacja dla przeglądarki, że jeżeli jest to możliwe, powinna wykorzystać metodę pobierania lokalizacji gwarantującą najwyższą dokładność. Domyślne ustawienie przyjmuje wartość `false`, jednak jego włączenie może nie spowodować żadnej różnicy lub uzyskanie informacji może zająć więcej czasu. Opcji tej należy używać ostrożnie.

■ **Uwaga.** Ustawienie wysokiej dokładności jest tylko przełącznikiem `true/false`. API nie pozwala na definiowanie różnych poziomów dokładności za pomocą wartości numerycznych. Być może zostanie to zmienione w kolejnych wersjach specyfikacji.

- `timeout` — wartość podawana jest w milisekundach — oznacza maksymalny czas pobierania danych geolokalizacyjnych. Jeżeli obliczenia nie zostaną zakończone w podanym przedziale czasu, wywoływana jest funkcja obsługująca błędy. Domyślnie limit nie jest ustawiany.
- `maximumAge` — wartość ta określa, po jakim czasie dane geolokalizacyjne muszą być powtórnie przeliczone — podawana jest w milisekundach. Domyślna wartość to 0, co oznacza, że przeglądarka musi przeliczyć lokalizację natychmiast.

■ **Uwaga.** Być może zastanawiasz się, jaka jest różnica pomiędzy `timeout` a `maximumAge`. Mimo że ich nazwa jest podobna, ich zastosowanie jest różne. Wartość `timeout` odnosi się do czasu potrzebnego na pobranie danych, natomiast `maximumAge` określa *częstotliwość* pobierania lokalizacji. Jeżeli czas pobierania lokalizacji będzie dłuższy niż podany w parametrze `timeout`, wywołwany jest błąd. Jeżeli jednak przeglądarka nie posiada aktualnej lokalizacji (młodszej niż wynikająca z parametru `maximumAge`), będzie musiała pobrać ją ponownie. Możliwe jest przypisanie specjalnych wartości: ustawienie wartości na 0 powoduje jej każdorazowe pobranie, ustawienie na `Infinity` oznacza, że wartość nigdy nie będzie powtórnie pobierana.

Geolocation API nie pozwala Ci na poinstruowanie przeglądarki, jak często ma aktualizować pozycję, leży to całkowicie w jej gestii. Jedyne, co możemy zrobić, to przekazać jej parametry `maximumAge`. Nie możemy niestety dokładnie kontrolować czasu pobierania danych geolokalizacyjnych.

Zmodyfikujmy wcześniejsze wywołanie, dodając opcje przy wykorzystaniu skróconej notacji obiektowej:

```
navigator.geolocation.getCurrentPosition(updateLocation, handleLocationError, {timeout:10000});
```

Nowe wywołanie gwarantuje, że jeżeli pobieranie lokalizacji będzie trwało dłużej niż 10 sekund (10 000 milisekund), wywołany zostanie błąd — funkcja `handleLocationError` zostanie uruchomiona z kodem błędu `timeout`. Możemy połączyć powyższe fragmenty kodu w celu wyświetlenia danych na stronie (rysunek 5.4).

Ciągła aktualizacja lokalizacji

Czasami musisz cyklicznie powtarzać pobieranie lokalizacji. Dzięki projektantom Geolocation API modyfikacja aplikacji w celu ciągłego pobierania danych geolokalizacyjnych jest niezwykle prosta. Wystarczy tylko podmienić wywołanie głównej metody, jak pokazano w następujących przykładach.



Rysunek 5.4. Dane geolokalizacyjne wyświetlane w telefonie komórkowym

Jednorazowa aktualizacja:

```
navigator.geolocation.getCurrentPosition(updateLocation, handleLocationError);
```

Ciągła aktualizacja:

```
navigator.geolocation.watchPosition(updateLocation, handleLocationError);
```

Ta prosta zmiana spowoduje, że usługa geolokalizacji będzie wywoływała metodę `updateLocation`, jeżeli lokalizacja klienta ulegnie zmianie. Działa, jakby aplikacja *obserwowała* lokalizację klienta — zostaniesz poinformowany, jeżeli dane geograficzne się zmieniają.

Po co miałbyś z tego korzystać?

Wyobraź sobie aplikację, która podaje kolejne wskazówki dotarcia do jakiegoś miejsca, lub stronę, która jest ciągle aktualizowana, pokazując Ci najbliższe stacje benzynowe, podczas gdy poruszasz się po mieście lub jedziesz autostradą. Można nawet stworzyć aplikację, która zapisuje Twoje pozycje i pozwoli Ci później prześledzić przebytą trasę. Wszystkie te aplikacje są łatwe do napisania, jeżeli przeglądarka będzie automatycznie informować o zmianach lokalizacji.

Wyłączenie aktualizacji jest również proste. Jeżeli aplikacja przestanie potrzebować aktualnej pozycji użytkownika, wystarczy wywołać funkcję `clearWatch()`, zgodnie z następującym przykładem:

```
navigator.geolocation.clearWatch(watchId);
```

Funkcja ta poinformuje usługę geolokalizacji, że już nie potrzebujesz ciągłych aktualizacji lokalizacji. Czym jednak jest zmienna `watchId` i skąd się wzięła? Jest to wartość zwracana z funkcji `watchPosition()`. Zwraca identyfikator usługi obserwacji lokalizacji, dzięki czemu możemy z niej zrezygnować. Jeżeli więc Twoja aplikacja będzie musiała zrezygnować z ciągłej aktualizacji, powinieneś utworzyć kod podobny do tego z listingu 5.5.

Listing 5.5. Wykorzystanie funkcji `watchPosition`

```
var watchId = navigator.geolocation.watchPosition(updateLocation, handleLocationError);
// wykonanie operacji aktualizujących pozycję!
// teraz jesteśmy gotowi do zatrzymania usługi aktualizacji
navigator.geolocation.clearWatch(watchId);
```

Budowa aplikacji wykorzystującej geolokalizację

Do tej pory skupialiśmy się na jednokrotnym pobieraniu lokalizacji. Zobaczmy, co możemy osiągnąć, wykorzystując ciągłą aktualizację odległości — zbudujemy małą, lecz użyteczną aplikację: stronę pozwalającą na śledzenie odległości.

Jeżeli kiedykolwiek potrzebowaliśmy informacji, jaką odległość przebyłeś w danym czasie, wykorzystywałeś prawdopodobnie urządzenie dedykowane, na przykład nawigację GPS lub krokomierz. Dzięki wykorzystaniu geolokalizacji możesz stworzyć aplikację, która zmierzy odległość, jaką przebyłeś od momentu jej uruchomienia w przeglądarce. Aplikacja nie będzie zbyt użyteczna dla użytkowników komputerów stacjonarnych, jednak dla milionów właścicieli telefonów wspierających geolokalizację będzie idealnym narzędziem. Wystarczy uruchomić naszą przykładową aplikację, udzielić pozwolenia na dostęp do informacji o lokalizacji, a co kilka minut przebyta odległość zostanie zaktualizowana i doliczona do wartości sumarycznej (rysunek 5.5).



Rysunek 5.5. Przykład aplikacji wykorzystującej geolokalizację

Przykład wykorzystuje omawianą wcześniej metodę `watchPosition()`. Za każdym razem, kiedy zostanie przekazana nowa lokalizacja, porównamy ją z poprzednią i obliczymy przebytą odległość. Do tego celu wykorzystamy funkcję `haversin`, pozwalającą obliczyć odległość między dwoma punktami na kuli, reprezentowanymi przez długość i szerokość. Listing 5.6 przedstawia wzór.

Listing 5.6. Funkcja `haversin`

$$d = 2R \arcsin \left(\sqrt{\sin^2 \left(\frac{\varphi_2 - \varphi_1}{2} \right) + \cos \varphi_1 \cos \varphi_2 \sin^2 \left(\frac{\lambda_2 - \lambda_1}{2} \right)} \right)$$

Jeżeli chciałbyś dowiedzieć się, jak działa ta funkcja, to się zawiedziesz. Zamiast tego pokażemy jej implementację w JavaScriptcie, którą możesz wykorzystać do obliczenia odległości pomiędzy dwoma lokalizacjami (listing 5.7).

Listing 5.7. *Implementacja funkcji haversin*

```
Number.prototype.toRadians = function() {
    return this * Math.PI / 180;
}
function distance(latitude1, longitude1, latitude2, longitude2) {
    // R oznacza promień Ziemi w kilometrach
    var R = 6371;

    var deltaLatitude = (latitude2- latitude 1).toRadians();
    var deltaLongitude = (longitude2-longitude1).toRadians();
    latitude = latitude1.toRadians(),latitude2 = latitude2.toRadians();

    var a = Math.sin(deltaLatitude/2) *
        Math.sin(deltaLatitude/2) +
        Math.cos(latitude1) *
        Math.cos(latitude2) *
        Math.sin(deltaLongitude/2) *
        Math.sin(deltaLongitude/2);
    var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
    var d = R * c;
    return d;
}
```

Jeżeli chcesz się dowiedzieć, czy lub dlaczego ten wzór działa, zajrzyj do podręcznika matematyki z liceum. Na nasze potrzeby dokonaliśmy konwersji ze stopni na radiany oraz stworzyliśmy funkcję `distance()` wyliczającą odległość pomiędzy dwoma punktami.

Jeżeli pozycja użytkownika będzie sprawdzana w regularnych i krótkich odstępach czasu, suma obliczanych odległości da w dobrym przybliżeniu długość przebytej trasy. Na potrzeby przykładu zakładamy, że użytkownik pomiędzy poszczególnymi pomiarami przemieszcza się w linii prostej.

Stworzenie kodu HTML

Zacznijmy od kodu HTML — jest stosunkowo prosty, ponieważ tym, co nas naprawdę interesuje, jest kod JavaScript. Na stronie wyświetlane są informacje o pozycji użytkownika. Dodatkowo umieścimy też informację pozwalającą użytkownikowi na sprawdzenie, jaką odległość przebył (listing 5.8).

Listing 5.8. *Kod HTML dla aplikacji mierzącej przebytą odległość*

```
<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" >
  <title>Geolokalizacja</title>
  <link rel="stylesheet" href="geo-html5.css" >
</head>

<body onload="loadDemo()">

  <header>
    <h1>Przebyta trasa</h1>
    <h4>Aktualne informacje!</h4>
```

```

</header>

<div id="container">

<section>
  <article>
    <header>
      <h1>Twoja lokalizacja</h1>
    </header>

    <p class="info" id="status">Twoja przeglądarka nie wspiera geolokalizacji.</p>

    <div class="geostatus">
      <p id="latitude">Szerokość: </p>
      <p id="longitude">Długość: </p>
      <p id="accuracy">Dokładność: </p>
      <p id="timestamp">Stempel czasowy: </p>
      <p id="currDist">Aktualnie przebyta odległość: </p>
      <p id="totalDist">Całkowita przebyta odległość: </p>
    </div>
  </article>
</section>

<footer>
  <h2>Zasilane przez HTML5 i Twoje stopy!</h2>
</footer>

</div>
.
.
.
</body>
</html>

```

Na razie wstawione zostały wartości domyślne, które zostaną zastąpione wartościami pobranymi z API.

Przetwarzanie danych geolokalizacyjnych

Pierwszy fragment kodu JavaScript jest już znany. Po załadowaniu strony uruchamiany funkcję `loadDemo()`. Skrypt sprawdzi, czy geolokalizacja jest wspierana, i za pośrednictwem funkcji aktualizujących status wyświetli wynik sprawdzenia na stronie. Następnie rozpocznie obserwację lokalizacji użytkownika (listing 5.9).

Listing 5.9. Funkcja `loadDemo()` oraz funkcje aktualizujące status

```

var totalDistance = 0.0;
var lastLat;
var lastLong;

function updateErrorStatus(message) {
  document.getElementById("status").style.background = "papayaWhip";
  document.getElementById("status").innerHTML = "<strong>Błąd</strong>: " + message;
}

function aktualizujStatus(message) {
  document.getElementById("status").style.background = "paleGreen";
  document.getElementById("status").innerHTML = message;
}

```



```
function loadDemo() {
  if(navigator.geolocation) {
    document.getElementById("status").innerHTML = "Geolokalizacja jest wspierana.";
    navigator.geolocation.watchPosition(updateLocation, handleLocationError, {maximumAge:20000});
  }
}
```

Zauważ, że wykorzystaliśmy ustawienie `{maximumAge:20000}` — jest to informacja dla usługi geolokalizacji, że nie interesują nas dane starsze niż 20 sekund (20 000 milisekund). Dzięki temu dane będą regularnie aktualizowane; możesz poeksperymentować z różnymi wartościami atrybutu.

Na potrzeby obsługi błędów wykorzystamy funkcję, którą stworzyliśmy we wcześniejszej części rozdziału — jest wystarczająco ogólna, abyśmy mogli jej użyć. Jeżeli wystąpi błąd, wewnątrz funkcji sprawdzimy jego kod i zaktualizujemy informacje na stronie (listing 5.10).

Listing 5.10. *Kod obsługujący błędy*

```
function handleLocationError(error) {
  switch(error.code){
    case 0:
      updateErrorStatus("Podczas pobierania lokalizacji wystąpił błąd: " + error.message);
      break;
    case 1:
      updateErrorStatus("Użytkownik nie zezwolił na dostęp do danych geolokalizacyjnych.");
      break;
    case 2:
      updateErrorStatus("Przeglądarka nie była w stanie uzyskać informacji o lokalizacji: " +
        error.message);
      break;
    case 3:
      updateErrorStatus("Maksymalny dozwolony czas pobierania danych został przekroczony.");
      break;
  }
}
```

Większość pracy będzie wykonywała funkcja `updateLocation()`. Wewnątrz tej funkcji zaktualizujemy wartości na stronie zgodnie z aktualnymi danymi oraz obliczymy przebytą odległość (listing 5.11).

Listing 5.11. *Funkcja `updateLocation`*

```
function updateLocation(position) {
  var latitude = position.coords.latitude;
  var longitude = position.coords.longitude;
  var accuracy = position.coords.accuracy;
  var timestamp = position.timestamp;

  document.getElementById("latitude").innerHTML = "Szerokość: " + latitude;
  document.getElementById("longitude").innerHTML = "Długość: " + longitude;
  document.getElementById("accuracy").innerHTML = "Dokładność: " + accuracy + " metrów";
  document.getElementById("timestamp").innerHTML = "Stempel czasowy: " + timestamp;
}
```

Pierwszą rzeczą, jaką robimy po uzyskaniu nowego zestawu współrzędnych, jest ich zapisanie. Pobieramy szerokość, długość, dokładność i stempel czasowy, a następnie wyświetlamy te informacje na stronie.

W swojej aplikacji możesz nie wyświetlać stempla czasowego. Data w takiej formie jest użyteczna przede wszystkim dla komputerów — dla użytkownika taki zapis jest niezrozumiały. Możesz zastąpić ją datą w bardziej czytelnej formie lub usunąć.

Dokładność podawana jest w metrach i na pierwszy rzut oka może się wydawać niepotrzebna. Wartość każdego danych jest jednak zależna od ich dokładności. Nawet jeżeli nie pokazujesz użytkownikowi, jaka jest dokładność, powinienś brać ją pod uwagę podczas obliczeń. Wyświetlanie niedokładnych wartości mogłoby

dać użytkownikowi błędne informacje na temat lokalizacji. W związku z tym wszystkie aktualizacje z małą dokładnością nie będą w naszym kodzie brane pod uwagę (listing 5.12).

Listing 5.12. Ignorowanie niedokładnych danych

```
// test zdrowego rozsądku... nie obliczamy odległości,
// jeżeli wartość dokładności jest zbyt duża
if (accuracy >= 30000) {
    updateStatus("Do obliczenia odległości potrzebne są dokładniejsze wartości.");
    return;
}
```

Najprostszy sposób podróżowania

Brian mówi: Sprawdzanie dokładności danych geolokalizacyjnych jest kluczowe. Nie masz dostępu do metod, z jakich przeglądarka korzysta, aby te dane uzyskać, masz jednak dostęp do informacji o dokładności. Wykorzystaj to!

Siedząc w hamaku w swoim ogrodzie, sprawdziłem swoją lokalizację za pośrednictwem telefonu komórkowego. Zdziwiłem się, ponieważ dowiedziałem się, że przez zaledwie kilka minut przemieściłem się o wiele kilometrów z różnymi prędkościami. Należy pamiętać, że dane są tylko tak dokładne, jak pozwala na to ich źródło.

Na koniec obliczymy przebytą odległość, zakładając, że otrzymaliśmy wcześniej przynajmniej jedną wystarczająco dokładną lokalizację. Zaktualizujemy całkowitą przebytą odległość i wyświetlimy odpowiednią informację na stronie. Aby uzyskać większą przejrzystość interfejsu, warto zaokrąglić obliczone wartości (listing 5.13).

Listing 5.13. Kod obliczający odległość

```
// obliczenie odległości
if ((lastLat != null) && (lastLong != null)) {
    var currentDistance = distance(latitude, longitude, lastLat, lastLong);

    document.getElementById("currDist").innerHTML =
        "Aktualnie przebyta odległość: " + currentDistance.toFixed(2) + " km";

    totalDistance += currentDistance;
    document.getElementById("totalDist").innerHTML =
        "Całkowita przebyta odległość: " + totalDistance.toFixed(2) + " km";
    updateStatus("Aktualizacja lokalizacji zakończona poprawnie.");
}

lastLat = latitude;
lastLong = longitude;
}
```

I to wszystko. Aby stworzyć aplikację śledzącą lokalizację użytkownika i demonstrującą praktycznie całe API geolokalizacji, potrzebowaliśmy mniej niż 200 linii kodu. Przykład jest mało interesujący na urządzeniach stacjonarnych — wypróbuj go w telefonie wspierającym geolokalizację i sprawdź, jaką odległość przebywasz w ciągu dnia.

Kompletny kod

Na listingu 5.14 przedstawiony został kompletny kod aplikacji.

Listing 5.14. *Kompletny kod aplikacji śledzącej przebytą odległość*

```

<!DOCTYPE html>
<html>

<head>
  <meta charset="utf-8" >
  <title>Geolokalizacja</title>
  <link rel="stylesheet" href="geo-htm15.css" >
</head>

<body onload="loadDemo()">

  <header>
    <h1>Przebyta trasa</h1>
    <h4>Aktualne informacje!</h4>
  </header>

  <div id="container">

    <section>
      <article>
        <header>
          <h1>Twoja lokalizacja</h1>
        </header>

        <p class="info" id="status">Twoja przeglądarka nie wspiera geolokalizacji.</p>

        <div class="geostatus">
          <p id="latitude">Szerokość: </p>
          <p id="longitude">Długość: </p>
          <p id="accuracy">Dokładność: </p>
          <p id="timestamp">Stempel czasowy: </p>
          <p id="currDist">Aktualnie przebyta odległość: </p>
          <p id="totalDist">Całkowita przebyta odległość: </p>
        </div>
      </article>
    </section>

    <footer>
      <h2>Zasilane przez HTML5 i Twoje stopy!</h2>
    </footer>

  </div>

  <script>
    var totalDistance = 0.0;
    var lastLat;
    var lastLong;

    Number.prototype.toRadians = function() {
      return this * Math.PI / 180;
    }

    function distance(latitude1, longitude1, latitude2, longitude2) {
      // R oznacza promień Ziemi w kilometrach
      var R = 6371;

      var deltaLatitude = (latitude2-latitude1).toRadians();

```

```

var deltaLongitude = (longitude2-longitude1).toRadians();
latitude1 = latitude1.toRadians(), latitude2 = latitude2.toRadians();
var a = Math.sin(deltaLatitude/2) *
    Math.sin(deltaLatitude/2) +
    Math.cos(latitude1) *
    Math.cos(latitude2) *
    Math.sin(deltaLongitude/2) *
    Math.sin(deltaLongitude/2);

var c = 2 * Math.atan2(Math.sqrt(a), Math.sqrt(1-a));
var d = R * c;
return d;
}

function updateErrorStatus(message) {
    document.getElementById("status").style.background = "papayaWhip";
    document.getElementById("status").innerHTML = "<strong>Błąd</strong>: " + message;
}

function updateStatus(message) {
    document.getElementById("status").style.background = "paleGreen";
    document.getElementById("status").innerHTML = message;
}

function loadDemo() {
    if(navigator.geolocation) {
        document.getElementById("status").innerHTML = "Geolokalizacja jest wspierana.";
        navigator.geolocation.watchPosition(updateLocation, handleLocationError,
            {maximumAge:20000});
    }
}

function updateLocation(position) {
    var latitude = position.coords.latitude;
    var longitude = position.coords.longitude;
    var accuracy = position.coords.accuracy;
    var timestamp = position.timestamp;

    document.getElementById("latitude").innerHTML = "Szerokość: " + latitude;
    document.getElementById("longitude").innerHTML = "Długość: " + longitude;
    document.getElementById("accuracy").innerHTML = "Dokładność: " + accuracy + " metrów";
    document.getElementById("timestamp").innerHTML = "Stempel czasowy: " + timestamp;

    // test zdrowego rozsądku... nie obliczamy odległości,
    // jeżeli wartość dokładności jest zbyt duża
    if (accuracy >= 30000) {
        updateStatus("Do obliczenia odległości potrzebne są dokładniejsze wartości.");
        return;
    }

    // obliczenie odległości
    if ((lastLat != null) && (lastLong != null)) {
        var currentDistance = distance(latitude, longitude, lastLat, lastLong);

        document.getElementById("currDist").innerHTML =
            "Aktualnie przebyta odległość: " + currentDistance.toFixed(2) + " km";

        totalDistance += currentDistance;
    }
}

```

```

        document.getElementById("totalDist").innerHTML =
            "Całkowita przebyta odległość: " + totalDistance.toFixed(2) + " km";
        updateStatus("Aktualizacja lokalizacji zakończona poprawnie.");
    }

    lastLat = latitude;
    lastLong = longitude;
}

function handleLocationError(error) {
    switch(error.code){
        case 0:
            aktualizujStatus("Podczas pobierania lokalizacji wystąpił błąd: " +
                error.message);
            break;
        case 1:
            aktualizujStatus("Użytkownik nie zezwolił na dostęp do danych
                geolokalizacyjnych.");
            break;
        case 2:
            aktualizujStatus("Przeglądarka nie była w stanie uzyskać informacji
                o lokalizacji: " + error.message);
            break;
        case 3:
            aktualizujStatus("Maksymalny dozwolony czas pobierania danych został
                przekroczony.");
            break;
    }
}
</script>
</body>
</html>

```

Praktyczne dodatki

Są techniki, które nie pasują do pokazywanych przykładów, ale są bardzo przydatne podczas tworzenia aplikacji w HTML5. Poniżej pokażemy kilka prostych, lecz przydatnych technik.

Jaki jest mój status?

Na pewno zauważyłeś już, że znaczna część Geolocation API dotyczy wartości czasowych. To nie powinno dziwić. Techniki sprawdzające lokalizację — triangulacja telefoniczna, GPS, sprawdzanie adresu IP — mogą sprawiać problemy: pobieranie lokalizacji może trwać bardzo długo lub w ogóle się nie zakończyć. Na szczęście API dostarcza narzędzia pozwalające na zbudowanie paska postępu.

Programista, ustawiając opcję `timeout` przy sprawdzaniu danych geolokalizacyjnych, zawiadamia przeglądarkę, że powinna wygenerować błąd, jeżeli operacja przekroczy zadany czas. Podczas sprawdzania lokalizacji możemy pokazać użytkownikowi informację mówiącą o aktualnym postępie. Wyświetlanie rozpoczyna się w momencie wysłania żądania pobrania pozycji, natomiast jego zakończenie jest definiowane przez wartość atrybutu `timeout`, niezależnie od tego, czy żądanie zakończy się sukcesem.

Na listingu 5.15 uruchamiamy licznik aktualizujący dane na stronie nową wartością oznaczającą postęp pobierania lokalizacji.

Listing 5.15. Monitorowanie statusu pobierania danych

```
function updateStatus(message) {
```

```

    document.getElementById("status").innerHTML = message;
}

function endRequest() {
    updateStatus("Zakończono.");
}

function updateLocation(position) {
    endRequest();
    // obsługa danych geolokalizacyjnych
}

function handleLocationError(error) {
    endRequest();
    // obsługa błędów
}

navigator.geolocation.getCurrentPosition(updateLocation,
                                        handleLocationError,
                                        {timeout:10000}); // maksymalny czas pobierania – 10 sekund

updateStatus("Pobieranie danych geolokalizacyjnych...");

```

Przyjrzyjmy się poszczególnym częściom. Jak poprzednio, mamy funkcję aktualizującą status na stronie:

```

function updateStatus(message) {
    document.getElementById("status").innerHTML = message;
}

```

Status podajemy w formie tekstu, jednak w podobny sposób moglibyśmy zbudować bardziej skomplikowaną, graficzną reprezentację postępu (listing 5.16).

Listing 5.16. Wyświetlenie statusu

```

navigator.geolocation.getCurrentPosition(updateLocation,
                                        handleLocationError,
                                        {timeout:10000}); // maksymalny czas pobierania – 10 sekund

updateStatus("Pobieranie danych geolokalizacyjnych...");

```

Po raz kolejny wykorzystujemy API geolokalizacji w celu pobrania aktualnej lokalizacji użytkownika, tym razem ustawiliśmy jednak maksymalny czas dla żądania. Po upływie podanego czasu żądanie zakończy się sukcesem lub błędem.

Natychmiast po tym wywołaniu aktualizujemy status na stronie, aby poinformować użytkownika, że pobieranie się rozpoczęło. Po pobraniu lokalizacji lub upływie 10 sekund modyfikujemy status, informując użytkownika o zakończeniu operacji (listing 5.17).

Listing 5.17. Końcowa aktualizacja statusu

```

function endRequest() {
    updateStatus("Zakończono.");
}

function updateLocation(position) {
    endRequest();
    // obsługa danych geolokalizacyjnych
}

```

Jest to prosty skrypt, łatwo go jednak rozbudować.

Ta metoda jest wygodna w przypadku jednorazowego pobierania lokalizacji. Wtedy programista dokładnie wie, kiedy rozpoczyna się pobieranie — oczywiście w momencie uruchomienia funkcji `getCurrentPosition()`. W przypadku ciągłej aktualizacji lokalizacji (funkcja `watchPosition()`) programista nie ma kontroli nad tym, kiedy rozpoczyna się pobieranie.

Co więcej, czas jest liczony dopiero od momentu udzielenia przez użytkownika zgody na pobieranie. Implementowanie aktualizacji statusu w przypadku ciągłej aktualizacji jest niepraktyczne, ponieważ strona nie jest informowana o momencie, w którym użytkownik udzielił pozwolenia.

Pokaż mnie na mapie

Częstym zadaniem przy korzystaniu z geolokalizacji jest pokazanie lokalizacji użytkownika na mapie — na przykład podczas korzystania z popularnej usługi Mapy Google. Jest to tak popularne, że Google wbudowało możliwości geolokalizacji w swój produkt. Wystarczy nacisnąć przycisk *Pokaż moją lokalizację* (rysunek 5.6); po jego kliknięciu wykorzystane zostanie Geolocation API (jeżeli będzie dostępne) do pobrania i wyświetlenia lokalizacji na mapie.



Rysunek 5.6. Mapy Google i pobieranie aktualnej lokalizacji

Możliwe jest także samodzielne wykonanie takiej operacji. Google Maps API zostało zaprojektowane tak, aby pobierać szerokość i długość geograficzną w formie liczb zmiennoprzecinkowych (i nie jest to zbieg okoliczności). W łatwy sposób możesz więc przekazać pobrane wartości do API (listing 5.18). Więcej na ten temat dowiesz się z książki *Beginning Google Maps Applications* (Apress, 2010, wydanie drugie).

Listing 5.18. Przekazanie lokalizacji do Google Maps API

```
// Załączenie biblioteki Google Maps
<script src="http://maps.google.com/maps/api/js?sensor=false"></script><script>

// Utworzenie mapy...
var map = new google.maps.Map(document.getElementById("map"));

function updateLocation(position) {
    // przekazanie współrzędnych do Google Maps API
    map.setCenter(new google.maps.LatLng(
        parseFloat(position.coords.latitude),
        parseFloat(position.coords.longitude)));
}
```

```
}  
navigator.geolocation.getCurrentPosition(updateLocation, handleLocationError);
```

Podsumowanie

W tym rozdziale omówiliśmy geolokalizację. Dowiedziałeś się, z czego składają się dane geolokalizacyjne (szerokość, długość i inne atrybuty) oraz z jakich źródeł mogą pochodzić. Dyskutowaliśmy także na temat problemów z prywatnością związanych z geolokalizacją. Pokazaliśmy również, jak, używając API, tworzyć aplikacje wykorzystujące informacje o lokalizacji użytkownika.

W następnym rozdziale pokażemy, w jaki sposób HTML5 pozwala na komunikację pomiędzy zakładkami i oknami, a także pomiędzy stronami i serwerami działającymi w różnych domenach.

Skorowidz

A

- AAC, 90
- accuracy, atrybut, 116
- addColorStop(), 54
- addEventListener(), 36
- Adobe Illustrator, 87
- alert(), 35
- altitude, atrybut, 116
- altitudeAccuracy, atrybut, 116
- animacje, 68, 69, 70
- aplikacje lokalne, 261, 263, 266
 - praca w trybie offline, 264
 - wsparcie przez przeglądarki, 263
- arc(), 52
- arcTo(), 52
- ARIA, 23
- article, element, 26, 27
- aside, element, 26, 27
- Asyncore, 157
- asyncore.dispatcher_with_send, 157
- audio
 - kodeki, 90
 - restrykcje, 91
 - w tle strony, 106
 - wsparcie przez przeglądarki, 91, 92
- Audio Video Interleave, *Patrz* avi
- audio, element, 26, 92
 - autobuffer, atrybut, 97
 - autoplay, atrybut, 96, 97
 - controls, atrybut, 95, 97
 - currentSrc, atrybut, 97

- currentTime, atrybut, 97
- duration, atrybut, 97
- Ended, atrybut, 97
- Error, atrybut, 97
- loop, atrybut, 97
- muted, atrybut, 97
- paused, atrybut, 97
- src, atrybut, 95
- startTime, atrybut, 97
- volume, atrybut, 97
- autobuffer, atrybut, 97
- autocomplete, atrybut, 183
- autofocus, atrybut, 183
- autoplay, atrybut, 96, 97
- avi, 89

B

- baza indeksowana, *Patrz* Indexed Database
- bąbelkowanie zdarzeń, 200
- beginPath(), 44, 45, 47
- bezierCurveTo(), 52
- bezpieczeństwo, 22
 - pochożenia, 133
- biblioteki
 - excanvas.js, 218
 - flashcanvas.js, 218
 - Modernizr, 92
- border-radius, 28
- bufferedAmount, atrybut, 165
- bufor aplikacji, 262
- button, element, 26

C

- canPlayType(), 92, 96, 97
- Canvas API, 42
- canvas, element, 23, 25, 26, 39, 40
 - a SVG, 40
 - addColorStop(), 54
 - animacje, 68, 69, 70
 - arc(), 52
 - arcTo(), 52
 - beginPath(), 44, 45, 47
 - bezierCurveTo(), 52
 - bezpieczeństwo, 64
 - cienie, 61
 - clearRect(), 50
 - closePath(), 47
 - createImageData(), 64
 - createPattern(), 55
 - CSS, 42
 - desenie tła, 55, 56
 - dodanie na stronę, 43
 - dostępność, 41
 - fill(), 47
 - fillRect(), 50
 - fillStyle, właściwość, 50
 - fillText(), 60
 - font, właściwość, 60
 - getImageData(), 62, 63
 - gradienty, 53, 54, 55
 - historia, 39
 - ID, atrybut, 43
 - koordynaty, 40
 - krzywe, 51, 52
 - lineCap, właściwość, 49
 - lineJoin, właściwość, 49
 - lineTo(), 44, 45, 47
 - linia ukośna, 43, 44
 - measureText(), 60
 - moveTo(), 44, 45, 47
 - obramowanie, 49
 - piksele, 62, 63, 64
 - putImageData(), 63
 - rotate(), 58
 - shadowBlur, właściwość, 61
 - shadowColor, właściwość, 61
 - shadowOffset, właściwość, 61
 - stroke(), 47
 - strokeRect(), 50
 - strokeStyle, właściwość, 49
 - strokeText(), 60
 - ścieżki, 47
 - tekst, 60, 74
 - textAlign, właściwość, 60
 - textBaseLine, właściwość, 60
 - toDataURL(), 64
 - transformacje, 45, 58
 - translate(), 45, 46
 - wsparcie przez przeglądarki, 42
 - wstawianie obrazów, 53
 - wypełnienie, 49, 50
 - zawartość zastępcza, 41
 - zmiana rozmiaru obiektów, 56, 57
- Carakan, 36
- cel upuszczenia, 198
- Chakra, 36
- CHECKING, status, 266
- checkValidity(), 188
- Chrome
 - prefiks, 31
 - V8, 36
 - wsparcie Indexed Database, 257
 - wsparcie Web SQL, 254
 - wspierane kodeki, 91
- ciasteczka, 235
- cienie, 61
- clearRect(), 50
- clearWatch(), 119
- close(), 165
- close, zdarzenie, 165
- closePath(), 47
- color, typ, 182
- Comet, 149
- console.log, 35
- controls, atrybut, 95, 97
- cookies, *Patrz* ciasteczka
- CORS, 140
- createElementNS(), 83
- createImageData(), 64
- createPattern(), 55
- Cross Document Messaging, 131
- Cross-Origin Resource Sharing, *Patrz* CORS
- CSS3, 28, 31
 - canvas, element, 42
 - in-range, pseudoklasa, 194
 - invalid, pseudoklasa, 194
 - optional, pseudoklasa, 194
 - out-of-range, pseudoklasa, 194
 - prefiksy, 31
 - required, pseudoklasa, 194
 - transformacje obracające element, 28
 - valid, pseudoklasa, 194
 - zaokrąglone rogi, 28
- currentSrc, atrybut, 97
- currentTime, atrybut, 97, 103

customError, 188
 czat, komunikacja, 132

D

dane, współdzielenie między oknami, 258
 dataList, element, 184
 dataTransfer, 199, 203
 addElement(), 203
 clearData(), 203
 dropEffect, właściwość, 203
 effectAllowed, właściwość, 203
 files, właściwość, 203
 getData(), 203
 items, właściwość, 203
 setData(), 203
 setDragImage(), 203
 types, właściwość, 203
 date, typ, 182
 datetime, typ, 182
 datetime-local, typ, 182
 debugowanie JavaScript, 34, 35
 defs, element, 78
 desenie
 canvas, 55, 56
 SVG, 79
 detekcja
 możliwości, 24
 przeglądarki, 24
 Developers Tools, 35
 device, element, 280
 dispatchEvent(), 36
 długość geograficzna, 110
 DOCTYPE, 25
 DOMStorage, 235
 dotyk, 282
 dotykowy, ekran, 280
 DOWNLOADING, status, 266
 drag, zdarzenie, 201
 drag-and-drop, *Patrz* przeciągnij i upuść
 dragend, zdarzenie, 202
 dragenter, zdarzenie, 201, 207
 draggable, atrybut, 202
 dragleave, zdarzenie, 201, 207
 Dragonfly, 35
 dragover, zdarzenie, 202
 dragstart, zdarzenie, 201
 drop, zdarzenie, 202
 dropzone, atrybut, 211
 duration, atrybut, 97
 DZSlides, 132, 133

dźwięk, 280
 w tle strony, 106
 wsparcie przez przeglądarki, 92

E

ekran dotykowy, 280
 elementy
 article, 26, 27
 aside, 26, 27
 audio, 26, 92
 autobuffer, atrybut, 97
 autoplay, atrybut, 96, 97
 controls, atrybut, 95, 97
 currentSrc, atrybut, 97
 currentTime, atrybut, 97
 duration, atrybut, 97
 Ended, atrybut, 97
 Error, atrybut, 97
 loop, atrybut, 97
 muted, atrybut, 97
 paused, atrybut, 97
 src, atrybut, 95
 startTime, atrybut, 97
 volume, atrybut, 97
 button, 26
 canvas, 23, 25, 26, 39, 40
 a SVG, 40
 addColorStop(), 54
 animacje, 68, 69, 70
 arc(), 52
 arcTo(), 52
 beginPath(), 44, 45, 47
 bezierCurveTo(), 52
 bezpieczeństwo, 64
 cienie, 61
 clearRect(), 50
 closePath(), 47
 createImageData(), 64
 createPattern(), 55
 CSS, 42
 desenie tła, 55, 56
 dodanie na stronę, 43
 dostępność, 41
 fill(), 47
 fillRect(), 50
 fillStyle, właściwość, 50
 fillText(), 60
 font, właściwość, 60
 getImageData(), 62, 63
 gradienty, 53, 54, 55
 historia, 39

elementy

- ID, atrybut, 43
- koordynaty, 40
- krzywe, 51, 52
- lineCap, właściwość, 49
- lineJoin, właściwość, 49
- lineTo(), 44, 45, 47
- linia ukośna, 43, 44
- measureText(), 60
- moveTo(), 44, 45, 47
- obramowanie, 49
- piksele, 62, 63, 64
- putImageData(), 63
- rotate(), 58
- shadowBlur, właściwość, 61
- shadowColor, właściwość, 61
- shadowOffset, właściwość, 61
- stroke(), 47
- strokeRect(), 50
- strokeStyle, właściwość, 49
- strokeText(), 60
- ścieżki, 47
- tekst, 60, 74
- textAlign, właściwość, 60
- toDataURL(), 64
- transformacje, 45, 58
- translate(), 45, 46
- wsparcie przez przeglądarki, 42
- wstawianie obrazów, 53
- wypełnienie, 49, 50
- zawartość zastępcza, 41
- zmiana rozmiarów obiektów, 56, 57

- dataList, 184
- defs, 78
- device, 280
- footer, 27
- form, 26
- g, 78
- h1, 26
- h2, 26
- header, 27
- hgroup, 26
- iframe, 26
- kbd, 26
- mark, 26
- nav, 27
- option, 184
- progress, 181
- script, 26
- section, 27
- sekcjonujące, 27
- small, 26

- source, 95
 - type, atrybut, 95, 96
- style, 26
- sub, 26
- sup, 26
- textarea, 26
- title, 26
- use, 78
- video, 26, 92
 - autobuffer, atrybut, 97
 - autoplay, atrybut, 96, 97
 - controls, atrybut, 97
 - currentSrc, atrybut, 97
 - currentTime, atrybut, 97, 103
 - duration, atrybut, 97
 - Ended, atrybut, 97
 - Error, atrybut, 97
 - height, atrybut, 99
 - loop, atrybut, 97
 - muted, atrybut, 97
 - paused, atrybut, 97
 - Poster, atrybut, 99
 - startTime, atrybut, 97
 - videoHeight, atrybut, 99
 - videoWidth, atrybut, 99
 - volume, atrybut, 97
 - width, atrybut, 99
- email, typ, 179, 180
- enableHighAccuracy, atrybut, 118
- Ended, atrybut, 97
- Error, atrybut, 97
- error, zdarzenie, 165
- event bubbling, *Patrz* bąbelkowanie zdarzeń
- event capture, *Patrz* przechwytywanie zdarzeń
- excanvas.js, 218
- executeSQL(), 255
- EXMAScript 5, 35

F

- fill(), 47
- fillRect(), 50
- fillStyle, właściwość, 50
- fillText(), 60
- Firebug, 34
- Firefox
 - Carakan, 36
 - Firebug, 34
 - JägerMonkey, 36
 - prefiks, 31
 - wsparcie Indexed Database, 257

wsparcie Web SQL, 254
 wspierane kodeki, 91
 Flash Video, *Patrz* flv
 flashcanvas.js, 218
 flv, 89
 font, właściwość, 60
 footer, element, 27
 form, element, 26
 formularze, 177, 178, 179, 182

- autocomplete, atrybut, 183
- autofocus, atrybut, 183
- color, typ, 182
- date, typ, 182
- datetime, typ, 182
- datetime-local, typ, 182
- email, typ, 179, 180
- list, atrybut, 184
- max, atrybut, 184
- min, atrybut, 184
- month, typ, 182
- numer, typ, 179
- placeholder, atrybut, 182
- range, typ, 179, 180
- required, atrybut, 185
- search, typ, 179, 180
- spellcheck, atrybut, 184
- step, atrybut, 185
- stylowanie, 194
- tel, typ, 179
- time, typ, 182
- url, typ, 179, 180
- valueAsNumber(), 185
- walidacja, 185, 188, 190, 193
- week, typ, 182
- wsparcie przez przeglądarki, 178

 framebusting, 148

G

g, element, 78
 Geolocation API, 109
 geolokalizacja

- accuracy, atrybut, 116
- altitude, atrybut, 116
- altitudeAccuracy, atrybut, 116
- ciągła aktualizacja, 119
- clearWatch(), 119
- enableHighAccuracy, atrybut, 118
- heading, atrybut, 117
- jednorazowa aktualizacja, 119
- jednorazowa prośba, 115
- koordynaty, 110

latitude, atrybut, 116
 longitude, atrybut, 116
 maximumAge, atrybut, 118
 na mapie, 129
 obchodzenie się z danymi, 114
 obsługa błędów, 117
 oparta na danych od użytkownika, 112
 oparta na GPS, 111
 oparta na IP, 111
 oparta na triangulacji, 112
 oparta na Wi-Fi, 111, 112
 PERMISSION_DENIED, 117
 pochodzenie informacji, 110
 POSITION_UNAVAILABLE, 117
 prywatność, 113
 speed, atrybut, 117
 szerokość i długość geograficzna, 110
 TIMEOUT, 117
 timeout, atrybut, 118
 watchPosition(), 120
 wsparcie przez przeglądarki, 112, 115
 wykorzystanie, 109
 gesty, 281

- zdarzenia, 281

 getElementById(), 32
 getElementsByName(), 32
 getElementsByTagName(), 32
 getImageData(), 62, 63
 GLSL, 278
 Google Chrome Developer Tools, 35
 GPS, 111
 gradienty

- canvas, 53, 54, 55
- SVG, 79

 grafika

- rastrowa, 73, 75
- tryb natychmiastowy, 75
- tryb opóźniony, 75
- wektorowa, 73

H

H.264, 90
 h1, element, 26
 h2, element, 26
 haversin, funkcja, 120, 121
 header, element, 27
 heading, atrybut, 117
 height, atrybut, 99
 hgroup, element, 26
 Hickson, Ian, 149, 151

- HTML, historia, 19
- HTML5, 19, 20
 - aplikacje lokalne, 261, 263, 266
 - praca w trybie offline, 264
 - wsparcie przez przeglądarki, 263
 - bezpieczeństwo, 22
 - elementy, 24
 - sekcjonujące, 27
 - formularze, 177, 178, 179, 182
 - autocomplete, atrybut, 183
 - autofocus, atrybut, 183
 - color, typ, 182
 - date, typ, 182
 - datetime, typ, 182
 - datetime-local, typ, 182
 - email, typ, 179, 180
 - list, atrybut, 184
 - max, atrybut, 184
 - min, atrybut, 184
 - month, typ, 182
 - numer, typ, 179
 - placeholder, atrybut, 182
 - range, typ, 179, 180
 - required, atrybut, 185
 - search, typ, 179, 180
 - spellcheck, atrybut, 184
 - step, atrybut, 185
 - stylowanie, 194
 - tel, typ, 179
 - time, typ, 182
 - url, typ, 179, 180
 - valueAsNumber(), 185
 - walidacja, 185, 188, 190, 193
 - week, typ, 182
 - nowości, 25, 26
 - pamięć podręczna, 261, 262
 - CHECKING, status, 266
 - DOWNLOADING, status, 266
 - IDLE, status, 266
 - OBSOLETE, status, 266
 - oncached, zdarzenie, 266
 - onchecking, zdarzenie, 266
 - ondownloading, zdarzenie, 266
 - onobsolete, zdarzenie, 266
 - onupdateready, zdarzenie, 266
 - UNCACHED, status, 266
 - UPDATEREADY, status, 266
 - wydajność, 268
 - priorytet użytkownika, 21
 - przeciągnij i upuść, 198, 199
 - cel upuszczenia, 198
 - dataTransfer, 199, 203
 - dla plików, 211, 212
 - drag, zdarzenie, 201
 - dragend, zdarzenie, 202
 - dragenter, zdarzenie, 201, 207
 - dragleave, zdarzenie, 201, 207
 - dragover, zdarzenie, 202
 - dragstart, zdarzenie, 201
 - drop, zdarzenie, 202
 - dropzone, atrybut, 211
 - modyfikacja wyglądu obiektu, 215
 - przepływ zdarzeń, 201
 - wsparcie przeciągania plików
 - przez przeglądarki, 212
 - źródło przeciągania, 198
 - przekaz strumieniowy, 91
 - separacja prezentacji i treści, 22
 - twórcy, 21
 - typy zawartości, 26
 - uniwersalny dostęp, 23
 - wątki robocze, 217
 - importScripts(), 219
 - inline, 219
 - komunikacja, 219
 - liczniki czasu, 223
 - obsługa błędów, 221
 - tworzenie, 219
 - wewnątrz innych wątków, 221
 - wsparcie przez przeglądarki, 218
 - współdzielone, 219
 - wykorzystanie, 217
 - zatrzymywanie, 221
 - Web Storage, 235, 236, 240
 - wsparcie w przeglądarkach, 236
 - zapisywanie i pobieranie danych, 237
 - zdarzenia, 242
 - wsparcie przez przeglądarki, 277
 - wtyczki, 23
 - zasady projektowe, 21
 - znaczniki semantyczne, 27
 - html5shiv, 32

I

- IDLE, status, 266
- IETF, 21
- iframe, element, 26
- image/jpeg, typ, 200
- image/png, typ, 200
- importScripts(), 219
- Indexed Database, 256
 - wsparcie przez przeglądarki, 257
- Inkscape, 87
- in-range, pseudoklasa, 194

Internet Explorer
 Chakra, 36
 Developers Tools, 35
 obsługa zdarzeń, 36
 prefiks, 31
 wsparcie Indexed Database, 257
 wsparcie Web SQL, 254
 wspierane kodeki, 91
 invalid, pseudoklasa, 194

J

JägerMonkey, 36
 JavaScript
 addEventListener(), 36
 alert(), 35
 canPlayType(), 92, 96, 97
 checkValidity(), 188
 close(), 165
 console.log, 35
 debugowanie, 34, 35
 dispatchEvent(), 36
 executeSQL(), 255
 getElementById(), 32
 getElementsByName(), 32
 getElementsByTagName(), 32
 importScripts(), 219
 JSON.parse(), 258
 JSON.stringify(), 258
 load(), 97
 logowanie, 34, 35
 Modernizr, biblioteka, 24
 odnajdywanie elementów, 32
 openDatabase(), 254
 pause(), 97
 play(), 97
 postMessage(), 131, 135
 bezpieczeństwo, 134
 struktury danych, 147
 wsparcie przez przeglądarki, 134
 send(), 144, 165
 silniki w przeglądarkach, 36
 stopPropagation(), 200
 transaction.executeSql(), 254
 valueAsNumber(), 185
 window.orientation, właściwość, 280
 wydajność, 36
 JIT, 36
 JPEG, 73
 JSON, 35, 134, 258
 parse(), 35, 258
 przechowywanie obiektów, 258
 stringify(), 35, 258

JSON with padding, *Patrz* JSONP
 JSON.parse(), 258
 JSON.stringify(), 258
 JSONP, 143

K

Kaazing WebSocket Gateway, 156
 kbd, element, 26
 kod semantyczny, 27
 kodeki
 audio, 90
 wideo, 90
 wsparcie, 90, 91
 komunikaty
 odbieranie, 135
 przesyłanie, 135
 kontenery wideo, 89
 formaty, 89
 koordynaty, 40
 krzywe, rysowanie, 51, 52

L

L, polecenie, 80
 latitude, atrybut, 116
 lineCap, właściwość, 49
 lineJoin, właściwość, 49
 lineTo(), 44, 45, 47
 linia ukośna, rysowanie, 43, 44
 list, atrybut, 184
 load(), 97
 loadstart, zdarzenie, 142
 localStorage, 237, 240
 porównanie z sessionStorage, 240
 longitude, atrybut, 116
 loop, atrybut, 97

M

M, polecenie, 80
 magazyn
 aktualizacja, 242
 lokalny, 240
 przeglądanie, 243
 manifestu, pliki, 264, 267
 CACHE, nagłówek, 265
 FALLBACK, nagłówek, 265, 266
 NETWORK, nagłówek, 265, 266
 mapa ciepła, 65
 mark, element, 26

mashup, 131
 Matroska, *Patrz* mkv
 max, atrybut, 184
 maximumAge, atrybut, 118
 measureText(), 60
 message, zdarzenie, 165
 MessageEvent, 135
 MIME, typy, 199, 200
 min, atrybut, 184
 mkv, 89
 mobilne, urządzenia
 dotyk, 282
 gesty, 280, 281
 zdarzenia, 281
 mod_pywebsocket, 156
 Modernizr, 24
 month, typ, 182
 moveTo(), 44, 45, 47
 -moz-, prefiks, 31
 Mozilla Firefox, *Patrz* Firefox
 mp4, 89
 MPEG-3, 90
 MPEG-4, *Patrz* mp4
 -ms-, prefiks, 31
 muted, atrybut, 97

N

nav, element, 27
 navigator.geolocation, 116
 navigator.onLine, właściwość, 264
 Netty, 156
 Nitro, 36
 node.js, 156
 numer, typ, 179

O

-o-, prefiks, 31
 obrazy, wstawianie do canvas, 53
 obsługa błędów
 geolokalizacja, 117
 wątki robocze, 221
 OBSOLETE, status, 266
 Ogg, *Patrz* ogv
 Ogg Theora, 90
 Ogg Vorbis, 90
 ogv, 89
 okna, współdzielenie danych, 258
 oncached, zdarzenie, 266
 onchecking, zdarzenie, 266
 ondownloading, zdarzenie, 266

ongesturechange, zdarzenie, 281
 ongestureend, zdarzenie, 281
 ongesturestart, zdarzenie, 281
 onobsolete, zdarzenie, 266
 ontouchcancel, zdarzenie, 282
 ontouchend, zdarzenie, 282
 ontouchmove, zdarzenie, 282
 ontouchstart, zdarzenie, 282
 onupdateready, zdarzenie, 266
 open, zdarzenie, 165
 openDatabase(), 254
 OpenGL, 278
 Opera
 Dragonfly, 35
 prefiks, 31
 wsparcie Indexed Database, 257
 wsparcie Web SQL, 254
 wspierane kodeki, 91
 option, element, 184
 optional, pseudoklasa, 194
 out-of-range, pseudoklasa, 194

P

P2P, 283
 pamięć podręczna, 261, 262, 266
 CHECKING, status, 266
 DOWNLOADING, status, 266
 IDLE, status, 266
 OBSOLETE, status, 266
 oncached, zdarzenie, 266
 onchecking, zdarzenie, 266
 ondownloading, zdarzenie, 266
 onobsolete, zdarzenie, 266
 onupdateready, zdarzenie, 266
 UNCACHED, status, 266
 UPDATEREADY, status, 266
 wydajność, 268
 parse(), 35
 pasek postępu, 181
 patternMismatch, 187
 pause(), 97
 paused, atrybut, 97
 peer-to-peer, sieci, 282
 PERMISSION_DENIED, 117
 placeholder, atrybut, 182
 play(), 97
 pliki manifestu, 264, 267
 CACHE, nagłówek, 265
 FALLBACK, nagłówek, 265, 266
 NETWORK, nagłówek, 265, 266
 PNG, 73

pochodzenie
 bezpieczeństwo, 133
 serializacja, 134
 POSITION_UNAVAILABLE, 117
 Poster, atrybut, 99
 postępu, zdarzenie, 142
 postMessage(), 131, 135
 bezpieczeństwo, 134
 struktury danych, 147
 wsparcie przez przeglądarki, 134
 prefiksy CSS3, 31
 -moz-, 31
 -ms-, 31
 -o-, 31
 -webkit-, 31
 progress, element, 181
 przechwytywanie zdarzeń, 200
 przeciągnij i upuść, 197, 198, 199
 cel upuszczenia, 198
 dataTransfer, 199, 203
 addElement(), 203
 clearData(), 203
 dropEffect, właściwość, 203
 effectAllowed, właściwość, 203
 files, właściwość, 203
 getData(), 203
 items, właściwość, 203
 setData(), 203
 setDragImage(), 203
 types, właściwość, 203
 dla plików, 211, 212
 drag, zdarzenie, 201
 dragend, zdarzenie, 202
 dragenter, zdarzenie, 201, 207
 dragleave, zdarzenie, 201, 207
 dragover, zdarzenie, 202
 dragstart, zdarzenie, 201
 drop, zdarzenie, 202
 dropzone, atrybut, 211
 historia, 197, 198
 modyfikacja wyglądu obiektu, 215
 przepływ zdarzeń, 201
 wsparcie przeciągania plików przez przeglądarki, 212
 źródło przeciągania, 198
 przeglądarki
 silniki JavaScript, 36
 wsparcie
 aplikacje lokalne, 263
 audio i wideo, 91, 92
 canvas, 42
 formularze, 178
 geolokalizacja, 112, 115
 HTML5, 277

 Indexed Database, 257
 postMessage(), 134
 przeciąganie plików, 212
 wątki robocze, 218
 Web SQL, 254
 Web Storage, 236
 WebSocket, 163, 164
 XMLHttpRequest Level 2, 142
 putImageData(), 63

Q

Q, polecenie, 80
 querySelector(), 33
 querySelectorAll(), 33
 Quirks, tryb, 26
 QUOTA_EXCEEDED_ERR, 241

R

range, typ, 179, 180
 rangeOverflow, 187
 rangeUnderflow, 187
 readystatechange, zdarzenie, 142
 rekomendacja
 kandydująca, 20
 proponowana, 20
 requestAnimationFrame(), 69, 70
 required, atrybut, 185
 required, pseudoklasa, 194
 rogi, zaokrąglone, 28
 rotate(), 58
 Rouget, Paul, 132

S

Safari
 Nitro, 36
 prefiks, 31
 SquirrelFish Extreme, 36
 Web Inspector, 35
 wsparcie Indexed Database, 257
 wsparcie Web SQL, 254
 wspierane kodeki, 91
 Scalable Vector Graphic, *Patrz* SVG
 script, element, 26
 search, typ, 179, 180
 section, element, 27
 selektory, 32, 34
 semantyczne, znaczniki, 27
 semantyczny, kod, 27

send(), 144, 165
 serializacja pochodzenia, 134
 sessionStorage, 237, 238, 240
 czas przechowywania danych, 238
 porównanie z localStorage, 240
 setInterval(), 68
 setTimeout(), 68
 shadery 3D, 278
 shadowBlur, właściwość, 61
 shadowColor, właściwość, 61
 shadowOffset, właściwość, 61
 shared Web Workers, *Patrz* wątki robocze
 współdzielone
 sieci peer-to-peer, 282
 silniki JavaScript
 Carakan, 36
 Chakra, 36
 JägerMonkey, 36
 Nitro, 36
 SquirrelFish Extreme, 36
 V8, 36
 SimpleHTTPServer, 264
 skalowalna grafika wektorowa, *Patrz* SVG
 small, element, 26
 source, element, 95
 type, atrybut, 95, 96
 speed, atrybut, 117
 spellcheck, atrybut, 184
 SquirrelFish Extreme, 36
 src, atrybut, 95
 startTime, atrybut, 97
 step, atrybut, 185
 stepMismatch, 187
 stopPropagation(), 200
 Storage, interfejs, 240
 clear(), 241
 getItem(), 241
 key(), 240
 length, atrybut, 240
 removeItem(), 241
 setItem(), 241
 StorageEvent, 242
 key, atrybut, 243
 oldValue, atrybut, 243
 storageArea, atrybut, 243
 url, atrybut, 243
 stringify(), 35
 stroke(), 47
 strokeRect(), 50
 strokeStyle, właściwość, 49
 strokeText(), 60
 style, element, 26

sub, element, 26
 sup, element, 26
 SVG, 40, 73, 74, 75
 a canvas, 40
 createElementNS(), 83
 defs, element, 78
 desenie, 79
 g, element, 78
 gradienty, 79
 grafika 2D, 76
 historia, 73
 interaktywna aplikacja, 83
 kształty, 77
 L, polecenie, 80
 M, polecenie, 80
 narzędzia, 87
 Q, polecenie, 80
 skalowanie, 75
 ścieżki, 80
 tekst, 74, 81
 transformacje, 78
 umieszczenie na stronie, 76
 use, element, 78
 Z, polecenie, 80
 SVG-edit, 87
 szerokość geograficzna, 110
 szklany panel, 68

Ś

ścieżki
 canvas, 47
 SVG, 80

T

Tamarin, 36
 TCPConnection, 151
 tekst
 canvas, 60, 74
 SVG, 74, 81
 tel, typ, 179
 text/cache-manifest, typ, 264
 text/plain, typ, 200
 text/x-age, typ, 200
 textAlign, właściwość, 60
 textarea, element, 26
 textBaseLine, właściwość, 60
 time, typ, 182
 TIMEOUT, 117
 timeout, atrybut, 118
 title, element, 26

toDataURL(), 64
 tooLong, 187
 transaction.executeSql(), 254
 transform: rotate(), 28
 transformacje elementu, 28
 translate(), 45
 triangulacja, 112
 tryb

- natychmiastowy, 75
- opóźniony, 75

 type, atrybut, 95, 96
 typeMismatch, 187
 typy zawartości, 26

- frazy, 26
- interaktywna, 26
- metadane, 26
- nagłówki, 26
- osadzona, 26
- sekcjonowanie, 26, 27
- układ, 26

U

UNCAHED, status, 266
 UPDATEREADY, status, 266
 url, typ, 179, 180
 urządzenia mobilne

- dotyk, 282
- gesty, 281
- zdarzenia, 281
- zmiana orientacji, 280

 use, element, 78

V

V8, 36
 valid, pseudoklasa, 194
 validationMessage, atrybut, 188
 ValidityState, 186
 valueAsNumber(), 185
 valueMissing, 186
 video, element, 26, 92

- autobuffer, atrybut, 97
- autoplay, atrybut, 96, 97
- controls, atrybut, 97
- currentSrc, atrybut, 97
- currentTime, atrybut, 97, 103
- duration, atrybut, 97
- Ended, atrybut, 97
- Error, atrybut, 97
- height, atrybut, 99

loop, atrybut, 97
 muted, atrybut, 97
 paused, atrybut, 97
 Poster, atrybut, 99
 startTime, atrybut, 97
 videoHeight, atrybut, 99
 videoWidth, atrybut, 99
 volume, atrybut, 97
 width, atrybut, 99
 videoHeight, atrybut, 99
 videoWidth, atrybut, 99
 volume, atrybut, 97
 Vorbis, 90
 VP8, 90

W

W3C, 19, 21
 WAI, 23
 walidacja formularzy, 185, 188, 190, 193
 watchPosition(), 120
 wątki robocze, 217

- importScripts(), 219
- inline, 219
- komunikacja, 219
- liczniki czasu, 223
- obsługa błędów, 221
- tworzenie, 219
- wewnątrz innych wątków, 221
- wsparcie przez przeglądarki, 218
- współdzielone, 219
- wykorzystanie, 217
- zatrzymywanie, 221

 Web Inspector, 35
 Web SQL, 254

- wsparcie przez przeglądarki, 254

 Web Storage, 235, 236, 240

- wsparcie w przeglądarkach, 236
- zapisywanie i pobieranie danych, 237
- zdarzenia, 242

 Web Video Text Tracks, *Patrz* WebVTT
 Web Workers, *Patrz* wątki robocze
 WebGL, 44, 278
 WebM, 91, 95
 WebSocket, 149, 151

- bufferedAmount, atrybut, 165
- close(), 165
- close, zdarzenie, 165
- error, zdarzenie, 165
- interfejs, 152, 153
- message, zdarzenie, 153, 165

WebSocket

- nawiązanie połączenia, 151
- obsługa zdarzeń, 165
- open, zdarzenie, 165
- połączenie z serwerem, 164
- ramki, 157
- send(), 153, 157, 165
- serwer, 156
- tworzenie aplikacji, 168
- uścisk dłoni, 151
- wsparcie przez przeglądarki, 163, 164
- wydajność, 153, 154, 155
- wysyłanie wiadomości, 165

WebSocketConnection, obiekt, 157

WebVTT, 93, 94

week, typ, 182

WHATWG, 19, 21

wideo

- kodeki, 90
- kontenery, 89
- odtworzenie po najechaniu kursorem, 107
- restrykcje, 91
- wsparcie przez przeglądarki, 91, 92
- wtyczki, 92

width, atrybut, 99

Wi-Fi, 112

willValidate, atrybut, 188

window.applicationCache, 266

- status, właściwość, 266

window.JSON, 35

window.orientation, właściwość, 280

World Wide Web Consortium, *Patrz* W3C

wtyczki, 23

wycieki danych, 238, 239

wystąpienia wiadomości, zdarzenie, 132

- data, właściwość, 132
- origin, właściwość, 132

X

XForms, 177

X-Frame-Options, 148

XHR, *Patrz* XMLHttpRequest

XHTML5, 22

XMLHttpRequest, 140

XMLHttpRequest Level 2, 140, 142

- dane binarne, 144
- send(), 144
- wsparcie przez przeglądarki, 142
- zdarzenia, 142

Z

Z, polecenie, 80

zaokrąglone rogi, 28

zawartości, typy, 26

- frazy, 26

interaktywna, 26

metadane, 26

nagłówki, 26

osadzona, 26

sekcjonowanie, 26, 27

układ, 26

zdarzenia

close, 165

drag, 201

dragend, 202

dragenter, 201, 207

dragleave, 201, 207

dragover, 202

dragstart, 201

drop, 202

error, 165

gestów, 281

loadstart, 142

message, 165

oncached, 266

onchecking, 266

on downloading, 266

ongesturechange, 281

ongestureend, 281

ongesturestart, 281

onobsolete, 266

ontouchcancel, 282

ontouchend, 282

ontouchmove, 282

ontouchstart, 282

onupdateready, 266

open, 165

pamięci podręcznej, 266

postępu, 142

readystatechange, 142

wystąpienia wiadomości, 132

- data, właściwość, 132

- origin, właściwość, 132

wywoływane przez magazyn, 242

związane z dotykiem, 282

znaczniki semantyczne, 27

Ź

źródło przeciągania, 198

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



- 1. ZAREJESTRUJ SIĘ**
- 2. PREZENTUJ KSIĄZKI**
- 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**

HTML5. Zaawansowane programowanie

HTML5 szturmem zdobywa rynek stron (a w zasadzie już aplikacji) WWW. Dziś ten odświeżony, bogaty w nowe możliwości język ma ogromny potencjał. Dlatego warto poświęcić mu trochę czasu i zacząć tworzyć jeszcze bardziej zaawansowane strony. To na pewno się opłaci!

Dzięki tej książce poznasz najbardziej zaawansowane możliwości HTML5. Nauczysz się wykorzystywać mechanizm geolokalizacji i będziesz w stanie dostarczać użytkownikom spersonalizowane treści w zależności od ich miejsca pobytu. Ponadto przekonasz się, jaki potencjał kryje element *canvas*. Za jego pomocą stworzysz niezwykle atrakcyjny interfejs bez konieczności stosowania dodatków do przeglądarki. W dalszych rozdziałach sprawdzisz, jak prosta może być komunikacja przeglądarki z serwerem za pośrednictwem WebSockets oraz jak zachować funkcjonalność Twojej aplikacji w przypadku braku połączenia z siecią. Dzięki tej książce zdążysz na czas opanować nowości z HTML5 i wykorzystasz je w Twoim kolejnym projekcie!

Sięgnij po tę książkę i:

- dostarasz użytkownikowi odpowiednią zawartość strony w zależności od jego lokalizacji
- przechowuj niezbędne dane w bazie danych przeglądarki
- twórz atrakcyjny interfejs dzięki elementowi *canvas*
- zastosuj najnowsze technologie!

**WYKORZYSTAJ MOC NAJNOWSZYCH TECHNOLOGII
WCHODZĄCYCH W SKŁAD HTML5!**

Apress

Nr katalogowy: 10476

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

Zamówienia telefoniczne:
0 801 339900
0 601 339900

helion.pl
księgarnia
internetowa

Sprawdź najnowsze promocje:
• <http://helion.pl/promocje>
Książki na chętniej czytane:
• <http://helion.pl/bestsellery>
Zamów informacje o nowościach:
• <http://helion.pl/novowci>

Helion

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



KOD KORZYŚCI

Cena 59,00 zł

ISBN 978-83-246-4809-2



9 788324 648092