Poznaj możliwości HTML5!

HTML5

nieoficjalny podręcznik

Matthew MacDonald

O'REILLY®

Helion

Tytuł oryginału: HTML5: The Missing Manual

Tłumaczenie: Maciej Reszotnik

ISBN: 978-83-246-3948-9

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/htm5np
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

4

ROZDZIAŁ

Udoskonalone formularze

Formularze HTML są zbiorem elementów, których przeznaczeniem jest pobranie informacji od osób odwiedzających stronę. Są wśród nich pola tekstowe, w które można wpisać treści, rozwijane listy, z których mogą wybrać opcję, pola wyboru, które można zaznaczać i usuwać znaczenie itp. Istnieje wiele sposobów na wykorzystanie formularzy HTML, a jeśli przebywałeś w sieci przez więcej niż tydzień, pewnie wiesz, że pojawiają się praktycznie wszędzie — od strony z informacjami o giełdzie, po stronę rejestracji skrzynki e-mail.

Formularze HTML są z nami od czasu wyklarowania się tego języka i od zeszłego stulecie nie zmieniły się ani trochę pomimo wielu prób. Twórcy standardów internetowych przez lata opracowywali ich następcę, określanego mianem XForms, który ostatecznie okazał się takim samym niewypałem jak XHTML 2 (strona 27). Choć XForm dość elegancko rozwiązywał pewne problemy z oryginalnym pomysłem, to jednak tworzył też kolejne. Kod XForms był rozwlekły i zakładał, że projektant strony jest dobrze obeznany z językiem XML. Największą wadą było to, że XForms nie był w żadnym względzie kompatybilny z HTML, co oznaczało tyle, iż twórcy stron musieliby zamknąć oczy i „przeskoczyć" na nowy standard, mając nadzieję, że jakoś to będzie. Co można było przewidzieć, większość webdeweloperów nie zawracała sobie głowy implementacją XForms — był to standard zbyt złożony, by nadawał się do powszechnego wykorzystania — zatem twórcy stron nigdy nie dokonali wymaganego skoku.

W HTML5 obrano inne podejście. Wprowadzono wiele usprawnień do już istniejącego modelu formularzy, dzięki czemu ulepszone ich wersje będą działać na starszych przeglądarkach, choć bez specjalnych wodotrysków (co jest dobrą wieścią, ponieważ z przeglądarek Microsoftu dopiero Internet Explorer 10 zapewni pełną obsługę nowych własności formularzy). W HTML5 dodano również opcje, z których twórcy stron korzystają już dziś. Różnica polega na tym, że HTML5 sprawia, iż są łatwo dostępne, i nie wymaga kodu lub narzędzi JavaScriptu dostarczanych przez zewnętrzne firmy.

W tym rozdziale zapoznasz się z nowymi własnościami formularzy HTML5. Dowiesz się, które z nich są powszechnie obsługiwane, a które obecnie nie cieszą się żadnym wsparciem. Przyjrzysz się również właściwości, która — choć nie jest częścią standardu HTML — wzbogaca warstwę interakcji przez umieszczenie złożonego edytora HTML na stronie.

Formularze

Najprawdopodobniej masz już doświadczenie w tworzeniu formularzy. Jeśli jednak nie jesteś pewien szczegółów, krótka powtórka odświeży Ci pamięć.

Formularz sieciowy to zbiór pól tekstowych, list, przycisków i innych komponentów, przy użyciu których użytkownik wprowadza pewne informacje na stronie. W sieci formularze występują na każdym kroku — pozwalają na rejestrację skrzynki e-mail, zakup produktów, wykonywanie transakcji bankowych itp. Najprostsze formularze, wykorzystywane w wyszukiwarkach internetowych, składają się z pojedynczego pola tekstowego, tak jak na rysunku 4.1.

Rysunek 4.1. *Prezentująca się spartańsko strona główna wyszukiwarki Google zbudowana jest z prostego formularza HTML. Korzysta się z niego, tak jak z każdego innego formularza — należy wprowadzić informacje (w tym wypadku poszukiwane słowa klucze) i kliknąć przycisk, aby przesłać je na serwer*

Wszystkie formularze sieciowe działają na tej samej zasadzie. Użytkownik podaje informacje, po czym klika przycisk. W odpowiedzi strona pobiera wprowadzone dane i wysyła je na serwer. Na serwerze jakaś aplikacja przetwarza te dane i wykonuje kolejny krok. Serwerowy program może zainicjować komunikację z bazą danych (aby odczytać bądź zapisać jakieś informacje) przed wysłaniem do użytkownika nowej strony.

PRZYSPIESZAMY

Obchodzenie mechanizmu przesyłania danych przy użyciu JavaScriptu

Warto wspomnieć, że formularze nie są jedynym rozwiązaniem, które umożliwia przesyłanie wprowadzonych danych na serwer (były inne dawno, dawno temu). Aktualnie chytry twórca stron może komunikować się z serwerem za pomocą napisanego w kodzie JavaScript obiektu XMLHttpRequest (strona 325). Wyszukiwarka Google korzysta z niego na dwa sposoby. Po pierwsze, żeby wyświetlać sugerowane hasła wyszukiwania w liście pod polem tekstowym; po drugie, by generować wyniki wyszukiwania równolegle z wpisywaniem nowych treści. Ta ostatni własność działa po włączeniu opcji wyszukiwania dynamicznego (*http://www.google.pl/instant/*).

Może Ci się wydać, że dzięki JavaScriptowi uda się w ogóle pominąć krok bezpośredniego wysyłania danych, tak jak ma to miejsce z wyszukiwaniem dynamicznym Google. Jednakże, choć warto zaoferować taką możliwość, nie może to być jedyna opcja. Wynika to z prostego faktu, że zaprezentowane podejście w języku JavaScript nie jest doskonałe (może powodować błędy przy wolniejszych łączach), oraz tego, iż wciąż są osoby, których przeglądarki nie obsługują tego języka lub opcja obsługi jest w nich wyłączona.

Oczywiście, nie ma nic złego w stronach, których formularze nie przesyłają nigdzie danych. Z pewnością widziałeś strony, na których można dokonywać prostych obliczeń (np. kalkulatory odsetek z kredytów hipotecznych). Obsługa serwera nie jest w nich wymagana, gdyż dołączony do tych dokumentów kod JavaScript może wykonać kalkulacje i wyświetlić wyniki.

Z perspektywy HTML5 nie ma znaczenia, czy prześlesz swój formularz na serwer w zwykły sposób, użyjesz danych z niego do obliczeń lokalnych, czy wyślesz je z pomocą obiektu XMLHttpRequest. W każdym z tych przypadków swój formularz skonstruujesz, posługując się standardowymi elementami HTML

Cały szkopuł tkwi w tym, że jest wiele sposobów na skonstruowanie programu działającego po stronie serwera (aplikacji przetwarzającej odczytane z formularza informacje). Niektórzy deweloperzy używają prostych skryptów, które pozwalają im na manipulowanie surowymi danymi, podczas gdy inni korzystają z dobrodziejstw wysokopoziomowych modeli, które opakowują informacje w wygodne obiekty. Niezależnie od podejścia, procedura sprowadza się do tego samego — pobrania danych, ich przetworzenia i zwrócenia rezultatów obliczeń na stronie.

Uwaga: W książce tej nie faworyzuje się żadnego z dostępnych narzędzi do programowania po stronie serwera. W zasadzie nie ma to większego znaczenia, ponieważ każde z rozwiązań korzysta z tych samych elementów formularza, którego dotyczą reguły HTML5.

Modernizowanie tradycyjnego formularza HTML

Najlepszą metodą nauki konstruowania formularzy w HTML5 jest ulepszenie już istniejącego, napisanego w standardowy sposób. Na rysunku 4.2 widać przykład, który zostanie omówiony.

Kod jest prosty jak konstrukcja cepa. Jeśli budowałeś wcześniej formularze, nie ujrzysz tu niczego nowego. Cały formularz został wyznaczony parą znaczników `<form>`.

```
<form id="zooKeeperForm" action="processApplication.cgi">
  <p><i>Proszę wypełnić ten formularz. Obowiązkowe pola oznaczono znakiem
</i><em>*</em></p>
...
```

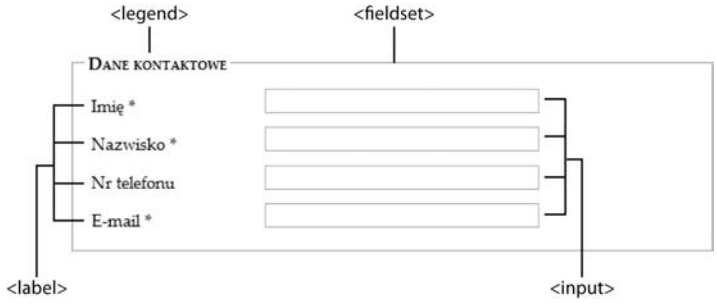
Rysunek 4.2. *Jeśli przemierzyłeś Internet wzdłuż i wszerz, formularz, taki jak ten, który pobiera od użytkownika strony podstawowe dane, nie powinien być dla Ciebie niczym nowym*

Element `<form>` zawiera wszystkie widgety formularza (zwane też **kontrolkami** lub **polami**). Informuje też przeglądarkę (z wykorzystaniem adresu URL w atrybucie `action`), gdzie ma wysłać zebrane dane. Jeśli całość operacji ma być wykonana przez kod JavaScript po stronie klienta, możesz nadać mu wartość #.

Uwaga: HTML5 dodaje mechanizm, dzięki któremu wolno zamieszczać kontrolki formularza poza jego obrębem. Cała sztuczka polega na użyciu atrybutu `form` (np. `form="zooKeeperForm"`) z wartością atrybutu `id`, który nadano znacznikowi `<form>` (np. `<form id="zooKeeperForm">…</form>`). Niestety, przeglądarki, które nie obsługują tej własności, pominą tak oznaczone dane w trakcie przesyłania formularza, co oznacza, że na razie można zapomnieć o używaniu tej właściwości.

Dobrze zaprojektowany formularz, taki jak ten dla pracownika zoo, dzieli się na logiczne części — co zapewnia użycie znacznika `<fieldset>`. Każda z tych części powinna posiadać tytuł, który wpisuje się w element `<legend>`. Oto obszar komponentu `<fieldset>` dla sekcji danych kontaktowych (rysunek 4.3).

```
<fieldset>
  <legend>Dane kontaktowe</legend>
    <label for="first name">Imię <em>*</em></label>
    <input id="first name"><br>
```



Rysunek 4.3. *Komponent <fieldset> pobiera cztery rodzaje informacji: imię, nazwisko, numer telefonu i adres e-mail. Dla każdej informacji podano podpis (wykorzystując element <label>) oraz kontrolkę, która pozwoli na zebranie danych (do tego celu nadają się elementy <input>, <textarea> i <select>)*

```
      <label for="family name">Nazwisko <em>*</em></label>
      <input id="family name"><br>
      <label for="telephone">Nr telefonu</label>
      <input id="telephone"><br>
      <label for="email">E-mail <em>*</em></label>
      <input id="email"><br>
  </fieldset>
```

Jak we wszystkich formularzach, i tu najważniejszymi elementami są znaczniki `<input>`, które pobierają tekst, tworzą pola wyborów, przyciski opcji itp. Oprócz nich w formularzach używa się komponentu `<textarea>`, który pozwala na wpisywanie wielu linii tekstu, i znacznika `<select>` tworzącego listy. Jeśli czujesz, że przydałaby Ci się powtórka, rzuć okiem na tabelę 4.1.

Oto pozostała część formularza dla opiekunów zwierząt w zoo z kilkoma nowymi szczegółami (listą ze znacznikiem `<select>`, polami wyboru i przyciskiem *Wyślij formularz*).

```
<fieldset>
    <legend>Szczegółowe dane osobowe</legend>
      <label for="age">Wiek <em>*</em></label>
      <input id="age"><br>
      <label for="gender">Płeć</label>
      <select id="gender">
        <option value="female">Kobieta</option>
        <option value="male">Mężczyzna</option>
      </select><br>
      <label for="comments">Kiedy zdałaś/łeś sobie sprawę, że opiekun
      ↪zwierząt to zawód dla Ciebie?</label>
      <textarea id="comments"></textarea>
  </fieldset>

<fieldset>
    <legend>Zaznacz swoje ulubione zwierzęta</legend>
    <label for="zebra"><input id="zebra" type="checkbox"> zebra</label>
    <label for="cat"><input id="cat" type="checkbox"> kot</label>
    <label for="anaconda"><input id="anaconda" type="checkbox">
    ↪anakonda</label>
    <label for="human"><input id="human" type="checkbox"> człowiek</label>
    <label for="elephant"><input id="elephant" type="checkbox"> słoń</label>
    <label for="wildebeest"><input id="wildebeest" type="checkbox">
    ↪antylopa</label>
    <label for="pigeon"><input id="pigeon" type="checkbox"> gołąb</label>
    <label for="crab"><input id="crab" type="checkbox"> krab</label>
  </fieldset>
  <p><input type="submit" value="Wyślij formularz"></p>
```

Tabela 4.1. *Kontrolki formularza*

Nazwa kontrolki	Znacznik HTML	Opis
Pole tekstowe (jeden wiersz)	`<input type="text">` `<input type="password">`	Definiuje pole tekstowe, w którym użytkownik może wpisywać informacje. Komponent `<input type="password">`, który służy do wpisywania haseł, nie wyświetla wpisanego tekstu. Zamiast liter użytkownik ujrzy w nim gwiazdki (*) lub kropki (·).
Wieloliniowe pole tekstowe	`<textarea>...</textarea>`	Definiuje duże pole tekstowe, w które można wpisać wiele linijek tekstu.
Pole wyboru	`<input type="checkbox">`	Definiuje pole wyboru, które można zaznaczać i usuwać z niego zaznaczenie.
Przycisk radiowy	`<input type="radio">`	Definiuje przycisk radiowy (kółka, które można zaznaczać i usuwać z nich zaznaczenie). Zwykle przyciski radiowe zbiera się w grupy o takiej samej wartości atrybutu `name`.

Przycisk	`<input type="submit">`
	`<input type="image">`
	`<input type="reset">`
	`<input type="button">`
		Definiuje standardowy przycisk. Przycisk typu `submit` powoduje odczytanie danych z formularza i ich przesłanie pod podany adres URL. Przycisk typu `image` działa tak samo, lecz wyświetla podany w nim obraz zamiast standardowej grafiki przycisku. Przycisk typu `reset` czyści wartości wszystkich kontrolek w formularzu. Z kolei przycisk typu `button` nie robi nic, chyba że dodasz do niego kod JavaScript.

Lista	`<select>...</select>`	Definiuje listę, z której użytkownik wybiera jedną lub wiele opcji. Każdą opcję na liście należy zawrzeć w elemencie `<option>`.

Stronę oraz załączony do niej prosty arkusz stylów znajdziesz wśród plików pobranych ze strony *helion.pl/ksiazki/htm5np.htm*. Wersję formularza napisaną w tradycyjnym HTML nazwano *ZookeperForm_Original.html*. Jeżeli interesują Cię bardziej wprowadzone w HTML5 modyfikacje, rzuć okiem na plik *ZookeperForm_Revised.html*.

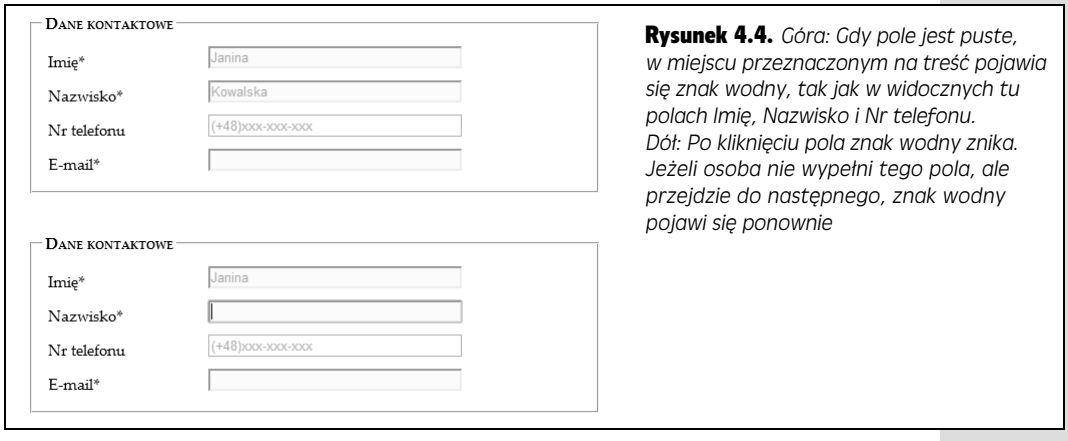
Uwaga: Jednym z podstawowych ograniczeń formularzy HTML jest to, że nie można w prosty sposób zmienić wyglądu generowanych przez przeglądarkę kontrolek. Nie da się np. zmienić zwykłego biało-szarego pola wyboru na duży czerwony znacznik. Możesz, oczywiście, utworzyć przy użyciu JavaScriptu inny element, który będzie zachowywał się tak jak pole wyboru, tzn. zmieniał wygląd po kliknięciu, przechodząc z jednego stanu w drugi i z powrotem.

HTML5 zachowuje to ograniczenie i nakłada je na wszystkie nowe kontrolki, o których dowiesz się więcej w dalszej części tego rozdziału. Oznacza to, że działanie formularzy może się nie spodobać osobom, które wymagają od strony wysublimowanych komponentów i pełnej kontroli nad jej wyglądem.

Zapoznałeś się już z formularzem — czas go zmodernizować przy użyciu HTML5. Zaczniesz od rzeczy prostych, czyli atrybutów `placeholder` i `autofocus`.

Znak wodny — dodawanie wskazówek

Po załadowaniu strony formularze są zwykle puste. Kolumna pustych pól tekstowych może jednak onieśmielać szczególnie wtedy, gdy nie wiadomo, co w nich ma być. Dlatego też zwykłą praktyką jest wypełnianie ich przykładowym tekstem przy użyciu atrybutu `placeholder`. Taki przykładowy tekst nazywa się **znakiem wodnym**, gdyż nadaje się mu jasnoszary kolor, aby odróżnić go od właściwych danych wprowadzonych przez użytkownika. Na rysunku 4.4 zaprezentowano znak wodny w działaniu.



Rysunek 4.4. *Góra: Gdy pole jest puste, w miejscu przeznaczonym na treść pojawia się znak wodny, tak jak w widocznych tu polach Imię, Nazwisko i Nr telefonu. Dół: Po kliknięciu pola znak wodny znika. Jeżeli osoba nie wypełni tego pola, ale przejdzie do następnego, znak wodny pojawi się ponownie*

Aby utworzyć znak wodny, po prostu wewnątrz znacznika dodaj atrybut `placeholder`.

```
<label for="family name">Nazwisko <em>*</em></label>
<input id="family name" placeholder="Kowalski"><br>
<label for="telephone">Nr telefonu</label>
<input id="telephone" placeholder="+48-xxx-xxx-xxx"><br>
<label for="email">E-mail <em>*</em></label>
<input id="email"><br>
```

Przeglądarki, które nie rozpoznają tego atrybutu, po prostu go zignorują (wśród nich jest, oczywiście, Internet Explorer). Na szczęście, nawet gdy tak się stanie, nie dojdzie do żadnej tragedii. W końcu znaki wodne są jedynie dodatkiem, który w żadnej mierze nie wpływa na działanie formularza. Jeśli jednak Ci to przeszkadza, na stronie *http://tinyurl.com/polyfills* pełno jest napisanych w JavaScripcie łatek, które uaktualnią nawet IE.

Obecnie nie ma sposobu na zmianę wyglądu tekstu w znaku wodnym (taką jak np. nadanie kursywy lub innego koloru). W przyszłości twórcy przeglądarek dodadzą obsługę arkusza stylów — gdy czytasz te słowa, być może już ją wprowadzili. Teraz w tym celu można eksperymentować ze specyficznymi dla każdej przeglądarki pseudoklasami (np. `-webkit-input-placeholder` i `-moz-placeholder`), ale najlepiej nie zawracać sobie tym głowy (wyjaśnienie działania pseudoklas znajdziesz na stronie 382).

PRZYSPIESZAMY

Prawidłowe znaki wodne

Nie musisz umieszczać znaków wodnych w każdym polu tekstowym. Używaj ich do wyjaśnienia potencjalnej dwuznaczności. Przykładowo nie trzeba wyjaśniać, co należy wpisać w pole *Nazwisko*, ale użycie słowa Godność mogłoby nie być zbyt oczywiste. Znak wodny daje jasno do zrozumienia, że jest to miejsce na nazwisko.

Często znakiem wodnym jest przykładowa wartość, czyli coś, co może być podstawową wartością. W wyszukiwarce przepisów Google (*http://www.google.com/landing/recipes*) wartością domyślną jest „chicken pasta" (makaron z kurczakiem), przez co jasne jest, że w pole należy wpisać nazwę potrawy, a nie listę składników lub nazwisko kucharza, który wymyślił przepis.

Znaków wodnych używa się czasem, żeby wskazać, jak wprowadzona wartość powinna być sformatowana. Pole *Nr telefonu* z rysunku 4.4 jest tego dobrym przykładem — wskazuje, że numery telefonu powinny składać się z numeru kodu kraju i dziewięciocyfrowego numeru telefonu. Taki znak wodny nie oznacza, że inaczej sformatowane dane nie zostaną przyjęte, lecz sugeruje prawidłowe rozwiązanie, które może pomóc niepewnym użytkownikom.

Nie powinieneś używać znaku wodnego do dwóch czynności. Po pierwsze, nie próbuj wcisnąć do pola opisów lub instrukcji. Załóżmy, że zdefiniowałeś pole, w które należy wpisać kod bezpieczeństwa karty kredytowej. W takim przypadku tekst: „Wpisz trzy cyfry wytypowane na odwrocie Twojej karty" nie nadaje się na znak wodny. Zamiast niego warto dodać podpis pod polem lub wykorzystać atrybut `title`, aby za każdym razem, gdy użytkownik najedzie na nie myszą, pojawiało się małe okienko z instrukcją.

```
<label for="promoCode">Kod
↪promocyjny</label>
<input id="promoCode" placeholder="QRBOO1"
↪title="Twój kod promocyjny zaczyna się
↪od trzech liter, po których występują
↪trzy cyfry">
```

Po drugie, nie powinieneś dodawać do znaków wodnych specjalnych znaków, aby je odróżnić od właściwego, wprowadzonego do pól tekstu. Przykładowo na niektórych stronach znak wodny zapisywany jest w nawiasach, np. *[Jan Kowalski]* zamiast *Jan Kowalski*. Nawiasy kwadratowe mają sugerować, że znak wodny jest jedynie przykładem. Taka konwencja jest zbędna i prowadzi do większego zamieszania.

Natomiast możesz bez przeszkód wykorzystać lepiej znany pseudoklasyk `focus`, by zmienić wygląd pola tekstowego, po tym jak zostanie wybrane. Załóżmy, że chcesz, żeby miało ciemniejszy kolor tła, tak by odróżniało się od reszty. Poniższa deklaracja CSS sprawi, że tak się stanie:

```
input:focus {
  background: #eaeaea;
}
```

Dobry punkt zaczepienia: właściwość focus

Zwykle, zaraz po załadowaniu formularza jego użytkownik chce po prostu zacząć go wypełniać. Niestety, nie może tego zrobić — przynajmniej dopóty, dopóki nie zaznaczy kliknięciem jednej z kontrolek lub nie przejdzie do niej przy użyciu klawisza *Tab*, w ten sposób ją aktywując.

Dowolną kontrolkę można aktywować domyślnie, wywołując w języku JavaScript metodę `focus()` na wybranym elemencie `<input>`. Wymaga to napisania całej linijki kodu i w niektórych przypadkach może doprowadzić do pojawienia się dziwnych błędów. Jeśli np. użytkownik kliknie kontrolkę zaraz po załadowaniu strony i zacznie ją wypełniać, jeszcze zanim metoda `focus()` zostanie wywołana, w trakcie pisania może zostać przeniesiony na pole, którym nie jest zainteresowany. Rozwią-

zaniem tego problemu byłoby zapewnienie przeglądarkom zdolności kontrolowania aktywacji komponentów. W ten sposób mogłaby ona aktywować jedną kontrolkę, pod warunkiem że inna nie została wcześniej wybrana przez użytkownika.

Właśnie ten pomysł przyczynił się do powstania atrybutu `autofocus`, który można dołączyć do elementów `<input>` i `<textarea>`, tak jak na poniższym przykładzie:

```
<label for="last name">Nazwisko <em>*</em></label>
<input id="name" placeholder="Janina Kowalska" autofocus><br>
```

Atrybut `autofocus` cieszy się identycznym poziomem wsparcia jak parametr `placeholder`, co oznacza, że wszystkie przeglądarki, z wyjątkiem Internet Explorera, go rozpoznają. Na szczęście i w tym przypadku łatwo załatać dziurę. Korzystając ze skryptu Modernizr (strona 52), możesz wykryć, czy przeglądarka obsługuje ten atrybut, i uruchomić swój własny kod automatycznego aktywowania elementu. Możesz też użyć gotowego wypełnienia, które doda odpowiednią obsługę (*http://tinyurl.com/polyfills*). Jednak szkoda zachodu na zabawę z tak mało znaczącą opcją, chyba że przy okazji dodasz obsługę innych opcji, takich jak omawiany niżej system walidacji.

Walidacja: wykrywanie błędów

Podstawową funkcją pól formularza jest pobranie od użytkownika strony informacji. Niecierpliwi lub zdezorientowani użytkownicy mogą pominąć ważne pole, wprowadzić złą treść lub kliknąć niewłaściwy przycisk. Wynikiem czegoś takiego jest otrzymanie mnóstwa bezużytecznych danych.

Każda porządna witryna powinna być wyposażona w mechanizm **walidacji**, odpowiedzialny za wykrywanie błędów (lub, w najlepszym wypadku, za zapobieganie ich powstawaniu). Przez lata deweloperzy tworzyli w tym celu własne procedury JavaScriptu lub wykorzystywali gotowe biblioteki. Oba rozwiązanie sprawdzają się wyśmienicie. Ze drugiej jednak strony, walidacja jest na tyle powszechnie wykorzystywanym mechanizmem (praktycznie każdy formularz wymaga zweryfikowania poprawności) i na tyle trudnym do zapewnienia (nikt nie chce pisać od podstaw podobnego kodu dla różnych formularzy, nie mówiąc już o jego *testowaniu*), że powinna istnieć inna metoda jego implementacji.

Twórcy HTML5 dostrzegli potencjalną niszę i wymyślili sposób na odebranie projektantom części odpowiedzialności za walidację i przerzucenie jej na przeglądarkę. Wynaleźli system walidacji *po stronie klienta* (szczegóły w ramce na stronie 126), który pozwala osadzić reguły sprawdzania błędów wewnątrz dowolnego znacznika `<input>`. Co najlepsze — system ten jest prosty w obsłudze — wystarczy posłużyć się właściwym atrybutem.

Proces walidacji w HTML5, krok po kroku

Podstawową rzeczą, o której należy pamiętać przy tworzeniu mechanizmu walidacji formularza, jest wskazanie, których elementów dotyczy, lecz nie musisz przy tym kłopotać się zbędnymi detalami. Przypomina to odrobinę awans do pracy w zarządzie, tyle że bez podwyżki.

PRZYSPIESZAMY

Dwie strony walidacji

Przez całe lata zmyślni twórcy witryn atakowali problem walidacji z różnych stron. Z biegiem czasu wyodrębniono najlepszą praktykę. W zwykłym formularzu wykorzystano dwie metody sprawdzania poprawności.

- **Walidacja po stronie klienta.** Ta forma walidacji zachodzi w przeglądarce użytkownika, jeszcze zanim prześle on formularz. Celem jest ułatwienie życia użytkownikom. Zamiast zwracać informacje o błędach po wypełnieniu wszystkich kontrolek i kliknięciu przycisku *Wyślij*, lepiej identyfikować je zaraz po wpisaniu złej wartości. W rezultacie możesz wywołać pomocną informację o przyczynie omyłki we właściwym miejscu, pozwalając użytkownikowi szybko naprawić błąd przed wysłaniem danych na serwer.
- **Walidacja po stronie serwera.** Dochodzi do niej już po przesłaniu formularza na serwer. W tym momencie zadaniem kodu serwera jest sprawdzenie szczegółów i upewnienie się, że wszystko jest w najlepszym porządku. Pamiętaj, niezależnie od wyniku walidacji po stronie klienta walidacja serwerowa jest po prostu konieczna. To najważniejsza linia obrony w walce z hakerami, którzy mogą umyślnie manipulować przesyłanymi na serwer danymi formularza. Jeśli walidacja po stronie serwera wykryje błędne lub podejrzane wpisy, powinna zwrócić stronę z wiadomością o błędzie.

Jak łatwo zauważyć, walidacja kliencka (której przykładem jest omawiany mechanizm w HTML5) ma ułatwić pracę użytkownikom formularza, podczas gdy walidacja po stronie serwera dba o poprawność. Musisz pamiętać, że obie formy walidacji są potrzebne w niemalże każdym przypadku — chyba że chodzi o bardzo prosty formularz, w którym trudno popełnić błąd lub w którym błędy nie stanowią problemu.

Załóżmy, że na Twoim formularzu znajduje się pole, które nie powinno pozostać puste — użytkownik musi wpisać w nie jakieś dane. W HTML5 wymóg ten zadeklarujesz, wpisując w znaczniku atrybut `required`:

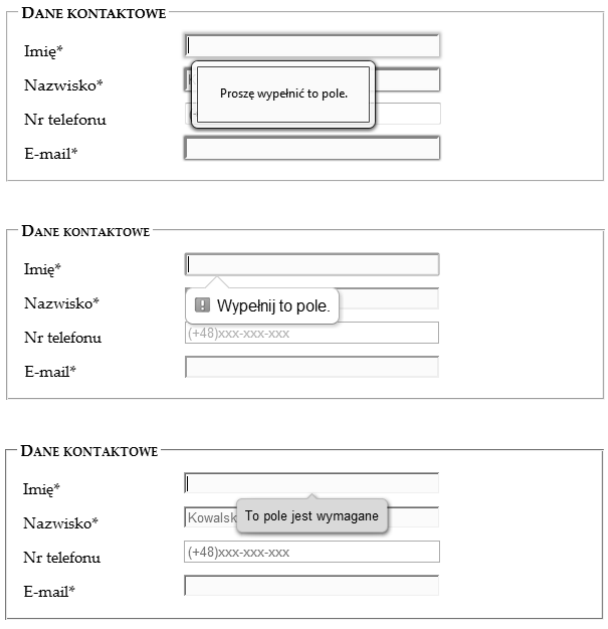
```
<label for="name">Nazwisko <em>*</em></label>
<input id="name" placeholder="Janina Kowalska" autofocus required><br>
```

Na pierwszy rzut oka to, że dane pole jest wymagane, nie jest w ogóle widoczne. Dlatego też może warto użyć wizualnych wskazówek w rodzaju innego koloru obramowania kontrolki bądź umieszczenia gwiazdki obok wymaganego pola (tak jak na stronie formularza dla opiekunów zwierząt).

Mechanizm walidacji jest uruchamiany po kliknięciu przez użytkownika przycisku wysyłania formularza. Jak widać na rysunku 4.5, jeśli przeglądarka obsługująca formularz HTML5 zauważy, że wymagane pole jest puste, przechwyci formularz przed wysłaniem i wyświetli wyskakujące wiadomości nad błędnie wypełnionymi kontrolkami.

Czytając kolejne punkty, przekonasz się, że różne atrybuty pozwalają na zdefiniowanie odmiennych reguł sprawdzania błędów. W jednym widgecie można określić wiele reguł, a jedną regułę można nanieść na wiele kontrolek (w tym elementy `<input>` i `<textarea>`). Wszystkie warunki walidacji muszą zostać spełnione, nim formularz zostanie przesłany na serwer.

Tu pojawia się zasadnicze pytanie. Co zrobić, kiedy wprowadzone dane łamią więcej niż jedną regułę? Co się stanie, gdy np. użytkownik zapomni wypełnić kilka pól?



Rysunek 4.5. *Oto to samo wymagane pole tekstowe w trzech przeglądarkach: Firefoksie (u góry), Chrome (środek) i Operze (u dołu). Przeglądarki wybierają sposób informowania użytkowników o błędzie, lecz wszystkie korzystają z wyskakujących wiadomości, które przypominają mechanizm podpowiedzi w systemie Windows. Niestety, nie ma możliwości zmiany wyglądu tych miniokien czy łatwej modyfikacji treści wiadomości — przynajmniej na razie*

Otóż, do momentu gdy użytkownik kliknie przycisk wysyłający formularz, nie stanie się nic. Potem jednak przeglądarka zacznie analizować zawartość pól — od góry w dół. Gdy odnajdzie pierwszą źle użytą kontrolkę, zaprzestaje dalszej analizy i wyświetla wyskakującą wiadomość o błędzie (jeśli dodatkowo po załadowaniu strony kontrolka z niepoprawnymi danymi będzie niewidoczna, przeglądarka przewinie dokument, by znalazła się u góry jej okna). Kiedy użytkownik naprawi błąd i kliknie przycisk przesyłania, przeglądarka wykryje i oznaczy kolejną niepoprawną wartość lub prześle dane na serwer.

Uwaga: Przeglądarki przeprowadzają walidację dopiero po kliknięciu przycisku wysyłania. Dzięki temu system walidacji działa wydajnie i prosto, więc sprawuje się wyśmienicie w każdych warunkach.

Niektórzy twórcy stron wolą jednak informować użytkownika o błędzie od razu po tym, jak opuści pole (np. klikając inną kontrolkę lub przyciskając klawisz *Tab*). Ta metoda walidacji sprawdza się szczególnie w długich formularzach, gdzie łatwo popełnić błędy w wielu polach. Niestety, obecnie HTML5 nie zawiera techniki dynamicznego sprawdzania, choć pewnie zostanie dodana do standardu w przyszłości. Jeśli zatem zależy Ci na natychmiastowym generowaniu informacji o błędach, napisz własny kod JavaScript lub wykorzystaj gotową bibliotekę.

Wyłączanie mechanizmu walidacji

Czasem zdarzają się sytuacje, w których trzeba wyłączyć mechanizm walidacji. Przypuśćmy, że chcesz szybko przetestować, jak dobrze identyfikuje błędy Twój kod po stronie serwera — warto wtedy dezaktywować walidację kliencką. Aby wyłączyć ją w całym formularzu, wystarczy wewnątrz znacznika `<form>` umieścić atrybut `novalidate`.

```
<form id="zooKeeperForm" action="processApplication.cgi" novalidate>
```

Innym rozwiązaniem jest dodanie przycisku przesyłania, który omija walidację. Technika ta czasem się przydaje. Możesz np. wymuszać walidację przy użyciu oficjalnego przycisku *Wyślij*, ale jednocześnie pozwolić użytkownikowi na zapisanie niedokończonych danych, aby mógł wysłać formularz później. W tym celu do znacznika `<input>` należy dołączyć atrybut `formnovalidate`:

```
<input type="submit" value="Zachowaj na później" formnovalidate>
```

Wiesz już, jak używać walidacji do wykrywania niewypełnionych pól. Pora, byś dowiedział się, jak szukać błędów w różnych typach danych.

Uwaga: Chcesz sprawdzić poprawność wpisanych liczb? Co prawda, żadna reguła walidacji nie wymusza stosowania cyfr w tekście, lecz pojawił się utworzony z myślą o liczbach nowy typ kontrolki — `number` (szczegóły na stronie 138). Niestety, jej obsługa wciąż jest niepełna.

Formatowanie kontrolek walidacyjnych

Mimo że nie da się obstylować wiadomości walidacyjnych, można zmienić wygląd pól tekstowych, w zależności od ich stanu. Można np. nadać źle wypełnionym kontrolkom inny kolor tła — taki widget zmieni wygląd zaraz po tym, jak przeglądarka wykryje problem.

Aby to osiągnąć, należy skorzystać z kilku nowych pseudoklas (strona 382). Oto szczególnie przydatne.

- Pseudoklasy `required` i `optional`, które nakładają formatowanie CSS w zależności od tego, czy element zawiera atrybut `required`.
- Pseudoklasy `valid` i `invalid`, które nanoszą style na kontrolki w zależności od tego, czy zawierają błędy. Pamiętaj, że przeglądarki wykryją błędne wartości dopiero wtedy, jak użytkownik spróbuje wysłać formularz, więc właściwe formatowanie nie pojawi się na stronie od razu.
- Pseudoklasy `in-range` i `out-of-range`, nakładające formatowanie arkusza na kontrolki, do których dodano atrybuty `max` i `min` (strona 138).

Jeśli np. chcesz, by wymagane pola `<input>` miały jasnożółte tło, łatwo zdefiniujesz taki wygląd przy użyciu pseudoklasy `required`:

```
input:required {
background-color: lightyellow;
}
```

Nie ma przeciwwskazań, by tym kolorem wyróżnić wymagane pola z błędnymi wartościami; musisz połączyć pseudoklasy `required` i `invalid` w taki oto sposób:

```
input:required:invalid {
background-color: lightyellow;
}
```

Takie ustawienia spowodują wyróżnienie pustych pól, gdyż łamią regułę narzucaną przez atrybut `required`.

Możesz wykorzystać inne sztuczki, np. połączyć pseudoklasę `validation` z pseudoklasą `focus`, użyć przesuniętego tła, w którym zawarłeś specjalną ikonkę do oznaczenia błędnych wartości itp. Warto jednak pamiętać o jednej podstawowej

zasadzie: możesz użyć tych własności, aby ulepszyć Twoją stronę, ale ze względu na kompatybilność ze starszymi przeglądarkami upewnij się, że prezentuje się dobrze bez nich.

Walidacja wyrażeń regularnych

Fundamentem najbardziej użytecznego (i złożonego) typu walidacji są tzw. **wyrażenia regularne**. JavaScript już od dawna obsługuje wyrażenia stałe, zatem dodanie tej własności do formularzy HTML to krok uzasadniony.

Wyrażenie regularne jest swego rodzaju schematem zdefiniowanym w specyficznym języku. Wyrażenia regularne są ustanawiane w celu wymuszenia poprawności pewnych wzorców tekstu. Przykładowo wyrażenie regularne może ustanawiać sekwencje wprowadzania cyfr i liter tylko tak, by tworzyły istniejący kod pocztowy, wymuszać zapisanie symbolu `@` w adresie e-mail lub narzucać dodanie przynajmniej dwuliterowego rozszerzenia w adresie domeny. Spójrz na tę formułę:

```
[A-Z]{3}-[O-9]{3}
```

Kwadratowy nawias na początku wyrażenia otwiera deklarację zakresu dozwolonych znaków. Zakres `[A-Z]` umożliwia wprowadzenie dużych liter od A do Z (niestety, tylko liter alfabetu łacińskiego). Występująca po nim cyfra pomiędzy nawiasami klamrowymi określa liczbę zdefiniowanych w zakresie liter. Tak więc w omawianym przykładzie formuła `{3}` mówi, że należy wpisać trzy duże litery. Myślnik w środku wyrażenia nie ma żadnego wyjątkowego znaczenia — po prostu wskazuje, że wymaga się wstawienia go po trzech pierwszych literach. Na koniec, formuła `[O-9]` pozwala na wpisanie cyfr od 0 do 9, a fraza `{3}` wymusza podanie trzech cyfr.

Porównywanie wyrażeń zwykłych przydaje się przy przeszukiwaniu (odnajdywaniu pasujących wzorców) i walidacji (sprawdzaniu, czy wartość pasuje do formuły). W HTML5 wyrażeń zwykłych używa się głównie w tym drugim celu.

Uwaga: Maniacy wyrażeń regularnych, zwróćcie uwagę, że nie musicie korzystać z czarodziejskich znaków `^` i `$`, żeby porównać początek i koniec wartości w polu. HTML5 zakłada wprowadzenie obu tych symboli, co oznacza, że *cała* wartość musi odpowiadać wzorcowi, by została uznana za prawidłową.

Po porównaniu ze zdefiniowaną wyżej regułą następujące wartości okazały się prawidłowe:

```
QRB-001
TTT-952
LAA-000
```

Te zaś wartości zostały zidentyfikowane jako błędne:

```
qrb-001
TTT-0952
LA5-000
```

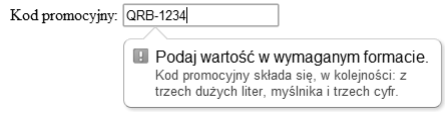
Wyrażenia regularne są zwykle o wiele bardziej złożone niż w omawianym tu przykładzie. Ich poprawne napisanie może przyprawić o ból głowy, toteż większość twórców stron korzysta z gotowych formuł, które sprawdzają interesujący ich typ danych.

Wskazówka: Aby dowiedzieć się więcej o wyrażeniach regularnych, wypróbuj dwa znakomite samouczki *www.w3schools.com/js/js_obj_regexp.asp* i *http://tinyurl.com/jsregex*. Gotowe formuły znajdziesz na stronie *http://regexlib.com*. Jeżeli masz aspirację zostać guru wyrażeń regularnych, przeczytaj książkę *Wyrażenia regularne*, autorstwa Jeffreya Friedla (Helion, 2001).

Gdy już przygotujesz wyrażenie regularne, możesz nałożyć je na elementy `<input>` lub `<textarea>`, dodając do nich atrybut `pattern`:

```
<label for="promoCode">Promotion Code</label>
<input id="promoCode" placeholder="QRB-001" title=
"Kod promocyjny składa się, w kolejności: z trzech dużych liter,
↪myślnika i trzech cyfr."
pattern="[A-Z]{3}-[0-9]{3}">
```

Na rysunku 4.6. zaprezentowano, co się dzieje, jeśli złamie się narzuconą przez wyrażenie regularne regułę.



Rysunek 4.6. *Najnowocześniejsze przeglądarki (np. widoczny na rysunku Google Chrome) nie tylko wyłapują błąd, lecz również pobierają tekst z atrybutu title i wyświetlają go wraz z komunikatem, żeby wyjaśnić użytkownikowi, co jest nie tak*

Wskazówka: Przeglądarki nie przeprowadzają walidacji pustych pól. W tym przykładzie puste pole z kodem promocyjnym przejdzie walidację. Jeśli nie chcesz na to pozwolić, równocześnie zastosuj atrybuty `pattern` i `required`.

Uwaga: Wyrażenia regularne wydają się wprost stworzone do definiowanie adresów e-mail i prawdą jest, że sprawdzają się w tym wyśmienicie. Na razie jednak powstrzymaj się od ich stosowania w tym celu. Specjalnie z myślą o adresach poczty elektronicznej HTML5 wprowadza nowy typ kontrolek z wbudowanym regularnym wyrażeniem (strona 137).

Własne reguły walidacji

Specyfikacja HTML5 wyróżnia zbiór gotowych właściwości JavaScriptu, które pozwolą Ci określić, czy podane wartości są poprawne (lub zmusić przeglądarkę do zrobienia tego za Ciebie). Najbardziej przydatna z nich wszystkich jest metoda `setCustomValidity()`, która umożliwia utworzenie własnych zasad sprawdzania poprawności dla określonych pól i ich współgrania z wbudowanym w HTML5 systemem walidacji.

Działa w następujący sposób. Najpierw należy sprawdzić, czy pole nie zawiera błędów. Można to osiągnąć, obsługując zdarzenie `onInput`, co nie jest niczym nowym:

```
<label for="comments"> Kiedy zdałaś/łeś sobie sprawę, że opiekun zwierząt
↪to zawód dla Ciebie?</label>
<textarea id="comments" oninput="validateComments(this)"></textarea>
```

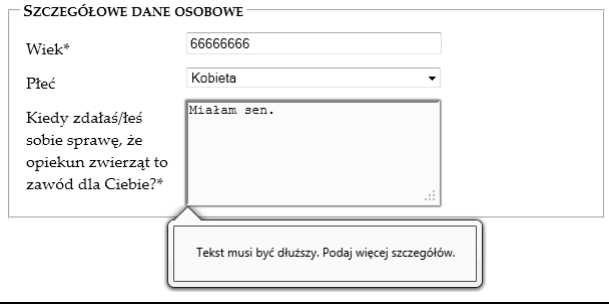
W tym przykładzie zdarzenie `onInput`, uruchamiane po podaniu przez użytkownika danych, wywołuje funkcję `validateComments()`. Napisanie tej funkcji, sprawdzenie wartości elementu `<input>` i wywołanie w niej metody `setCustomValidity()` należy do Ciebie.

Jeśli pobrana wartość jest nieprawidłowa, do wywołania `setCustomValidity()` powinieneś przekazać wiadomość o błędzie. Z drugiej strony, jeżeli okaże się w pełni poprawna, do tej samej metody należy przekazać pusty typ łańcuchowy. W ten sposób usuniesz dowolną inną wiadomość, którą ustawiłeś wcześniej.

Podany niżej przykład wymusza wpisanie w pole tekstowe przynajmniej 20 znaków:

```
function validateComments(input) {
  if (input.value.length < 20) {
    input.setCustomValidity("Tekst musi być dłuższy. Podaj więcej
    ↪szczegółów.");
  }
  else {
    // Poprawna wartość. Czyści wiadomość o błędzie.
    input.setCustomValidity("");
  }
}
```

Na rysunku 4.7. pokazano, co się stanie, jeśli ktoś złamie narzuconą regułę i spróbuje wysłać formularz.



Rysunek 4.7. *Jeżeli do metody setCustomValidity dodasz jako parametr własną wiadomość, przeglądarka zinterpretuje ją jak wybudowany w nią tekst. Po wysłaniu niepoprawnej wiadomości wyświetlone zostanie napisana przez Ciebie wyskakująca wiadomość*

Oczywiście, zaprezentowany tu scenariusz da się rozwiązać przy użyciu wyrażeń regularnych, narzucających wpisanie długich haseł. Jednak nadają się one do walidacji niektórych typów danych, a własnoręcznie skonstruowany kod może zweryfikować poprawność każdej informacji — od skomplikowanych działań algebraicznych, po szyfr połączenia z sieciowym serwerem.

Uwaga: Pamiętaj, że użytkownik Twojej strony może podejrzeć wszystko, co się dzieje w kodzie JavaScript, więc nie nadaje się on do wykonywania szyfrowych algorytmów. Ty jako programista możesz np. wiedzieć, że w prawidłowym kodzie promocyjnym suma cyfr jest zawsze równa 12. Ostatnią rzeczą, którą chcesz zrobić, jest zdradzenie tej informacji wewnątrz procedury walidacji, ponieważ pomogłoby to oszustom wygenerować fałszywe kody. Taki typ walidacji powinien zachodzić na serwerze.

Obsługa mechanizmu walidacji

Twórcy przeglądarek dodają kolejne fragmenty obsługi mechanizmów walidacji, kawałek po kawałku. W rezultacie część przeglądarek obsługuje niektóre własności weryfikacji, ignorując przy tym inne. W tabeli 4.2 znajdziesz informacje o najstarszych wersjach przeglądarek, na których wszystkie z zaprezentowanych sztuczek działają.

Tabela 4.2. *Obsługa mechanizmów walidacji*

	IE	Firefox	Chrome	Safari	Opera	Safari iOS	Android
Najstarsza wersja	10*	4	10	5 (wersja na system Windows)	10	-	-

*Aktualnie przeglądarka ta jest dostępna wyłącznie we wczesnej wersji Beta.

Z uwagi na to, że walidacja w HTML5 nie zastąpi tego rodzaju mechanizmów działających po stronie serwera, możesz uznać, że jest ona zaledwie dodatkiem, i jakoś znieść nierówne wsparcie. Przeglądarki, które nie implementują tego rozwiązania, jak choćby Internet Explorer 9, pozwalają ich użytkownikom wpisywać niepoprawne daty, bo przecież i tak zostaną one wykryte na etapie weryfikacji na serwerze i ukazane po ponownym załadowaniu strony.

Z drugiej strony, na Twojej witrynie mogą mieścić się złożone formularze, w których łatwo o pomyłkę, i może Ci zależeć na tym, żeby nie przysparzać użytkownikom IE niepotrzebnych frustracji. W takim przypadku masz do wyboru dwie możliwości: napisany przez Ciebie awaryjny kod walidacji lub bibliotekę JavaScriptu, która doda brakującą funkcjonalność. Twój wybór zależy od stopnia złożoności wymaganej walidacji.

Jeżeli wszystkie formularze na witrynie wymagają mechanizmu weryfikacji, warto dodać własne instrukcje sprawdzające. Korzystając ze skryptu Modernizr (patrz strona 52), możesz przetestować obsługę wielu własności HTML5. Przykładowo instrukcja `Modernizr.input.pattern` sprawdzi, czy przeglądarka rozpoznaje atrybut `pattern`:

```
if (!Modernizr.input.pattern) {
// Wybrana przeglądarka nie przeprowadza walidacji wyrażeń regularnych.
// Warto użyć wyrażeń regularnych JavaScriptu.
...
}
```

Uwaga: Atrybut `pattern` jest jedną z wielu własności testowanych przez obiekt `Modernizr.input`. Inne własności, których obsługę warto ustalić, to: `placeholder`, `autofocus`, `required`, `max`, `min` i `step`.

Oczywiście, w podanym przykładzie nie określono, kiedy należy przeprowadzać weryfikację i jak reagować na jej wyniki. Jeśli zależy Ci na tym, aby Twój system walidacji naśladował zdefiniowany przez HTML5, przeprowadzaj ją dopiero po wysłaniu formularza na serwer. W tym celu należy obsłużyć dołączane do znacznika `<form>` zdarzenie `onSubmit` funkcją, która zwróci wartość `true` (prawda — co oznacza, że formularz został wypełniony prawidłowo i przeglądarka może go przesłać dalej) lub `false` (fałsz — w formularzu znaleziono błąd i należy przerwać operację).

```
<form id="zooKeeperForm" action="processApplication.cgi"
onsubmit="return validateForm()">
```

Oto przykład prostej procedury walidacji, która wymaga wypełnienia wszystkich pól:

```
function validateForm() {
  if (!Modernizr.input.required) {
    // Jeśli przeglądarka nie rozpoznaje atrybutu required, należy samodzielnie sprawdzić,
    // czy wymagane pola zostały poprawnie wypełnione.

    // Najpierw utwórz tablicę ze wszystkimi elementami.
    var inputElements = document.getElementById("zooKeeperForm").elements;

    // Następnie przeszukaj tablicę element po elemencie.
    for(var i = 0; i < inputElements.length; i++) {

      // Sprawdź, czy element posiada atrybut required.
      if (inputElements[i].hasAttribute("required")) {
        // Jeśli znaleziono atrybut required, sprawdź, jaką ma wartość.
        // Jeśli nie zidentyfikowano go, formularz nie przechodzi walidacji,
        // a cała funkcja zwraca wartość false (fałsz).
        if (inputElements[i].value == "") {
          return false;
        }
      }
    }
    // Jeśli przeglądarka dodarła do tego punktu, walidacja się powiodła
    // i formularz może zostać wysłany.
    return true;
  }
}
```

Wskazówka: Ten blok kodu powstał w oparciu o kilka technik języka JavaScript, takich jak przeszukiwanie elementów, pętla i instrukcje warunkowe. Aby dowiedzieć się o nich więcej, zapoznaj się z treścią dodatku B.

Jeśli Twój formularz jest skomplikowany, a Ty chcesz zaoszczędzić sobie wysiłku (lecz mimo to przygotować się na przyszłość), użycie prostej łatki JavaScriptu rozwiąże wszystkie Twoje problemy. Z technicznego punktu widzenia, podejście polega na tym samym — Twoja strona sprawdzi, czy walidacja jest obsługiwana przez przeglądarkę i przeprowadzi ją ręcznie, jeśli okaże się to konieczne. Różnica polega na tym, że biblioteka zawiera już cały potrzebny kod.

Pod adresem *http://tinyurl.com/polyfills* znajdziesz całą onieśmielającą listę bibliotek, które próbują zapewnić poszukiwaną funkcjonalność. Jedna z najlepszych nosi nazwę webforms2 i jest dostępna na stronie *https://github.com/westonruter/webforms2*. Aby ją ściągnąć, odszukaj dobrze ukryty przycisk *Download* (ściągnij).

Biblioteka webforms2 implementuje wszystkie poznane tu atrybuty. Aby jej użyć, umieść jej pliki w folderze głównym witryny (lub, co jest częstszą praktyką, w podfolderze) i dodaj ścieżkę do niej na swojej stronie:

```
<head>
<title>...</title>
<script src="webforms2.js"></script>
...
<head>
```

NIEOSZLIFOWANY DIAMENT

Kilka pominiętych atrybutów kontrolek

HTML5 rozpoznaje kilka innych atrybutów, które kontrolują zachowanie przeglądarki w trakcie edytowania formularza, lecz nie są używane do walidacji. Nie wszystkie te własności działają równie dobrze na wszystkich przeglądarkach. Mimo to, warto z nimi poeksperymentować.

- Atrybut **`spellcheck`**. Niektóre przeglądarki pomagają uniknąć wstydu, bo sprawdzają poprawność ortograficzną słów wpisywanych w pola tekstowe. Oczywistym problemem jest to, że nie zawsze tekst musi składać się z ortograficznie prawidłowych słów, a użytkownik zniesie tylko ograniczoną liczbę podkreślonych wyrazów, nim się zdenerwuje. Ustaw wartość atrybutu `spellcheck` na `false`, jeśli nie chcesz, aby przeglądarka sprawdzała ortografię użytkownika, lub na `true`, kiedy właśnie na tym Ci zależy. Warto zauważyć, że przeglądarki różnią się pod względem domyślnej obsługi tego atrybutu — są takie, które wspierają go bez żadnej deklaracji.
- Atrybut **`autocomplete`**. Niektóre przeglądarki chcą Ci zaoszczędzić czasu, oferując podpowiedzi z niedawno wpisanymi w pola wartościami. Takie zachowanie nie zawsze jest pożądane — jak stwierdzono w specyfikacji HTML5, niektóre informacje mogą być zbyt istotne (np. kod odpalenia rakiet z ładunkiem jądrowym) lub ważne tylko przez krótki czas (np. jednorazowy kod logowania banku). W takich przypadkach zaleca się nadanie atrybutowi `autocomplete` wartości `off` (wyłączony). Możesz też przestawić ją na `on` (włączony) dla pojedynczego pola.
- Atrybuty **`autocorrect` i `autocapitalize`**. Własności te kontrolują mechanizm autokorekty i automatycznego pisania dużych liter na niektórych urządzeniach przenośnych, szczególnie na działającej na iPhonie lub iPadzie przeglądarce Safari.
- Atrybut **`Multiple`**. Projektanci stron już od dawna dodawali atrybut `multiple` do znacznika `<select>`, aby utworzyć listę elementów wielokrotnego wyboru. Teraz jednak możesz go dołączyć do znacznika `<input>`, w tym do pola wyboru pliku (używanego do ładowania go na serwer) i pola e-maila (patrz strona 137). Dzięki niemu na przeglądarkach, które obsługują tę własność, użytkownik może wybrać kilka plików lub dodać kilka adresów poczty jednocześnie.

Biblioteka webforms współpracuje z inną użyteczną łatką JavaScriptu, zwaną html5Widgets. Skrypt ten dodaje obsługę kilku własności, z którymi zapoznasz się później. Więcej o tym pliku dowiesz się na stronie *www.useragentman.com/tests/html5Widgets*. Obie przedstawione biblioteki oferują solidne wsparcie dla formularzy, jednak nawet używając ich, często natkniesz się na drobne błędy i potknięcia. Tylko czas pokaże, czy te już dość imponujące biblioteki będą usprawniane i utrzymywane.

Nowe typy znacznika input

Chyba największym dziwactwem formularzy HTML jest to, że pojedynczy element, nazwany ogólnie `<input>`, jest używany do tworzenia różnego rodzaju kontrolek: od pól wyboru, przez pola tekstowe, po przyciski. Atrybut `type` określa, co dany element `<input>` tworzy po wczytaniu strony.

Jeżeli przeglądarka natrafi na element `<input>` o typie, którego nie rozpoznaje, traktuje go jak zwykłe pole tekstowe. Oznacza to, że następujące trzy elementy zostaną wczytane tak samo na każdej przeglądarce:

```
<input type="text">
<input type="super-udziwniony-typ-znacznika">
<input>
```

HTML5 wykorzystuje ten sposób interpretacji. Jego twórcy dodali kilka nowych typów elementu `<input>`, mając świadomość, że starsze przeglądarki zinterpretują je jak zwykłe pola tekstowe. Jest to przydatne, gdy np. chcesz utworzyć pole do wprowadzania adresów e-mail; bez przeszkód możesz zadeklarować nowy typ `email`:

```
<label for="email">E-mail <em>*</em></label>
<input id="email" type="email"><br>
```

Jeśli otworzysz stronę w przeglądarce, która nie rozpoznaje kontrolki typu `email` (np. w Internet Explorerze 9), wygenerowane zostanie zwykłe pole tekstowe, co można zaakceptować. Przeglądarki obsługujące HTML5 są jednak trochę bardziej zmyślne. Oto, co potrafią.

- **Oferują wygodne edytowanie**. Inteligentne przeglądarki pozwalają przenieść adres e-mail z komputerowej książki adresów na kontrolkę.
- **Zapobiegają potencjalnym błędom**. Przykładowo przeglądarki mogą zignorować wpisywane litery, gdy w polu należy umieścić wyłącznie cyfry, lub odrzucić nieprawidłowe daty (bądź zmusić użytkownika do wybrania ich z minikalendarza, co jest prostszą i bezpieczniejszą praktyką).
- **Przeprowadzają walidację**. Przeglądarki mogą dokonać bardziej złożonej weryfikacji po kliknięciu przycisku wysyłania formularza. Najnowocześniejsze mogą rozpoznać nieprawidłowy adres e-mail i odmówić wykonania operacji.

Specyfikacja HTML5 nie daje twórcom przeglądarek żadnych wskazówek dotyczących pierwszego punktu. Innymi słowy, przeglądarki mogą rozwiązać problem wyświetlania i edytowania różnych typów danych w dowolny racjonalny sposób, a różne przeglądarki mogą zawierać odmienne mechanizmy ułatwiające. Przykładowo przeglądarki mobilne wykorzystują brak ograniczeń, ukrywając klawisze interfejsu, które nie są potrzebne do wypełnienia wybranego pola (rysunek 4.8).

Dużo ważniejsze są mechanizmy zapobiegania i wykrywania błędów. Jako minimum przeglądarki obsługujące formularze HTML5 nie pozwalają na przesłanie formularza, jeśli zawiera dane, które łamią narzucone ograniczenia. Jeżeli więc przeglądarce nie uda się zapobiec błędom (według obserwacji z drugiego punktu listy powyżej), musi je zweryfikować, po tym jak użytkownik potwierdzi wysłanie informacji (punkt trzeci na liście).

Niestety, nie wszystkie przeglądarki spełniają te trzy wymagania. Niektóre z nich rozpoznają nowe typy danych i wprowadzają ciekawe gadżety do edytowania zawartości, lecz nie obsługują walidacji. Inne prawidłowo interpretują tylko niektóre typy danych. Szczególnych problemów przysparzają przeglądarki mobilne — co prawda, pełno w nich udogodnień edytowania, lecz nie działają na nich żadne wbudowane mechanizmy walidacji.

W tabeli 4.3 znajdziesz listę typów danych i przeglądarek, które w pełni je obsługują — co oznacza, że przerywają wysyłanie formularza, jeśli wykryją w nim nieprawidłowe dane.

Wskazówka: Tak na marginesie, łatwo zweryfikować obsługę Modernizra, wykorzystując właściwości obiektów `inputtypes`. Przykładowo wywołanie funkcji `Modernizr.inputtypes.range` zwróci wartość `true` (prawda) w przeglądarkach, które obsługują typ danych `range`.

Rysunek 4.8. *Osoby korzystające z urządzeń przenośnych przy wypełnianiu formularza są na ogół pozbawione luksusu, jakim jest pełna klawiatura. iPod ułatwia życie takim użytkownikom strony, dopasowując wirtualną klawiaturę do spodziewanego typu danych. Pole tekstowe wymagające podania numeru telefonu wywoła klawiaturę numeryczną (po lewej), podczas pole z adresem e-mail spowoduje pojawienie się klawiatury z dedykowanym przyciskiem @ i mniejszym klawiszem spacji (po prawej)*

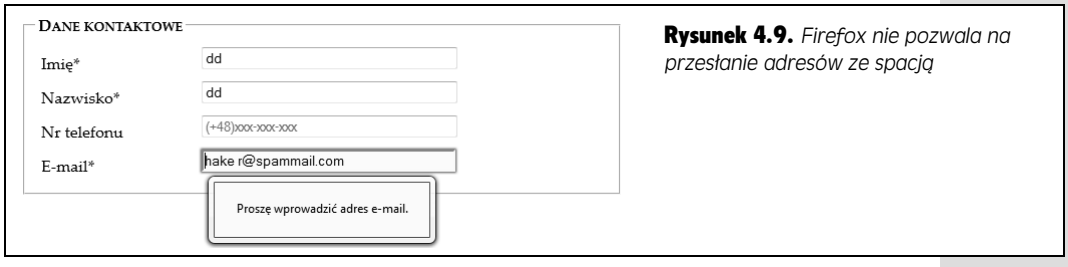
Tabela 4.3. *Kompatybilność przeglądarek a nowe typy elementu <input>*

Wartość atrybutu	IE	Firefox	Chrome	Safari	Opera	Safari iOS*	Android
`email`	-	4	10	5 (tylko wersja Windows)	10.6	-	-
`url`	-	4	10	5 (tylko wersja Windows)	10.6	-	-
`search`*	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy
`tel`*	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy	nie dotyczy
`number`	-	-	10	5	-	-	-
`range`	-	-	6	5	11	-	-
`datetime`, `date`, `month`, `week`, `time`	-	-	10	-	11	-	-
`color`	-	-	-	-	11	-	-

* Standard HTML5 nie wymaga walidacji dla tego typu danych.
** Choć przeglądarka iOS nie obsługuje walidacji, jej zdolność do dynamicznego przystosowania klawiatury (rysunek 4.8) jest istotnym udogodnieniem, więc wciąż warto używać specjalistycznych typów danych.

Adresy e-mail

Pola adresów e-mail deklaruje się, nadając atrybutowi `type` wartość `email`. Zwykle poprawny adres e-mail jest ciągiem znaków (w którym nie dozwolono użycia pewnych symboli). Adres e-mail musi zawierać znak `@` oraz przynajmniej jedną kropkę. Co więcej, między nimi muszą znajdować się przynajmniej dwa znaki. Te zasady dotyczą w praktyce wszystkich adresów poczty elektronicznej. Zdefiniowanie solidnych reguł walidacji lub wyrażenia regularnego dla adresów e-mail okazuje się jednak nad wyraz trudnym zadaniem, które przerosło siły wielu webdeweloperów. To sprawia, że przeglądarki, które obsługują atrybut e-mail i przeprowadzają jego walidację automatycznie, są wspaniałe (rysunek 4.9).



Rysunek 4.9. *Firefox nie pozwala na przesłanie adresów ze spacją*

Kontrolka typu `email` może zawierać atrybut `multiple`, który pozwala przenieść do pola kilka adresów na raz. Domyślnie adresy te zleją się w jeden ciąg tekstu, dlatego należy rozdzielić je przecinkiem.

Uwaga: Pamiętaj, że niewstawienie żadnej wartości w polu omija walidację. Jeśli chcesz zmusić użytkownika do wprowadzenia prawidłowego adresu, do kontrolki typu `email` dodaj atrybut `required` (strona 126).

Adresy URL

Pole adresów URL deklaruje się, nadając atrybutowi `type` wartość `url`. To, co konstytuuje prawidłowy adres URL, jest przedmiotem zaciekłych dyskusji. W większości przeglądarek używa się jednak dość mało restrykcyjnych algorytmów walidacji. Adres URL wymaga podania przedrostka (który może być poprawny, jak `http://`, lub wymyślony, niczym `bonk://`). Można w nim też umieszczać spacje i inne symbole, z wyjątkiem dwukropka (:).

Niektóre przeglądarki przy wpisywaniu generują listę sugerowanych adresów, bazując na historii ostatnio odwiedzanych witryn.

Pola wyszukiwania

Pola wyszukiwania powstają po zadeklarowaniu w kontrolce atrybutu `type` o wartości `search`. Pole to ma mieścić słowa klucze, których używa się przy wyszukiwaniu haseł. Mogą być użyte do przeszukania całej sieci (tak jak w widocznej na

rysunku 4.1 wyszukiwarce Google), pojedynczej strony lub katalogu z interesującą informacją. Pole wyszukiwania zachowuje się i wygląda jak zwykłe pole tekstowe.

Na niektórych przeglądarkach (np. Safari) wyglądają one odrobinę inaczej — mają zaokrąglone rogi. Ponadto podczas wpisywanie w nie treści na przeglądarce Safari i Chrome po prawej stronie kontrolki pojawia się mała ikona X, kliknięcie której spowoduje oczyszczenie pola. Poza tymi drobnymi różnicami, pola wyszukiwania *są* polami tekstowymi. Cała różnica tkwi w semantyce. Innymi słowy, typu danych `search` używa się, aby umożliwić przeglądarkom i oprogramowaniu dla osób niepełnosprawnych łatwiejszą identyfikację kontrolki. W rezultacie są one w stanie naprowadzić użytkownika we właściwe miejsce na stronie i zaoferować dodatkowe opcje kontroli — to drugie być może w przyszłości.

Telefon

Numery telefonów deklaruje się za pomocą typu `tel`. Numery telefonów występują w wielu formach. Niektóre składają się z cyfr, inne zawierają spacje, myślniki, znak + i nawiasy. Być może z powodu takiej różnorodności standard HTML5 nie wymaga od przeglądarek wykonania walidacji numerów telefonów. Trudno jednak nie życzyć sobie, aby pole typu `tel` odrzucało litery (które przyjmuje).

Obecnie jedynym powodem, dla którego warto zwrócić uwagę na kontrolkę `tel`, jest to, iż na przeglądarkach urządzeń mobilnych wywołuje ona zmodyfikowaną wirtualną klawiaturę numeryczną.

Liczby

Standard HTML5 definiuje dwa numeryczne typy danych. Typ danych `number` powstał z myślą o zwykłych liczbach.

Sposób jego zastosowania jest niemal oczywisty. Zwykłe pola tekstowe przyjmują dowolne wartości: liczby, litery, spacje, znaki interpunkcyjne i symbole używane do przedstawiania przekleństw w komiksowych dymkach. Dlatego też jedną z najczęściej wykonywanych form walidacji jest weryfikowanie, czy wprowadzana wartość jest liczbą i mieści się w narzuconym zakresie. Teraz jednak wystarczy wprowadzić kontrolkę typu `number` i przeglądarka zignoruje naciśnięcia nieliczbowych klawiszy. Oto przykład:

```
<label for="age">Wiek<em>*</em></label> <input id="age"
type="number"><br>
```

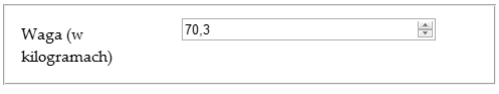
Oczywiście, jest wiele rodzajów liczb i nie wszystkie nadają się do wpisywania każdego typu danych. Zaprezentowany wyżej kod HTML pozwala na wpisanie w polu wieku takich wartości jak 43 000 i –6. Z myślą o takich scenariuszach zaprojektowano atrybuty `min` i `max`. W przykładzie niżej zakres wieku zdefiniowano w rozsądnych granicach od 0 do 120 lat.

```
<input id="age" type="number" min="0" max="120"><br>
```

Zazwyczaj typ `number` akceptuje wyłącznie liczby całkowite, więc liczba 30,5 zostałaby uznana za błędnie napisaną. Niektóre przeglądarki nie pozwolą nawet wpisać przecinka. Umożliwia to dodanie atrybutu `step` wraz z wartością ułamka dziesiętnego, który wskaże najmniejszą wartość po przecinku. Przykładowo podstawienie wartości `0.1` sprawi, że liczby 0, 0,1, 0,2, 0,3 itd. będą rozpoznawane jako prawidłowe (zwróć uwagę, że w deklaracji użyto angielskiego formatowania ułamka dziesiętnego; niestety, obecnie tylko przeglądarka Chrome poprawnie interpretuje znak przecinka w ułamku — w innych przeglądarkach, wpisując ułamki, należy posługiwać się kropką). Jednak próba przesłania liczby `0,15` zakończy się pojawieniem znanego Ci już wyskakującego komunikatu o błędzie. Domyślnie atrybut `step` ma wartość `1`.

```
<label for="weight">Waga (w kilogramach)</label>
<input id="weight" type="number" min="0.1" max="650" step="0.1"
value="70"><br>
```

Jak widać na rysunku 4.10, atrybut `step` wpływa też na działanie przycisków po prawej stronie kontrolki.



Rysunek 4.10. *Najnowsze przeglądarki dołączają przyciski do pól numerycznych. Po każdym kliknięciu górnego przycisku liczba w polu jest zwiększana o zdefiniowaną atrybutem step wartość. Podobnie kliknięcie dolnego przycisku spowoduje zmniejszenie liczby*

Suwak

Innym liczbowym typem danych w HTML5 jest typ `range`. Podobnie jak typ `number`, może on reprezentować liczby całkowite i ułamki. Obsługuje również te same atrybuty zakresu dozwolonych wartości (czyli `min` i `max`). Oto przykład:

```
<label for="weight">Waga (w kilogramach)</label>
<input id="weight" type="range" min="0" max="560" value="70"><br>
```

Różnica tkwi w sposobie reprezentowania informacji. Zamiast po prostu wpisywać wartość w polu tekstowym, przeglądarki nowej generacji wczytują suwak wartości (rysunek 4.11).

Rysunek 4.11. *Prawdopodobnie korzystałeś już z podobnych kontrolek do zwiększania głośności i ściszania dźwięku. Sprawdzają się świetnie, gdy w grę wchodzi pobranie danych z pewnego zakresu, w których nie jest ważna dokładna wartość, ale położenie na skali między minimum a maksimum*

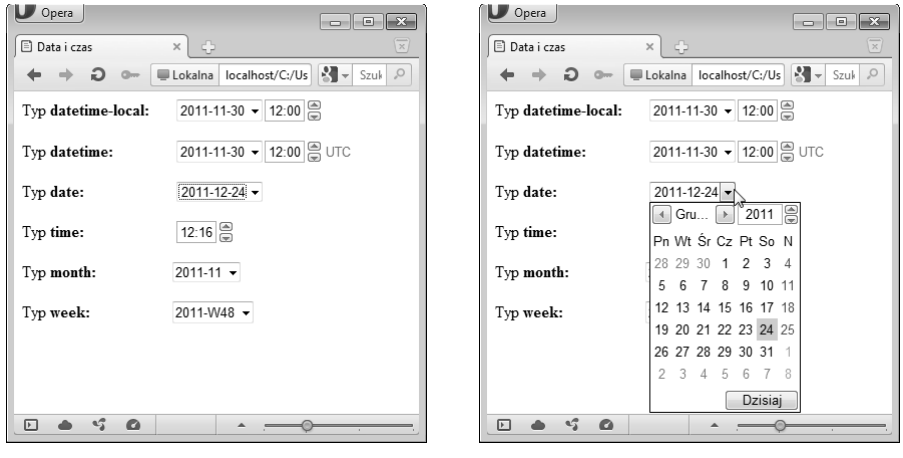
Wartość na tej kontrolce ustawia się, przesuwając suwak w granicach zdefiniowanego zakresu. Przeglądarki, które rozpoznają typ `range`, nie zwracają żadnych informacji o wartości. Jeśli te dane są Ci w jakimś celu potrzebne, musisz dodać kilka linijek kodu JavaScript, który zareaguje na zmianę położenia suwaka (np. poprzez obsługę zdarzenia `onChange`) i wyświetli informacje w pobliżu. Oczywiście,

warto sprawdzić, czy wybrana przeglądarka obsługuje typ `range` (choćby przy użyciu narzędzie Modernizr). Jeśli okaże się, że tak nie jest, nie trzeba implementować kodu, gdyż ustawiona wartość pokaże się w zwykłym polu tekstowym.

Czas: daty i godziny

HTML5 definiuje kilka typów danych. Przeglądarki, które potrafią je zidentyfikować, wyświetlają wygodne kalendarze, z których da się wybrać datę. W rezultacie nie tylko unikniesz zamieszania z formatowaniem czasu, ale zapobiegniesz też przypadkowemu (lub celowemu) wybraniu daty, która nie może istnieć. Inteligentne przeglądarki przyszłości pójdą krok dalej — zintegrują pole z osobistym kalendarzem.

Obecnie, pomimo oczywistych zalet, typy dat nie cieszą się powszechnym wsparciem. Opera jest jedyną przeglądarką, która wyświetla rozwijane kalendarze (rysunek 4.12). Chrome zapewnia jedynie podstawową obsługę: wyświetla pola tekstowe z przyciskami, takie same jak dla typu `number`, i odrzuca zły format daty.



Rysunek 4.12. *Element <input> prezentuje się trochę inaczej, gdy ma służyć do przechowania informacji o dacie i godzinie (po lewej). Największym udogodnieniem w Operze jest rozwijalny kalendarz, który pozwala wybrać datę i uniknąć męki jej formatowania (po prawej)*

Wskazówka: Jeśli zdecydujesz się użyć jednego z typów dat, warto zastosować wypełnienie, w rodzaju opisanej wcześniej biblioteki html5Widgets (*www.useragentman.com/tests/html5Widgets*). W końcu osobom używającym przeglądarek innych niż najnowsza Opera czy Chrome łatwo wpisać datę w złym formacie, a walidacja tego typu danych oraz podawanie wskazówek jest trudnym i niewdzięcznym zadaniem. Z tego względu napisane w języku JavaScript kontrolki dat cieszą się takim wzięciem w sieci.

W tabeli 4.4 znajdziesz opis sześciu formatów dat HTML5.

Wskazówka: Przeglądarki, które obsługują typy dat, wspierają również użycie atrybutów `min` i `max`. Możesz więc zdefiniować maksymalną lub minimalną datę, pod warunkiem że zapiszesz ją we właściwym formacie. Gdybyś chciał ograniczyć wartości w polu dat, mógłbyś wpisać `<input type="date" min="2012-01-01" max="2012-12-31">`.

Tabela 4.4. *Typy dat*

Wartość atrybutu type	Opis	Przykład
`date`	Data w formacie `YYYY-MM-DD`.	25 stycznia 2012 — 2012-01-25
`time`	Dokładny czas na 24-godzinnym zegarze w formacie `GG:mm:ss.ss`.	14:35 (i 52 sekundy) — 14:35 lub 14:35:50.2
`datetimelocal`	Data i godzina oddzielone dużą literą *T* (format: `YYYY-MM-DDTHH:mm:ss`).	25 stycznia 2012, 14:35 — 2012-01-15T14:35
`datetime`	Data i godzina z różnicą w strefie czasowej. Przyjmuje taki sam format jak w elemencie `<time>` (`YYYY-MM-DDTHH:mm:ss-HH:mm`) omówionym na stronie 93.	25 stycznia 2012 r., 14:35 w Warszawie — 2012-01-15T14:35+02:00
`month`	Rok i miesiąc w formacie `YYYY-MM`.	Styczeń 2012 r. — 2012-01
`week`	Rok i numer tygodnia w roku. Zauważ, że w roku są 52 lub 53 tygodnie.	Drugi tydzień 2012 roku — 2012-W02

Kolor

Kontrolka kolorów definiowana jest za pomocą typu `color`. Ten typ danych jest bardzo ciekawym, choć rzadko używanym gadżetem, który pozwala gościowi strony pobrać z niej kolor przy użyciu wysuwanej pipety kolorów, która wygląda podobnie jak w programie Paint, w systemie Windows. Aktualnie tylko Opera zapewnia pełną obsługę. W innych przeglądarkach należy własnoręcznie wpisać szesnastkowy kod koloru. Można też posłużyć się biblioteką html5Widgets, do ściągnięcia ze strony *www.useragentman.com/tests/html5Widgets*.

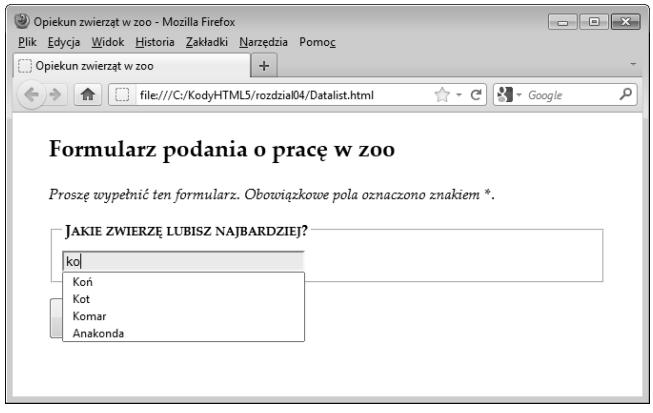
Nowe elementy

Do tej pory dowiedziałeś się, w jakim stopniu HTML5 rozszerza funkcjonalność formularzy dzięki nowym własnościom walidacji oraz jak wiele nowych typów kontrolek do niego dodano. Są to najbardziej praktyczne i najszerzej obsługiwane opcje, lecz nie stanowią jedynych nowości w formularzach HTML5.

W HTML5 wprowadzono też kilka innych innowacji, które wypełniają luki w zapotrzebowaniu i dodają większą funkcjonalność. Nowe elementy, o których tu mowa, pozwolą Ci osadzać na stronie rozwijane listy z sugestiami, paski stanu, paski narzędzi i wiele innych komponentów. Cały szkopuł w tym, że starsze przeglądarki nigdy nie będą ich rozpoznawać, a nawet w nowych ich wersjach są wprowadzane bardzo niechętnie, z uwagi na częste zmiany w specyfikacji. W konsekwencji, elementy te zaliczają się do *najgorzej* obsługiwanych własności w tym rozdziale. Najprawdopodobniej zechcesz zobaczyć je w akcji, lecz będziesz musiał poczekać z ich wykorzystaniem, chyba że nie przeszkadza Ci bycie pionierem w krainie błędów i niekompatybilności.

Sugerowane odpowiedzi i element <datalist>

Element <datalist> umożliwia zespolenie listy z sugerowanymi odpowiedziami ze zwykłym polem tekstowym. Dzięki temu użytkownik formularza może wybrać z listy dostępnych opcji lub wpisać własną (rysunek 4.13).



Rysunek 4.13. *W miarę jak piszesz, przeglądarka wyświetla potencjalne odpowiedzi. Wpisanie np. liter „ko" spowoduje wywołanie wszystkich zwierząt z tą sekwencją liter w nazwie (niekoniecznie od początku)*

Listy danych używa się wspólnie ze standardowym polem tekstowym. Załóżmy, że wprowadziłeś na stronę taki element <input>:

```
<legend>Jakie zwierzę lubisz najbardziej?</legend> <input id="favoriteAnimal">
```

Aby dodać rozwijalną listę sugestii, zadeklaruj element <datalist>. Teoretycznie, możesz umieścić ten komponent w dowolnym miejscu. Jest to spowodowane tym, że element datalist nie pojawia się na stronie — zamiast tego dostarcza danych używanych przez kontrolkę <input>. Rozsądnie jednak będzie umieścić element <datalist> w pobliżu pola tekstowego, z którym ma być zespolony. Oto przykład:

```
<datalist id="animalChoices">
        <option label="Koń" value="koń">
        <option label="Zebra" value="zebra">
        <option label="Kot" value="kot">
        <option label="Komar" value="komar">
        <option label="Gąsienica" value="gąsienica">
        <option label="Anakonda" value="anakonda">
        <option label="Człowiek" value="człowiek">
        <option label="Słoń" value="słoń">
        <option label="Gnu" value="gnu">
        <option label="Gołąb" value="gołąb">
        <option label="Krab" value="krab">
</datalist>
```

Podobnie jak w tradycyjnym elemencie <select>, w komponencie <datalist> zagnieżdża się znaczniki <option>. Każdy element <option> oznacza odrębną sugestię wyświetlaną pod polem tekstowym w trakcie wypełniania. Atrybut label pokazuje tekst, który pojawi się pod kontrolką, podczas gdy własność value — wartość, która zostanie wysłana na serwer, jeśli użytkownik wybierze jej opcję z listy. Lista danych jest całkowicie niewidoczna. Nim zacznie wyświetlać sugestie, trzeba ją załączyć do pola tekstowego, dodając do niego atrybut list i nadając mu wartość atrybutu id znacznika <datalist>.

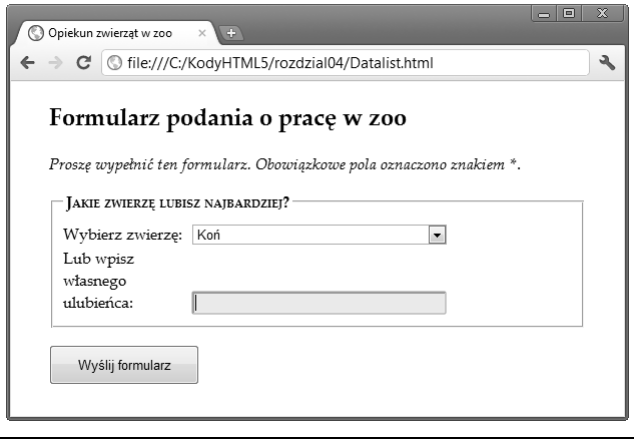
```
<input id="favoriteAnimal" list="animalChoices">
```

W przeglądarce, która rozpoznaje element `<datalist>` — obecnie są to wyłącznie Opera 10 i Firefox 4 (lub ich nowsze wydania) — użytkownicy ujrzą stronę z rysunku 4.13. Inne przeglądarki zignorują atrybut `list` i cały kod `<datalist>`, co sprawi, że lista sugestii stanie się bezużyteczna.

Istnieje jednak pewna awaryjna sztuczka, który zmusi wszystkie przeglądarki do właściwego zachowania. Polega ona na umieszczeniu dodatkowych treści wewnątrz komponentu `<datalist>`. Sprawdza się wyśmienicie, gdyż przeglądarki, które obsługują znacznik `<datalist>`, zauważają wyłącznie osadzony wewnątrz niego element `<option>` i ignorują inne treści. Oto ulepszony kod, który wykorzystuje to zachowanie. Pogrubiony kod oznacza fragmenty, które zostaną zignorowane przez przeglądarki rozpoznające element `<datalist>`.

```
<legend>Jakie zwierzę lubisz najbardziej?</legend>
    <datalist id="animalChoices">
    <span class="Label">Wybierz zwierzę:</span>
      <select id="favoriteAnimalPreset">
        <option label="Koń" value="koń">
        <option label="Zebra" value="zebra">
        <option label="Kot" value="kot">
        <option label="Komar" value="komar">
        <option label="Gąsienica" value="gąsienica">
        <option label="Anakonda" value="anakonda">
        <option label="Człowiek" value="człowiek">
        <option label="Słoń" value="słoń">
        <option label="Gnu" value="gnu">
        <option label="Gołąb" value="gołąb">
        <option label="Krab" value="krab">
      </select>
    <br>
    <span class="Label">Lub wpisz własnego ulubieńca: </span>
    </datalist>
    <input list="animalChoices" id="favoriteAnimal">
```

Po usunięciu pogrubionego kodu otrzymałbyś poprzednią zawartość dokumentu. Przeglądarki, które rozpoznają element `<datalist>`, wygenerują pojedyncze pole tekstowe i rozwijaną listę sugestii, podobną do tej z rysunku 4.13. Na innych przeglądarkach dodane komponenty umożliwią użytkownikom wpisanie zwierzaka lub wybranie go z listy (rysunek 4.14).



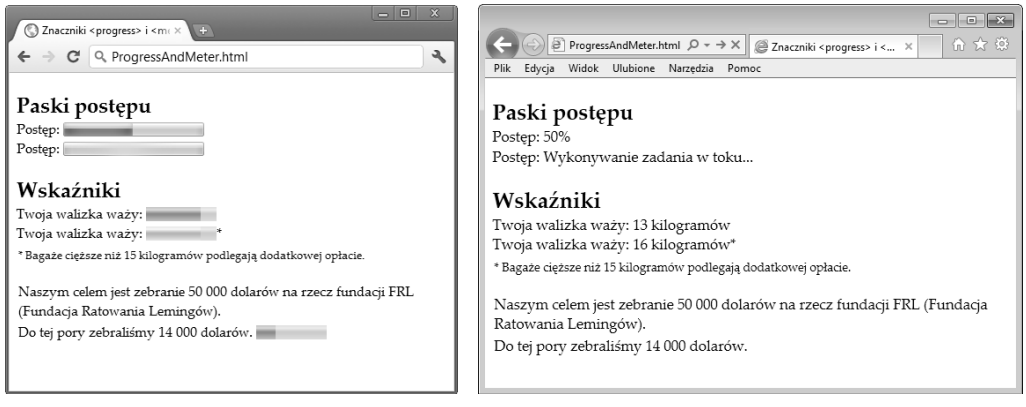
Rysunek 4.14. *Nawet pomimo braku wsparcia dla listy z sugestiami, nie musisz z niej rezygnować. Wystarczy, że osadzisz ją w tradycyjnym elemencie listy <select>*

Rozwiązanie nie jest doskonałe. Przed wysłaniem danych na serwer będziesz musiał sprawdzać zarówno wartości wybrane z tradycyjnej listy (tutaj o identyfikatorze `favouriteAnimalPreset`), jak i z pola tekstowego (identyfikator `favoriteAnimal`). Mimo tej małej wady, możesz łatwo dodać nowoczesne udogodnienie, bez zostawiania większości swoich użytkowników w tyle.

Uwaga: Na początku element `<datalist>` miał posiadać własność, która umożliwiałaby pobranie opcji z zewnętrznego źródła, np. serwera, który mógłby je wyciągnąć z bazy danych. Być może własność ta zostanie dodana w kolejnych wersjach HTML, lecz na razie podobną funkcję da się zapewnić za pomocą napisanego w JavaScripcie obiektu `XMLHttpRequest` (strona 322).

Pasek stanu i miernik

Elementy `<progress>` i `<meter>` są graficznie bardzo podobne, ale używa się ich w innych celach (rysunek 4.15).



Rysunek 4.15. *Elementy <progress> i <meter> są przydatnymi udogodnieniami, przynajmniej na przeglądarkach, które je rozpoznają (po lewej). Pozostałe przeglądarki po prostu wyświetlają zapisaną w nich wartość w formie tekstu (po prawej)*

Znacznik `<progress>` wskazuje postęp jakiegoś zadania. Jest to szary komponent w pewnej części wypełniony pulsującym, zielonym paskiem. Element `<progress>` przypomina bardzo paski stanu, z którymi spotkałeś się wcześniej (przypomnij sobie choćby pasek wyświetlany w systemie Windows w trakcie przesyłania plików), choć jego dokładny wygląd zależy od przeglądarki, na której otwarto stronę.

Element `<meter>` oznacza wartość w znanym zakresie. Po załadowaniu prezentuje się bardzo podobnie do komponentu `<progress>`, choć zawarty w nim zielony pasek ma trochę inny odcień i nie pulsuje. W zależności od przeglądarki pasek miernika może zmieniać kolor, gdy nadana mu wartość jest „za mała" lub „zbyt duża" — np. w drugim przypadku Chrome zmienia barwę paska na żółtą. Najważniejsza różnica pomiędzy dwoma znacznikami wynika z ich odmiennego interpretowania przez przeglądarkę.

Uwaga: Technicznie rzecz ujmując, nowe elementy `<meter>` i `<progress>` nie muszą się znajdować w obrębie formularza. W zasadzie nie są one nawet kontrolkami (ponieważ nie pobierają żadnych danych od użytkownika). Mimo to, oficjalne wytyczne HTML5 zaliczają je do grupy komponentów formularza, gdyż oba te znaczniki wyglądają jak widgety i służą do wyświetlania informacji w graficznej formie.

Aktualnie elementy `<progress>` i `<meter>` działają w Chrome (wersja 9. i późniejsze), Operze (wersja 11. i nowsze) oraz Safari (wersja 5.1 lub nowsze). Firefox rozpoznaje znacznik `<progress>` (ale nie `<meter>`), począwszy od wersji 6. Tylko Internet Explorer nie obsługuje obu.

Obydwoma komponentami łatwo się posługiwać. Najpierw przyjrzyj się znacznikowi `<progress>`. Przyjmuje on atrybut `value`, którego wartość wyrażona w ułamku *dziesiętnym* (od 0 do 1) ustawia procent postępu operacji (a tym samym długość zielonego wypełnienia). Można np. ustawić wartość parametru `value` na `0.25`, co odpowiada 25% ukończenia zadania.

```
<progress value="0.25"></progress>
```

Możesz też wykorzystać atrybut `max` i zmienić skalę paska. Gdybyś ustawił maksymalną wartość na `200`, wartość parametru `value` musiałaby się zmieścić między `0` a `200`. Dlatego też wpisanie do niego liczby `50` spowoduje wypełnienia paska stanu do `25%`, tak jak w poprzednim przykładzie.

```
<progress value="50" max="200"></progress>
```

Ustawienia skali są kwestią wygody. Użytkownik strony nie widzi wartości paska.

Uwaga: Element `<progress>` po prostu powoduje wyświetlenie cieniowanego paska postępu. Sam *nie robi* nic. Przypuśćmy, że chcesz użyć paska stanu do pokazania stopnia ukończenia pewnego zadania (np. przy użyciu omówionych na stronie 359 własności pracowników). Do Ciebie należy napisanie kodu, który wykryje element `<progress>` i zmieni jego wartość.

Przeglądarki, które nie rozpoznają elementu `<progress>`, po prostu go ignorują. Częściowym rozwiązaniem tego problemu jest podanie awaryjnej wartości, co robi się w taki oto sposób:

```
<progress value="0.25">25%</progress>
```

Pamiętaj, że zawarta między znacznikami treść nie pojawi się na przeglądarkach, które *rozpoznają* ten komponent.

Paska postępu można użyć również w inny sposób. Można wyświetlić *nieoznaczony* pasek postępu, który wskazuje, że zadanie jest wykonywane, lecz nie wiadomo, kiedy się zakończy (w takim przypadku myśl o pasku jak o wymyślnym sposobie przekazania wiadomości „ładowanie w toku"). Nieoznaczony pasek postępu wygląda jak zwykły szary słupek, po którym co pewien czasu porusza się zielony błysk, od lewej do prawej. Aby utworzyć taki element, po prostu pomiń atrybut `value` w deklaracji:

```
<progress>Wykonywanie zadania w toku... </progress>
```

Element `<meter>` korzysta z podobnego modelu, lecz wskazuje pewną miarę. Komponent ten czasem nazywany jest **miarką** lub **wskaźnikiem**. Często dokładna wartość wskaźnika będzie odpowiadać rzeczywistym pomiarom (np. sumom pieniężnym, liczbie dni, wadze itd.). Sposób, w jaki znacznik `<meter>` prezentuje te

informacje, można zmodyfikować z wykorzystaniem wartości maksymalnych i minimalnych (czyli atrybutów `min` i `max`).

```
Twoja walizka waży: <meter min="2" max="40" value="">13
kilogramów</meter>
```

Tak jak ma to miejsce w elemencie `<progress>`, zawartość elementu `<meter>` jest wyświetlana tylko wtedy, gdy przeglądarka go nie rozpoznaje. Oczywiście, mogą zdarzyć się sytuacje, kiedy warto pokazać dokładną liczbę, którą komponent ma odzwierciedlać. W takim przypadku należy ją po prostu dodać na stronie samemu — nie trzeba wtedy polegać na awaryjnej treści. W następującym przykładzie zaprezentowano, jak to zrobić. Metoda ta pozwala zawrzeć wszelkie informacje i dodaje pasek wskaźnika na obsługujących go przeglądarkach.

```
<p>Naszym celem jest zebranie 50 000 dolarów na rzecz fundacji FRL
➥(Fundacja Ratowania Lemingów).</p>
<p>Do tej pory zebraliśmy 14 000 dolarów. <meter max="50000"
value="14000"></meter>
```

Element `<meter>` jest również wyposażony w gadżety, które umożliwiają oznaczenie zbyt wysokich lub niskich wartości, przy jednoczesnym prawidłowym ich wyświetlaniu. W tym celu używa się atrybutów `low` i `high`. Przykładowo wartość parametru `value`, która przekracza liczbę w atrybucie `high` (ale jest mniejsza od maksimum), jest uznana za zbyt wysoką, lecz dozwoloną. Podobnie wartość poniżej dołączonej do atrybutu `low` liczby zostanie oznaczona jako zbyt niska:

```
Twoja walizka waży: <meter min="2" max="40" high="15" value="16">16
➥kilogramów</meter>* <p><small>* Bagaże cięższe niż 15 kilogramów
➥podlegają dodatkowej opłacie.  </small></p>
```

Przeglądarki mogą wykorzystać te dodatkowe atrybuty lub nie. I tak Chrome zmienia kolor paska na żółty dla zbyt wysokich wartości (jak w poprzednim przykładzie). Nie reaguje jednak przy zbyt małych liczbach. Na koniec, możesz oznaczyć pewną liczbę jako optymalną wartość za pomocą atrybutu `optimum`, lecz nie zmieni to sposobu ładowania komponentu na stronie we współczesnych przeglądarkach.

Podsumowując, można stwierdzić, że znaczniki `<progress>` i `<meter>` są drobnymi fajerwerkami, które staną się użyteczne, gdy więcej przeglądarek zacznie je rozpoznawać.

Paski narzędzi i menu — znaczniki <command> i <menu>

Prawdopodobnie wśród wszystkich własności, które nie zostały zaimplementowane, elementy `<command>` i `<menu>` są najważniejsze. Podstawą tych dwóch elementów jest koncepcja odzwierciedlenia akcji uruchamianych przez użytkownika (w komponencie `<command>`) i zgrupowania ich w jednym miejscu (przy użyciu znacznika `<menu>`). W zależności od podejścia do problemu oraz użytych sztuczek formatowania, element `<menu>` mógłby przekształcić się we wszystko: od paska narzędzi osadzonego obok krawędzi okna, po wyskakujące menu, które pojawia się, gdy użytkownik kliknie gdzieś na stronie. Niestety, obecnie żadna przeglądarka nie rozpoznaje tych komponentów, więc przyjdzie poczekać, nim będzie Ci dane sprawdzić, czy będą tak wspaniałe, jak spodziewają się twórcy stron.

Edytor HTML na stronie

Jak przekonałeś się w rozdziale 1., twórcy HTML5 obrali sobie za cel brukowanie ścieżek — innymi słowy, włączenie do specyfikacji HTML5 często wykorzystywanych przez współczesnych deweloperów, aczkolwiek niestandardowych, rozwiązań. Chyba najlepszym przykładem tego procesu są dwa dziwaczne atrybuty `contentEditable` i `designMode`, które pozwalają przekształcić zwykłą przeglądarkę w edytor HTML-u.

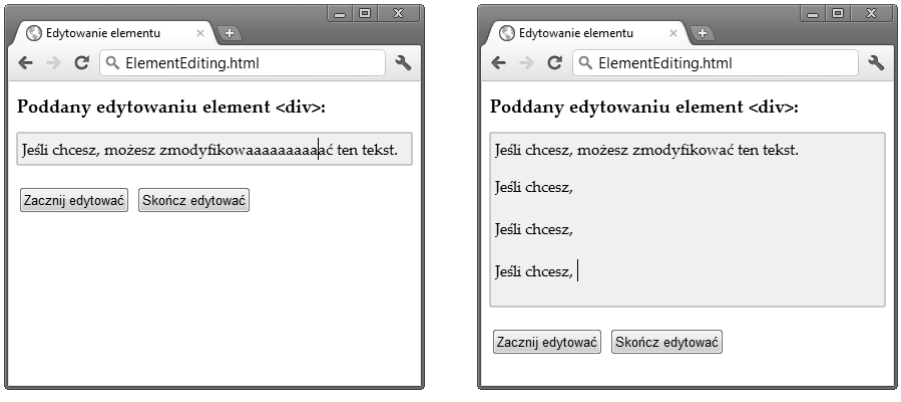
Te dwa atrybuty nie są nowością. W rzeczywistości dodano je do Internet Explorera 5, w ciemnych wiekach Internetu. W owym czasie większość twórców stron uznała je za sieciowe rozszerzenie przeznaczone wyłącznie dla systemu Windows i dlatego je zignorowano. Z biegiem czasu coraz więcej przeglądarek obrało praktyczne, choć niepozbawione wad, podejście do edytowania kodu HTML. Obecnie wszystkie przeglądarki na komputery stacjonarne obsługują te atrybuty, mimo że nigdy wcześniej nie były częścią żadnego oficjalnego standardu.

Edytowanie zawartości za pomocą contentEditable

Pierwszym narzędziem do dynamicznego edytowania HTML-u jest parametr `contentEditable`. Wystarczy wpisać parametr w dowolnym znaczniku, aby można było go edytować.

```
<div id="editableElement" contentEditable>Jeśli chcesz, możesz
↪zmodyfikować ten tekst.</div>
```

W pierwszej chwili zapewne nie zauważysz żadnej różnicy. Jeżeli jednak załadujesz stronę z tym kodem i klikniesz wewnątrz elementu `<div>`, pojawi się kursor pisania tekstu, zwany też **karetką** (rysunek 4.16).



Rysunek 4.16. *Po kliknięciu edytowalnego obszaru będziesz mógł się po nim poruszać, używając klawiszy strzałek, oraz kasować tekst lub wpisywać nową treść (po lewej). Możesz też zaznaczyć słowa, przytrzymując klawisz Shift i dowolnie je kopiować, wycinać i wklejać (po prawej). Przypomina to pisanie w edytorze tekstu, z tą różnicą, że użytkownik nie może wyjść poza obręb elementu i zmodyfikować treść reszty strony*

PRZYSPIESZAMY

Kiedy warto implementować funkcję edytowania HTML-u?

Nim wypróbujesz funkcję edytowania HTML-u, warto zadać pytanie, czemu ma ona służyć. Należy pamiętać, że edytowanie HTML-u jest specjalistyczną własnością, której użycie jest kontrowersyjne. Umieszczanie jej na stronie ma sens, jeśli chcesz pozwolić użytkownikowi szybko i bezboleśnie zmodyfikować treść, aby np. dodać posty na blogu, wpisać własne recenzje, umieścić dozwolone reklamy lub wpisywać wiadomości przeznaczone dla innych użytkowników.

Jeśli nawet okaże się, że taka opcja jest potrzebna, atrybuty `contentEditable` i `designMode` mogą nie być najlepszym wyborem. Wynika to z prostego faktu, że żaden z tych parametrów nie oferuje opcji dostępnych w prawdziwych narzędziach projektowania stron, w rodzaju zmieniających kod instrukcji, możliwości edytowania dokumentu źródłowego HTML, sprawdzania pisowni itp. Z drugiej strony, posługując się nimi, przy odrobinie wysiłku *da się* skonstruować o wiele bardziej złożony edytor. Jeśli jednak naprawdę potrzebujesz funkcji edytowania strony, może lepiej będzie użyć gotowego rozwiązania, które po prostu do niej dołączysz? Informacje o najpopularniejszych opcjach znajdziesz na blogu *http://ajaxian.com/archives/richtexteditors-compared*.

W tym przykładzie oznaczony atrybutem `contentEditable` element `<div>` zawiera wyłącznie tekst. Nic nie stoi na przeszkodzie, aby umieścić w nim inny komponent. W zasadzie ten element `<div>` mógłby objąć całą stronę, co sprawi, że stanie się otwarta na wprowadzanie zmian przez użytkowników. Idąc dalej tym tropem, atrybut `contentEditable` możesz zadeklarować w kilku komponentach, dzięki czemu na stronie pojawi się wiele sekcji, jakie można będzie do woli modyfikować.

Wskazówka: Niektóre przeglądarki rozpoznają pewne kombinacje klawiszy. Przykładowo w przeglądarce Internet Explorer istnieje opcja pogrubienia, podkreślenia lub nałożenia kursywy na tekst, aktywowana wciśnięciem klawiszy (odpowiednio) — *Ctrl+B*, *Ctrl+U* oraz *Ctrl+I*. W podobny sposób, naciskając klawisze *Ctrl+Z*, można cofnąć ostatnio wykonaną operację w Firefoksie. Wszystkie te skróty działają w przeglądarce Chrome. Jeżeli chcesz dowiedzieć się więcej o komendach formatowania i poznać metodę tworzenia paska narzędzi, który mógłby je obsługiwać, przeczytaj dwuczęściowy artykuł opublikowany na witrynie Opery, pod adresami *http://tinyurl.com/htmlEdit1* i *http://tinyurl.com/htmlEdit2*.

Atrybut `contentEditable` rzadko deklaruje się bezpośrednio na stronie. Zamiast tego włącza się go przy użyciu JavaScriptu i wyłącza po dokonaniu zmian na stronie. Dwie poniższe funkcje powstały właśnie w tym celu:

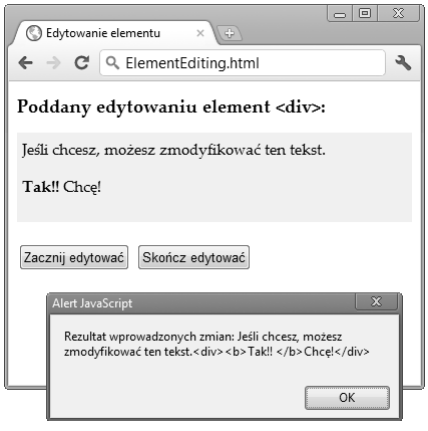
```
function startEdit() {
  // Włącza opcję edytowania wybranego elementu.
  var element = document.getElementById("editableElement");
  element.contentEditable = true;
}

function stopEdit() {
  // Przywraca wybrany element do poprzedniego stanu.
  var element = document.getElementById("editableElement");
  element.contentEditable = false;

  // Wyświetla kod w polu wiadomości.
  alert("Rezultat wprowadzonych zmian: " + element.innerHTML);
}
```

Oto dwa przyciski, w których je zaimplementowano:

```
<button onclick="startEdit()">Zacznij edytować</button>
<button onclick="stopEdit()">Skończ edytować</button>
```

Upewnij się, że nie umieściłeś przycisków w edytowanym obszarze — jeśli tak zrobiłeś, przestaną reagować na zdarzenia i nie będą mogły zostać użyte do uruchamiania kodu.

Na rysunku 4.17 przedstawiono wynik zmian po zmodyfikowaniu elementu i pogrubieniu tekstu (wszystko dzięki kombinacji klawiszy *Ctrl+B*).



Rysunek 4.17. *Rysunek jest dowodem, że edytowanie elementu naprawdę zmienia zawartość strony. W tym przykładzie treść umieszczono w wyskakującym okienku. Jednak profesjonalnie napisana strona wysłałaby tak otrzymane dane na sieciowy serwer, korzystając z opisanego na stronie 325 obiektu XMLHttpRequest*

Uwaga: W sposobie działania funkcji dynamicznego edytowania kodu HTML na stronie, można w zależności od przeglądarki wyróżnić wiele subtelnych różnic. Przykładowo wciśnięcie kombinacji klawiszy *Ctrl+B* na przeglądarce Chrome spowoduje dodanie elementu `<b>`, podczas gdy na Internet Explorerze będzie to komponent `<strong>`. Do podobnej sytuacji dochodzi po naciśnięciu klawisza *Enter* w celu dodania nowej linii lub klawisza *Backspace*, aby usunąć znacznik. Jednym z powodów standaryzacji własności dynamicznego edytowania dokumentów HTML5 jest chęć narzucenia spójnego modelu jej działania.

Edytowanie strony za pomocą atrybutu designMode

Własność `designMode` działa podobnie do parametru `contentEditable`, ale w odróżnieniu od niego pozwala na modyfikację całej strony. Może się to wydać trochę niedorzeczne. W końcu jeśli cała strona ma być poddana temu procesowi, jak użytkownik uzyska dostęp do kontrolujących edytowanie przycisków? Rozwiązaniem jest umieszczenie modyfikowanej strony wewnątrz elementu `<iframe>`, który działa jak pole edycji tekstu (rysunek 4.18).

Cały użyty w tym przykładzie kod jest wyjątkowy prosty. Oto zawartość elementu `<body>` tej strony:

```
<h1>Poddawana modyfikacji strona</h1>
<iframe id="pageEditor" src="ApocalypsePage_Revised.html"></iframe>
<div>
  <button onclick="startEdit()">Zacznij edytować</button>
  <button onclick="stopEdit()">Skończ edytować</button>
</div>

<h1>Zmodyfikowany kod HTML</h1>
<div id="editedHTML"></div>
```

Rysunek 4.18. *Na tej stronie umieszczono dwa pola. W pierwszym polu, utworzonym przez element <iframe>, widać omawianą w rozdziale 2. apokaliptyczną stronę. Drugie pole jest zwykłym elementem , który wyświetla kod dokumentu HTML po tym, jak został zmodyfikowany. Dwa przyciski pośrodku strony służą do przełączania komponentu <iframe> w tryb projektowania i z powrotem w tryb normalny*

Jak widać, podobnie jak na poprzednio omawianej stronie, przykład ten działa dzięki metodom `startEdit()` i `stopEdit()`. Zmodyfikowano jednak kod JavaScript po to, żeby dodawał atrybut `designMode` zamiast `contentEditable`:

```
function startEdit() {
  // Włącza tryb projektowania w elemencie <iframe>.
  var editor = document.getElementById("pageEditor");
  editor.contentWindow.document.designMode = "on";
}

function stopEdit() {
  // Wyłącza tryb projektowania w elemencie <iframe>.
  var editor = document.getElementById("pageEditor");
  editor.contentWindow.document.designMode = "off";

  // Wyświetla zmodyfikowany kod HTML (po to tylko, aby pokazać, że wciąż tam jest).
  var htmlDisplay = document.getElementById("editedHTML");
  htmlDisplay.textContent =
editor.contentWindow.document.body.innerHTML;
}
```

Ten przykład pozwoli Ci lepiej poznać skalę możliwości edytowania dokumentów HTML. Zauważ, jak po kliknięciu grafiki na stronie przeglądarka pozwala dowolnie nią manipulować. Możesz zmieniać jej rozmiar, przemieszczać po stronie lub usunąć ją jednym kliknięciem lub wciśnięciem klawisza *Delete*. W podobny sposób da się modyfikować kontrolki formularza, jeśli są obecne na edytowanej stronie.

Naturalnie, nim będziesz zdolny zmienić ten przykład w coś działającego bardziej praktycznie, musisz pokonać istotną lukę. Najpierw warto by było dodać lepsze kontrolki edytowania. Po raz kolejny na ratunek przychodzi zespół twórców Opery, którzy wesprą Cię w poznawaniu modelu kontroli wykraczającego poza zakres tego rozdziału (odwiedź strony *http://tinyurl.com/htmlEdit1* oraz *http://tinyurl.com/htmlEdit2*). Poza tym przydałoby się zrobić coś pożytecznego ze zmienionym przez Ciebie kodem, np. wysłać go na serwer za pośrednictwem obiektu `XMLHttpRequest` (więcej szczegółów na stronie 322).

Należy też wspomnieć o jednej ważnej rzeczy. Jeśli uruchomisz ten przykład z dysku twardego Twojego własnego komputera, może nie działać w niektórych przeglądarkach (Internet Explorer i Chrome natrafiają na ograniczenia bezpieczeństwa, podczas gdy Firefox śmiga bez większych problemów). Problemu tego możesz uniknąć, uruchamiając ten kod na utworzonej przez nas stronie *www.prosetech.com/html5*.

Skorowidz

E

F

G

H

N

O

P

Q

R

S

Ś

T

U

V

W

HTML5

nieoficjalny podręcznik

HTML5 to coś więcej niż język służący do tworzenia stron WWW — to zbiór kilkunastu niezależnych standardów sieciowych pod jednym wspólnym szyldem. Z rozmachem wkracza do codziennego życia projektantów stron internetowych. Jego nowe możliwości naprawdę robią wrażenie: obejmują ścisłą integrację ze środowiskiem przeglądarki internetowej, usługi geolokalizacyjne, doskonałe wsparcie dla multimediów i aplikacji offline. Jeszcze niedawno takie możliwości nie śniły się żadnym webmasterom, a dziś są w zasięgu każdego!

Dzięki kolejnej książce z serii *Nieoficjalny podręcznik* nie musisz odkrywać tajników HTML5 na własną rękę. Znajdziesz tu wszystkie istotne informacje, dzięki którym błyskawicznie zaczniesz korzystać z dobrodziejstw HTML5. W trakcie lektury nauczysz się dynamicznie rysować elementy, używać geolokalizacji oraz przechowywać dane użytkowników w lokalnych magazynach danych. Ponadto poznasz nowe znaczniki oraz ich przeznaczenie. HTML5 to przyszłość sieci, dlatego już dziś warto poznać jego możliwości!

HTML5 to:

- wsparcie dla plików multimedialnych
- usługi geolokalizacyjne
- wygodne przechowywanie danych
- aplikacje offline
- standard nowoczesnej sieci Internet!

Sprawdź, jak będzie wyglądać sieć jutra, i zacznij korzystać z tego już dziś!

helion.pl
księgarnia internetowa

Nr katalogowy: 8629

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

ISBN 978-83-246-3948-9

Cena 69,00 zł

Informatyka w najlepszym wydaniu