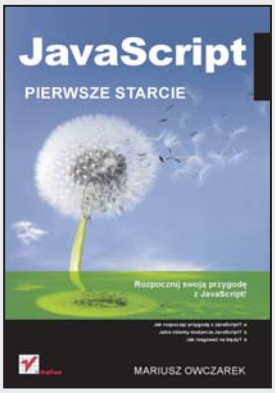
JavaScript. Pierwsze starcie



Autor: Mariusz Owczarek
ISBN: 978-83-246-2076-0
Format: 158×235, stron: 144

Rozpocznij swoją przygodę z JavaScript!

- Jak rozpocząć przygodę z JavaScript?
- Jakie obiekty dostarcza JavaScript?
- Jak reagować na błędy?

Język JavaScript, choć ma już blisko dwanaście lat, swoimi możliwościami wciąż potrafi zafascynować niejednego projektanta stron internetowych. Ma już za sobą gorsze dni, jednak aktualnie dzięki technologii AJAX znów jest na topie. Wykorzystując go w odpowiedni sposób, sprawisz, że twój serwis WWW stanie się bardziej interaktywny i dynamiczny.

Ta książka pozwoli Ci wyjść zwycięsko z pierwszego starcia z tym językiem! Dowiesz się z niej, jak używać zmiennych, operatorów oraz funkcji. Nauczysz się reagować na zdarzenia oraz wykorzystywać okna dialogowe. Ponadto zdobędziesz wiedzę na temat pracy z obiektami DOM HTML oraz na temat sposobów reagowania na błędy w skryptach. Autor przedstawia tu także dostępne obiekty JavaScript oraz pokazuje, jak wykonywać operacje związane z czasem. Ogromnym atutem tej książki jest przejrzystość i usystematyzowany sposób prezentowania informacji. Dzięki temu również Ty szybko i bezboleśnie poznasz JavaScript!

- Typowe konstrukcje języka JavaScript
- Wykorzystanie zmiennych
- Zastosowanie funkcji
- Reagowanie na zdarzenia
- Sposoby użycia okien dialogowych
- Wykonywanie operacji związanych z czasem
- Dostępne obiekty JavaScript
- Obiekty DOM HTML
- Przygotowanie własnych obiektów
- Dziedziczenie w JavaScript
- Obsługa błędów

Przejdź bezboleśnie pierwsze starcie z JavaScript!

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl

Spis treści

Rozdział 9.

Praca z dokumentami DOM HTML i XML

Drzewo obiektów

Jak już wspomniałem w poprzednim rozdziale, specyfikacja DOM pozwala na traktowanie strony Web jako zbioru elementów. Elementy tworzą drzewo, podobnie jak pliki i foldery na dysku. Zamiast plików i folderów mamy tu węzły (node) i elementy. W przypadku dokumentu HTML elementami są obiekty HTML. Zamiast tworzyć je bezpośrednio w dokumencie, można utworzyć je za pomocą skryptu, uzyskując dynamiczną stronę Web. To wszystko może być połączone z pobieraniem danych z dokumentów XML.

Funkcje i właściwości obiektu document do zarządzania elementami

Elementy HTML są częścią dokumentu, który jest reprezentowany przez obiekt `document`, dlatego obiekt ten zwiera funkcje do tworzenia elementów i węzłów. Będziemy używać dwóch funkcji pokazanych w tabeli 9.1.

Tabela 9.1. *Funkcje obiektu document do tworzenia elementów*

Funkcja	Opis
`createElement(nazwa)`	Tworzy element, parametr nazwa to nazwa znacznika HTML podana w cudzysłowie, np. `„div”`, `„table”`, `„p”` itd.
`createTextNode(tekst)`	Tworzy węzeł tekstowy zawierający tekst `tekst`.

Element jest obiektem typu element i po utworzeniu nie jest związany do dokumentem, tzn. że nie jest umieszczony od razu w drzewie obiektów dokumentu. Jest zmienną tak jak inne zmienne w JS. Aby ustawić go w drzewie lub zmienić jego parametry (na przykład wygląd tabeli), trzeba użyć funkcji będących częścią elementu. Funkcje obiektu element przedstawia tabela 9.2.

Tabela 9.2. *Funkcje obiektu element*

Nazwa	Opis
appendChild(element)	Dodaje element jako podrzędny do elementu, na którym wywołujemy tę funkcję.
cloneNode(parametr)	Kopiuje istniejący element lub węzeł. Funkcja zwraca kopie elementu, na którym została wywołana. Jeśli parametr ma wartość false, to kopiowany jest tylko dany element, jeśli tru — to cała gałąź drzewa, do którego należy.
removeChild(element)	Usuwa element z drzewa. Jako parametr podajemy konkretny obiekt element, który ma być usunięty.
applyElement(element)	Dodaje element jako nadrzędny do elementu, na którym została wywołana ta funkcja.
setAttribute(nazwa atrybutu, wartości)	Ustawia właściwość elementu podaną w parametrze nazwa atrybutu na wartość. Ponieważ do formatowania zaleca się stosowanie stylów, najczęściej będziemy używać tej funkcji do przyporządkowania elementowi identyfikatora id, określonego w arkuszu stylów.

Będziemy też wykorzystywać dwie własności obiektu element, które służą do sprawdzania elementów podrzędnych w stosunku do danego. Własności te zawiera tabela 9.3.

Tabela 9.3. *Własności obiektu element*

Własność	Opis
firstChild	Obiekt, który został dołączony do danego obiektu jako pierwszy obiekt podrzędny.
childNodes[]	Tabela zawierająca wszystkie obiekty bezpośrednio podporządkowane danemu.
nodeName	Nazwa węzła.
nodeType	Typ węzła elementu. Jest on identyfikowany liczbą całkowitą. Kilka możliwych wartości to: 1 — węzeł prowadzi do innego elementu; 3 — węzeł tekstowy; 8 — węzeł komentarza, np. komentarz w dokumencie HTML; 9 — węzeł prowadzi do innego dokumentu.
nodeValue	Wartość węzła, np. tekst w węźle tekstowym.

Tych kilka funkcji wystarczy do tworzenia elementów na stronie Web. Czas na przykład. Stworzymy stronę z paragrafem <p> i obiektem , oczywiście tworzonymi nie poprzez kod HTML, ale za pomocą naszych funkcji.

Dodawanie elementów do dokumentu

Przykład 9.1. Wstawianie podstawowych elementów do drzewa strony

1. Zaczynamy już tradycyjnie od podstawowego pliku HTML.
2. Sekcję <body> wyposażymy w identyfikator, ponieważ będziemy się na niego powoływać, „dowiązując" do sekcji <body> kolejne elementy HTML.

   ```
   <body id = "body1">
   ```

3. W sekcji <body> napiszemy skrypt — wyjaśnię go linia po linii. Rozpoczynamy jak zwykle:

   ```
   <script type = "text/javascript">
   ```

   Teraz za pomocą funkcji `createElement()` utworzymy paragraf. Jako parametr funkcji podajemy rodzaj obiektu HTML bez znaków <>:

   ```
   var paragraf = document.createElement("p");
   ```

   Zmienna paragraf reprezentuje obiekt paragrafu. Tekst do paragrafu będzie w postaci węzła tekstowego:

   ```
   var tekst1 = document.createTextNode("Tekst w paragrafie");
   ```

   Na razie te elementy nie są ze sobą powiązane. Aby „wpisać" tekst do paragrafu, trzeba ustawić węzeł tekstowy jako element podrzędny w stosunku do paragrafu:

   ```
   paragraf.appendChild(tekst1);
   ```

   Pozostaje jeszcze dowiązanie paragrafu do ciała strony, musimy tu użyć znanej już funkcji `getElementById()`, aby odwołać się do sekcji <body>:

   ```
   document.getElementById("body1").appendChild(paragraf);
   ```

4. Paragraf już gotowy, teraz zajmiemy się elementem . Tworzymy go analogicznie, podam od razu cały kod:

   ```
   var div1 = document.createElement("div");
   var tekst2 = document.createTextNode("Tekst we wnętrzu div")
   div1.appendChild(tekst2);
   document.getElementById("body1").appendChild(div1);
   ```

5. Zapisz plik i otwórz go w przeglądarce, otrzymasz tekst w paragrafie i tekst w div jak na rysunku 9.1. Na razie może nie wygląda to zbyt efektownie, ale to dopiero początek możliwości.

Rysunek 9.1. *Elementy utworzone dynamicznie na stronie*

Aby nadać obiektom kolor i kształt, można użyć funkcji `setAttribute()` bezpośrednio do ustawienia tych parametrów. Nowocześniej jednak jest zdefiniować arkusz stylów, a funkcji setAttribute() użyć tylko do ustawienia parametru `id`, tak aby obiekt był rysowany z użyciem stylu z formularza. Jak to zrobić, pokazuje następny przykład.

Zmiana parametrów elementów

Przykład 9.2. Przyporządkowywanie stylów do obiektów funkcją setAttribute()

1. Zaczniemy z plikiem z poprzedniego przykładu.
2. Na początku sekcji nagłówkowej zdefiniujemy arkusz stylów CSS z dwoma stylami. Jeden, `pstyl1`, przeznaczony będzie dla paragrafu, a drugi, `divstyl1`, dla obiektu div.

```
<style>
#pstyl1 {
  background-color: blue;
}
#divstyl1 {
float:left;color:black;background-color:yellow;width:20%;
}
</style>
```

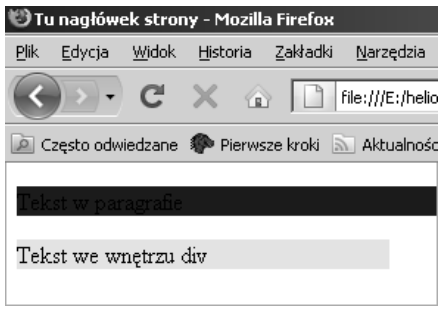
3. W skrypcie w sekcji <body> na obiektach `paragraf` i `div1` wywołujemy funkcje `setAttribute()`. Za pomocą tej funkcji ustawiamy identyfikator `id` dla obiektów na wartość `pstyl1` dla paragrafu i `divstyl1` dla div. Spowoduje to użycie stylu z arkusza dla tych elementów. Miejsce wywołania tej funkcji w zasadzie nie ma znaczenia, ja umieściłem ją po dodaniu elementu do ciała strony. I tak dla paragrafu:

```
document.getElementById("body1").appendChild(paragraf); //ta linia istnieje
paragraf.setAttribute("id","pstyl1"); //tu zmieniamy parametr id
```

   a dla obiektu div:

```
document.getElementById("body1").appendChild(div1); //ta linia istnieje
div1.setAttribute("id","divstyl1");//tu zmieniamy parametr id
```

4. I to już wszystko, zapisz plik i otwórz go w przeglądarce. Wynik jest przedstawiony na rysunku 9.2.



Rysunek 9.2. *Elementy z formatowaniem*

Parametr `id` można zmieniać w czasie działania skryptu, a co za tym idzie — można zmieniać wygląd elementu w bardzo prosty sposób. Pokaże to następny przykład.

Przykład 9.3. Zmiana stylu elementu

1. Zaczniemy od pliku z poprzedniego przykładu.
2. Do arkusza stylów dodaj drugi styl dla paragrafu i obiektu div, aby było w czym wybierać. Oto cały uzupełniony arkusz:

```
<style>
#pstyl1 {
  background-color: blue;
}
#pstyl2 {
  background-color: green;
}
#divstyl1 {
float:left;color:black;background-color:yellow;width:20%;
}
#divstyl2 {
float:left;color:black;background-color:lightgreen;width:40%;
}
</style>
```

3. W dalszym ciągu sekcji `<head>` trzeba napisać funkcję zmieniającą parametr `id` elementów w zależności od ustawienia list rozwijalnych, które zaraz umieścimy na stronie.

```
<script type="text/javascript">
function zmien(element1,element2)
{
var stylpara = new Array(5);
var styldiv = new Array(5);
stylpara[0] = "pstyl1"
stylpara[1] = "pstyl2"
styldiv[0] = "divstyl1"
styldiv[1] = "divstyl2"
element1.setAttribute("id",stylpara[document.getElementById("lista1").sele
➥ctedIndex])
element2.setAttribute("id",styldiv[document.getElementById("lista2").
➥selectedIndex])
}
</script>
```

   Lista nazw stylów, czyli możliwych parametrów `id` dla odpowiednich elementów, jest przechowywana w oddzielnych tabelach. Przy zmianie stylu powołujemy się na indeks w tabeli, numer tego indeksu będzie ustawiony na liście rozwijalnej.

4. W sekcji `<body>`, poza kodem skryptu, dodaj formularz z dwiema listami rozwijalnymi i przyciskiem. Na listach będziemy ustawiać żądany wygląd paragrafu i diva.

```
<form>
<select id="lista1">
<option value="Op1"> Styl 1</option>
<option value="Op2"> Styl 2</option>
```

```
</select>
<select id="lista2">
<option value="Op1"> Styl 1</option>
<option value="Op2"> Styl 2</option>
</select>
<input type = "button" id = "przycisk" value = "Zmień styl">
</form>
```

5. Na końcu skryptu w sekcji <body> umieszczamy obsługę zdarzenia naciśnięcia przycisku.

```
var odn = document.getElementById("przycisk");
if (odn.addEventListener) {odn.addEventListener("click",function()
{zmien(paragraf,div1);},true);}
else if (odn.attachEvent) {odn.attachEvent("onclick",function()
{zmien(paragraf,div1);});}
```

   Po naciśnięciu przycisku będzie wywoływana funkcja `zmien()`. Parametrami tej funkcji są elementy strony, tu paragraf i div.

6. Zapisz plik i otwórz w przeglądarce. Zmieniając wybór na liście i klikając przycisk, zmieniasz styl elementów.

Usuwanie elementów z drzewa

Podobnie jak można elementy dodać, można je także usunąć. Nie jest to całkowite usunięcie, ale przerwanie łączności z drzewem. I tak na przykład odłączenie węzła tekstowego od paragrafu spowoduje zniknięcie tekstu. Zróbmy przykład.

Przykład 9.4. Odłączanie elementów z drzewa

1. Zaczniemy od podstawowego dokumentu HTML z ćwiczenia 1.1.
2. W nagłówku <head> umieścimy funkcję, która będzie odłączała od drzewa całego dokumentu element podany jako parametr.

```
<script>
function kasuj(element) {
document.getElementById("body1").removeChild(element);
}
</script>
```

3. Do sekcji <body> dokumentu dodajemy identyfikator `id`.

```
<body id="body1">
```

4. Na początku tej sekcji umieścimy formularz z jednym przyciskiem.

```
<form>
<input type = "button" id = "przycisk" value = "Skasuj">
</form>
```

5. Teraz czas na skrypt. We wnętrzu skryptu najpierw tworzymy element div i dodajemy do niego węzeł tekstowy.

```
<script type="text/javascript">var div1 = document.createElement("div");
var tekst2 = document.createTextNode("Tekst we wnętrzu div")
```

```
div1.appendChild(tekst2);
document.getElementById("body1").appendChild(div1);
```

Następnie dodajemy obsługę zdarzenia naciśnięcia przycisku i kończymy skrypt.

```
var odn=document.getElementById("przycisk");
if (odn.addEventListener) {odn.addEventListener("click",function()

{kasuj(div1);},true);}
else if (odn.attachEvent) {odn.attachEvent("onclick",function()
{kasuj(div1);});}
</script>
```

Naciśnięcie przycisku wyzwoli funkcję `kasuj()`, przy czym jako element do skasowania podajemy utworzony div.

6. Zapisz plik i otwórz w przeglądarce. Po naciśnięciu przycisku div zostanie odłączony od drzewa dokumentu i zniknie.

Budowanie tabeli

Czasem na stronie Web chcemy zaprezentować dane w formie tabeli, przy czym tabela ta musi mieć różną długość i pobierać dane z zewnętrznego źródła, na przykład dokumentu XML. Pokażę najpierw, jak zbudować taką tabelę, wykorzystując znane już funkcje obiektu document, a w dalszym ciągu — jak zaimportować dane z XML.

Tabelę budujemy podobnie jak elementy strony pokazane w poprzednich przykładach, z tym że drzewo obiektów tabeli jest bardziej skomplikowane. Weźmy na początek prostą tabelę 2×2 pola, pokazaną w tabeli 9.4.

Tabela 9.4. *Przykładowa tabela na stronie Web*

Pole 1,1	Pole 2,1
Pole 1,2	Pole 2,2

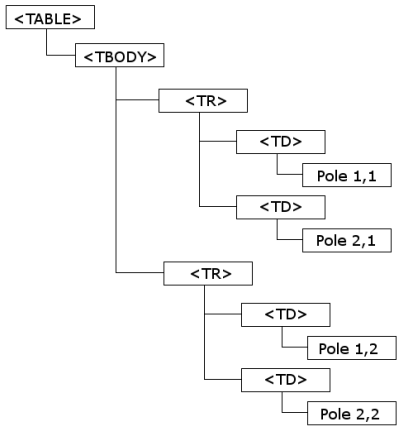
Jak pewnie wiesz, tabele w HTML tworzymy za pomocą znacznika `<table>`. Tabela 9.1 zapisana w HTML w tradycyjny sposób będzie miała postać:

```
<table>
<tr>
<td>Pole 1,1</td><td> Pole 2,1 <td>
</tr>
<tr>
<td>Pole 1,2</td><td>Pole 2,2</td>
</tr>
</table>
```

My narysujemy tę tabelę na stronie, budując jej drzewo. Wygląd tego drzewa przedstawia rysunek 9.3. Jest ono, oczywiście, zbudowane ze znanych elementów HTML.

Takie właśnie drzewo zbudujemy za pomocą poznanych funkcji, zrobimy to w następnym przykładzie. Tak skonstruowana tabela może być dowolnie zmieniana podczas działania skryptu.

Rysunek 9.3.
Drzewo elementów w tabeli 9.1



Przykład 9.5. Budowa tabeli

1. Rozpoczynamy jak zwykle — od podstawowego pliku HTML.
2. Nadajemy identyfikator sekcji `<body>`, tak jak w poprzednich przykładach.

   ```
   <body id = "body1">
   ```

3. Teraz czas na skrypt tworzący tabelę. Objaśnię kolejne linie — należy je wpisywać w ciągu skryptu. Zaczynamy standardowo:

   ```
   <script type="text/javascript">
   ```

   Następnie tworzymy zasadniczy element tabeli:

   `var tabela = document.createElement("TABLE");` Oprócz tego potrzebny jest element `<tbody>`, którego w tradycyjnej składni zwykle nie wypisujemy jawnie, ale tu musimy:

   ```
   var tabbody=document.createElement("TBODY");
   ```

   Łączymy element `TABLE` z `TBODY` — zgodnie z rysunkiem 9.3:

4. `tabela.appendChild(tabbody);` Są już ramy, teraz pierwszy wiersz tabeli.

   ```
   var wiersz1=document.createElement("TR");
   ```

   I pierwsze pole w tym wierszu:

   ```
   var pole11 = document.createElement("TD")
   ```

   Napis w polu jest w postaci węzła tekstowego:

   ```
   var tekst11= document.createTextNode("Pole 1,1");
   ```

   Węzeł tekstowy jest podrzędnym elementem pola, a pole dołączamy do pierwszego wiersza tabeli:

   ```
   pole11.appendChild(tekst11);
   wiersz1.appendChild(pole11);
   ```

W taki sam sposób tworzymy drugie pole i dołączamy do pierwszego wiersza:

```
var pole21 = document.createElement("TD")
var tekst21= document.createTextNode("Pole 2,1");
pole21.appendChild(tekst21);
wiersz1.appendChild(pole21);
```

Cały zaś wiersz dołączamy do tabeli:

```
tabbody.appendChild(wiersz1);
```

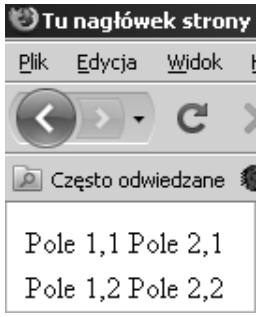
5. Drugi wiersz z polami tworzymy tak samo.

```
var wiersz2=document.createElement("TR");
var pole12 = document.createElement("TD")
var tekst12= document.createTextNode("Pole 1,2");
pole12.appendChild(tekst12);
wiersz2.appendChild(pole12);
var pole22 = document.createElement("TD")
var tekst22= document.createTextNode("Pole 2,2");
pole22.appendChild(tekst22);
wiersz2.appendChild(pole22);
tabbody.appendChild(wiersz2);
```

6. Na końcu całą tabelę dodajemy do sekcji <body>.

```
document.getElementById("body1").appendChild(tabela;
```

7. Wszystko już gotowe. Zapisz plik i otwórz go w przeglądarce. Powinieneś otrzyma obraz jak na rysunku 9.4.



Rysunek 9.4. *Tabela stworzona przez skrypt*

Jak widać, tabela nie ma obramowania. Wykonamy je za pomocą arkusza stylów CSS. Jest to bardzo proste, wystarczy przyporządkować styl border do wszystkich pól <TD> oraz samego elementu <TABLE>. Czyli formatujemy je tak samo jak tabele zapisane statycznie.

Przykład 9.6. Formatowanie tabeli

1. Zaczynamy od pliku z poprzedniego przykładu.
2. W sekcji <head> dopisujemy arkusz stylów, dodający ramkę o szerokości jednego piksela.

```
<style>
table, td {
```

```
    border: solid 1px;
  }
  </style>
```

3. Zapisz plik i otwórz w przeglądarce. Efekt będzie jak na rysunku 9.5.

Rysunek 9.5. *Tabela z prostym formatowaniem*

Na razie sens stosowania takich tabel może wydawać się cokolwiek mglisty, ponieważ stworzenie ich zajmuje o wiele więcej czasu niż tabel statycznych. Zalety tabel dynamicznych ujawnią się, kiedy będziemy je tworzyć z danych z zewnętrznego źródła, na przykład dokumentu XML.

Import danych z dokumentów XML

Dokument XML jest sposobem na uporządkowane przechowywanie danych. Drzewo danych jest zapisane za pomocą znaczników, podobnie jak dokument HTML. Za pomocą JavaScriptu można pobrać dane z dokumentów XML i wyświetlić praktycznie w dowolnej postaci na stronie Web. Aby to zrobić, najpierw wczytujemy dokument XML do pamięci w postaci elementów i węzłów DOM, a następnie wykonujemy operacje na tych węzłach znanymi już metodami. Niestety, sposób wczytania dokumentu XML jest różny dla różnych przeglądarek. Tutaj pokażę metodę dla Firefoksa i Internet Explorera.

Wczytywanie dokumentów XML w Firefoksie

Wczytanie dokumentu ma dwa etapy. Najpierw tworzymy pusty obiekt dokument, a następnie do tego dokumentu wczytujemy plik z danymi w języku XML. Po wczytaniu można już wyświetlić elementy na stronie Web.

Wczytany dokument XML będzie, oczywiście, w drzewie głównego dokumentu HTML, czyli obiektu dokument. Utworzymy go za pomocą interfejsu `implementation` obiektu `document`. Interfejs ten zawiera trzy funkcje, które zestawiono w tabeli 9.5.

Tabela 9.5. *Funkcje interfejsu implementation obiektu document*

Funkcja	Opis
`createDocument(PrzestrzeńNazw, ↪IdDokumentu, TypDokumentu)`	Tworzy obiekt dokument w podanej przestrzeni nazw XHTML, z podanym identyfikatorem `IdDokumentu` i o typie `TypDokumentu`.
`hasFeature(NazwaModułu, ↪WersjaModułu)`	Sprawdza, czy dana wersja DOM wspiera określony moduł, na przykład HTML 2.0.
`createDocumentType(Nazwa, ↪IdentyfikatorPubliczny, ↪IdentyfikatorSystemowy)`	Tworzy obiekt typu dokumentu. Obiekt może być użyty jako trzeci argument funkcji `createDocument()`.

Do utworzenia pustego dokumentu posłużymy się funkcją `createDocument()`, przy czym jako argumenty podamy same wartości null, ponieważ nie zależy nam na razie na typie dokumentu. Po utworzeniu wczytamy plik XML za pomocą funkcji `load()` obiektu `document`. Funkcji tej jeszcze nie znasz, ale jej użycie jest bardzo proste, jedynym argumentem jest plik do wczytania. Przejdźmy do przykładu, napiszemy skrypt wyświetlający dowolny plik XML w postaci tabeli na stronie Web.

Przykład 9.7. Wczytanie pliku XML — wersja dla Firefoksa

1. Zaczynamy od standardowego dokumentu HTML. Potrzebny będzie także plik XML, z dowolnymi danymi, ja zrobiłem opis trzech samochodów. Poniższy listing wpisz do oddzielnego pliku tekstowego nazwanego *auta.xml*. Plik możesz napisać w Notatniku.

```
<?xml version="1.0" encoding="ISO-8859-2"?>
<samochody>
  <samochod>
     <marka>Opel</marka>
     <pojemnosc> 1.4 </pojemnosc>
     <kolor> zielony </kolor>
     <ile_osob> 5 </ile_osob>
  </samochod>
  <samochod>
     <marka>Audi</marka>
     <pojemnosc> 1.8 </pojemnosc>
     <kolor> szary </kolor>
     <ile_osob> 5 </ile_osob>
  </samochod>
  <samochod>
     <marka>Mercedes</marka>
     <pojemnosc> 2.0 </pojemnosc>
     <kolor> biały </kolor>
     <ile_osob> 5 </ile_osob>
  </samochod>
 </samochody>
```

2. Zasadnicza funkcja wpisująca zawartość pliku do tabeli będzie w sekcji nagłówkowej `<head>`, ale najpierw zaczniemy od skryptu w sekcji `<body>`. Po pierwsze, trzeba przypisać identyfikator sekcji `<body>`, ponieważ będzie potrzebny do stworzenia tabeli.

```
<body id="body1">
```

3. Skrypt w tej sekcji będzie wczytywał plik z dysku do pamięci jako obiekty. Najpierw tworzymy pusty obiekt `document` w drzewie głównego obiektu `document`, reprezentującego stronę Web. Teraz do tego nowego dokumentu podłączamy metodę obsługi zdarzenia `load`, zachodzącego w momencie wczytania treści dokumentu z pliku. W funkcji obsługi tego zdarzenia będzie tworzona tabela. W końcu wczytujemy plik *auta.xml* do wnętrza dokumentu za pomocą funkcji `load()`.

```
<script type="text/javascript">
var dokumentxml = document.implementation.createDocument("","",null)
dokumentxml.addEventListener("load", documentLoaded, false);
dokumentxml.load("auta.xml");
</script>
```

4. Czas na funkcję w sekcji `<head>` rysującą tabelę. W momencie wywołania tej funkcji mamy już wczytane dane z pliku XML do dodatkowego obiektu `document`. Z obiektu dokument przepiszemy te dane do zwykłej tablicy `Array`, a następnie tablicę tę wyświetlimy w tabeli HTML. Będę objaśniał skrypt w miarę pisania, należy go umieścić w sekcji `<head>` dokumentu HTML. Zaczynamy skrypt, a następnie definiujemy funkcję `documentLoaded()`.

```
<script type="text/javascript">
function documentLoaded() {
var dane = new Array();
dane = dokumentxml.getElementsByTagName("samochod");
```

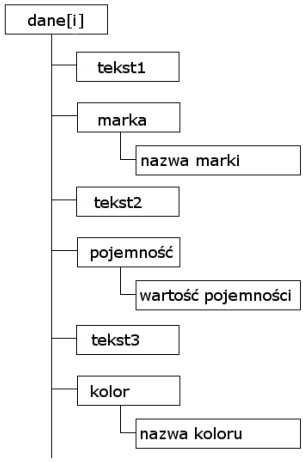
5. Wszystkie elementy zbioru *auta.xml* są już w tablicy `dane[]`, przy czym jedno pole tej tablicy zawiera dane zapisane między znacznikami `<samochod>` i `</samochod>` w pliku XML. Strukturę tablicy `dane[]` przedstawia rysunek 9.6. Teraz pozostaje wyświetlić odpowiednie komórki z tablicy `dane` w tabeli HTML, skonstruowanej identycznie jak w poprzednim przykładzie. Tym razem tabela będzie miała dodatkowo wiersz nagłówkowy. Wiersz taki zawiera się we wnętrzu elementu `THEAD`, pełniącego podobną funkcje jak `TBODY` dla pozostałych elementów tabeli.

```
var tabela=document.createElement("TABLE");
var tabhead=document.createElement("THEAD");
tabela.appendChild(tabhead);

var wierszng=document.createElement("TR");
for (var i=0;i<dane[0].childNodes.length;i++) {

   if(dane[0].childNodes[i].nodeType==1) {
      var poleng = document.createElement("TH");
      var tekstng= document.createTextNode(dane[1].childNodes[i].nodeName);
      poleng.appendChild(tekstng);
      wierszng.appendChild(poleng);
     }
}
tabhead.appendChild(wierszng);
```

Nie wszystkie węzły są wyświetlane, tylko te, które prowadzą do gałęzi węzłów tekstowych, tylko one reprezentują elementy ograniczone znacznikami `<marka>` `</marka>`, `<kolor>` `</kolor>` itp. Jak widać z rysunku 9.6, mamy w drzewie dodatkowe puste węzły tekstowe przy głównym pniu. Węzły te są dodawane wszędzie między znacznikami, dlatego są też na przykład między znacznikiem `</marka>` i `<pojemnosc>`. Są one niepotrzebne, ale nie da się ich uniknąć.



Rysunek 9.6. *Struktura pola tabeli dane[]*

6. Główną częęść tabeli wyświetlamy podobnie, z tym że interesuje nas wyświetlanie zawartości węzłów tekstowych, a nie nazw. Na końcu gotową tablicę dodajemy do sekcji <body>.

```
var tabbody=document.createElement("TBODY");

tabelaappendChild(tabbody);

for (var j=0;j<dane.length;j++) {
    var wiersz=document.createElement("TR");
    for (var i=0;i<dane[j].childNodes.length;i++) {
          if(dane[j].childNodes[i].nodeType==1) {
              var pole = document.createElement("TD");
              var tekst= document.createTextNode(dane[j].childNodes[i].
              ↪firstChild.nodeValue);
              pole.appendChild(tekst);
              wiersz.appendChild(pole);}
    }
    tabbody.appendChild(wiersz);
}
document.getElementById("body1").appendChild(tabela);
}
</script>
```

7. Aby poprawić wygląd tabeli, na początku sekcji <head> dodamy arkusz stylów.

```
<style>
table, td, th {
 border: solid 1px;
}
</style>
```

8. Wszystko jest gotowe, zapisz plik i otwórz go w przeglądarce Firefox. Otrzymasz obraz jak na rysunku 9.7.

Skrypt z przykładu 9.7 jest uniwersalny, może służyć do prezentacji innych plików XML.

Rysunek 9.7. *Wynik działania skryptu z przykładu 9.6. Dokument XML prezentowany w postaci tabeli*

Wczytywanie dokumentów XML w Internet Explorerze

Interfejs dokument.implementation nie działa w przeglądarce Internet Explorer. Zamiast tego obiekt reprezentujący dokument XML w pamięci będzie w postaci kontrolki ActiveX. Powołamy ją do życia za pomocą znanej już z rozdziału 8 funkcji ActiveXObject(). Przy tworzeniu dokumentu XML parametrem tej funkcji będzie `Microsoft.XMLDOM`. W następnym kroku nie skorzystamy z obsługi zdarzeń, tylko wczytamy dokument w trybie synchronicznym, to znaczy wykonanie skryptu zostanie zawieszone do czasu wczytania dokumentu. Po wczytaniu zostanie wywołana funkcja tworząca tabelę, identyczna z tą dla Firefoksa. Oznacza to, że zamiany ograniczą się tylko do skryptu w sekcji `<body>`.

Przykład 9.8. Wczytanie pliku XML — wersja dla Microsoft Internet Explorera

1. Otwórz plik z poprzedniego przykładu.
2. Usuń zawartość skryptu w sekcji `<body>`. Dla IE skrypt będzie wyglądał następująco:

```
<script type="text/javascript">
dokumentxml = new ActiveXObject("Microsoft.XMLDOM");
dokumentxml.async="false";
dokumentxml.load("auta.xml");
documentLoaded();
</script>
```

   Wszystko w tym skrypcie jest zgodnie z tym, co napisałem w akapicie poprzedzającym przykład. Komentarza wymaga jedynie właściwość `async`. Ustawienie tej właściwości na wartość `false` spowoduje pracę w trybie synchronicznym, czyli zawieszenie wykonywania skryptu podczas wczytywania pliku za pomocą funkcji `load()`.
3. Pozostałe części skryptu nie wymagają zmiany. Zapisz plik i otwórz go w Internet Explorerze. Otrzymasz tabelę taką jak na rysunku 9.6.

Wczytywanie uniwersalne dla IE i FF

Skrypt uniwersalny napiszemy, stosując instrukcje warunkowe wykonujące kod odpowiednio dla Firefoksa lub Microsoft Internet Expolera. Przeglądarkę Firefox wykryjemy poprzez sprawdzenie funkcji `addEventListener()`, tak jak przy detekcji przeglądarki, dzięki której można wybrać sposób obsługi zdarzeń. Internet Explorera wykryjemy natomiast przez obecność funkcji `ActiveXObject()`. Oto przykład:

Przykład 9.9. Wczytanie pliku XML — wersja uniwersalna

1. Otwórz plik z poprzedniego przykładu.
2. Różnica będzie polegała tylko na zmianie skryptu w sekcji `<body>`. Zgodnie z tym, co napisałem wcześniej, będzie miał on postać:

```
<script type="text/javascript">
if (window.ActiveXObject) { //IE
dokumentxml = new ActiveXObject("Microsoft.XMLDOM");
dokumentxml.async="false";
dokumentxml.load("auta.xml");
documentLoaded(); }

if (document.addEventListener) { //FF
var dokumentxml = document.implementation.createDocument("","",null)
dokumentxml.addEventListener("load",documentLoaded,false);
dokumentxml.load("auta.xml")
}
</script>
```

3. Zapisz plik, powinien działać zarówno w Firefoksie, jak i MS Internet Explorerze.

Test

1. Do tworzenia elementu (nie węzła tekstowego) służy funkcja obiektu document:
   a) makeElement(),
   b) createElement(),
   c) createTextNode().
2. Po utworzeniu elementu:
   a) jest on od razu związany w gałęzi drzewa sekcji `<body>`;
   b) jest on od razu związany w gałęzi drzewa sekcji `<head>`;
   c) trzeba go umieścić w drzewie za pomocą funkcji appendChild().
3. Aby ustawiać wygląd elementów utworzonych dynamicznie na stronie”
   a) można skorzystać z funkcji setAttribute() do ustawienia danej własności, aczkolwiek jest to sposób przestarzały;
   b) można posłużyć się arkuszem stylu, a za pomocą funkcji setAttribute() ustawić jedynie parametr id na nazwę stylu z arkusza;

c) można posłużyć się wyłącznie arkuszem stylu, przypisując styl do danego typu obiektu.

4. Dokumenty XML:

a) Są nowym formatem arkusza kalkulacyjnego.

b) Są sposobem na uporządkowany zapis danych.

c) Zapisuje się je w postaci plików tekstowych.

5. W MS Internet Explorerze dokument XML tworzymy:

a) za pomocą `document.implementation.createDocument();`

b) w ogóle nie można utworzyć dokumentu XML;

c) za pomocą kontrolki ActiveX.