Java Script.

Księga eksperta

Autor: Danny Goodman
Tłumaczenie: Adam Żytka
ISBN: 83-7197-179-6
Tytuł oryginału: JavaScript Bible
Format: B5, stron: 1096
oprawa twarda
Zawiera CD-ROM

Interaktywne, pełne dynamiki strony WWW - to cel, który często stawiasz przed sobą. Marzysz o wykorzystaniu apletów Javy bez przeciążania serwera, o projektowaniu stron w języku DHTML. Jeden z najbardziej znanych i poważanych ekspertów w branży, Danny Goodman, odkryje przed Tobą tajniki i potęgę języka JavaScript. W typowym dla siebie przejrzystym i precyzyjnym stylu autor przedstawia kompletny wykład na temat JavaScriptu, prowadząc nas od zagadnień elementarnych aż do zaawansowanych technik programowania. Dodatkowym atutem książki jest obszerny leksykon obiektów, funkcji i innych elementów języka.
Ta książka to wyczerpujący opis:

- Języka JavaScript
- Tworzenia animowanych przycisków
- Obiektów i innych kluczowych elementów języka, w tym struktur sterujących, funkcji, operatorów
- Wykorzystania JavaScriptu do weryfikacji danych wprowadzonych przez użytkownika
- Użycia apletów Javy na stronach WWW
- Różnic pomiędzy implementacjami języka zrealizowanymi przez firmy Netscape i Microsoft
- Niezależnych od przeglądarki aplikacji DHTML
- Debuggera przeglądarki Netscape

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

Część II Kurs JavaScriptu

Część IV Praktyczne wykorzystanie JavaScriptu

Dodatki

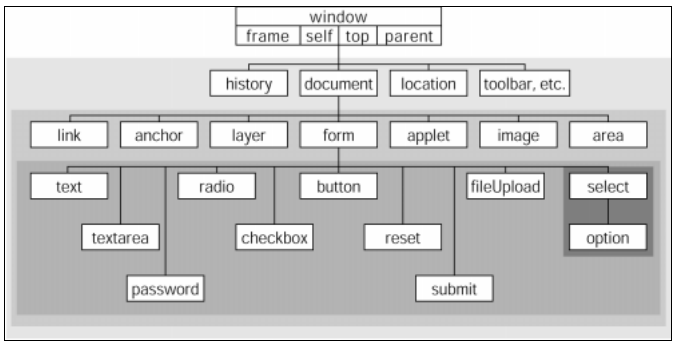
Rozdział 8.

Okna i obiekty dokumentu

Teraz, kiedy masz już podstawowe wiadomości z zakresu programowania, łatwiej będzie zademonstrować, jak za pomocą skryptów obsługiwać obiekty dokumentów. Począwszy od tego rozdziału kurs JavaScriptu powraca do modelu obiektów dokumentu, bliżej zajmując się obiektami, z których często będziesz korzystać we własnych dokumentach.

Obiekty dokumentu

Dla przypomnienia przeanalizuj model obiektów dokumentu Netscape Navigatora zamieszczony ponownie na rysunku 8.1.



Rys. 8.1. *Model obiektów dokumentu Netscape Navigatora*

Ten rozdział poświęcony jest obiektom umieszczonym na szczycie lub blisko szczytu hierarchii – obiektom: `window`, `location`, `history` i `document`. Oprócz podawania podstawowych wiadomości umożliwiających pisanie prostych skryptów, rozdział stanowić ma również wprowadzenie do dogłębnej analizy każdego z tych obiektów, jego właściwości, metod i obsługi zdarzeń, którą przedstawimy w części III. Na tym etapie kursu omawiam jedynie podstawowe właściwości, metody i zdarzenia – o wiele więcej dowiesz się z części III. Do analizy przykładów tam umieszczonych niezbędna jest znajomość podstaw programowania omówiona we wcześniejszych rozdziałach.

Obiekt window

Na samym szczycie hierarchii obiektów dokumentu znajduje się obiekt `window`. W piramidzie obiektów zajmuje on eksponowane miejsce – jest bowiem głównym kontenerem zawartości stron oglądanych w przeglądarce. O ile okno przeglądarki jest otwarte – nawet jeśli nie ma w nim żadnych dokumentów – w aktualnym modelu pamięci definiowany jest obiekt `window`.

Oprócz obszaru okna, w którym umieszczane są dokumenty, do sfery wpływów obiektu `window` zalicza się także wymiary okna oraz wszystkie elementy okna otaczające zawartość strony. Netscape obszar ten (w którym umieszczane są paski przewijania, paski narzędzi, paski stanu i – na platformach innych niż Macintosh – także pasek menu) nazywa krojem lub *wykończeniem* okna. Nie wszystkie przeglądarki, ani nawet nie wszystkie wersje Navigatora, mogą w pełni kontrolować wykończenie głównego okna, łatwo można jednak opracować skrypty tworzące dodatkowe okna o zadanych rozmiarach, w których umieszczane są tylko te elementy wykończenia, na które się zdecydujesz.

Chociaż ramkami zajmiemy się dopiero w rozdziale 11., mogę już teraz wspomnieć, że każda ramka jest także traktowana jako obiekt typu `window`. Po chwili zastanowienia, należy przyznać, że jest to rozwiązanie logiczne – w każdej ramce może być umieszczony inny dokument. W strukturze obiektów ramkę, w której wyświetlany jest dokument zawierający skrypt, tenże skrypt traktuje jako obiekt `window`.

Jak się za chwilę przekonasz, metody wyświetlające modalne okna dialogowe oraz modyfikujące tekst na pasku stanu na dole okna przeglądarki wygodnie jest dołączyć do obiektu `window`. Obiekt `window` udostępnia także metodę umożliwiającą tworzenie na ekranie samodzielnych okien dialogowych. Analizując właściwości, metody i zdarzenia zdefiniowane dla obiektu `window` w modelu obiektów Netscape'a (zobacz dodatek A), jasne powinno się stać, dlaczego umieszczone są one właśnie tam – porównaj ich zasięg z zasięgiem okna przeglądarki.

Korzystanie z właściwości i metod obiektu window

Odwołania skryptu do właściwości i metod obiektu `window` można zapisywać na kilka sposobów, kierując się raczej kaprysem i swoimi upodobaniami, a nie wymaganiami składniowymi. Najbardziej logiczny i najczęściej stosowany sposób tworzenia takich odwołań korzysta z obiektu `window`:

```
window.nazwaWłaściwości
window.nazwaMetody([parametery])
```

Gdy skrypt odwołuje się do okna zawierającego dokument, obiekt `window` posiada synonim. Synonimem tym jest słowo `self`. Składnia odwołania jest wtedy następująca:

```
self.nazwaWłaściwości
self.nazwaMetody([parametery])
```

Początkowych nazw obiektów możesz w odwołaniach używać zamiennie, ja skłaniam się jednak ku wykorzystywaniu słowa `self` w bardziej złożonych skryptach, w których umieszczono wiele ramek i okien. Słowo `self` bardziej dobitnie określa aktualne okno, w którym umieszczony jest dokument zawierający skrypt. Według mnie skrypt taki łatwiej się czyta – zarówno autorowi jak i innym osobom.

W rozdziale 4. wspomniałem, że podczas działania dowolnego skryptu, zawsze dostępny jest obiekt `window`. Obiekt ten można więc opuścić z odwołań do innych obiektów umieszczonych w oknie. W zapisie składniowym podanym poniżej przeglądarka przyjmuje, że właściwości i metody odwołują się do aktualnego okna:

```
nazwaWłaściwości
nazwaMetody([parametery])
```

Jak się za chwilę przekonamy, zapis niektórych metod może okazać się bardziej czytelny, jeśli opuścisz odwołanie do obiektu `window`. Bez względu na to, na które rozwiązanie się zdecydujesz, oba działać będą poprawnie.

Utworzenie okna

Skrypt nie tworzy głównego okna przeglądarki. Robi to użytkownik, uruchamiając przeglądarkę lub otwierając adres URL, czy jakiś plik za pomocą menu (jeśli w przeglądarce nie jest jeszcze otwarty żaden dokument). Po otwarciu głównego okna, skrypt umieszczony w dokumencie tego okna może natomiast utworzyć dowolną liczbę podokien.

Metodą, która tworzy nowe okno jest metoda `window.open()`. Metoda ta może zawierać do trzech parametrów, w których można określić charakterystyczne cechy okna, takie jak adres URL dokumentu, który ma być otwarty, nazwę okna (stosowaną w HTML-owych odwołaniach wykorzystujących słowo `TARGET`) oraz wygląd okna (czyli jego rozmiar i elementy wykończenia). Nie będę na razie zajmował się szczegółami dotyczącymi poszczególnych parametrów (dokładniej są one omówione w rozdziale 14.), natomiast chciałbym zwrócić uwagę na pewien istotny element metody `window.open()`.

Przyjrzyj się poniższej instrukcji powodującej otwarcie nowego okna o podanym rozmiarze i zawierającego dokument HTML pobrany z serwera:

```
var subWindow = window.open("definition.html", "def",
↳"HEIGHT=200,WIDTH=300")
```

Należy zwrócić uwagę, że jest to instrukcja przypisania. Coś przypisywane jest do zmiennej `subWindow`. Cóż to jest? Okazuje się, że uruchomienie metody `window.open()` powoduje nie tylko otwarcie nowego okna, stosownie do wyspecyfikowanych parametrów – metoda oprócz tego przyjmuje wartość odwołania do nowego okna. W żargonie programistów mówi się, że metoda *zwraca* wartość, w tym przypadku odwołanie do prawdziwego obiektu. Właśnie ta zwrócona wartość przypisywana jest do zmiennej.

Skrypt może teraz wykorzystać tę zmienną jako pełnoprawne odwołanie do podokna. Jeśli chcesz uzyskać dostęp do jego właściwości lub metod, odwołując się do nich powinieneś skorzystać z tejże zmiennej. Na przykład, aby zamknąć podokno za pomocą

skryptu w oknie głównym, odwołanie do metody `close()` tego podokna powinno wyglądać w następujący sposób:

```
subWindow.close
```

Jeśli skrypt w głównym oknie wywołałby metodę `window.close()`, `self.close()` lub po prostu `close()`, spowodowałaby ona zamknięcie głównego okna, a nie podokna. Odwołując się do metod lub właściwości innego okna, nie wolno opuszczać odwołania do niego. Wywiera to wpływ na tworzony kod, zechcesz bowiem prawdopodobnie, aby zmienna zawierająca odwołanie do podokna była dostępna tak długo, jak długo dokument główny dostępny jest w przeglądarce. Żeby tak się stało, zmienną należy zainicjalizować jako zmienną globalną, a nie jako zmienną lokalną wewnątrz funkcji (choć wewnątrz funkcji można następnie przypisać do zmiennej wartość). W ten sposób jedna funkcja może otwierać okno, a druga je zamykać.

Na listingu 8.1 przedstawiona jest strona, na której umieszczono przyciski do otwierania nowego, pustego okna i zamykania go z poziomu okna głównego. Aby sprawdzić działanie skryptów, zmniejsz główne okno przeglądarki, tak aby zajmowało mniej niż pół ekranu. Następnie, utworzywszy nowe, małe okno, przesuń je, tak by było ono widoczne także po wyświetleniu na pierwszym planie okna głównego. W kodzie na listingu 8.1 kluczowe znaczenie odgrywa to, że zmienną `newWindow` zdefiniowano jako zmienną globalną, dzięki czemu dostęp do niej mają obydwie funkcje. Po zadeklarowaniu zmiennej, jeśli nie zostanie dokonane do niej przypisanie, jej wartość to `null`. Okazuje się, że wartość `null` w warunkach spełnia taką samą rolę jak wartość `false`, natomiast dowolna wartość różna od wartości `null` odpowiada wartości `true`. Warunek w funkcji `closeNewWindow()` sprawdza się więc do sprawdzenia, czy przed wywołaniem metody `close()` otwarto podokno.

Listing 8.1. *Odwołania do obiektów window*

```
<HTML>
<HEAD>
<TITLE>Otwieranie i zamykanie okien</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var newWindow
function makeNewWindow() {
  newWindow = window.open("","","HEIGHT=300,WIDTH=300")
}
function closeNewWindow() {
  if (newWindow) {
    newWindow.close()
  }
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
  <INPUT TYPE="button" VALUE="Utwórz nowe okno"
↳onClick="makeNewWindow()">
  <INPUT TYPE="button" VALUE="Zamknij nowe okno"
↳onClick="closeNewWindow()">
</FORM>
</BODY>
</HTML>
```

Właściwości i metody obiektu window

Jedna właściwość i trzy metody obiektu `window` opisane w tym rozdziale, wywierają bezpośredni wpływ na interakcje użytkownika ze stroną. Działają one we wszystkich obsługujących skrypty przeglądarkach. Złożone przykłady demonstrujące wykorzystanie poszczególnych właściwości i metod obiektu `window` można znaleźć w części III.

Właściwość window.status

Na pasku stanu na dole przeglądarki w chwili wskazania myszką łącza, zazwyczaj wyświetlany jest adres URL. Podczas otwierania dokumentu, inicjalizacji apletów Javy itd. pojawiają się na nim inne komunikaty. Za pomocą JavaScriptu na pasku stanu można jednak wyświetlać własne komunikaty, jeśli tylko miałyby one okazać się pomocne dla osób odwiedzających strony. Przykładowo, zamiast adresu URL, do którego prowadzi łącze, można na pasku stanu wyświetlić jakiś inny, bardziej czytelny opis strony (lub obydwa te elementy, co zadowoli zarówno żółtodziobów, jak i sieciowych weteranów).

Tekst do właściwości `window.status` można przypisać w dowolnej chwili. Aby na pasku stanu w odpowiednim momencie pojawiały się opisy łącz, umieszczony na nim tekst musi być zmieniany przez obsługę zdarzenia `onMouseOver=` obiektu `link` (łącza). Specyfika zmieniającej tekst na pasku stanu obsługi zdarzenia `onMouseOver=` polega na konieczności zastosowania instrukcji `return true`. W JavaScripcie wymagane jest to niezwykle rzadko, w tym konkretnym przypadku okazuje się jednak niezbędne do pomyślnego nadpisania przez skrypt tekstu na pasku stanu.

Ze względu na prostotę przypisywania wartości do właściwości `window.status`, bardzo często instrukcje skryptów umieszcza się bezpośrednio w znacznikach, w definicjach obsługi zdarzeń. Jest to rozwiązanie korzystne w przypadku krótkich skryptów, nie jest bowiem wówczas konieczne definiowanie samodzielnej funkcji oraz umieszczanie na stronie znaczników `<SCRIPT>`. Instrukcje umieszczane są po prostu wewnątrz znacznika połączenia `<A>`:

```
<A HREF="http://home.netscape.com" onMouseOver="window.status=
↳'Odwiedź stronę firmy Netscape (home.netscape.com)';
↳return true">Netscape</A>
```

Przyjrzyj się dobrze dwóm instrukcjom skryptu, umieszczonym w obsłudze zdarzenia `onMouseOver=`.

Te dwie instrukcje to:

```
window.status='Odwiedź stronę firmy Netscape (home.netscape.com)'
return true
```

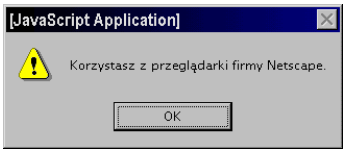
W przypadku umieszczania instrukcji wewnątrz znaczników należy oddzielać je od siebie za pomocą średników. Ważne jest też to, aby wokół nich umieszczony był podwójny cudzysłów (`"..."`). Aby łańcuch przypisywany do właściwości `window.status`

umieścić wewnątrz skryptu (atrybutu znacznika) zamkniętego w podwójnym cudzysłowie, łańcuch ten otaczamy apostrofami (`'...'`).

Zmiana napisów na pasku stanu wymaga niewielkiego nakładu pracy, a jest niezwykle użyteczna. Efektem ubocznym udostępnienia możliwości obsługi za pomocą skryptów pasków stanu są pojawiające się na nich – na niektórych stronach – okropne, przewijające się napisy. Ohyda!

Metoda window.alert()

Z metody `window.alert()` w niniejszym kursie korzystałem już wielokrotnie. Metoda ta wyświetla okno dialogowe z tekstem, który przekazywany jest do niej jako parametr (zobacz rysunek 8.2). Zamknięcie okna umożliwia umieszczony w nim przycisk OK (jego wyglądu nie można zmieniać).



Rys. 8.2. *Okno dialogowe z komunikatem utworzone za pomocą JavaScriptu*

Wygląd tego i dwóch innych, opisanych niżej okien dialogowych nieznacznie się zmieniał w różnych wersjach przeglądarek obsługujących skrypty. Niektóre wersje Navigatora na pasku tytułowym zamiast tekstu JavaScript Application, który jasno wskazuje, że mamy do czynienia z aplikacją JavaScriptu, wyświetlają napis Netscape. Napisu tego nie można zmieniać za pomocą skryptów. Zmieniana może być tylko treść komunikatu.

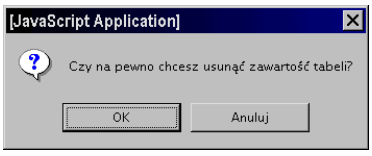
Wszystkie trzy metody wyświetlające okna dialogowe dobrze ilustrują możliwość użycia metod obiektu `window` bez konieczności podawania w odwoływaniu nazwy tego obiektu. Pomimo tego, że metoda `alert()` pod względem formalnym jest metodą obiektu `window`, pomiędzy oknem generującym komunikat a oknem z komunikatem nie zachodzą żadne bliższe związki. W stronach, które tworzę zazwyczaj nie korzystam więc z pełnego odwołania:

```
alert("Okno komunikatu utworzone za pomocą JavaScriptu." )
```

Metoda window.confirm()

W kolejnym typie okna dialogowego, którym się teraz zajmiemy, umieszczone są dwa przyciski. W większości przeglądarek, bez względu na wersję i platformę sprzętową, są to przyciski Cancel (Anuluj) i OK. Okno to, pokazane na rysunku 8.3, nazywane jest oknem potwierdzenia. Co ważniejsze, metoda, która je wyświetla zwraca wartość – jest to wartość `true`, jeśli użytkownik naciśnie przycisk OK i wartość `false` w przypadku naciśnięcia przycisku Cancel (Anuluj). Okno i zwracana przez nie wartość może być wykorzystana do tego, aby zadecydować, które instrukcje skryptu należy wykonać.

Rys. 8.3.
Okno dialogowe z prośbą o potwierdzenie, utworzone za pomocą JavaScriptu



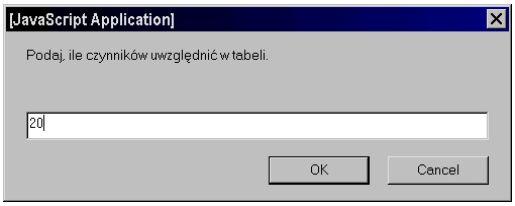
Ponieważ omawiana metoda zawsze zwraca wartość boolowską, możemy wykorzystać ją w warunkach tworzonych za pomocą słowa `if` lub w konstrukcji `if...else`. W poniższym fragmencie kodu pytamy na przykład użytkownika, czy wykonywanie aplikacji ma się rozpocząć od początku. Jeśli tak, w przeglądarce otwierana jest domyślna strona witryny.

```
if (confirm("Czy na pewno chcesz rozpocząć od początku?")) {
  location = "index.html"
}
```

Metoda window.prompt()

Ostatnie okno dialogowe tworzone za pomocą metody obiektu `window` zawiera określony w skrypcie tekst komunikatu i pole tekstowe umożliwiające użytkownikowi wpisanie danych (zobacz rysunek 8.4). Okna można się pozbyć za pomocą jednego z dwóch przycisków: Cancel (Anuluj) lub OK. Pierwszy z nich powoduje anulowanie odpowiedzi, a drugi zaakceptowanie tekstu wpisanego do okna.

Rys. 8.4.
Okno dialogowe z prośbą o wpisanie tekstu, utworzone za pomocą JavaScriptu



Metoda `window.prompt()` korzysta z dwóch parametrów. Pierwszym z nich jest komunikat, stanowiący objaśnienie dla użytkownika. W polu tekstowym można zasugerować domyślną odpowiedź, podając jako drugi parametr zmienną typu `string`. Jeśli nie chcesz, aby w oknie pojawiała się odpowiedź domyślna, skorzystaj po prostu z łańcucha pustego (zapisywanego za pomocą znaków cudzysłowu bez odstępu w środku).

Po naciśnięciu przez użytkownika dowolnego przycisku, metoda zwraca wartość. Po naciśnięciu przycisku Cancel (Anuluj) zwracana jest wartość `null`, bez względu na to, co wpisano do pola. Naciśnięcie przycisku OK powoduje natomiast, że zwracany jest łańcuch wpisany przez użytkownika. Z informacji tych skrypty mogą korzystać w konstrukcjach sterujących `if` bądź `if...else`. Wartość `null` w warunkach traktowana jest jak wartość `false`. W ten sam sposób traktowane są również łańcuchy puste. Dzięki temu w warunku można łatwo sprawdzić, czy do pola wpisano jakieś znaki, co ułatwia znacznie jego zapisanie. Widać to na poniższym przykładzie:

```
var answer = prompt("Jak masz na imię?", "")
if (answer) {
  alert("Cześć, " + answer + "!")
}
```

Metoda `alert()` wywoływana jest jedynie wówczas, gdy użytkownik wpisał w oknie jakiś tekst i nacisnął przycisk OK.

Obsługa zdarzenia onLoad=

Obiekt `window` obsługuje kilka zdarzeń systemowych i zdarzenia wywoływane przez użytkownika. Zdarzeniem, z którego będziesz zapewne korzystał najczęściej jest zdarzenie zachodzące w chwili, gdy wszystkie elementy strony zostaną otwarte przez przeglądarkę. Zdarzenie to nie zachodzi dopóki przeglądarka pobiera rysunki, aplety Javy i pliki danych dla plug-inów. Próba odczytania obiektów dokumentu podczas otwierania strony może okazać się niebezpieczna. Jeśli bowiem obiekt nie został jeszcze odczytany (na przykład ze względu na wolne połączenie sieciowe lub przeciążenie serwera) skrypt spowoduje błąd. Zaletą używania obsługi zdarzenia `onLoad=` do wywoływania funkcji jest to, że mamy pewność, iż wszystkie obiekty dokumentu są już pobrane i umieszczone w modelu obiektów dokumentu przeglądarki.

Definicje obsługi wszystkich zdarzeń obiektu `window` umieszczane są wewnątrz znaczników `<BODY>`.

Mimo tego, że z czasem zaczniesz kojarzyć atrybuty znacznika `<BODY>` z właściwościami obiektu `document`, definicje obsługi zdarzeń umieszczane w jego wnętrzu dotyczą obiektu `window` (z obiektem `document` nie są skojarzone żadne zdarzenia).

Obiekt location

Czasem obiekt w hierarchii przedstawia coś, co nie ma fizycznego odpowiednika, takiego jak okno czy przycisk. Sytuacja taka zachodzi w przypadku obiektu `location`. Obiekt ten reprezentuje adres URL dokumentu odczytanego w oknie. Jest on zatem czymś innym niż obiekt `document` omówiony w dalszej części rozdziału. Obiekt `document` to właściwa zawartość strony, podczas gdy obiekt `location` to po prostu adres URL.

O ile nie znasz stosunkowo dobrze mechanizmów działania sieci Web, zapewne nie zdajesz sobie sprawy z tego, że adres URL składa się z wielu komponentów określających lokalizację i sposób transferu danych pliku. Na adres URL składa się nazwa protokołu (na przykład `http:`) i nazwa hosta (na przykład `www.giantco.com`). Wszystkie te elementy można odczytać jako właściwości obiektu `location`. Najczęściej jednak skrypty, które będziesz tworzył, korzystać będą tylko z jednej właściwości – właściwości `href` opisującej pełny adres URL.

Przypisanie wartości do właściwości `location.href` to najprostszy sposób pozwalający na sterowane za pomocą skryptów przechodzenia przeglądarki na inne strony, bądź

to w aktualnym oknie bądź w innej ramce. Do strony we własnej witrynie można przejść podając względny adres URL (tzn. adres względny wobec aktualnie otwartej strony) zamiast adresu pełnego, zawierającego informacje o protokole i nazwie hosta.

Skrót dobrze sprawdzający się w przypadku Navigatora polega na opuszczeniu w odwołaniu właściwości `href`. Można po prostu przypisać do obiektu `location` adres URL (względny lub pełny), a Navigator sam przejdzie do żądanej strony. Dlatego wykonanie obydwu poniższych instrukcji daje ten sam efekt:

```
location="http://www.dannyg.com"
location.href="http://www.dannyg.com"
```

Jeśli strona ma być otwarta w innym oknie lub ramce, w instrukcji należy umieścić odwołanie do odpowiedniego obiektu. Przykładowo, jeśli skrypt otwiera nowe okno i do zmiennej `newWindow` przypisuje odwołanie do niego, otwarcie w tym oknie nowego dokumentu umożliwi poniższa instrukcja:

```
newWindow.location = "http://www.dannyg.com"
```

Obiekt history

Kolejnym obiektem, który nie ma fizycznego odpowiednika na stronie jest `history`. W każdym oknie przechowywana jest lista ostatnio odwiedzanych przez przeglądarkę stron. Chociaż na liście umieszczone są adresy URL tych stron, odczytywanie adresów za pomocą skryptu, nie jest w zasadzie możliwe. Metody obiektu `history` pozwalają natomiast poruszać się do przodu i do tyłu po liście adresów (czyli historii) obowiązującej dla aktualnie otwartej strony.

Obiekt document

W obiekcie `document` umieszczana jest właściwa zawartość strony. Właściwości oraz metody obiektu `document` mają ogólnie rzecz biorąc wpływ na wygląd oraz zawartość dokumentu wyświetlanego w oknie. Pominąwszy niektóre zaawansowane opcje Dynamicznego HTML-a wprowadzone w Internet Explorerze 4, po odczytaniu dokumentu, tekst na stronie nie może być za pomocą skryptu odczytywany ani zmieniany.

Jednak jak miałeś okazję przekonać się już w pierwszym skrypcie w rozdziale 3., metoda `document.write()` umożliwia dynamiczne sterowanie zawartością strony podczas jej otwierania. Wiele właściwości obiektu `document` ustalanych jest za pomocą atrybutów w znaczniku `<BODY>`. Pozostałe właściwości to często tablice innych obiektów w dokumencie.

Uzyskanie dostępu do właściwości i metod obiektu `document` nie jest skomplikowane, co widać w poniższych konstrukcjach składniowych:

```
[window.]document.nazwaWłaściwości
[window.]document.nazwaMetody
```

Odwołanie do obiektu `window` nie jest obowiązkowe wówczas, gdy skrypt odwołuje się do obiektu `document`, w którym sam się zawiera.

Właściwość document.forms[ ]

Jednym z obiektów często umieszczanych w dokumencie jest obiekt typu `form`. Ponieważ nie ma powodów, dla których w dokumencie nie można byłoby umieścić więcej niż jednego formularza, kolejne formularze przechowywane są jako elementy tablicy we właściwości `document.forms[]`. Jak sobie zapewne przypominasz z opisu tablic umieszczonego w rozdziale 7., numer indeksu wewnątrz nawiasów kwadratowych wskazuje na jeden z elementów tablicy. Aby więc uzyskać dostęp do pierwszego formularza w dokumencie, należy skorzystać z następującego odwołania:

```
document.forms[0]
```

Zazwyczaj lepszym rozwiązaniem jest jednak korzystanie z nazw formularzy przypisywanych do nich za pomocą atrybutu `NAME`. Odwołanie takie wygląda następująco:

```
document.nazwaFormularza
```

Za pomocą obydwu sposobów uzyskujemy dostęp do tego samego obiektu. Jeśli natomiast skrypt ma obsługiwać obiekty wewnątrz formularza, pełne odwołanie do nich powinno zawierać słowa `document` i `form`.

Właściwość document.title

Nie wszystkie właściwości obiektu `document` opisuje się wewnątrz znacznika `<BODY>`. Tytuł dokumentu, czyli właściwość `document.title` jest określana wewnątrz znaczników `<TITLE>` w sekcji `<HEAD>`. Ustawienie to pełni w zasadzie rolę kosmetyczną, pozwalając na nadanie stronie zwykłej nazwy tekstowej. Nazwa ta widoczna jest na pasku tytułowym, jak również w historii i zakładkach.

Metoda document.write()

Metoda `document.write()` może być wykorzystywana zarówno w skryptach tworzących treść strony podczas jej otwierania, jak i w skryptach wywoływanych z opóźnieniem, używanych do tworzenia nowych stron w tym samym bądź innym oknie.

W wywołaniu metody podaje się jeden łańcuch, stanowiący fragment HTML-owego kodu wpisywanego do okna lub ramki. Parametrami łańcuchowymi mogą być zmienne lub dowolne inne wyrażenia przyjmujące wartości łańcuchowe. Bardzo często we wpisywanej do okna zawartość umieszczane są HTML-owe znaczniki.

Pamiętaj, że po otwarciu strony, strumień wyjściowy przeglądarki jest automatycznie zamykany. Po tym fakcie, metoda `document.write()` wykonana w odniesieniu do aktualnej strony, powoduje natychmiastowe wymazanie jej bieżącej zawartości (w tym zmiennych i innych wartości w dokumencie). Jeśli więc zawartość aktualnej strony ma być zastąpiona nowym HTML-owym kodem wpisywanym do dokumentu za pomocą skryptu, kod ten należy umieścić w zmiennej i wpisać go za pomocą tylko jednej metody `document.write()`. Nie jest konieczne usuwanie starej zawartości strony ani otwieranie nowego strumienia danych. Wszystkim zajmie się metoda `document.write()`.

Ostatnia dobra rada dotycząca metody `document.write()` wiąże się z pokrewną jej metodą `document.close()`. Skrypt musi zamknąć strumień wyjściowy po zakończeniu zapisywania zawartości do okna (aktualnego bądź dowolnego innego). Po wykonaniu ostatniej metody `document.write()`, umieszczonej w skrypcie wywoływanym z opóźnieniem należy pamiętać, aby wywołać metodę `document.close()`. Jeśli tego zaniechamy, to może się okazać, że rysunki i formularze nie będą widoczne, także dowolna inna metoda `document.write()` wywołana później dopisze jedynie tekst do strony, zamiast wyczyścić aktualną jej zawartość i rozpocząć pisanie od nowa. Przykład użycia metody `document.write()` znaleźć można w dwóch poniższych wersjach tej samej aplikacji. W jednej z nich tekst wpisywany jest do tego samego dokumentu, który zawiera skrypt, a w drugiej do samodzielnego okna. Wpisz każdy z dwóch dokumentów, zapisz go i otwórz w przeglądarce.

Kod z listingu 8.2 tworzy przycisk, którego naciśnięcie powoduje wpisanie do dokumentu nowego HTML-owego kodu, w tym HTML-owych znaczników określających tytuł dokumentu i atrybut `BGCOLOR` dla znacznika `<BODY>`. Następnie za pomocą jednej instrukcji `document.write()`, cały gotowy już kod wpisywany jest do dokumentu, wymazując wszelkie pozostałości elementów strony zapisanych na listingu 8.2. Instrukcja `document.close()` jest natomiast niezbędna do odpowiedniego zamknięcia strumienia wyjściowego. Po odczytaniu dokumentu oraz naciśnięciu przycisku polecenia zmieniającego zawartość strony, warto zwrócić uwagę, że zmieniony został również tytuł dokumentu w oknie przeglądarki. Wróć do dokumentu oryginalnego za pomocą przycisku Back (Wstecz) i ponownie naciśnij przycisk polecenia w dokumencie, zwracając tym razem uwagę na to, że dynamiczne tworzenie strony przebiega bardzo szybko, szybciej nawet niż ponowne odczytanie dokumentu oryginalnego.

Listing 8.2. *Wykorzystanie metody document.write() w odniesieniu do aktualnego okna*

```
<HTML>
<HEAD>
<TITLE>Wpisywanie tekstu to bieżącego dokumentu</TITLE>
<SCRIPT LANGUAGE="JavaScript">
function reWrite() {
  // przygotowanie zawartości nowego okna
  var newContent = "<HTML><HEAD><TITLE>Nowy dokument
↳</TITLE></HEAD>"
  newContent += "<BODY BGCOLOR='aqua'><H1>Ten dokument jest
↳całkiem nowy.</H1>"
  newContent += "Naciśnij przycisk Back (Wstecz), aby
↳zobaczyć dokument oryginalny."
  newContent += "</BODY></HTML>"
```

```
  // wpisanie dokumentu HTML do nowego okna
  document.write(newContent)
  document.close() // koniec wpisywania tekstu
}
</SCRIPT>
</HEAD>
<BODY>
<FORM>
<INPUT TYPE="button" VALUE="Zmień treść dokumentu"
↳onClick="reWrite()">
</FORM>
</BODY>
</HTML>
```

Na listingu 8.3 mamy do czynienia z nieco bardziej złożoną sytuacją, bowiem skrypt tworzy podokno, do którego wpisuje następnie generowany automatycznie dokument. Aby w obu wykorzystywanych w skrypcie funkcjach możliwe było odwoływanie się do nowego okna, zmienna `newWindow` deklarowana jest jako zmienna globalna. Natychmiast po otwarciu strony, obsługa zdarzenia `onLoad=` wywołuje funkcję `makeNewWindow()`. Funkcja ta tworzy puste podokno. W trzecim parametrze metody `window.open()` podałem właściwość powodującą wyświetlenie w podoknie paska stanu.

Przycisk na stronie wywołuje funkcję `subWrite()`. Pierwsze zadanie funkcji polega na sprawdzeniu właściwości `closed` podokna. Właściwość ta (dostępna jedynie w nowszych wersjach przeglądarek) zwraca wartość `true`, w przypadku gdy okno, do którego się odwołujemy jest zamknięte. Jeśli rzeczywiście zachodzi taka sytuacja (użytkownik mógł samodzielnie zamknąć okno) ponownie wywoływana jest funkcja `makeNewWindow()`, która po raz kolejny otwiera okno.

Teraz, gdy nowe okno jest już otwarte, w zmiennej łańcuchowej przygotowujemy jego zawartość. Tak jak w przypadku kodu na listingu 8.2, wszystko wpisywane jest do okna za jednym razem (choć nie jest to konieczne, gdy tekst wpisywany jest do samodzielnych okien). Następnie wywoływana jest metoda `close()`. Warto zwrócić uwagę na istotną różnicę w wywołaniach metod zarówno `close()` jak i `open()` – w przeciwieństwie do kodu na listingu 8.2 w jednym i drugim przypadku podawane jest odwołanie do podokna.

Listing 8.3. *Wykorzystanie metody document.write() w samodzielnym oknie*

```
<HTML>
<HEAD>
<TITLE>Wpisywanie tekstu to okna potomnego</TITLE>
<SCRIPT LANGUAGE="JavaScript">
var newWindow
function makeNewWindow() {
  newWindow = window.open("","","status,height=200,width=300")
}
function subWrite() {
  // utworzenie okna, jeśli ktoś je wcześniej zamknął
  if (newWindow.closed) {
    makeNewWindow()
  }
  // przygotowanie zawartości nowego okna
```

```
  var newContent = "<HTML><HEAD><TITLE>Nowy dokument
↳</TITLE></HEAD>"
  newContent += "<BODY BGCOLOR='coral'><H1>Ten dokument jest
↳całkiem nowy.</H1>"
  newContent += "Naciśnij przycisk Back (Wstecz), aby zobaczyć
↳dokument oryginalny."
  newContent += "</BODY></HTML>"
  // wpisanie dokumentu HTML do nowego okna
  newWindow.document.write(newContent)
  newWindow.document.close() // koniec wpisywania tekstu
}
</SCRIPT>
</HEAD>
<BODY onLoad="makeNewWindow()">
<FORM>
<INPUT TYPE="button" VALUE="Zmień treść dokumentu"
↳onClick="subWrite()">
</FORM>
</BODY>
</HTML>
```

Obiekt link

W hierarchii obiektów, do obiektu `document` przyporządkowany jest również obiekt `link`, czyli łącze. Dokument może zawierać dowolną liczbę łącz, tak więc w razie potrzeby w odwołaniach do tych obiektów najczęściej używa się składni korzystającej z tablic i indeksów:

```
document.links[n].nazwaWłaściwości
```

Najczęściej łącz nie obsługuje się za pomocą skryptów. Wiąże się jednak z nimi pewien ciekawy aspekt języka. Jeśli kliknięcie łącza ma powodować wykonanie skryptu, a nie natychmiastowe przejście pod wskazany adres URL, w atrybucie `HREF` znacznika można dokonać przekierowania, tak aby wskazywana była funkcja skryptu.

Technika ta wiąże się z zastosowaniem pseudoadresu URL, rozpoczynającego się od łańcucha `javascript:`. Jeśli za tym łańcuchem umieścimy nazwę funkcji, przeglądarki obsługujące skrypty uruchomią ją. Funkcja ta, aby nie wprowadzać zamieszania, powinna ostatecznie dokonywać jakiejś nawigacji, zastosowanie skryptu pozwala jednak na wykonanie również innych czynności, na przykład na jednoczesną zmianę zawartości dwóch ramek w zdefiniowanym układzie.

Konstrukcja, którą należy umieścić wewnątrz łącza wygląda następująco:

```
<A HREF="javascript:void nazwaFunkcji
↳([parametr1]...[parametrN])">...</A>
```

Dzięki zastosowaniu słowa kluczowego `void` łącze nie będzie próbowało wyświetlać wartości, które mogą być zwracane przez funkcję. O stosowaniu adresu `javascript:` należy pamiętać w przypadku wszystkich znaczników, zawierających atrybuty `HREF` i `SRC`. Jeśli w danym atrybucie możliwe jest podanie adres URL, poprawny będzie rów-

nież adres rozpoczynający się od łańcucha `javascript:`. Może okazać się to pomocne przy pisaniu skryptów obsługujących mapy po stronie klienta, które niekoniecznie prowadzą w jakieś miejsce, ale powodują, że na stronie zaczyna się coś dziać.

Kolejny logiczny krok w poznawaniu hierarchii obiektów dokumentu wiąże się z formularzem. Zajmiemy się nim w kolejnym rozdziale.

Ćwiczenia

1. Rozstrzygnij, które z poniższych odwołań są poprawne, a które nie. Omów błędy występujące w odwołaniach nieprawidłowych.
   a. `window.document.form[0]`
   b. `self.entryForm.entryField.value`
   c. `document.forms[2].name`
   d. `entryForm.entryField.value`
   e. `newWindow.document.write("Hej!")`
2. Zapisz instrukcję JavaScriptu, wyświetlającą wiadomość na pasku stanu, która wita użytkowników na stronie sieci Web.
3. Napisz instrukcję JavaScriptu, wyświetlającą tę samą wiadomość w dokumencie jako nagłówek `<H1>` strony.
4. Utwórz stronę, która w czasie otwierania pyta się użytkownika o jego imię (za pomocą okna dialogowego), a następnie wita użytkownika po imieniu za pomocą tekstu wyświetlanego w części `<BODY>`.
5. Utwórz stronę, po otwarciu której automatycznie wyświetlane byłoby okno dialogowe informujące użytkownika o aktualnym adresie URL.