# Dreamweaver 4. Biblia


Autor: Joseph W. Lowery
Tłumaczenie: Piotr Ducher
ISBN: 83-7197-591-0
Tytuł oryginału: Dreamweaver 4 Bible
Format: B5, stron: 1036
Zawiera CD-ROM

Dreamweaver jest jednym z najbardziej znanych narzędzi do tworzenia stron WWW. Tworzenie stron to połączenie sztuki i rzemiosła. Program ten jest doskonałym narzędziem dla każdego – zarówno dla profesjonalisty poganianego terminami, jak i dla natchnionego amatora. Dreamweaver jest nie tylko pierwszym narzędziem umożliwiającym wizualną edycję strony, ale także czołowym pod względem łatwości wykorzystania na stronie skryptów napisanych w JavaScript.

Aby móc w pełni wykorzystać wiadomości zawarte w tej książce, potrzebujesz dwóch rzeczy – Dreamweavera i chęci tworzenia niebanalnych stron WWW (w zasadzie nie potrzebujesz Dreamweavera, aby zacząć pracę – na dołączonym do książki CD-ROM-ie znajdziesz w pełni funkcjonalną wersję próbną programu). Dreamweaver ułatwia i przyspiesza pracę twórcy stron WWW na każdym jej etapie – od wykonania projektu do zarządzania witryną. Niestety, nawet Dynamiczny HTML, wspaniale zresztą obsługiwany przez Dreamweavera, nie potrafi poradzić sobie z wszystkimi aspektami nowoczesnej strony WWW. W tej książce znajdziesz jednak szczegółowe instrukcje dotyczące rozwiązywania niemal każdego zadania związanego z tworzeniem strony.

Pod prostym i intuicyjnym interfejsem użytkownika Dreamweavera kryje się skomplikowany i rozbudowany program ułatwiający dostęp do wszystkich zaawansowanych narzędzi. Zadaniem tej książki jest pomoc w opanowaniu wszystkich elementów programu. Tworzysz prostą stronę w edytorze wizualnym? Chcesz rozbudować możliwości programu, tworząc własne obiekty? Wiadomości zawarte w tej książce umożliwią Ci wykonanie tych zadań.

Dreamweaver jest przeznaczony dla szerokiego kręgu projektantów. Ponieważ program nie modyfikuje postaci kodu, może być wykorzystywany przez „weteranów" posiadających własny styl zapisywania kodu. Z racji możliwości automatyzacji wielu czynności program zainteresuje również początkujących projektantów. Ta książka przeznaczona jest dla wszystkich – od początkujących projektantów, szukających podstawowych informacji, do profesjonalistów poszukujących nowych sposobów rozwiązywania problemów występujących przy projektowaniu. Co więcej – książka ta jest doskonałym podręcznikiem i źródłem informacji dla każdego, kto na co dzień korzysta z Dreamweavera.

Jeśli przeczytasz tę książkę od początku do końca, staniesz się zaawansowanym użytkownikiem Dreamweavera. Możesz jednak wykorzystywać tylko niektóre informacje w niej zawarte, potrzebne przy wykonywaniu określonych czynności. Aby ułatwić takie korzystanie z książki, została ona podzielona na osiem części poświęconych różnym zagadnieniom. Możesz również wykorzystywać tę książkę jako źródło wiedzy podczas pracy z Dreamweaverem.

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

# Rzut oka na książkę

# Spis treści

# Rozdział 20.
# Tworzenie behawiorów

W rozdziale:

- Podstawowe informacje na temat tworzenia behawiorów
- Obiektowy model dokumentu (DOM)
- Interfejs programowania (API) Dreamweavera
- Często wykorzystywane funkcje
- Techniki tworzenia określonych funkcji behawiorów

Każdy, kto zna języki HTML i JavaScript, może pisać własne behawiory. Określenie „pisać behawiory" jest określeniem nieco na wyrost. Nigdy nie zmodyfikujesz tej części behawiora, która jest odpowiedzialna za zdarzenia. Możesz tworzyć jedynie pliki akcji. Aby pomóc twórcom behawiorów, firma Macromedia udostępniła dokumentację *Extending Dreamweaver* zawierającą opisy wszystkich funkcji i rozszerzeń JavaScriptu i obiektowego modelu dokumentu (DOM) rozpoznawanych przez program Dreamweaver.

W Dreamweaverze 4 znacznie rozbudowano możliwości behawiorów. Dzięki znacznie większej liczbie funkcji z DOM Dreamweaver może wpływać na niemal każdy element strony WWW. Możesz nawet wykorzystać behawiory do otwierania istniejących dokumentów lub tworzenia nowych. JavaScript API (interfejs programowania) Dreamweavera umożliwia wykorzystanie ponad 400 funkcji. W tym rozdziale opisane są jedynie funkcje przydatne przy tworzeniu behawiorów, ale opis większości funkcji API Dreamweavera wykracza znacznie poza zakres tematyczny tej książki. Zanim poznasz szczegółowe zasady tworzenia behawiorów, musisz zapoznać się z ogólnym zarysem tego procesu.

## Tworzenie behawiora od podstaw

Tworzenie behawiora nie jest procesem skomplikowanym, jeśli podzielisz go na etapy.

- *Etap 1. Zdefiniuj behawior*. Behawior to metoda dołączenia do strony określonej funkcji JavaScript. Najlepszym sposobem rozpoczęcia tworzenia behawiora jest napisanie tej właśnie funkcji. Funkcja zostanie dołączona do pliku akcji.

- *Etap 2. Stwórz plik akcji*. Jedną z najważniejszych funkcji używanych w behawiorach jest funkcja `behaviorFunction()`, która powoduje wstawienie kodu do sekcji `<head>` dokumentu. Dreamweaver umożliwia wstawianie do kodu grupy funkcji lub pojedynczej funkcji.
- *Etap 3. Stwórz interfejs użytkownika*. Wszystkie behawiory posiadają okno dialogowe. Okno to opiera się na formularzu stworzonym w języku HTML i często nazywane jest formularzem wprowadzania parametrów (*parameter form*).
- *Etap 4. Dołącz behawior do znacznika*. Behawior składa się ze zdarzenia i akcji. Funkcja `applyBehavior()` umożliwia dołączenie Twojej funkcji do znacznika i zdarzenia. Funkcja ta przekazuje również argumenty funkcjom znajdującym się w sekcji `<head>`.
- *Etap 5. Zbadaj behawior*. Z punktu widzenia użytkownika, tworzenie strony WWW to proces oparty na metodzie prób i błędów. Próbujesz działanie jednego zestawu parametrów, a jeśli to, co zobaczysz, nie spełnia Twoich oczekiwań, próbujesz inny zestaw parametrów. Aby zmodyfikować parametry behawiora, należy dwukrotnie kliknąć jego nazwę, żeby otworzyć okno dialogowe. Funkcja `inspectBehavior()` umieszcza w polach okna dialogowego poprzednio ustalone wartości.
- *Etap 6. Przetestuj behawior*. Ostatnią czynnością jest testowanie. Przetestuj działanie behawiora w różnych przeglądarkach i usuń błędy, jeśli jakieś wystąpią (a tego możesz być pewny).

W kilku następnych podrozdziałach przedstawiony jest proces tworzenia behawiora *Set Layer Z Index* autorstwa Massimo Fotiego. Zmiana położenia warstwy w stosie (czyli wartości parametru *z-index*) jest łatwa podczas tworzenia strony, ale zrealizowanie tego samego na „żywej" stronie nie jest łatwe. M. Foti zaprojektował behawior umożliwiający interaktywną kontrolę położenia warstwy w stosie. Behawior *Set Layer Z Index* jest stosunkowo prosty, ale bardzo elegancko napisany i przez to stanowi idealny materiał do nauki.


Do Dreamweavera dołączany jest behawior *Change Property*, za pomocą którego również można zmienić kolejność warstw w stosie, ale nie jest on zgodny z obydwoma typami przeglądarek. Aby osiągnąć za jego pomocą taki efekt, jak za pomocą behawiora napisanego przez M. Fotiego, musiałbyś przypisać go dwa razy — raz dla Netscape Navigatora, a drugi raz dla Internet Explorera.

## Etap 1. Zdefiniuj behawior

Behawiory powstają w wyniku konkretnej potrzeby. Wielokrotne powtarzanie tej samej operacji prowadzi zwykle do konkluzji: „Musi istnieć na to lepszy sposób". „Lepszy sposób" oznacza zwykle automatyzację tego procesu. Jeśli procesem tym jest wstawianie do kodu strony funkcji JavaScript, to rozwiązaniem jest behawior.

Takie postawienie problemu jest jednocześnie pierwszą fazą tworzenia behawiora — określeniem jego zadania. Jeśli zadaniem tym miałoby być wstawienie na stronę kodu umożliwiającego dynamiczną zmianę kolejności warstw w stosie, kod wyglądałby następująco:

```
<html>
<head>
<title>Untitled Document</title>
<meta http-equiv="Content-Type" content="text/html; charset=iso-8859-1">
<script language="JavaScript">
<!--
function tmt_LayerIndex(theTarget, theValue) {
    if (document.layers) {
        target = eval(theTarget);
        target.zIndex = theValue;
    }
    if (document.all) {
        eval("theTarget=theTarget.replace(/.layers/gi, '.all')");
        eval(theTarget + ".style.zIndex = theValue");
    }
}
//-->
</script>
</head>
<body bgcolor="#FFFFFF">
<a href="#" onClick="tmt_LayerIndex('document.layers[\'backLayer\']','4')" ¬
>Click to Bring Layer to Front</a>
<div id="backLayer" style="position:absolute; left:125px; top:95px; ¬
width:327px; height:142px; z-index:1; background-color: #CCCCFF; ¬
layer-background-color: #CCCCFF; border: 1px none #000000">
  <p>Back layer</p>
</div>
<div id="frontLayer" style="position:absolute; left:238px; top:45px; ¬
width:162px; height:258px; z-index:2; background-color: #FFCCCC; ¬
layer-background-color: #FFCCCC; border: 1px none #000000">
  <p>Front Layer</p>
</div>
</body>
</html>
```

Zwróć uwagę na linie wyróżnione pogrubioną czcionką. To najważniejsze elementy kodu — funkcja w sekcji `<head>` (akcja) i funkcja dołączona do tekstu (zdarzenie). Testy w różnych przeglądarkach wypadły pomyślnie, zatem można na podstawie tej funkcji stworzyć behawior.

Tworząc behawior w ten sposób, określasz od razu argumenty, których wartości będzie można modyfikować. W naszym przykładzie są dwa takie argumenty: `theTarget` i `theValue`. W idealnym przypadku akcja powinna być na tyle elastyczna, aby możliwe było określenie wartości dowolnego argumentu. W naszym przykładzie są dwa atrybuty, których wartości będzie można ustalać w oknie dialogowym.

Po stworzeniu i przetestowaniu funkcji w Dreamweaverze zapisz ją na dysk. Często podczas tworzenia pliku akcji powracam do oryginalnego pliku, aby sprawdzić, czy nie popełniłem żadnego błędu.

## Etap 2. Stwórz plik akcji

W kolejnym etapie procesu tworzenia behawiora stworzysz szkielet pliku akcji i rozpoczniesz dodawanie do niego niezbędnych funkcji JavaScriptu. W każdym pliku akcji muszą znajdować się co najmniej cztery funkcje.

- `canAcceptBehavior()`. Określa, czy behawior może być dostępny. Jeśli nie jest, nie można wybrać go z menu *Add Action*.
- `behaviorFunction()`. Wstawia funkcje do sekcji `<head>` dokumentu.
- `applyBehavior()`. Dołącza funkcje do zaznaczonego znacznika i wstawia wybrane zdarzenie.
- `inspectBehavior()`. Umożliwia ponowne otwarcie okna dialogowego behawiora i zmianę wartości parametrów.


Najłatwiejszym sposobem utworzenia pliku akcji jest zmodyfikowanie istniejącego. Możesz otworzyć i modyfikować dowolny plik akcji dołączony do Dreamweavera. Pamiętaj jednak o tym, aby zapisać zmodyfikowany plik poleceniem *Save As*.

Do behawiorów i innych rozszerzeń można dołączać zewnętrzne pliki z kodem JavaScriptu za pomocą znacznika `<script language="javascript" src="script.js"> </script>`. Możliwość tę wykorzystują wszystkie behawiory dołączane do Dreamweavera. Zaletą takiego rozwiązania jest możliwość wielokrotnego wykorzystania tego samego kodu przez różne behawiory. Możesz oczywiście umieścić interfejs użytkownika i funkcje JavaScriptu w jednym pliku, ale obecnie zaleca się przechowywać je w dwóch osobnych plikach — okno dialogowe w pliku HTML, a funkcje w pliku o identycznej nazwie, ale z rozszerzeniem *.js*.


Nasz przykład jest bardzo prosty. Nie wykorzystamy w nim opisywanej wyżej techniki.

Zaczynamy tworzenie pliku akcji.

1. Wybierz z menu polecenie *File⇨New*.
2. Wybierz z menu polecenie *Modify⇨Page Properties*, aby zmienić tytuł behawiora.
3. Wybierz z menu polecenie *File⇨Save As* i zapisz plik HTML w katalogu *Configuration/Behaviors/Actions*.
4. Przejdź do okna widoku kodu (*Show Code View*), otwórz inspektor *Code* lub edytor tekstu, w którym będziesz wpisywał kod źródłowy akcji.


Najlepiej pracuje się nad formularzem z parametrami (interfejsem użytkownika) w oknie otwieranym kliknięciem przycisku *Show Design View*, a nad plikiem JavaScript — w oknie otwieranym przyciskiem *Show Code View*.

5. Wpisz do sekcji `<head>` dokumentu znacznik `<script language="javascript">...</script>`.
6. Otwórz plik zawierający funkcję.
7. Skopiuj kod funkcji do pliku behawiora.

   W naszym przykładzie skopiowana została funkcja `tmt_LayerIndex()`.


Upewnij się, że nazwa behawiora jest unikalna, czyli że żaden inny behawior nie posiada takiej nazwy. Przy pierwszym otwarciu palety *Behaviors Dreamweaver* sprawdza, czy nie ma w katalogu behawiorów o powtarzających się nazwach. Jeśli są, Dreamweaver uwzględnia tylko ten, który został stworzony wcześniej.

**8.** Dodaj do znacznika `<script>` następujące funkcje:

```
function canAcceptBehavior(){
    return true;
}

function behaviorFunction(){
    return tmt_LayerIndex;
}

function applyBehavior(){
    return "";
}

function inspectBehavior(msgStr){
}
```

Tylko jedna z powyższych funkcji jest „gotowa” — `behaviorFunction()`. Reszta to dopiero podstawy do napisania funkcji.


Funkcja `behaviorFunction()` może zwracać więcej niż jedną funkcję. Więcej informacji na ten temat znajdziesz w podrozdziale „Techniki tworzenia behawiorów” znajdującym się w dalszej części tego rozdziału.

Po stworzeniu podstawowej struktury behawiora należy zdefiniować sytuacje, w których behawior może zostać wykorzystany. Realizuje się to za pomocą funkcji `canAcceptBehavior()`. Jeśli behawior nie wymaga istnienia na stronie określonych elementów, możesz pozostawić tę funkcję bez zmian. Nasz przykładowy behawior wykorzystuje warstwy, więc powinien być dostępny tylko wtedy, gdy na stronie znajdują się warstwy. Sprawdzenia obecności warstw dokonuje się za pomocą następującego kodu:

```
function canAcceptBehavior(){
  var nameArray = getObjectRefs("NS 4.0","document","LAYER");
  return (nameArray.length > 0);
}
```

Jeśli funkcja `getObjectRefs()` znajdzie na stronie warstwy, rozmiar tablicy `nameArray` będzie większy od zera i funkcja `canAcceptBehavior()` zwróci wartość *true*. Jeśli funkcja zwróci wartość *false*, behawior będzie znajdował się w menu *Add Action*, ale jego wybranie nie będzie możliwe.

## Etap 3. Stwórz interfejs użytkownika

Interfejsem użytkownika w przypadku behawiora jest formularz wprowadzania parametrów, stworzony za pomocą elementów formularza dostępnych w języku HTML. Liczba i typ argumentów wymaganych przez funkcję są głównymi wyznacznikami tego, co powinno znaleźć się w formularzu.

W naszym przykładzie funkcja wymaga dwóch argumentów: nazwy warstwy (argument `theValue`) i liczby określającej położenie warstwy w stosie (argument `theValue`). Interfejs powinien więc umożliwiać użytkownikowi wprowadzenie wartości tych argumentów.

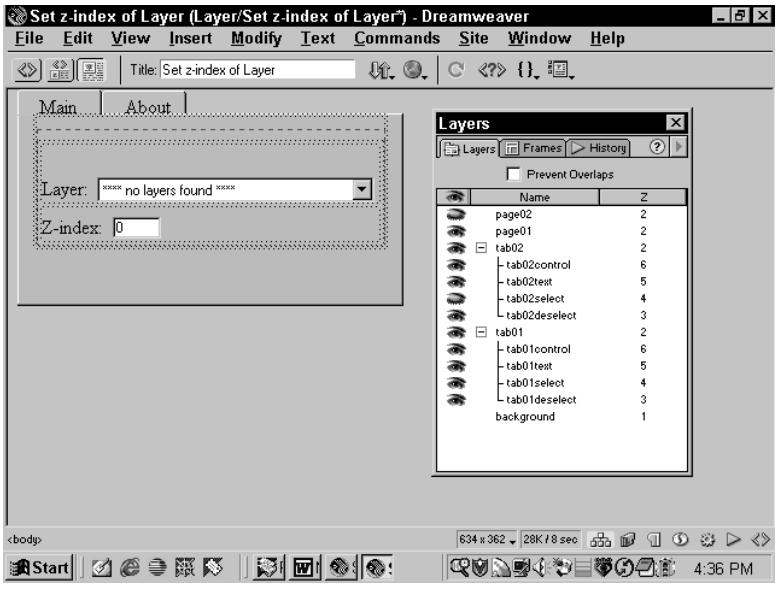
Wszystkie elementy interfejsu użytkownika znajdują się w sekcji `<body>` pliku HTML. Możesz do ich tworzenia wykorzystać narzędzia Dreamweavera. Wielu projektantów wykorzystuje tabele do wyrównywania elementów formularza. Jeśli Ty również zamierzasz to zrobić, pamiętaj o tym, aby umieścić tabelę wewnątrz znacznika `<form>`, a nie odwrotnie. W przeciwnym przypadku mógłbyś umieścić formularz jedynie w jednej komórce tabeli.

Tworzymy interfejs użytkownika.

1. Otwórz plik akcji w Dreamweaverze.
2. W oknie dokumentu wybierz z menu polecenie *Insert⇨Form* lub kliknij przycisk *Insert Form* znajdujący się w palecie *Objects*. Nadaj formularzowi nazwę.
3. Umieść w formularzu tabelę, wybierając z menu polecenie *Insert⇨Table* lub klikając przycisk *Insert Table* w palecie *Objects*.
4. Dodaj do formularza wszystkie niezbędne elementy. Pamiętaj o nadawaniu każdemu z nich nazwy (wyjątkiem są grupy przycisków opcji — tu nadajesz nazwę całej grupie).


Podobnie jak w przypadku obiektów, tu również, tworząc formularz wprowadzania parametrów, nie zobaczysz w nim przycisków *OK*, *Cancel* i *Help*. Dreamweaver automatycznie umieszcza je w prawym górnym rogu interfejsu.

Interfejs użytkownika dla akcji *Set Layer Z Index*, przedstawiony na rysunku 20.1, wykorzystuje listę opcji, w której znajdują się nazwy dostępnych na stronie warstw oraz pole tekstowe do wpisania wartości parametru *z-index*.




**Rysunek 20.1.** *W formularzu wykorzystane są pola tekstowe przekazujące dane do odpowiednich funkcji*

Ostatnią czynnością w procesie tworzenia interfejsu użytkownika jest napisanie funkcji inicjalizującej go i umieszczającej kursor tekstowy w odpowiednim polu. Do tego celu służy funkcja `initializeUI()`, znajdująca się w sekcji *Local Functions* kodu źródłowego. W naszym przykładzie w interfejsie użytkownika wszystkie znajdujące się na stronie warstwy powinny być wyświetlane w liście. Aby to zrealizować, funkcja `initializeUI()` wywołuje inną funkcję:

```
function initializeUI() {
  var niceNamesArray;
//Get all the layers available includingparent frames
//Then turn those objects references into nice names
niceNamesArray = niceNames (getAllObjectRefs9"NS
4.0","LAYER"), TYPE_Layer);
//Populate the select element
populateSelect("LAYER_LIST",niceNamesArray);
//Set focus on textbox
findObject("Z-INDEX").focus();
//Set insertion point into textbox
findObject("Z-INDEX").select();
}
```

Do inicjalizacji okna dialogowego Massimo Foti stosuje kombinację funkcji autorskich i tych opracowanych przez firmę Macromedia. Funkcja Macromedii `niceNames()` zmienia referencje obiektu JavaScript, na przykład fragment `document.layer['onLayer'] .document.theForm` zamieniany jest na bardziej czytelną postać: `form "theForm" in layer "onLayer"`. W wyniku tego funkcja `niceNames()` znajduje zastosowanie do wypełniania listy rozwijanej za pomocą opracowanej przez Massimo funkcji `populateSelect()`. Dwie linie, które zaczynają się funkcją `findObject()`, stosowane są do umieszczenia kursora w polu tekstowym po otwarciu okna dialogowego — to wspaniałe potwierdzenie profesjonalizmu Massimo Fotiego.

Na koniec należy przyłączyć funkcję `initializeUI()` do znacznika `<body>` za pomocą zdarzenia `onLoad`. Możesz zrealizować to dwoma sposobami. Pierwszy z nich polega na wpisaniu do znacznika `<body>` dodatkowych atrybutów:

```
<body onLoad="initializeUI()">
```

Drugi sposób polega na wykorzystaniu behawiora *Call JavaScript* (opisanego w rozdziale 19.). W tym przypadku wpisz tylko `initializeUI()` w oknie dialogowym behawiora.

## Etap 4. Dołącz behawior do znacznika

Teraz możesz napisać kod łączący funkcję ze znacznikiem i zdarzeniem. Proces ten można rozłożyć na trzy fazy:

1. Sprawdzenie, czy użytkownik wpisał odpowiednie informacje.
2. Przekonwertowanie informacji z formularza do bardziej użytecznego formatu.
3. Wywołanie odpowiedniej funkcji.

Wszystkie te czynności są realizowane przez funkcję `applyBehavior()` znajdującą się w kodzie źródłowym.

Informacje z formularza wprowadzania parametrów pobiera się w podobny sposób, w jaki było to realizowane w przypadku obiektów. Wykorzystując techniki omówione w rozdziale 18., można odczytać dane i przekonwertować je na zmienne lokalne. Liczba zmiennych jest równa liczbie wymaganych argumentów funkcji.


Jeśli przewidujesz, że część danych z formularza będzie wysyłana do serwera, musisz zakodować je tak, aby mogły być odczytane przez serwery pracujące pod kontrolą systemu Unix. Za pomocą funkcji `escape` możesz przekonwertować spacje i znaki specjalne na kody wykorzystywane przez Unix. Funkcja `unescape` ma odwrotne działanie i jest wykorzystywana w funkcji `inspectBehavior()`.

Teraz utworzymy funkcję `applyBehavior()`.

1. Zdefiniuj niezbędne zmienne:

```
var theTarget
var theValue
```

2. Pobierz informacje z formularza. Przebieg tego procesu zależy od typu pól wykorzystywanych w formularzu. W przypadku listy opcji należy znaleźć odpowiednią wartość w tablicy zawierającej nazwy warstw:

```
divArray = getObjectRefs("NS 4.0", "document", "LAYER");
theTarget="'" + escQ(divArray[theForm.le.selectedIndex]) + "'";
```

   Funkcja `escQ()` konwertuje nazwę warstwy do odpowiedniego formatu.

3. Wywołaj funkcję uruchamiającą behawior z odpowiednimi parametrami. Funkcja `applyBehavior()` musi zwrócić łańcuch składający się z wartości zmiennych. Zamknij wartości argumentów w apostrofach. Jeśli w tekście znajdują się cudzysłowy, musisz poprzedzić je znakiem ukośnika wstecznego.

4. Sprawdź, czy wpisane zostały wszystkie wartości. Jeśli nie, poinformuj o tym użytkownika.

```
myErr = (myErr=="")? "":"The following fields must be ¬
filled: "+myErr;
if (!myErr)
return "tmt_LayerIndex("+ theTarget+ ", " + theValue ¬
+ ")";
else return myErr;
```

Przed przetestowaniem akcji musisz wykonać jeszcze jedną czynność.

## Etap 5. Zbadaj behawior

Nadszedł czas na dodanie do kodu behawiora funkcji `inspectBehavior()`. Funkcja ta jest wywoływana, gdy użytkownik dwukrotnie kliknie nazwę behawiora w palecie *Behaviors*. Wyświetla ona poprzednio wprowadzone przez użytkownika dane w formularzu wpisywania parametrów i umożliwia ich zmianę. W wielu przypadkach funkcja ta może być uważana za przeciwieństwo funkcji `applyBehavior()` — zamiast pobierać dane z formularza i przesyłać je do kodu, pobiera dane z kodu i umieszcza je w formularzu.

Interpretacja danych z formularza określana jest mianem przetwarzania łańcucha (*parsing*). Akcja *Set Layer Z Index* przekazuje łańcuch w następującej formie:

```
onClick="tmt_LayerIndex('document.layers[\'backLayer\']','4')"
```

Dreamweaver podczas przetwarzania łańcuchów wykorzystuje kilka funkcji, ale najważniejszą z nich jest funkcja `getTokens()`. Funkcja ta przyjmuje jako argumenty łańcuch tekstowy oraz listę separatorów, a zwraca tablicę złożoną z łańcuchów tekstowych. Pierwszym argumentem funkcji `getTokens()` jest łańcuch, który ma zostać przetworzony. Drugi argument powinien zawierać nawiasy, apostrof i przecinek, czyli znaki wykorzystywane w łańcuchu jako separatory:

```
var argArray = getTokens(msgStr,"()',");
```

Kiedy łańcuchy zostaną umieszczone w tablicy, mogą zostać z niej odczytane i umieszczone ponownie w formularzu wprowadzania parametrów. Wykonując poniższe czynności, stworzysz funkcję `inspectBehavior()`.

1. Zdefiniuj zmienną i przypisz jej wartość do wartości zwracanej przez funkcję `getTokens()`.
2. Przypisz elementy tablicy tym samym zmiennym, których użyłeś w funkcji `applyBehavior()`:

   ```
   var theValue=unescQ(argArray[1]);
   ```

3. Umieść wartości zmiennych w odpowiednich polach formularza:

   ```
   theForm.le.selectedIndex=j;
   theForm.theZvalue.value=argArray[2]
   ```

4. Gotowa funkcja `inspectBehavior()` wygląda następująco:

   ```
   function inspectBehavior(msgStr){
       aargArray = extractArgs(msgStr);
       var argArray = getTokens(msgStr,"()',");
       var ii=0; var j=0;
       divArray = getObjectRefs("NS 4.0", "document", "LAYER");
       for (j=0;j<divArray.length;j++){
           myImg=unescQ(argArray[1]);
           if (myImg==divArray[j]){
               theForm.le.selectedIndex=j;
           }
       }
       theForm.theZvalue.value=argArray[2]
   }
   ```


W naszym przykładzie funkcja `inspectBehavior()` jest stosunkowo prosta. Pamiętaj jednak, że im więcej danych w behawiorze wprowadza użytkownik, tym bardziej skomplikowany będzie proces ich wyświetlania w formularzu. Podobnie jak w innych etapach procesu tworzenia behawiora, tu również najlepszym sposobem stworzenia odpowiedniej funkcji `inspectBehavior()` będzie przeanalizowanie behawiorów dołączanych do Dreamweavera.

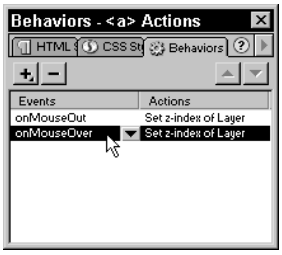
## Etap 6. Przetestuj behawior

Testowanie i usuwanie błędów z behawiora jest ostatnim i niezbędnym etapem w procesie jego tworzenia. Aby przetestować behawior, wykonaj następujące czynności:

1. Uruchom ponownie Dreamweavera.
2. Wstaw na stronę obrazek lub łącze.

3. Zaznacz element, który będzie wyzwalał akcję.
4. Otwórz paletę *Behaviors*.
5. Wybierz zdarzenie `onClick` z menu *Add Event*.
6. Z menu *Add Action* wybierz nazwę swojego behawiora.
7. Wpisz odpowiednie parametry w formularzu.

   Nazwa akcji zostanie wyświetlona w panelu *Action* (zobacz rysunek 20.2).




**Rysunek 20.2.** *Odpowiednio napisane behawiory są wyświetlane w palecie Behaviors*

8. Kliknij dwukrotnie nazwę akcji, aby sprawdzić, czy wartości, które poprzednio wprowadziłeś, są wyświetlane w polach formularza.
9. Przetestuj działanie behawiora w różnych przeglądarkach.


Kiedy po raz pierwszy wybierzesz zdarzenie, Dreamweaver bada wszystkie pliki znajdujące się w katalogu *Actions*. Jeśli wystąpi problem, taki jak obecność dwóch plików o identycznych nazwach, zostaniesz o tym poinformowany, a w liście wyświetlony zostanie jedynie starszy plik. Będziesz wówczas musiał skorygować nazwy plików i ponownie uruchomić program.

Jeśli planujesz rozpowszechnianie behawiora, przetestuj go bardzo dokładnie, szczególnie działanie interfejsu użytkownika. Jako twórca behawiora będziesz wiedział, jakie wartości mają być wpisane w formularzu i jak uniknąć potencjalnych błędów. Użytkownik nie będzie posiadał tak dokładnych informacji. Szczególną uwagę zwróć na informacje wpisywane w pola tekstowe. Jeśli informacja ta nie jest treścią komunikatu wyświetlanego na ekranie lub listwie statusowej, musi zostać zweryfikowana pod kątem poprawności danych. Samo poinformowanie użytkownika o tym, że powinien wpisać liczby z określonego przedziału, nie daje gwarancji, że to właśnie zrobi.

## Usuwanie błędów z behawiora

Wystąpienie błędu w programie jest chyba najmniej lubianą przez programistów sytuacją, ale za to usunięcie tego błędu jest wielkim sukcesem i źródłem satysfakcji. W przypadku wystąpienia problemu najpierw powinieneś sięgnąć po podstawowe techniki usuwania błędów ze skryptów JavaScript, takie jak np. zastosowanie funkcji `alert()` w celu wyświetlania wartości zmiennych. Dreamweaver wyświetla komunikaty błędów w taki sam sposób jak przeglądarki. Obsługa błędów w Dreamweaverze rozwiązana jest bardzo dobrze. Wiele komunikatów o błędach wskazuje od razu błędny kod.

Jeśli błędy są na tyle poważne, że Dreamweaver nie rozpoznaje kodu jako akcji, plik nie jest wymieniany w liście *Action*. W takiej sytuacji musisz ponownie uruchomić program po usunięciu błędu. Jeśli jednak usuwasz z kodu drobne błędy, możesz uniknąć konieczności ponownego uruchamiania Dreamweavera, wykorzystując poniższe wskazówki.

1. Otwórz plik akcji i usuń z niego błędy. Zapisz plik na dysk.
2. Przyłącz akcję do znacznika i otwórz okno dialogowe behawiora. Nie wpisuj żadnych parametrów, tylko kliknij przycisk *Cancel*.
3. Usuń akcję z palety *Actions*.
4. Przypisz akcję ponownie. Dreamweaver załaduje nową wersję pliku.


Pamiętaj, że w funkcjach JavaScript rozróżniane są duże i małe litery. Jeśli wyświetlony zostanie komunikat o niemożliwości znalezienia funkcji, sprawdź, czy poprawnie wpisałeś jej nazwę.

# Dokumentacja Extending Dreamweaver

Aby pomóc programistom tworzącym behawiory, firma Macromedia udostępniła dokumentację *Extending Dreamweaver* (rozbudowywanie Dreamweavera). Dokumentacja ta opisuje funkcje przydatne przy tworzeniu behawiorów. Omawia podstawy działania, tworzenia i wykorzystywania behawiorów oraz wykorzystywania rozszerzeń i wbudowanych funkcji. Dokumentację *Extending Dreamweaver* znajdziesz w menu *Help*.

W dokumentacji *Extending Dreamweaver* opisane są wszystkie rodzaje rozszerzeń Dreamweavera, ale dla twórców behawiorów najbardziej przydatne będą trzy jej części: *Document Object Model* (opisująca obiektowy model dokumentu), *Dreamweaver JavaScript API* (interfejs JavaScriptu w Dreamweaverze) oraz *Behaviors* (omawiająca tworzenie behawiorów). Im więcej będziesz wiedział o różnych komponentach i ich funkcjach, tym łatwiej będzie Ci tworzyć kolejne behawiory.


Materiał opisany w tym podrozdziale przeznaczony jest dla programistów znających język JavaScript, a więc dla Czytelników bardzo zaawansowanych.

## Obiektowy model dokumentu

JavaScript to interpretowany język programowania, odwołujący się do elementów przeglądarki i strony WWW w sposób hierarchiczny. Aby uzyskać dostęp do właściwości każdego obiektu znajdującego się na stronie, JavaScript wykorzystuje obiektowy model dokumentu (*DOM*). DOM rozbija stronę na małe elementy, tak aby można było zidentyfikować każdy z jej elementów.

Jak wspomniałem we wprowadzeniu, Dreamweaver 4 posiada bardzo rozbudowany DOM, w którym zawarte są elementy obiektowego modelu dokumentu używanego przez Netscape Navigatora oraz elementy DOM określonego przez W3C. Oprócz tego

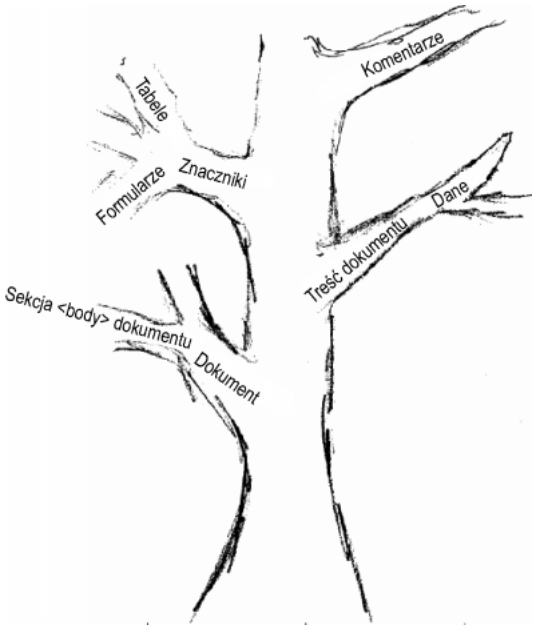
można znaleźć w nim niezwykle użyteczne funkcje nieobecne w żadnej innej implementacji DOM. Chyba największe rozszerzenie DOM Dreamweavera jest związane z paletą History. Wcześniej każda akcja użytkownika, która miała być powtarzana, wymagała obecności odpowiedniego kodu JavaScript.

## Koncepcja węzłów

DOM Dreamweavera udostępnia niemal każdy element strony WWW. DOM jest zbudowany analogicznie do drzewa, w którym dokument HTML jest pniem. Głównymi gałęziami tego drzewa nie są jednak sekcje <head> i <body>. DOM Dreamweavera, na wzór DOM zdefiniowanego przez W3C, wykorzystuje cztery oddzielne gałęzie (zwane również węzłami — *node*):

- *DOCUMENT_NODE* — umożliwia dostęp do obiektów związanych z całym dokumentem;
- *ELEMENT_NODE* — zawiera odwołania do wszystkich znaczników HTML w dokumencie;
- *TEXT_NODE* — opisuje bloki tekstu zawarte pomiędzy znacznikami HTML;
- *COMMENT_NODE* — reprezentuje wszystkie komentarze znajdujące się w dokumencie.

Jedna gałąź drzewa może rozdzielać się na mniejsze i — analogicznie — jeden węzeł może zawierać inne. Warstwa może zawierać tabelę, w której z kolei znajdują się komórki z zapisanymi danymi. Takie połączenie węzłów nazywane jest połączeniem hierarchicznym (*parent-child relationship*), a węzeł, który nie zawiera innych węzłów, nazywany jest liściem (*leaf node*). Rysunek 20.3 przedstawia schemat koncepcji węzłów w dokumencie.




**Rysunek 20.3.** *Koncepcja węzłów jest wykorzystywana do określenia struktury dokumentu HTML i jego powiązań z przeglądarką*

### Właściwości obiektowego modelu dokumentów (DOM)

Składnia odwołania do określonego znacznika opiera się na kolejnych elementach, od najwyżej położonego w hierarchii do najniższego. Na przykład załóżmy, że chcesz sprawdzić, co użytkownik wpisał w pole tekstowe zwracające parametr o nazwie *value*. Odwołujesz się więc do dokumentu i posuwasz się w dół hierarchii:

```
var theText = document.theForm.textbox.value
```

DOM określa, jakie właściwości i w jakiej formie są dostępne. Nie wszystkie właściwości i metody można bezpośrednio wykorzystać. Np. nie możesz odwołać się bezpośrednio do wartości zwracanej przez przycisk w formularzu. Musisz przypisać tę wartość do pola ukrytego i dopiero wtedy będziesz mógł ją wykorzystać.

Składniki DOM związane z formularzami i ich elementami zostały opisane w rozdziale 18. Reguły wykorzystywania i ograniczenia dotyczące używania formularzy w obiektach dotyczą również wykorzystywania formularzy w behawiorach. Dodatkowo, DOM Dreamweavera udostępnia inne obiekty, co przedstawione jest w tabeli 20.1. Właściwości nadające się tylko do odczytu są oznaczone symbolem „*”, pozostałe można odczytywać i ustawiać.

### Metody DOM

Metoda (*method*) to w programowaniu określenie funkcji dołączonej do określonego obiektu, na przykład do dokumentu. DOM Dreamweavera zawiera wiele metod ułatwiających pracę z dokumentami HTML. Dzięki węzłom możesz używać tych metod w odniesieniu do dokumentu, zestawu ramek, ramki lub zaznaczonego obiektu.

Wykorzystując te metody, behawiory mogą analizować stronę i modyfikować lub nawet usuwać atrybuty znaczników. W tabeli 20.2 znajduje się lista metod DOM Dreamweavera.

## Rozszerzenia API Dreamweavera

Interfejs programowania (API) Dreamweavera 4 został rozbudowany tak, że posiada funkcje odpowiadające niemal wszystkim operacjom wykonywanym przez program Dreamweaver. Zawiera on ponad 400 funkcji, z których wiele zostało stworzonych z myślą o współpracy z paletą *History*. Dokładne omówienie API wykracza daleko poza ramy tej książki, ale twórcy behawiorów powinni zapoznać się z jego możliwościami. Część *JavaScript API* dokumentacji *Extending Dreamweaver* jest podzielona na następujące kategorie:

- *Behaviors* — behawiory,
- *Clipboard* — schowek systemowy,
- *Command* — polecenia,
- *Conversion* — konwersja,
- *CSS Style* — style CSS,
- *External Application* — zewnętrzne aplikacje,
- *File Manipulation* — operacje na plikach,

**Tabela 20.1.** *Właściwości DOM Dreamweavera*

| Właściwość | Węzły | Opis | Zwracane wartości |
|---|---|---|---|
| nodeType* | Wszystkie | Zwraca nazwę węzła, w którym znajduje się zaznaczony element | DOCUMENT_NODE<br>ELEMENT_NODE<br>TEXT_NODE<br>COMMENT_NODE |
| parentNode* | Wszystkie | Zwraca nazwę węzła nadrzędnego, lub — jeśli zaznaczony jest znacznik — nazwę dokumentu | Dowolny węzeł |
| parentWindow* | DOCUMENT_NODE | Zwraca nazwę obiektu JavaScript odpowiadającego głównemu oknu dokumentu | Łańcuch tekstowy |
| childNodes* | Wszystkie | Zwraca listę węzłów znajdujących się w hierarchii pod zaznaczonym | Tablica |
| documentElement* | DOCUMENT_NODE | Odpowiada znacznikowi <html> aktualnego dokumentu | "<html><br><head>... </body><br></html>"<br>(jeśli użyta w połączeniu z właściwością outerHTML) |
| body* | DOCUMENT_NODE | Odpowiada znacznikowi <body> aktualnego dokumentu | "<body>... </body>"<br>(jeśli użyta w połączeniu z outerHTML) |
| URL* | DOCUMENT_NODE | Zwraca ścieżkę dostępu do dokumentu | "FILE://C\|/DOCS/NEW.HTML" — przykład |
| tagName* | ELEMENT_NODE | Zwraca nazwę znacznika | "IMG" or "TABLE"<br>— przykład |
| attrName | ELEMENT_NODE | Zwraca wartość podanego atrybutu | "grey" or "#33CC66"<br>— przykład |
| innerHTML | ELEMENT_NODE | Zwraca kod źródłowy HTML znajdujący się wewnątrz określonego znacznika | "First Name" |
| outerHTML | ELEMENT_NODE | Zwraca kod źródłowy HTML znajdujący się wewnątrz określonego znacznika wraz z tym znacznikiem | " First Name " |
| data | TEXT_NODE<br>COMMENT_NODE | Zwraca łańcuch tekstowy znajdujący się wewnątrz określonego znacznika | "J. Lowery" dla znacznika J. Lowery |

**Tabela 20.2.** *Metody DOM Dreamweavera*

| Metoda | Zwracana wartość | Opis |
|---|---|---|
| `getElementsByTagName (nazwa_znacznika)` | Lista węzłów | Tworzy tablicę wystąpień określonego znacznika w kodzie strony |
| `getTranslatedAttribute (nazwa_znacznika)` | Wartość | Pobiera wartość określonego atrybutu. Używana w połączeniu z filtrami (*translators*) |
| `hasChildNodes()*` | Wartość logiczna | Określa, czy zaznaczony element posiada elementy potomne w hierarchii |
| `hasTranslatedAttributes()` | Wartość logiczna | Określa, czy zaznaczony znacznik posiada atrybuty interpretowane za pomocą filtrów |
| `getAttribute (nazwa_atrybutu)` | Wartość | Pobiera wartość wskazanego atrybutu |
| `setAttribute(nazwa_ atrybutu,wartość_atrybutu)` | Brak | Przypisuje określonemu atrybutowi podaną wartość |
| `removeAttribute(nazwa_ atrybutu)` | Brak | Usuwa określony atrybut |

- *Find/Replace* — wyszukiwanie i zastępowanie,
- *Frame and Frameset* — ramki,
- *General Editing* — edycja,
- *Global Applications* — aplikacje,
- *Global Document* — dokument,
- *History* — paleta *History*,
- *HTML Style* — style HTML,
- *Keyboard* — klawiatura,
- *Layer and Image Map* — warstwy i mapy obrazków,
- *Menu* — menu,
- *Path* — ścieżki dostępu do plików,
- *Quick Tag Editor* — edytor *Quick Tag*,
- *Selection* — zaznaczanie elementów,
- *Site* — okno *Site*,
- *String Manipulation* — operacje na tekstach,
- *Table Editing* — edycja tabel,
- *Toggle* — przełączniki,
- *Translation* — filtry,
- *Visual Layout* — wizualne projektowanie układu strony,
- *Window* — okna.

Twórców behawiorów szczególnie zainteresują funkcje opisane w częściach *Behaviors*, *File Manipulation*, *Global Document*, *Path*, *Selection* i *String Manipulation*.

Programiści z firmy Macromedia opracowali również API dla operacji wejścia-wyjścia, dziennika projektu, integracji z Fireworks i połączeń HTTP. Extending Dreamweaver zawiera ogólny opis każdego z nich.

Aby ułatwić poznanie API Dreamweavera, zamieściłem w następnych podrozdziałach opisy najczęściej używanych rozszerzeń API. Opisy oczywiście nie wyczerpują tematu, ale są źródłem najważniejszych informacji na temat sposobu działania funkcji API.


Zauważ, że nazwy rozszerzeń API mają przedrostek „.dreamweaver" lub „.dom". Przedrostek „.dreamweaver" może być skrócony do postaci „.dw", na przykład „dw.getDocumentDOM()". Funkcje z przedrostkiem „.dom" odwołują się do DOM dokumentu zwracanego przez funkcję `getDocumentDOM()`, opisaną w następnym podrozdziale.

## Funkcja dreamweaver.getDocumentDOM()

Funkcja `getDocumentDOM()` jest bazą wielu operacji wykonywanych za pomocą JavaScriptu w Dreamweaverze. Przypisanie wartości tej funkcji do zmiennej powoduje zwrócenie całego DOM określonego dokumentu, co umożliwia jego odczytywanie i edycję. Ogólnie, funkcja ta jest wykorzystywana w następujący sposób:

```
var theDom = dreamweaver.getDocumentDOM()("document")
```

W tym przykładzie zmienna *theDom* określa dokument i wszystkie znajdujące się w nim elementy. Po odwołaniu się do DOM w ten sposób możesz odwołać się do bardziej szczegółowych informacji. Np. jeśli chcesz zbadać sekcję `<body>` dokumentu, możesz użyć następujących funkcji:

```
var theDom = dreamweaver.getDocumentDOM()("document")
var theBody = theDom.body
```

Możesz również zapisać to w postaci skróconej:

```
var theBody = dreamweaver.getDocumentDOM()("document").body
```


Wiele behawiorów odwołuje się do DOM dokumentu bardzo często, więc dobrze jest przypisać go na początku skryptu do zmiennej.

Funkcja `getDocumentDOM()` wymaga jednego argumentu, *sourceDoc*, określającego dokument. Argument może przyjmować następujące postaci:

- `"document"`. Określa aktualnie otwarty dokument. Argument ten może być użyty w dowolnym miejscu skryptu, ale wszelkie wykonywane za jego pomocą modyfikacje muszą być wywoływane z funkcji `applyBehavior()`, `deleteBehavior()` lub `objectTag()` (lub dowolnej funkcji z inspektora *Property* bądź *Command*).
- `"parent"`. Określa dokument nadrzędny względem aktualnego. Argument ten jest zwykle wykorzystywany do określania, czy dokument znajduje się w ramce:

```
var frameset = dreamweaver.getDocumentDOM("parent");
if (frameset) { ... kod ... }
```

- `"parent.frames[liczba]"` lub `"parent.frames['nazwa_ramki']"`. Takie postaci argumentu są wykorzystywane do odwoływania się do innego dokumentu, znajdującego się w tym samym zestawie ramek. Pierwsza forma jest wykorzystywana w sytuacjach, gdy nazwy ramek są nieznane lub gdy trzeba przeszukać określoną liczbę ramek. Druga postać określa dokument znajdujący się w ramce o podanej nazwie.
- Adres URL. Czasami twórca behawiora odwołuje się do istniejących dokumentów znajdujących się na dysku lokalnym lub w sieci. Wykorzystanie w roli argumentu adresu URL (w postaci bezwzględnej lub względnej) umożliwia uzyskanie informacji o niemal dowolnym dokumencie. Jeśli używasz adresu względnego, pamiętaj, że musisz określić go względem położenia behawiora:

```
var idRoot = dreamweaver.getDocumentDOM¬
('../../../Help/contextID.html')
```


Jeśli określona funkcja API wymaga DOM dokumentu, jak na przykład funkcja `dom.getSelection()` i inne omówione w następnych podrozdziałach, musisz najpierw uzyskać do niego dostęp. W następnych przykładach założono, że zmiennej `theDom` przypisano już odpowiednią wartość:

```
var theDOM = dreamweaver.getDocumentDOM("document")
var theSel = theDOM.getSelection()
```

### Funkcja dom.getSelection()

Działanie behawiora jest często determinowane przez to, jaki znacznik został zaznaczony przez użytkownika przed dołączeniem behawiora. Wykorzystanie funkcji `getSelection()` jest pierwszym krokiem w zbieraniu informacji niezbędnych do kontrolowania behawiora na podstawie zaznaczonego znacznika. Podkreślam sformułowanie *pierwszym krokiem*, ponieważ funkcja ta zwraca informację o zaznaczonym znaczniku w postaci *offsetu* (przesunięcia) mierzonego w bajtach. *Offset* to liczba wskazująca adres komórki pamięci. W przypadku wartości zwracanych przez funkcję `getSelection()` dwa offsety wskazują adresy początku i końca zaznaczenia. Jeśli wpiszesz na pustej stronie w Dreamweaverze zdanie: „Dwa główne założenia" i zaznaczysz pierwsze słowo, funkcja `getSelection()` użyta w następujący sposób:

```
var selArray = theDOM.getSelection()
alert(selArray)
```

spowoduje wyświetlenie w oknie komunikatu dwóch liczb:

```
161,164
```

które wskazują pierwszy i ostatni bajt offsetu zaznaczonego słowa. Jeśli pierwszy i ostatni bajt są takie same (na przykład 164 i 164), oznacza to, że nic nie zostało zaznaczone. Fakt ten przydaje się w sytuacji, gdy chcesz upewnić się, że użytkownik zaznaczył jakiś element na stronie.

Aby określić, co znajduje się pod adresem wskazywanym przez offsety zwrócone przez funkcję `getSelection()`, musisz użyć funkcji `offsetsToNode()`, omówionej w jednym z dalszych podrozdziałów.

### Funkcja dom.setSelection()

Funkcja `dom.setSelection()` działa przeciwnie do funkcji `getSelection()`, mianowicie definiuje nowe offsety i przez to nowy obszar zaznaczenia w dokumencie. Funkcja ta wymaga podania dwóch argumentów: *offsetBegin* (początek offsetu) i *offsetEnd* (koniec offsetu).

Funkcja `setSelection()` jest zwykle wykorzystywana do przywracania zaznaczenia określonego przez użytkownika. W poniższym przykładzie obszar zaznaczenia zapisywany jest w zmiennej za pomocą funkcji `getSelection()`, a później, po modyfikacjach dokumentu, odtwarzany za pomocą funkcji `setSelection()`:

```
var currSelection = theDOM.getSelection()
// tutaj kod funkcji modyfikujących dokument
theDOM.setSelection(currSelection[0],currSelection[1])
```


Jeśli nowy obszar zaznaczenia obejmuje, na przykład, atrybuty w znaczniku, jest rozszerzany tak, aby objął cały znacznik.

Możesz również wykorzystać funkcję `setSelection()` do usunięcia zaznaczenia po wykonaniu behawiora. W tym przypadku oba argumenty funkcji powinny mieć tę samą wartość. Kontynuując poprzedni przykład, poniższy kod:

```
theDOM.setSelection(currSelection[1],currSelection[1])
```

spowoduje umieszczenie kursora za zaznaczeniem, a kod:

```
theDOM.setSelection(currSelection[0],currSelection[0])
```

ustawi kursor przed zaznaczeniem.

### Funkcja dom.offetsToNode()

Funkcja `offsetsToNode()` umożliwia przekonwertowanie wartości zwróconych przez funkcję `getSelection()` do postaci bardziej czytelnej. Z tego powodu często używana jest następująca kombinacja funkcji:

```
selArr = theDOM.getSelection();
selObj = theDOM.offsetsToNode(selArr[0],selArr[1]);
```

w której funkcja `getSelection()` zwraca tablicę, gdzie elementami są wartości offsetów, a funkcja `offsetsToNode()` to obiekt określony tymi wartościami. Jak widać, funkcja `offsetsToNode()` wymaga dwóch argumentów: *offsetBegin* i *offsetEnd*, zwykle wyrażanych w postaci pierwszego (czyli o indeksie 0) i drugiego (o indeksie 1) elementu tablicy.

Po wykorzystaniu funkcji `offsetsToNode()` do pobrania zaznaczonego obiektu, możesz użyć go w innych funkcjach. W poniższym przykładzie, zaczerpniętym z polecenia *Replicator* (znajdującego się na dołączonym do książki CD-ROM-ie), wykorzystałem tę funkcję do sprawdzania, czy zaznaczono właściwe elementy (tekst) i — jeśli nie — do wywołania okna systemu pomocy:

```
var offsets = theDOM.getSelection()
var selObj = theDOM.offsetsToNode(offsets[0],offsets[1])
if (selObj.nodeType == Node.TEXT_NODE) {
  helpMe2()
}
```

## Funkcja dom.nodeToOffsets()

Funkcja `nodeToOffsets()` jest przeciwieństwem funkcji `offsetsToNode()`. Konwertuje odnośnik do obiektu na odpowiadające mu offsety. Jest ona przydatna przy manipulowaniu częściami zaznaczenia, na przykład kilkoma znakami zaznaczonego tekstu.

W poniższym przykładzie, zaczerpniętym z polecenia *Change Case* (znajdującego się na dołączonym do książki CD-ROM-ie), zaznaczony obiekt jest pobierany za pomocą funkcji `getSelection()` i `offsetsToNode()`, a funkcja `nodeToOffsets()` zapisuje go w tablicy, której elementy można zmieniać na pisane wielkimi lub małymi literami za pomocą jednego kliknięcia. Oto fragment kodu źródłowego funkcji `upperCase()`, konwertującej zaznaczony tekst na tekst pisanym wielkimi literami:

```
var theDom = dreamweaver.getDocumentDOM("document");
var offsets = theDom.getSelection()
var theNode = theDom.offsetsToNode(offsets[0],offsets[1])
if (theNode.nodeType == Node.TEXT_NODE) {var nodeOffsets = ¬
theDom.nodeToOffsets(theNode)
offsets[0] = offsets[0]-nodeOffsets[0]
offsets[1] = offsets[1]-nodeOffsets[0]
var nodeText = theNode.data
theNode.data = nodeText.substring(0,offsets[0]) +
 nodeText.substring(offsets[0], offsets[1]).toUpperCase() +
 nodeText.substring(offsets[1], nodeText.length);
```

Funkcja `nodeToOffsets` zwraca dwie wartości, więc możesz wykorzystać je jako argumenty funkcji `setSelection` do zaznaczenia obiektu na stronie. Jeśli chciałbyś zaznaczyć pierwsze łącze znajdujące się na stronie, mógłbyś użyć następującego kodu:

```
var theDom = dreamweaver.getDocumentDOM("document")
var theLink = theDom.links[0]
var offsets = theDom.nodeToOffsets(theLink)
theDom.setSelection(offsets[0],offsets[1])
```

## Funkcja dreamweaver.getTokens()

Funkcja `getTokens()` jest często używana wewnątrz funkcji `inspectBehavior()`, ponieważ służy do przetwarzania łańcuchów tekstowych. Przez pojęcie *token* określana jest grupa znaków nie zawierająca żadnego ze zdefiniowanych wcześniej separatorów. *Separatory* w funkcjach to nawiasy otaczające ich argumenty i przecinki oddzielające argumenty od siebie.

Funkcja `getTokens()` wykorzystuje dwa argumenty: przetwarzany łańcuch i separatory, a zwracane wartości umieszcza w tablicy. Jako przykład wykorzystajmy następujący łańcuch tekstowy:

```
doGroovoid('false', 'Fanfare-Arrival')
```

Aby wydobyć argumenty z takiego wywołania funkcji, użyj funkcji `getTokens()` w następujący sposób:

```
getTokens("doGroovoid('false','Fanfare-Arrival')","'(),")
```

Jeśli przypiszesz wartość tej funkcji do tablicy o nazwie `argArray`, otrzymasz następujący rezultat:

```
argArray[0] = 'doGroovoid'
argArray[1] = 'false'
argArray[2] = 'Fanfare-Arrival'
```

Zwykle pierwszy element tablicy, czyli nazwa funkcji, jest pomijany.

### Funkcja dreamweaver.getElementRef()

Funkcja `getElementRef()` jest wykorzystywana do pobierania specyficznych dla określonej przeglądarki odwołań do obiektów i umieszczania ich w tablicy.

Funkcja `getElementRef()` wymaga dwóch argumentów. Pierwszy z nich przyjmuje wartość *NS 4.0* lub *IE 4.0*, co oznacza, odpowiednio: Netscape Navigator 4.0 i Internet Explorer 4.0, a drugi to badany znacznik. Zwracany łańcuch tekstowy określa nazwę określonego znacznika w formacie wykorzystywanym w wybranej przeglądarce. Np. jeśli użyjesz funkcji `getElementRef()` do pobrania odnośnika do określonej warstwy w przeglądarce Netscape:

```
var theObjNS = dreamweaver.getElementRef("NS 4.0", tagArr[i])
```

zwrócona wartość będzie miała postać:

```
document.layers['newLayer']
```

Ta sama warstwa w przeglądarce Internet Explorer:

```
var theObjNS = dreamweaver.getElementRef("IE 4.0", tagArr[i])
```

będzie określana przez wartość:

```
document.all.newLayer1
```

Funkcje `getElementRef()` i `getObjectRef()` zwracają wykorzystywane w obu przeglądarkach odnośniki do następujących znaczników: `<a>`, `<area>`, `<applet>`, `<embed>`, `<select>`, `<option>`, `<textarea>`, `<object>` i `<img>`. Odnośniki do znaczników `<div>`, `<span>` i `<input>` są zwracane prawidłowo dla Internet Explorera, a do znaczników `<layer>` i `<ilayer>` — dla Netscape Navigatora. Odnośniki do znaczników `<div>` i `<span>` z pozycjonowaniem bezwzględnym są zwracane prawidłowo dla Netscape Navigatora, ale inne zwracają tekst `"cannot reference <tag>"` (nie można odwołać się do znacznika).


Nadawanie obiektom i warstwom nazw jest niezwykle istotne, szczególnie w przypadku opisywanych wyżej funkcji. Dreamweaver nie zwróci odnośników do obiektów nie posiadających nazw, zwróci jedynie tekst `unnamed <tag>` (znacznik nie posiadający nazwy). Dreamweaver nie zwróci również odnośnika do obiektu posiadającego nazwę, ale znajdującego się w warstwie nie posiadającej nazwy. Dreamweaver automatycznie nadaje nazwy wstawianym do dokumentu warstwom, ale nazwy formularzy projektant musi wpisać w inspektorze *Property*.

### Funkcja dreamweaver.getBehaviorTag()

Funkcja `getBehaviorTag()` zwraca znacznik, do którego przyłączony jest behawior. Może ona również zostać dołączona do tej części kodu behawiora, która odpowiada za jego konfigurację, aby odpowiednio pokierować działaniami użytkownika.

Funkcja `getBehaviorTag()` zwraca cały znacznik wraz z atrybutami, wartościami i tekstem otaczanym przez niego. Z tego powodu należy wydobyć z wartości funkcji jedynie pewną część informacji. Jedną z technik umożliwiających wykonanie tego zadania jest wykorzystanie właściwości `indexOf`, określającej czy znacznik znajduje się w zwróconym łańcuchu tekstowym. Kod z poniższego przykładu sprawdza, czy zaznaczony znacznik jest znacznikiem `<img>` i — jeśli nie — informuje o tym użytkownika.

```
function initializeUI(){
var theTag = dreamweaver.getBehaviorTag().toUpperCase();
if (theTag.indexOf('IMG') != -1)){
// Inicjalizacja interfejsu użytkownika behawiora
} else{
alert("Ten behawior wymaga zaznaczenia obrazka.")
}
}
```


Ta technika różni się od zastosowania funkcji `canAcceptBehavior` w celu zablokowania dostępu do znacznika. Tu użytkownik zostanie poinformowany o błędnym postępowaniu.

### Funkcja dreamweaver.getBehaviorElement()

Innym sposobem sprawdzenia, jaki znacznik został zaznaczony dla behawiora, jest wykorzystanie funkcji `getBehaviorElement()`. Główną różnicą pomiędzy tą funkcją a opisywaną wcześniej funkcją `getBehaviorTag()` jest fakt, że pierwsza z nich zwraca odnośnik do znacznika wykorzystywany przez DOM, a druga — jedynie znacznik. Mając odnośnik do znacznika wykorzystywany przez DOM, możesz uzyskać sporo informacji na temat samego znacznika i jego atrybutów.

Podobnie jak `getBehaviorTag()`, funkcja `getBehaviorElement()` jest zwykle wykorzystywana do określenia, czy użytkownik zaznaczył właściwy znacznik. Jeśli zaznaczony jest niewłaściwy znacznik, można wyświetlić informację na ten temat. Funkcja `getBehaviorElement()` zwraca odnośnik do znacznika lub wartość `null`. Wartość `null` jest zwracana w następujących okolicznościach:

- funkcja nie została wywołana ze skryptu uruchamianego przez inspektor *Behavior*;
- behawior jest częścią animacji *timeline*;
- funkcja została wywołana ze skryptu uruchamianego przez funkcję `dreamweaver.popupAction()`;
- funkcja została wywołana przez inspektor *Behavior* dołączający zdarzenia do łącza (`<a href="#">...</a>`, które nie zostało jeszcze utworzone;
- funkcja została wywołana spoza behawiora.

W poniższym przykładzie założyłem, że behawior musi zostać dołączony do znacznika osadzającego na stronie procedurę plugin z wyświetlonym interfejsem:

```
function initializeUI(){
var theTag = dreamweaver.getBehaviorElement();
var tagGood = (theTag.tagName == "EMBED" && theTag.getAttribute("HIDDEN") ¬
== null);
if (tagGood) {
// Kod interfejsu użytkownika
} else{
alert("Tego behawiora nie można przyłączać do ukrytych procedur plug-in.")
}
}
```

### Funkcja dreamweaver.browseForFileURL()

Funkcja `browseForFileURL()` umożliwia użytkownikowi, zamiast wpisywania nazwy pliku i ścieżki dostępu w polu tekstowym, wskazanie pliku w oknie dialogowym. Możesz określić, czy ma zostać otwarte okno *Open*, *Save* czy *Select*, i podać treść etykiety okna wyświetlanej na jego pasku tytułowym. Możesz nawet włączyć w oknie panel *Preview*, w którym wyświetlane będą miniaturki plików graficznych. Niezależnie od opcji, funkcja `browseForFileURL()` zwraca ścieżkę dostępu i nazwę pliku w postaci względnego adresu URL.

Funkcja `browseForFileURL()` wykorzystuje następującą składnię:

```
browseForFileURL('Open'|'Save'|'Select', 'Etykieta okna', true|false)
```

Pierwszy argument, `Open`, `Save` lub `Select`, określa typ okna dialogowego. W oknie typu *Select* wyświetlana jest dodatkowo informacja o katalogu głównym witryny. Drugi argument określa treść etykiety okna dialogowego. Jeśli chcesz, aby pasek tytułowy okna był pusty, musisz zastosować dwa apostrofy:

```
browseForFileURL('open','',false)
```

Ostatni argument jest wartością logiczną i określa, czy w oknie ma zostać również wyświetlony panel *Preview* (taki, jaki wyświetlany jest w oknie *Select Image*). Jeśli nie określisz treści etykiety okna i nadasz ostatniemu argumentowi wartość *true*, na pasku tytułowym okna wyświetlona zostanie etykieta *Select Source Image*.

Funkcja `browseForFileURL()` jest zwykle umieszczana wewnątrz innej funkcji wywoływanej przez zdarzenie `onClick` przyłączone do przycisku *Browse* (*Choose*), który z kolei znajduje się obok pola tekstowego, w którym użytkownik może wpisać adres pliku.

Do Dreamweavera dołączana jest funkcja `browseFile()` (umieszczona w pliku *_common.js*) wymagająca jednego argumentu, `fieldToStoreURL` (pole, w którym ma zostać zapisany adres). Przykładowy kod przycisku *Browse* (*Choose*) może wyglądać następująco:

```
<input type="text" name="textFile">
<input value="Browse..." type="button" ¬
onClick="browseFile(document.theForm.textFile.value)" name="button">
```

Następnie funkcja `browseFile()` wywołuje funkcję `browseForFileURL()`, która otwiera okno dialogowe *Select File* i — jeśli okno dialogowe zwróci nazwę pliku — przypisuje ją zmiennej. W funkcji `browseFile()` przedstawionej poniżej zwrócona nazwa pliku jest przypisywana do wartości wyświetlanej w określonym polu tekstowym.

```
function browseFile(fieldToStoreURL){
  var fileName = "";
  fileName = browseForFileURL();  //returns a local filename
  if (fileName) fieldToStoreURL.value = fileName;
}
```

Funkcja `browseForFileURL()` nie zwraca adresów URL w postaci bezwzględnej.

### Funkcja dreamweaver.getDocumentPath()

Dreamweaver wyposażony jest w kilka funkcji wspomagających operacje odczytu, edycji i zapisu dokumentów. Jedną z nich jest `getDocumentPath()`. Zwraca ona ścieżkę dostępu do dokumentu. Ścieżka zwracana jest w formacie file://URL, więc jeśli wykorzystasz ją w odniesieniu do pliku c:\witryny\index.html, zwróci ona ścieżkę file:// C|witryny.

Funkcja `getDocumentPath()` wymaga określenia jednego argumentu — dokumentu. Można tu użyć następujących określeń dokumentu: `"document"`, `"parent"`, `"parent.frames[numer]"` lub `"parent.frames[nazwa_ramki]"`, podobnie jak w przypadku funkcji `getDocumentDOM()`. Jeśli dokument nie został zapisany, funkcja `getDocumentPath()` zwraca pusty łańcuch.

### Funkcja dreamweaver.getConfigurationPath()

Katalog *Configuration* można nazwać bazą rozbudowywania Dreamweavera. Znajdują się w nim nie tylko pliki HTML, takie jak behawiory i obiekty, ale również inne pliki kontrolujące wygląd i działanie menu. Z tego powodu przydatna może być możliwość zlokalizowania folderu *Configuration*, aby można było uzyskać dostęp do znajdujących się w nim plików. Do tego właśnie celu służy funkcja `getConfigurationPath()`.

Przykładowym zastosowaniem tej funkcji jest obiekt *Rollover*. Właściwie rollower nie jest obiektem — jest poleceniem. Funkcja `getConfigurationPath()` pełni kluczową rolę w kodzie tej komendy, zapisanym w pliku *rollover.js*:

```
var rolloverCmdURL = dreamweaver.getConfigurationPath() + ¬
"/Commands/Rollover.htm";
   var rolloverDoc    = dreamweaver.getDocumentDOM( rolloverCmdURL );
```

W pierwszej linii kodu funkcja `getConfigurationPath()` jest użyta do zlokalizowania pliku *Rollover.htm* i przypisania do zmiennej ścieżki dostępu do niego. Umożliwia to obiektowi pobranie DOM i manipulowanie nim za pomocą funkcji `getDocumentDOM()`.


Funkcja `getConfigurationPath`, podobnie jak `getDocumentPath()`, zwraca ścieżkę w postaci *file:///URL*.

### Funkcja dreamweaver.getSiteRoot()

Większość operacji związanych z zarządzaniem witryną w Dreamweaverze opiera się na zdefiniowaniu katalogu głównego witryny. Możliwość określenia jego położenia jest wyjątkowo ważna dla behawiorów lub innych rozszerzeń działających na poziomie katalogu głównego witryny.

Funkcja `getSiteRoot()` nie wymaga żadnego argumentu, a zwraca adres katalogu głównego witryny, w której znajduje się aktualnie przetwarzany plik, w postaci *file:// URL*. Jeśli zwrócony zostanie pusty łańcuch, oznacza to, że plik nie został zapisany.

### Funkcja dreamweaver.releaseDocument()

Pracując nad skomplikowanym dokumentem z wieloma elementami graficznymi, warstwami, tabelami i tekstem, obrabiasz duży plik HTML. Dostęp do DOM takiego dokumentu może spowodować, że zajęty zostanie duży obszar pamięci. Zakończenie pracy behawiora powoduje automatyczne zwolnienie obszaru pamięci zajmowanego przez wykorzystywane przez niego dane, ale jeśli behawior edytuje kilka dużych dokumentów jednocześnie, może zająć całą pamięć przed zakończeniem swojej pracy. Dzięki funkcji `releaseDocument()` możesz zwolnić obszar pamięci zajmowany przez DOM określonego dokumentu.

Jedynym argumentem funkcji `releaseDocument()` jest DOM dokumentu, uzyskiwany dzięki funkcji `getDocumentDOM()`. Działanie tej funkcji możesz zobaczyć w pliku *displayHelp.js* służącym do wyświetlania systemu pomocy kontekstowej.

### Funkcja dreamweaver.browseDocument()

Jeśli plik pomocy będzie zbyt duży, aby można było wyświetlić go w zwykłym oknie dialogowym, możesz wyświetlić go w oknie domyślnej przeglądarki internetowej. Funkcja `browseDocument()`umożliwia wyświetlenie dowolnego pliku w przeglądarce. Funkcja ta wymaga podania jednego argumentu — ścieżki dostępu do wyświetlanego pliku:

```
dreamaweaver.browseDocument("http://www.idest.com/help/etable.htm")
```

Jak wspomniałem w rozdziale 18., możesz wykorzystać funkcję `browseDocument()` do wyświetlenia pliku określonego bezwzględnym adresem URL lub pliku znajdującego się na dysku lokalnym. Jeśli chcesz wyświetlić plik znajdujący się na dysku lokalnym, musisz połączyć działanie funkcji `browseDocument()` z działaniem funkcji takiej jak `getConfigurationPath()`. W poniższym przykładzie funkcje te zostały zastosowane w celu wyświetlenia pliku *InsertMenu.htm*:

```
function displayMenu() {
  var menuPath = dreamweaver.getConfigurationPath() + ¬
"/Objects/InsertMenu.htm"
  dreamweaver.browseDocument(menuPath)
}
```

### Funkcje dreamweaver.openDocument() i dreamweaverCreateDocument()

Funkcje `openDocument()`i `createDocument()`mają podobne możliwości i podobne ograniczenia. Funkcja `openDocument()` jest równoznaczna poleceniu *File⇨Open* i wybraniu pliku w oknie dialogowym. Funkcja `createDocument()` umożliwia stworzenie pustego dokumentu na podstawie szablonu *Default.htm*. W obu przypadkach dokument jest umieszczany w oknie dokumentu, które jest następnie wyświetlane nad innymi oknami.

Funkcja `createDocument()` nie wymaga żadnego argumentu, a zwraca DOM nowego dokumentu. Poniższa definicja:

```
var theNewDoc = dreamweaver.createDocument()
```

jest równoznaczna z zastosowaniem funkcji `getDocumentDOM()` wobec pustego dokumentu.

Funkcja `openDocument()` wymaga argumentu w postaci *file://URL*. Jeśli URL jest podawany w postaci względnej, to jest przyjmowany względem pliku, z którego nastąpiło wywołanie funkcji. Np. jeśli chcesz otworzyć plik znajdujący się w katalogu nadrzędnym względem katalogu *Commands*, musiałbyś odwołać się do niego w sposób następujący:

```
dreamweaver.openDocument("../Extensions.txt")
```

Za pomocą techniki omówionej w opisie funkcji `browseDocument()` możesz odwoływać się do plików, adresując je względem katalogu *Configuration*.


Funkcje `openDocument()` i `createDocument()`nie mogą być wykorzystywane w behawiorach, ale mogą zostać wywołane z polecenia lub inspektora Property. Dlatego możliwe jest wykorzystanie funkcji `popupCommand()` do uruchomienia polecenia wykorzystującego funkcję `openDocument()`lub `createDocument()`.

### Funkcja dreamweaver.saveDocument()

Po zakończeniu modyfikacji pliku trzeba go zapisać. Do tego celu służy funkcja `saveDocument()`. Funkcja ta wymaga podania dwóch argumentów: *documentObject* i *fileURL*. Pierwszy z nich określa DOM zapisywanego dokumentu, a drugi adres, pod którym plik ma zostać zapisany. W tym przypadku również adres URL podawany jest względem pliku, z którego nastąpiło wywołanie funkcji.

Funkcja `saveDocument()`zwraca wartość *true*, jeśli zapisywanie pliku zakończyło się pomyślnie, lub wartość *false*, jeśli operacja nie została dokończona. Jeśli plik posiada znacznik „tylko do odczytu”, Dreamweaver próbuje odblokować go. Jeśli proces ten się nie powiedzie, wyświetlony zostanie komunikat o błędzie.

### Funkcja dreamweaver.editLockedRegions()

Szablony Dreamweavera to połączenie zablokowanych i edytowalnych obszarów (*regions*). Zwykle obszary te określa się w oknie dokumentu, ale możesz wykorzystać funkcję `editLockedRegions()` do programowego blokowania i odblokowywania obszarów. Funkcja `editLockedRegions()` wymaga argumentu *true*, przy odblokowywaniu wszystkich zablokowanych obszarów, lub argumentu *false* — przy ponownym ich blokowaniu. Po zakończeniu działania funkcji, z której nastąpiło wywołanie funkcji `editLockedRegions()`, status wszystkich obszarów zostaje przywrócony do wartości domyślnych.


Z racji możliwości wystąpienia nieprzewidzianych efektów działania, firma Macromedia zaleca, aby funkcja `editLockedRegions()` była wykorzystywana jedynie w filtrach (*translators*).

### Funkcje dreamweaver.popupAction() i dreamweaver.runCommand()

Funkcje `popupAction()` i `runCommand()` nie mogą być wykorzystywane w behawiorach, ale umożliwiają wzajemną współpracę innych rozszerzeń Dreamweavera. Za pomocą tych funkcji można wywołać istniejący behawior lub polecenie i wyświetlić jego okno dialogowe. Wyjątkiem jest sytuacja, gdy funkcje te są wykorzystywane do wywołania behawiora lub polecenia z poziomu obiektu, innego polecenia lub inspektora *Property*.

Funkcja `popupAction()` wymaga dwóch argumentów: nazwy pliku akcji i funkcji wywołującej akcję. Plik akcji musi znajdować się w katalogu *Actions*. Przykładowy kod wywołujący behawior *Sound Control* może wyglądać następująco:

```
var goCS = dreamweaver.popupAction("Control Sound.htm","MM_controlSound(..)")
```


Jeśli chcesz wywołać akcję znajdującą się w podkatalogu katalogu *Actions*, musisz określić ścieżkę dostępu do niej. Np. jeśli chcesz wywołać jedną z akcji z grupy *Timeline*, musisz wyrazić jej nazwę w postaci „Timeline/Go to Frame.htm".

Nazwa funkcji wywołującej behawior znajduje się przy końcu funkcji `applyBehavior()`, w miejscu, gdzie określana jest wartość zwracana, lub w wartości zwracanej przez funkcję `behaviorFunction()`. Funkcja `popupAction()` zwraca pełną postać wywołania funkcji, wraz z określonymi przez użytkownika parametrami. Kontynuując poprzedni przykład, jeśli użytkownik wybrał *Play* i wskazał plik *brazil.mid*, zmienna *goCS* będzie miała następującą wartość:

```
"MM_controlSound('play',document.CS911946210190,'brazil.mid')"
```


Drugi argument to nazwa generowana przez program Dreamweaver.

Wszystkie te dane, z wyjątkiem procedury obsługi zdarzenia i odpowiadającego jej wywołania funkcji, są zapisywane do strony. Za obsługę zdarzenia i wywołanie funkcji odpowiada obiekt, polecenie lub inspektor *Property*.

Funkcja `runCommand()` jest nieco prostsza. Wymaga tylko jednego argumentu, nazwy pliku polecenia. Plik ten musi znajdować się w katalogu *Commands*. Funkcja `runCommand()` nie zwraca żadnej wartości. Uruchamia jedynie podane polecenie.

### Funkcje dreamweaver.latin1ToNative() i dreamweaver.nativeToLatin1()

Dreamweaver posiada dwie funkcje ułatwiające dostosowanie behawiorów do pracy w różnych wersjach językowych systemów operacyjnych. W wielu krajach wykorzystywane jest inne kodowanie znaków niż standardowo wykorzystywany w Stanach Zjednoczonych i większości krajów Europy Zachodniej Latin1.

Aby przekonwertować łańcuch tekstowy z kodowania Latin1 na kodowanie wykorzystywane w systemie operacyjnym zainstalowanym na komputerze użytkownika, wykorzystaj funkcję `latin1ToNative()`. Argument, czyli łańcuch tekstowy, powinien zostać wcześniej przetłumaczony na odpowiedni język. Aby dokonać odwrotnej konwersji, zastosuj funkcję `nativeToLatin1()`.


Żadna z tych funkcji nie zadziała poprawnie w systemie Windows bazującym na kodowaniu Latin1.

### Funkcja dreamweaver.relativeToAbsoluteURL()

Ponieważ coraz więcej programów umożliwia tworzenie plików HTML (przykładami mogą być Fireworks i Director 7), behawiory i inne rozszerzenia mogą zostać wykorzystane do obsługi tak stworzonych dokumentów. Często niezbędne jest określenie bezwzględnego adresu URL takiego pliku w celu pobrania jego DOM-u lub otwarcia go. Funkcja `relativeToAbsoluteURL()` zwraca taką informację na podstawie trzech argumentów:

- *docBaseURL*. Fragment ścieżki dostępu do dokumentu z wyłączeniem nazwy pliku. Na przykład, jeśli analizowanym plikiem byłby plik *images/austria.gif*, argument *docBaseURL* przyjąłby wartość *images/*.
- *siteRootURL*. Adres URL katalogu głównego witryny, zwrócony przez funkcję `getSiteRoot()`.
- *relativeURL*. Pełny względny adres URL badanego pliku, na przykład *images/austria.gif*.

Składnia funkcji wygląda następująco:

```
var absoluteURL = dreamweaver.relativeToAbsoluteURL(docBaseURL,siteRootURL,relativeURL)
```

Określenie wartości argumentu *docBaseURL* może być nieco kłopotliwe. Po określeniu wartości argumentu *relativeURL* za pomocą funkcji `browseForFileURL()`, musisz wydobyć z niej fragment znajdujący się przed nazwą pliku. Aby to zrealizować, użyj funkcji `lastIndexOf` do znalezienia znaku „/” i wydobycia znajdującego się przed nim łańcucha. Oto przykład:

```
function docBase() {
var docURL = dreamweaver.getDocumentPath("document");
```

```
var index = docURL.lastIndexOf('/');
if ( -1 == index ){ // jeśli nie ma określenia katalogu
  return "";        // zwróć pusty łańcuch
  }
  else {
  return docURL.substring(0, index);
  }
}
```

## API dla behawiorów

Większość funkcji znajdujących się w API dla behawiorów poznałeś już, czytając podrozdział „Stwórz plik akcji". Ta część API jest wykorzystywana podczas pisania behawiorów. Umieszczone w tej części funkcje zostały wyszczególnione w poniższej tabeli.

| Funkcja | Zadanie funkcji |
|---|---|
| `canAcceptBehavior()` | Określa, czy akcja jest dostępna dla zaznaczonego znacznika |
| `windowDimensions()` | Definiuje rozmiary okna dialogowego behawiora |
| `applyBehavior()` | Dołącza funkcję do zaznaczonego znacznika |
| `inspectBehavior()` | Przywraca wpisane poprzednio wartości do pól okna dialogowego |
| `behaviorFunction()` | Zapisuje funkcję do sekcji `<head>` dokumentu HTML |
| `deleteBehavior()` | Usuwa behawior z pliku HTML |
| `identifyBehaviorArguments()` | Zapisuje te argumenty behawiora, które powinny zostać zmodyfikowane, gdy plik jest umieszczany w innym obszarze dysku |
| `displayHelp()` | Umieszcza w oknie dialogowym behawiora przycisk *Help* |

Funkcje `canAcceptBehavior()`, `applyBehavior()`, `inspectBehavior()` i `behaviorFunction()` zostały omówione w poprzednich podrozdziałach. Poniżej znajdują się opisy pozostałych funkcji.

### Funkcja windowDimension()

Za pomocą funkcji `windowDimension()` można określić rozmiary okna dialogowego behawiora, co może przyspieszyć jego wyświetlanie. Jeśli funkcja ta nie znajduje się w kodzie behawiora, rozmiary okna są wyznaczane automatycznie. Funkcja wymaga określenia jednego argumentu, *platform*, określającego system operacyjny Windows lub MacOS. Zwracaną wartością jest łańcuch tekstowy zawierający szerokość i wysokość okna dialogowego wyrażone w pikselach. Oto przykład:

```
function windowDimensions(platform){
if (platform.charAt(0) == 'm'){ // Macintosh
       return "650,500";
       }
       else { // Windows 95 lub NT
       return "675,525";
}
}
```

Funkcja ta została zastosowana w kilku behawiorach dołączanych do Dreamweavera. Firma Macromedia zaleca, aby używać ją jedynie w sytuacji, gdy okno dialogowe ma być większe niż 640×480.

### Funkcja deleteBehavior()

Zwykle Dreamweaver automatycznie usuwa z kodu strony obsługę zdarzenia i funkcje JavaScript w momencie, gdy użytkownik kliknie przycisk *Remove Behavior* w inspektorze *Behavior*. Jednak gdy behawior jest skomplikowany i dodaje do kodu źródłowego strony dodatkowe elementy, niezbędne jest wykorzystanie funkcji `deleteBehavior()`. Najłatwiej będzie zrozumieć działanie tej funkcji, analizując przykłady.

Funkcja `deleteBehavior()` jest wykorzystywana w dwóch standardowych behawiorach: *Control Sound* i *Swap Image*. Behawior *Control Sound* wstawia do kodu strony znacznik `<embed>` z atrybutem *ID*. Aby usunąć ten zapis z dokumentu HTML, funkcja `deleteBehavior()` odczytuje łańcuch wywołania funkcji (taki, jak zwracany jest przez funkcję `applyBehavior()`). Jeśli funkcja znajdzie znacznik `<embed>` z określoną wartością atrybutu *ID*, do którego nie odwołuje się żaden inny element strony, usuwa go z kodu strony. W kodzie źródłowym behawiora *Control Sound* wygląda to następująco:

```
function deleteBehavior(fnCallStr) {
  var argArray,sndName,doc,tagArray,i,embedName;

  argArray = extractArgs(fnCallStr);
  if (argArray.length > 2) {
    sndName = dreamweaver.getTokens(argArray[2],".")[1]; //remove ¬
    "document.", use unique name
    //Find all EMBED calls that we created (name starts with "CS"), add ¬
      to menu
    doc = dreamweaver.getDocumentDOM("document"); //get all
    tagArray = doc.getElementsByTagName("EMBED");
    for (i=0; i<tagArray.length; i++) {  //with each EMBED tag
      embedName = tagArray[i].name;
      if (embedName == sndName) { //if same embed
        if ( -1 == doc.body.outerHTML.indexOf( argArray[2] ) ) // and embed ¬
        ref'd no where else
           tagArray[i].outerHTML = "";
        break;
  } } }
}
```

Behawior *Swap Image* nie dodaje do kodu strony dodatkowych znaczników, ale wstawia dodatkowe zdarzenia umożliwiające działanie rollowera. Kiedy behawior *Swap Image* jest usuwany ze strony, usunięte muszą zostać również te zdarzenia. Aby to zrealizować, funkcja `deleteBehavior()` przeszukuje łańcuch tekstowy wywołania funkcji behawiora w poszukiwaniu argumentu *PreloadID*. *PreloadID* to nazwa nadawana przez program w momencie, gdy użytkownik zaznaczy w oknie dialogowym behawiora opcję *Preload*. Jeśli argument ten zostanie znaleziony, z kodu usuwane jest wywołanie funkcji realizującej ładowanie obrazków do pamięci podręcznej przeglądarki — `onLoad=MM_preloadImages()`. Następnie funkcja `deleteBehavior()` sprawdza, czy włączona została opcja *Restore*, a jeśli tak, usuwa jej kod oraz wywołujące ją zdarzenie.

### Funkcja identifyBehaviorArguments()

Jeśli kiedykolwiek przenosiłeś witrynę z jednego katalogu do innego, to wiesz na pewno, ile pracy wymaga sprawdzenie poprawności łączy po takiej operacji. Dreamweaver ułatwia to zadanie. Jeśli zapiszesz plik za pomocą polecenia *Save As*, wszystkie ścieżki w atrybutach znaczników HTML są automatycznie aktualizowane. Dreamweaver umożliwia wykorzystanie podobnych możliwości w adresach URL znajdujących się w kodzie źródłowym behawiora.

Wyobraź sobie, że stworzyłeś stronę, do której dodałeś behawior *Check Browser*, aby skierować użytkowników różnych przeglądarek na odpowiednie strony. Jeśli zapisałbyś tę stronę w innym katalogu, Dreamweaver automatycznie uaktualniłby adresy stron.

Aby mogło to nastąpić, musisz dodać do kodu behawiora funkcję `identifyBehaviorArguments()`. Funkcja ta przekazuje argumenty behawiora do Dreamweavera, który dzięki temu może uaktualnić występujące w nich adresy URL. Funkcja ta umożliwia również znalezienie wykorzystywanych przez behawior warstw, które Dreamweaver musi uwzględnić przy realizowaniu polecenia *Convert Layers to Tables*.

Argumentem funkcji `identifyBehaviorArguments()` jest łańcuch tekstowy zawierający wywołanie funkcji behawiora wraz z argumentami. Argumenty te są wydobywane z łańcucha i umieszczane w tablicy. Funkcja rozpoznaje, które z nich są adresami URL, które określeniami warstw, a które innymi elementami. Zwracane są cztery wartości:

- *URL* — jeśli argument jest określeniem pliku lub ścieżki dostępu do pliku;
- *NS4.0ref* — jeśli argument określa warstwę w składni używanej w przeglądarce Netscape — `document.layers[\'Layer1'\]`;
- *IE4.0ref* — jeśli argument określa warstwę w składni używanej w przeglądarce Internet Explorer — `document.all[\'layer1\']`;
- *Other* — jeśli argument nie jest żadnym z powyższych.

Funkcja `identifyBehaviorArguments()` wykorzystywana jest w behawiorze *Check Plugin*:

```
function identifyBehaviorArguments(fnCallStr) {
  var argArray;

  argArray = extractArgs(fnCallStr);
  if (argArray.length == 5) {
    return "other,URL,URL,other";
  }
}
```

Podobnie jak w przypadku funkcji `inspectBehavior()`, liczba elementów tablicy jest o jeden większa od liczby argumentów. Pierwszym elementem tablicy jest nazwa funkcji.

### Funkcja displayHelp()

Funkcja `displayHelp()` powoduje umieszczenie w oknie dialogowym behawiora przycisku *Help*. Przycisk ten jest umieszczany poniżej przycisków *OK* i *Cancel*. Funkcja ta nie wymaga żadnych argumentów i jest zwykle wykorzystywana do wyświetlania pliku pomocy. Najczęściej używa się jej w połączeniu z metodą `alert()` i funkcją `browseDocument()`.

Aby wyświetlić krótki komunikat, zastosuj metodę `alert()`:

```
function displayHelp() {
  alert("Ten behawior działa tylko z plikami .rmf.")
}
```

Jeśli chcesz wyświetlić dłuższy tekst, użyj funkcji `browseDocument()`:

```
function displayHelp() {
  dreamweaver.browseDocument("http://www.idest.com/help/rep.htm")
}
```

Za pomocą funkcji `browseDocument()` możesz również wyświetlać pliki znajdujące się na dysku lokalnym. Opis funkcji `browseDocument()` znajdziesz we wcześniejszej części tego rozdziału.


Nie wywołuj z behawiorów funkcji znajdującej się w pliku *displayHelp.js*. Ta funkcja jest wykorzystywana przez Dreamweavera do wyświetlania systemu pomocy.

# Inne użyteczne funkcje

W językach zorientowanych obiektowo funkcję tworzy się raz, a później używa się jej w razie potrzeby. Dreamweaver posiada dużą bibliotekę gotowych i bardzo użytecznych funkcji, znajdującą się w katalogu *Shared/Macromedia/Scripts/CMN*. Funkcje te są pogrupowane w kategorie i zapisane w plikach JavaScript. W katalogu znajduje się 13 plików, w tym *docInfo.js*, *DOM.js*, *file.js* i *string.js*. Są one wykorzystywane w behawiorach dołączonych do Dreamweavera, ale możesz również zastosować je w swoich. Aby móc ich używać, musisz wstawić do kodu JavaScript behawiora dodatkową linię:

```
<script src="../../Shared/MM/Scripts/CMN/string.js></script>
```

W tabeli 20.3 znajdziesz krótkie opisy niektórych funkcji znajdujących się w folderze *Shared* wraz z nazwą pliku, w którym możesz je znaleźć.

# Techniki tworzenia behawiorów

Aby utworzyć behawior, zwykle nie wystarczy połączyć kilku gotowych funkcji. Istnieje wiele technik stosowanych do określonych celów. Jeśli ich nie poznasz, spędzisz mnóstwo czasu, ponownie wymyślając koło. Czytając ten podrozdział, poznasz kilka technik, dzięki którym będziesz mógł szybciej i sprawniej tworzyć behawiory.

## Określanie zdarzenia

W Dreamweaverze każdy znacznik, do którego można dołączyć behawior, posiada domyślnie określone zdarzenie. Możesz zmienić domyślne zdarzenia dla znaczników, modyfikując pliki znajdujące się w katalogu *Events*, ale zmiana ta będzie obowiązywała jedynie w Twoim systemie, a nie w systemach innych użytkowników. Możesz jednak określić zdarzenie, a właściwie grupę zdarzeń, z poziomu behawiora.

**Tabela 20.3.** *Pozostałe funkcje*

| Funkcja | Plik | Opis |
|---|---|---|
| `extractArgs()` | *string.js* | Wydobywa z wywołania funkcji argumenty i umieszcza je w tablicy |
| `escQuotes()` | *string.js* | Przegląda łańcuch i wstawia znak „/” przed każdym apostrofem, cudzysłowem i ukośnikiem |
| `unescQuotes()` | *string.js* | Usuwa znaki „/” z łańcucha |
| `browseFile()` | *file.js* | Otwiera okno dialogowe *Select File* i umieszcza ścieżkę dostępu do wskazanego pliku w określonym polu tekstowym |
| `stripStar()` | *menuItem.js* | Usuwa znaki „*” z końca łańcucha |
| `stripValue()` | *menuItem.js* | Usuwa określone wartości z końca łańcucha |
| `addStarToMenuItem()` | *menuItem.js* | Dodaje znak „*” do zaznaczonego elementu menu |
| `addValueToMenuItem()` | *menuItem.js* | Dodaje określoną wartość do zaznaczonego elementu menu |
| `niceNames()` | *niceName.js* | Zmienia odwołania do obiektów wykorzystywane przez JavaScript, na przykład `document.layers['onLayer'].document.theForm` na bardziej czytelne: `form "theForm" in layer "onLayer"` |
| `nameReduce()` | *niceName.js* | Wydobywa nazwy obiektów i numery elementów tablicy |
| `errMsg()` | *errmsg.js* | Łączy łańcuchy podane jako argumenty. Przykładowo `errMsg("Teraz nadszedł %s na %s,var1,var2")` zwróci tekst *Teraz nadszedł czas na odpoczynek*, jeśli zmiennej *var1* przypisano wartość *czas*, a zmiennej *var2* — wartość *odpoczynek* |
| `findObject()` | *UI.js* | Zwraca odwołanie do wskazanego obiektu w formacie wykorzystywanym przez JavaScript. Np. jeśli w warstwie *onLayer* znajduje się obrazek *imgOne*, funkcja `findObject("imgOne")` zwróci `document.layers['onLayer'].imgOne` |
| `getParam()` | *string.js* | Zwraca tablicę zawierającą nazwy obiektów znajdujących się wewnątrz określonego znacznika |
| `badChars()` | *string.js* | Usuwa z łańcucha nieprawidłowe znaki: ~!@#$%^&*()_+\|`-=\\{}[]:\";'<>,./ oraz spacje |
| `getAllObjectRefs()` | *docInfo.js* | Zwraca tablicę składającą się z odnośników do obiektów dla określonego znacznika, znajdujących się wewnątrz aktualnego dokumentu lub we wszystkich ramkach |
| `getAllObjectTags()` | *docInfo.js* | Zwraca tablicę zawierającą znaczniki dla określonego obiektu, znajdujące się wewnątrz aktualnego dokumentu lub we wszystkich ramkach |

Określenie zdarzenia znajduje się w funkcji `canAcceptBehavior()`. Zwykle funkcja ta zwraca wartość *true* lub *false* w zależności od tego, czy spełnione są warunki niezbędne do uruchomienia behawiora. Jeśli warunki są spełnione, a dodatkowo chcesz określić, jakie zdarzenie ma zostać zastosowane, możesz tak zdefiniować funkcję `canAcceptBehavior()`, by zwróciła ona łańcuch składający się z nazw zdarzeń.

Kod z poniższego przykładu analizuje stronę i — jeśli znajdzie na niej warstwę, zmienia domyślne zdarzenie na onKeyDown.

```
function canAcceptBehavior(){
  var nameArray = dreamweaver.getObjectRefs("NS 4.0","document","LAYER");
  if (nameArray.length > 0){
    return "onKeyDown";
  }else{
  return FALSE;
}
```

Możliwe jest również określenie grupy zdarzeń. Ważna jest tu kolejność — od najmniej pożądanego do najważniejszego:

```
return "onKeyDown,onKeyPress,onKeyUp"
```

Jeśli jedno ze zdarzeń jest niedostępne (na przykład w danej wersji przeglądarki), wybierane jest następne zdarzenie z listy.

## Zwracanie wartości

Większość zdarzeń nie wymaga, aby zwrócona została do nich wartość, ale niektóre z nich, takie jak onMouseOver i onMouseOut, tego wymagają. Zwykle w behawiorach nie jest to uwzględniane, ale jest możliwe. Możesz zrealizować to, deklarując zmienną document.MM_returnValue. Działanie tej zmiennej widoczne jest w behawiorze *Set Text of Status Bar*.

Zmienna document.MM_returnVariable jest deklarowana w ostatniej linii definicji funkcji. Kod funkcji wygląda następująco:

```
function MM_displayStatusMsg(msgStr) { //v2.0
  status=msgStr;
  document.MM_returnValue = true;
}
```

Oczywiście, zwracana wartość może być również równa *false*.

## Funkcje zwracające wywołania innych funkcji

Możliwość zwracania przez funkcje wywołań innych funkcji wprowadzono już w Dreamweaverze 1.2. Wcześniej funkcje w behawiorach nie mogły wywoływać żadnych funkcji pomocniczych. Teraz można zrealizować to za pomocą funkcji behaviorFunctions(). Wszystkie zwracane funkcje są zapisywane w dokumencie wewnątrz znacznika <script>...</script>.

Wywoływanie jednej funkcji przez drugą jest łatwe do zrealizowania. Najpierw wymień funkcje w liście w funkcji behaviorFunction(), oddzielając je przecinkami. Pamiętaj, aby funkcja, która ma być wywoływana przez zdarzenie, była ostatnią pozycją na liście. Widać to w poniższym fragmencie kodu, zaczerpniętym z napisanego przeze mnie behawiora *Resize Layer Patch*:

```
function behaviorFunction(){
  return 'reDo,resizePatch';
}
```

Tu główną funkcją jest funkcja `resizePatch()`, wykorzystana w funkcji `applyBehavior()`:

```
function applyBehavior() {
  return 'resizePatch()';  //zwróć wywołanie funkcji z argumentami
}
```

# Podsumowanie

Napisanie behawiora nie jest łatwym zadaniem, ale daje dużą satysfakcję, zarówno twórcy, jak i użytkownikowi. Dreamweaver umożliwia zautomatyzowanie tworzenia zaawansowanych elementów strony dzięki dostępowi do obiektowego modelu dokumentu. Poniżej zamieściłem kilka uwag podsumowujących wiadomości zawarte w niniejszym rozdziale.

- Jeśli możesz wykonać coś za pomocą JavaScriptu, możesz stworzyć behawior automatyzujący tę czynność.
- Dreamweaver posiada rozbudowany DOM (*Document Object Model* — obiektowy model dokumentu), który umożliwia programiście dostęp do niemal każdego elementu strony.
- Tworząc akcje, możesz wykorzystać wbudowane w Dreamweavera rozszerzenia JavaScript i funkcje z API.
- Rozszerzenia JavaScript wbudowane w Dreamweavera umożliwiają otwieranie istniejących dokumentów oraz tworzenie nowych.
- Wiele użytecznych funkcji można znaleźć w katalogu *Configuration/Shared/CMN*.

W następnym rozdziale nauczysz się dostosowywać Dreamweavera do swoich potrzeb za pomocą dodatkowych poleceń, inspektorów *Property* i innych elementów.