

## XUL – część II

# Zabawa z klockami

W pierwszej części naszego wstępu do programowania XUL omówiliśmy kilka rodzajów elementów GUI i możliwości oferowane przez XUL, które ułatwiają tworzenie interfejsów graficznych. Mimo że stworzony przez nas interfejs wyglądał przyzwoicie, nie miał żadnego praktycznego zastosowania.

JONO BACON



**W** tym miesiącu poznamy sposoby budowania elementów interfejsu, nadając im jednocześnie prawdziwą funkcjonalność, która pozwoli stworzyć moduły do budowy nowych aplikacji.

## Moduły interakcyjne

Każda forma narzędzia graficznego (w tym także XUL) jest beзуżyteczna bez środków umożliwiających interakcję z użytkownikiem. Kiedy klikamy przycisk, przesuwamy pasek przewijania, wybieramy zakładkę czy pozycję z menu lub klikamy na przycisk znajdujący się na pasku narzędzi, oczekujemy konkretnej reakcji ze strony programu. Przykładowo, jeżeli z menu Plik wybierzemy polecenie Zakończ, możemy spodziewać się zakończenia pracy danego programu. W interakcji tego rodzaju występują dwa procesy: obsługa zdarzeń i funkcja.

Niektórzy Czytelnicy z pewnością spotkali już tego rodzaju funkcjonalność w innych narzędziach graficznych, np. Qt korzysta z systemu Sygnał/Slot. Generalnie chodzi o to, że każdy rodzaj sterowania graficznego (czyli przyciski, paski przewijania, menu, paski narzędzi, pola tekstowe) oferuje inny sposób interakcji z użytkownikiem. Przykładowo, przycisk klikamy wskaźnikiem myszy, z listy

w ramce wybieramy odpowiednią pozycję, a przy pomocy paska przewijania przesuwamy zawartość okna. Każdy z podanych tutaj przykładów interakcji nazywamy zdarzeniem. Niektóre elementy okna (czyli interaktywne elementy graficzne) oferują czasem wiele rodzajów zdarzeń dla różnych sposobów interakcji z użytkownikiem.

To wszystko będzie beзуżyteczne, jeżeli nie będziemy mogli zareagować odpowiednio na dane zdarzenie i właśnie dlatego potrzebujemy funkcji. Funkcja to po prostu fragment kodu, który wykonuje konkretne zadanie. Może to być np. zmiana tekstu interfejsu lub dołączenie informacji do jednego z elementów okna, przetworzenie informacji lub cokolwiek innego, co jest użyteczne. Dużą zaletą wszystkich funkcji jest to, że mogą one być swobodnie wykorzystywane przez różne zdarzenia. Przykładowo, jeżeli utworzymy funkcję generującą nowy dokument, z pewnością chcielibyśmy ją podłączyć do pozycji menu Plik->Nowy i jednocześnie do przycisku paska narzędzi, który będzie otwierał nowy dokument. W ten sposób wszystko zostaje połączone w jedną całość.

## Panowanie nad kodem

Po tej dawce teorii można się zastanawiać, w jaki sposób to wszystko działa w praktyce.

W jaki sposób możemy stworzyć obsługę zdarzeń, jak tworzymy funkcję i jak łączymy wszystko w całość? Odpowiedzią na te pytania jest Javascript.

Zanim przejdziemy dalej, kilka słów o Javascript. Wiele osób uśmiecha się szyderczo na samą myśl o języku Javascript twierdząc, że jest przydatny tylko do tworzenia irytujących komunikatów pojawiających się na pasku stanu lub menu albo wyskakujących okienek, z powodu których strona internetowa nie działa prawidłowo we wszystkich przeglądarkach. Trzeba przyznać, że jest to w dużej mierze prawda, jednak Javascript to niewielki, doskonale opracowany i przydatny język programowania.

W przypadku XUL korzystamy ze specjalnej cechy, zwanej obiektowym modelem dokumentu (ang. Document Object Model [DOM]). Celem tego modelu jest udostępnienie w kodzie wszystkich elementów na stronie internetowej, tak aby każdy użytkownik mógł przeglądać i zmieniać ich zawartość. Przykładowo, można w ten sposób zmieniać nagłówki strony, datę czy edytować etykiety przycisków. Typowym przykładem funkcjonalności tego rodzaju jest dowolne oprogramowanie umożliwiające stworzenie forum internetowego. Jeżeli podczas

pisania wiadomości skorzystamy z przycisków formatujących tekst, do treści naszej wiadomości dołączone zostaną znaczniki formatujące go w wybrany przez nas sposób. Skąd dany przycisk wie, jak komunikować się z oknem tekstowym wiadomości po jego naciśnięciu? Wykorzystuje obiektowy model dokumentu (DOM).

Zasada działania DOM polega na tworzeniu drzewa elementów znajdujących się na danej stronie internetowej. Spójrzmy na poniższy przykład:

```
<html>
<head>
  <title>Moja strona</title>
</head>
<body>
  <div>
    <h1>Witajcie na mojej stronie</h1>
  </div>
  <div>
    Zawartość strony.
  </div>
</body>
</html>
```

Każdy ze znaczników tego kodu zagnieżdża się wewnątrz kolejnego z nich, poza znacznikiem <html>, który jest znacznikiem nadrzędnym. Znacznik nadrzędny jest

traktowany jak korzeń drzewa, a wszystkie inne znaczniki – jak gałęzie wychodzące z jednego pnia. Na przykład, jeżeli spojrzymy na znacznik <body>, wewnątrz niego zobaczymy dwa zagnieżdżone znaczniki <div>. Są one jak dwie gałęzie drzewa. Poniżej możemy dokładnie zobaczyć formowanie drzewa wewnątrz znacznika nadrzędnego <html>:

```
<html>
  <head>
    <title>Moja strona</title>
  </head>
  <body>
    <div>
      <h1>Witajcie na mojej stronie</h1>
    </div>
    <div>
      Zawartość strony.
    </div>
  </body>
</html>
```

Dzięki znacznikom ułożonym w formie drzewa możemy wykorzystywać model DOM do wyszukiwania poszczególnych znaczników, wyświetlania ich lub pobierania z nich dodatkowych danych. Poza wyszukiwaniem właściwych znaczników, model DOM potrafi także je aktualizować.

## DOM i XUL

Zacznijmy od prostego przykładu, w którym stworzymy przycisk w systemie zdarzenie/funkcja. Przede wszystkim stworzymy no-

### Listing 1: Plik pierwszy.xul

```
<?xml version='1.0'?>
<?xml-stylesheet href='chrome://global/skin/' type='text/css'?>

<window
  id='test-window'
  title='Test Program'
  xmlns='http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul'>

  <button id='name' label='My button' />

</window>
```

wy plik o nazwie pierwszy.xul i wpisujemy kod programu z Listingu 1.

Kiedy odczytamy ten plik w przeglądarce Mozilla, zobaczymy zwykły przycisk. Pierwszym krokiem do stworzenia obsługi zdarzeń dla tego przycisku jest określenie typu interakcji, na którą ma on reagować. Każde zdarzenie wywoływane jest specjalnym programem obsługi, określającym typ interakcji, który nas interesuje. W tabeli zdarzeń XUL umieściliśmy

## Zdarzenia XUL

Zdarzenie	Opis
onclick	Zdarzenie wywołane po naciśnięciu i zwolnieniu przycisku myszy na elemencie okna. Zdarzenia tego należy używać tylko wtedy, gdy chcemy, aby przycisk reagował wyłącznie na naciśnięcie myszy. W przypadku przycisków, pozycji menu i podobnych elementów należy użyć zdarzenia oncommand, które bierze pod uwagę także użytkowników korzystających z klawiatury i innych urządzeń wskazujących.
onmousedown	Zdarzenie wywołane po naciśnięciu przycisku myszy na elemencie okna. Program obsługi zdarzeń zostanie wywołany w chwili naciśnięcia przycisku myszy, bez względu na to, czy został on zwolniony, czy nie.
onmouseup	Zdarzenie wywołane w chwili zwolnienia przycisku myszy na elemencie okna.
onmouseover	Zdarzenie wywołane w chwili przesunięcia wskaźnika myszy na element okna. Można je wykorzystywać do wyróżnienia elementu, jednakże kaskadowe arkusze stylów (CSS) umożliwiają automatyczne wyróżnienie przycisku, więc nie zalecamy korzystania z tego zdarzenia. Wyjątkiem jest sytuacja, kiedy chcemy wyświetlić tekst pomocy na pasku stanu.
onmousemove	Zdarzenie wywołane w chwili poruszenia wskaźnika myszy znajdującego się nad elementem okna. Wywołane wielokrotnie, przy każdym ruchu wskaźnika, więc nie polecamy korzystania z tego zdarzenia.
onmouseout	Zdarzenie wywołane w chwili przesunięcia wskaźnika myszy poza element okna. Można w ten sposób wygasić wyróżnienie lub usunąć tekst z paska stanu.
oncommand	Zdarzenie wywołane w chwili wybrania przycisku lub pozycji menu. W przypadku menu wystarczy dodać ten program obsługi do pozycji menu. Zalecamy korzystanie z tego zdarzenia, gdyż użytkownik może do naciśnięcia przycisku użyć zarówno myszy, jak i skrótu klawiaturowego, czy też innego urządzenia wskazującego.
onkeypress	Zdarzenie wywołane w chwili naciśnięcia i zwolnienia klawisza (gdy przycisk jest zaznaczony). Można w ten sposób dołączyć dodatkową obsługę skrótu klawiaturowego lub sprawdzić poprawność znaków w polu. Tworzenie skrótów klawiaturowych zostanie omówione w dalszej części artykułu.
onkeydown	Zdarzenie wywołane w chwili naciśnięcia klawisza (gdy przycisk jest zaznaczony). Zdarzenie zostanie wywołane natychmiast po naciśnięciu klawisza, bez względu na to, czy klawisz zwolniono, czy nie. Rzadko używane.
onkeyup	Zdarzenie wywołane w chwili zwolnienia klawisza (gdy przycisk jest zaznaczony).
onfocus	Zdarzenie wywołane w chwili zaznaczenia elementu przy pomocy klawisza TAB lub kliknięcia myszą. Zdarzenie można wykorzystać do wyróżnienia elementu okna lub wyświetlenia tekstu pomocy.
onblur	Zdarzenie wywołane w chwili utracenia zaznaczenia, gdy użytkownik kliknie inny element okna lub naciśnie klawisz TAB. Można w ten sposób weryfikować informacje lub zamykać wyskakujące okienka. Lepiej jednak weryfikować pola po kliknięciu klawisza OK.
onload	Zdarzenie wywołane w chwili pierwszego otwarcia okna. Zwykle dołącza się ten program obsługi do znacznika okna w celu jego inicjacji. Dzięki temu można przypisać do wszystkich pól wartości domyślne na warunkach określonych w skrypcie.
onunload	Zdarzenie wywołane w chwili zamknięcia okna. Zwykle dołącza się ten program obsługi do znacznika okna w celu zapisania informacji przed zamknięciem okna.

spis obsługiwanych przez XUL zdarzeń (spis pochodzi ze strony XULPlanet.com).

Naszą podróż po zdarzeniach rozpoczniemy od najczęściej spotykanego programu obsługi zdarzeń: *onclick*. Aby dołączyć ten program, należy dopisać *onclick* w linii kodu, w której tworzymy przycisk. Należy też określić, jaka czynność ma być wykonana. Zmieniamy zatem linię z kodem przycisku na:

```
<button id='name' label='My button' onclick='alert('hello');'/>
```

W cudzysłowie zapisujemy czynność, którą ma uruchomić program obsługi. W naszym prostym przykładzie wykorzystujemy funkcję *alert* języka Javascript do wyświetlenia okna dialogowego z napisem *hello*. Jeżeli zapiszemy teraz dokonane zmiany i odświeżymy zawartość strony, a następnie klikniemy przycisk, zobaczymy wyskakujące okienko.

## Funkcjonalność przede wszystkim

Dołączenie funkcjonalności do elementu okna jest całkiem proste – widać to na powyższym przykładzie. Jeżeli umieścimy całą niezbędną funkcjonalność w cudzysłowie zdarzenia *onclick*, to pomimo pozorowanej prostoty takiego rozwiązania, napotkamy na duży problem – nasz kod strony zostanie znacznie wydłużony. Co jednak zrobić, gdy potrzebujemy wykonać skomplikowane oblicze-

nia matematyczne czy złożone przetwarzanie danych, które wymaga wielu linii kodu? W tym celu zwinimy naszą funkcję.

Funkcja to część kodu, którą można wywołać inny kod. Funkcję można przyrównać do książki w bibliotece. Możemy czytać jedną książkę i czegoś w niej nie rozumieć, więc sięgamy po kolejną, opisującą dokładniej to, czego nie rozumiemy. Funkcja pozwala na pakowanie i dostęp do skomplikowanych fragmentów kodu przy pomocy jednej linii kodu. Mimo że funkcje mogą być bardzo rozbudowane i skomplikowane, to większość z nich jest bardzo prosta – pokażemy to na przykładzie użycia funkcji Javascript.

Aby skorzystać z dobrodziejstw funkcji, zalecamy umieszczenie wszystkich funkcji w osobnym pliku. Dostęp do nich będziemy uzyskiwać z poziomu skryptu XUL. Tworzy-

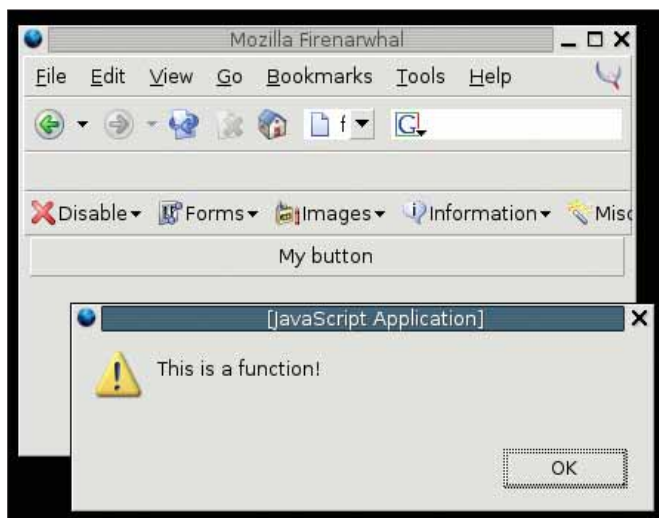
my zatem plik o nazwie *functions.js* i dopiszemy następujące linie:

```
function func_showdialog()
{
    alert('This is a function!');
}
```

Kod zawiera kilka elementów kluczowych. W pierwszej linii tworzymy funkcję (*function*), a następnie chcemy wywołać naszą funkcję (*func\_showdialog*). Puste nawiasy służą do wskazania, że nowa funkcja niczego nie przetwarza. Wspólną cechą wszystkich funkcji jest przekazywanie do nich informacji umieszczonych w nawiasach, tak przekazane dane zostaną następnie przetworzone w celu uzyskania wyniku. W naszym przypadku po prostu uruchamiamy kod funkcji, nie przetwarzając żadnych danych (stąd puste nawiasy).

Nawiasy klamrowe określają rozmiar funkcji. Kod funkcji zaczyna się znakiem { i kończy znakiem }. Nasza funkcja składa się tylko z jednej linii, powodującej wyświetlenie znanego już z wcześniejszego przykładu okna dialogowego. Niezwykle istotny jest znak średnika stawiany po każdej instrukcji wewnątrz funkcji. Określa on koniec linii w języku Javascript.

Dysponując gotową funkcją, możemy dołączyć ją do kodu XUL. Na początek należy załadować plik *functions.js* do naszego pliku XUL. Do określenia źródła



Rysunek 1: Pierwszy skrypt z kodem Javascript.

### Listing 2: Pełny kod po modyfikacjach

```
<?xml version='1.0'?>
<?xml-stylesheet href='chrome://global/skin/' type='text/css'?>

<window
  id='test-window'
  title='Test Program'
  xmlns='http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul'>

  <script src='functions.js'/>

  <button id='name' label='My button' onclick='func_showdialog()'/>

</window>
```

### Listing 3: Plik drugi.xul

```
<?xml version='1.0'?>
<?xml-stylesheet href='chrome://global/skin/' type='text/css'?>

<window
  id='test-window'
  title='Test Program'
  xmlns='http://www.mozilla.org/keymaster/gatekeeper/there.is.only.xul'>

  <script src='second.js'/>

  <textbox id='textbox'/>
  <button id='showinfo' label='Show Information' onclick='func_getinfo()'/>

</window>
```

### Listing 4: Plik *drugi.js*

```
function func_getinfo()
{
  var info=document.
  getElementById('textbox');
  alert(info.value);
}
```

pliku używamy znacznika `<script>` wewnątrz innego znacznika – `<window>`. Należy ponadto zmienić wartość atrybutu `onclick` na nazwę naszej funkcji. Na Listingu 2 umieściliśmy pełny kod ze wszystkimi wymaganymi zmianami.

Podaliśmy tutaj cały kod, aby pokazać, jak umieścić nowy kod we właściwym miejscu. Gdybyśmy wstawili znacznik `<script>` pod znacznikami `<?xml>` (typowy błąd popełniany przez osoby uważające, że skrypt dotyczy całego pliku), otrzymalibyśmy komunikat o błędzie. Należy też upewnić się, czy skrypt znajduje się wewnątrz znaczników `<window>`.

### Tajemnice modelu DOM

Czas na odkrycie tajemnic obiektowego modelu dokumentu (DOM). Na początek stwórzmy jednak nowy kod programu, utwórzmy nowy plik pod nazwą `drugi.xul` i przepiszmy do niego kod z Listingu 3.

Powyższy kod zawiera dwa główne elementy graficzne: okno dialogowe i przycisk. Szczególnie istotne jest odnotowanie atrybutu identyfikatora (*id*) każdego elementu okna, dzięki któremu otrzymuje niepowtarzalną nazwę. Powinniśmy zawsze nadawać unikalne atrybuty, które będą jednocześnie łatwe do zapamiętania. W naszym przykła-

dzie zmieniliśmy także nazwę funkcji na `func_getinfo()`, a nazwę pliku przechowującego funkcje na `drugi.js`.

Kolejnym plikiem, w którym należy dokonać zmian, jest plik `drugi.js`. Dopusujemy do niego kod z Listingu 4.

Dopiero tutaj zaczyna się prawdziwe działanie skryptu. Skrypt ma za zadanie umożliwić użytkownikowi wpisanie dowolnego tekstu w oknie dialogowym, tak aby później wpisany tekst pojawił się w ramce ostrzegawczej. Aby to osiągnąć, trzeba skomunikować się jakoś z oknem dialogowym, pobrać wprowadzone przez użytkownika informacje i umieścić je w ramce ostrzegawczej.

W pierwszej linii kodu funkcji tworzymy nową zmienną o nazwie *info*. Do określenia, że *info* jest zmienną, używamy słowa kluczowego *var*. Dla osób niezorientowanych w typach danych przypominamy, że zmienna umożliwia przechowywanie informacji w pamięci komputera i korzystanie z nich poprzez nazwę zmiennej. Zmienną można porównać do kartonowego pudełka z zapisaną na jednym z boków jego nazwą. Jeżeli włożymy coś do pudełka, będziemy mogli mówić o tym, wymieniając nazwę tego pudełka. W naszym przypadku pudełko otrzymało nazwę *info*.

W tej samej linii co *var info* ustalamy także zawartość zmiennej (dane umieszczone po prawej stronie znaku `=`). Kod ten powoduje uzyskanie informacji wpisanych w oknie dialogowym. Do uzyskania dostępu do elementu okna przy pomocy identyfikatora *textbox* skorzystaliśmy z polecenia języka Javascript – `getElementById`. Polecenie `getElementById` występuje zaraz obok słowa *document*. Zasada działania polega na tym, że *document* odnosi się do głównego dokumentu XUL i umieszczonych w nim elementów okna. Znak kropki (`.`) oznacza, że polecenia umieszczone po prawej stronie znaku kropki powinny być zastosowane wobec kodu programu znajdującego się po lewej stronie tego znaku. W naszym przykładzie szukamy w głównym dokumencie elementu okna o identyfikatorze *textbox*. Dzięki temu uzyskujemy zawartość okna dialogowego

### Inne języki programowania i XUL

Użytkownicy XUL często zadają pytanie, czy istnieją jakieś inne języki programowania, których można użyć do tworzenia skryptów dla XUL. Obecnie jedynym obsługiwany językiem jest Javascript, ale prowadzone są prace nad wprowadzeniem innych języków programowania w kolejnych wersjach przeglądarki Mozilla. Mogą to być m.in. języki takie jak Python, C# lub inne. Wiele zależy od wyników dyskusji na temat utworzenia kolejnego poziomu funkcjonalności Mozilli – wirtualnej maszyny Mozilla (ang. Mozilla Virtual Machine). Więcej informacji na temat postępów rozwoju projektu Mozilla znajduje się na stronach Mozillazine (<http://www.mozillazine.org>) i Planet Mozilla (<http://planet.mozilla.org/>).

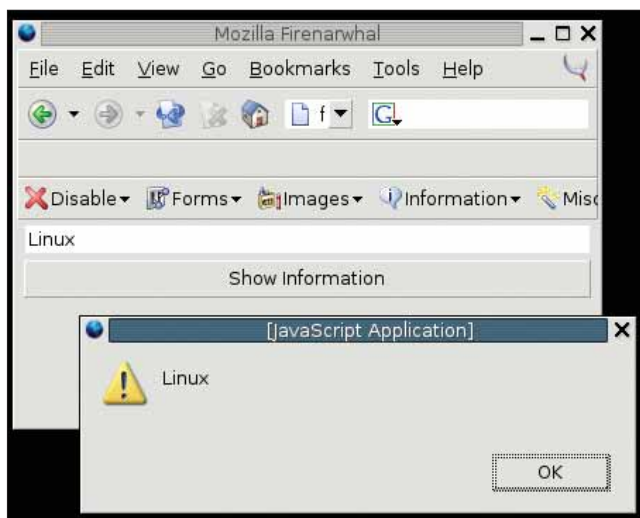
przechowywanego w zmiennej *info*.

Określanie za pomocą znaku kropki miejsca, gdzie należy zastosować daną funkcję, jest cechą typową wielu języków programowania. Skorzystaliśmy z niej także w drugiej linii naszego kodu, w której użyliśmy funkcji *value* do pobrania wartości zmiennej *info*. Zagnieźdźmy ten kod w funkcji `alert` i w ten sposób wyświetlamy wpisany wcześniej tekst w nowym oknie. Warto zauważyć, że nie użyliśmy tutaj podwójnych znaków cudzysłowu – wykorzystujemy je tylko do 'cytowania' konkretnego tekstu (ciągu znakowego).

### Podsumowanie

W bieżącym numerze zrobiliśmy pierwsze poważne kroki w kierunku stworzenia funkcjonalnego interfejsu XUL. Nadal przed nami jeszcze wiele do zrobienia, ale mamy już solidne podstawy, które umożliwią w przyszłości łatwiejsze pisanie skryptów. Dobre zrozumienie podstawowych zasad jest zawsze istotne w nauce języków programowania.

W następnym numerze pójdziemy o krok naprzód i stwórzmy szczegółowe funkcje oraz poznamy nowe możliwości łączenia różnych elementów interfejsów XUL w całość. Do tego czasu zapoznajcie się ponownie z przedstawionymi skryptami i spróbujcie zrozumieć jak najwięcej z dzisiejszej lekcji. Jeżeli coś jest nadal niejasne, nie ma powodu do zmartwień! W kolejnych numerach Linux Magazine omówimy dokładniej niektóre z przedstawionych tutaj zagadnień. A więc powodzenia!



Rysunek 2: Wykorzystanie modelu DOM do scalenia elementów okna ze sobą.