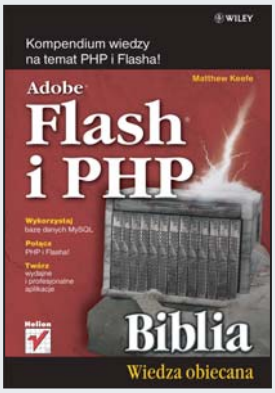
Adobe Flash i PHP. Biblia



Autor: Matthew Keefe
Tłumaczenie: Paweł Jabłoński
ISBN: 83-7197-641-0
Tytuł oryginału: Flash and PHP Bible
Format: 172×245, stron: 496

Kompendium wiedzy na temat PHP i Flasha!

- Wykorzystaj bazę danych MySQL
- Połącz PHP i Flasha!
- Twórz wydajne i profesjonalne aplikacje

Technologie Flash i PHP znane są ze swoich możliwości tworzenia dynamicznych rozwiązań. Flash po stronie klienta, PHP po stronie serwera. Mieszanka Flasha i PHP musi być wybuchowa. I tak właśnie jest! To połączenie powoduje eksplozję nowych możliwości, które wykorzystasz na Twoich stronach WWW. Teraz atrakcyjne i dynamiczne rozwiązania są w zasięgu Twoich rąk.

Jednak pojawia się pytanie: „Jak zapanować nad tym duetem?". Książka „Adobe Flash i PHP. Biblia" stanowi kompletne źródło informacji na ten temat. Dzięki niej poznasz podstawy technologii Flash i PHP, a także sposoby używania bazy danych MySQL oraz weryfikacji połączenia Flash – PHP. Ponadto nauczysz się obsługiwać dane w różnych formatach, tworzyć formularze, wykorzystywać ciasteczka oraz stosować dodatkowe biblioteki. Zdobędziesz również wiedzę na temat korzystania z gniazd, gwarancji bezpieczeństwa aplikacji oraz tworzenia rozbudowanych serwisów WWW. Książka ta zasłuży z pewnością na uznanie specjalistów w tej dziedzinie, jednak i dla początkujących będzie ona stanowiła świetny podręcznik do nauki tych technologii.

- Instalacja serwera Apache
- Instalacja bazy danych MySQL oraz PHP
- Konfiguracja środowiska pracy
- Podstawy PHP
- Zasady pracy w środowisku Flash
- Nawiązanie połączenia pomiędzy PHP i Flashem
- Obsługa danych w formacie XML i nie tylko
- Projektowanie i tworzenie formularzy
- Bezpieczeństwo pobieranych danych
- Wykorzystanie gniazd
- Rozszerzenia do Flasha i PHP
- Programowanie obiektowe
- Debugowanie i sposoby rozwiązywania najpopularniejszych problemów
- Optymalizacja i zapewnienie wydajności tworzonym aplikacjom
- Projektowanie i tworzenie prostych oraz rozbudowanych aplikacji

Twórz dynamiczne i atrakcyjne wizualnie strony WWW!

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

Spis treści

Rozdział 11.
Tworzymy proste aplikacje

W tym rozdziale:

- O składnikach aplikacji
- Piszemy klienta czata
- Tworzymy galerię w PHP
- Tworzymy inne dynamiczne aplikacje, korzystające z MySQL-a

W niniejszym rozdziale napiszemy kilka w pełni działających aplikacji. Jego celem jest utworzenie kilku takich aplikacji, a jednocześnie poznanie kilku podstawowych pułapek oraz sposobów ich omijania. Przyswoimy również nowe techniki, takie jak tworzenie procedur obsługi wielu zdarzeń, pisanie klas pochodnych czy korzystanie ze zdalnych usług.

W rozdziale omówimy również najlepsze sposoby pracy z MySQL-em, uwzględniające zarówno efektywność, jak i bezpieczeństwo. Na koniec powiemy o najlepszych metodach konserwowania aplikacji oraz technikach programowania, pozwalających zachować możliwość dalszego rozwijania tworzonego oprogramowania.

O składnikach aplikacji

Tworzenie aplikacji nie polega na tym, że „odpalamy" Flasha lub Twój ulubiony edytor i wklepujemy kod linijka po linijce. Najlepszym pomysłem na rozpoczęcie pisania programu jest etap oceny. Chodzi o moment, w którym przypatrujemy się liście zadań, jakie aplikacja powinna realizować, zadajemy sobie pytanie, kto ma być jej użytkownikiem oraz w jaki sposób będziemy ją rozwijać. W tabeli 11.1 umieściłem trzy punkty, składające się na etap oceny aplikacji:

Tabela 11.1. *Trzy punkty, składające się na etap oceny*

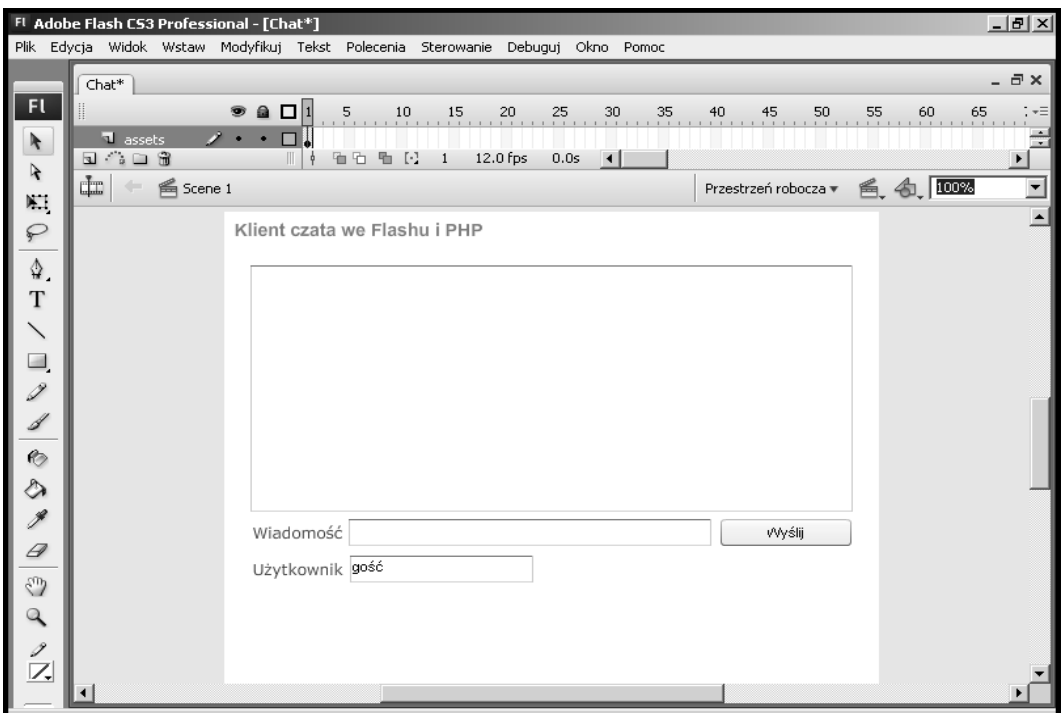
Co	Co aplikacja powinna robić?
Kto	Potencjalni użytkownicy aplikacji
Jak	Techniki stosowane podczas tworzenia aplikacji

W przykładach prezentowanych poniżej pomijamy punkt „jak”, bo przecież jest to książka o Flashu i PHP. Po rozprawieniu się z punktem „jak” możemy przejść do punktu „kto”. W punkcie tym określamy oczywiście, kto będzie potencjalnym użytkownikiem naszej aplikacji. W przypadku niektórych programów nie jest możliwe sztywne zdefiniowanie potencjalnego użytkownika; dotyczy to zwłaszcza serwisów WWW. Mimo to najlepiej jest spróbować go scharakteryzować, a najlepszym sposobem realizacji tego zadania jest zastanowienie się nad typem rozwijanej aplikacji. Czy to ma być na przykład oprogramowanie typu e-commerce? A może odtwarzacz wideo? Wiedza o tym, kto będzie korzystał z aplikacji, jest kluczem do sukcesu.

Gdy już rozpatrzyliśmy punkty „jak” i „kto”, musimy się zastanowić, co właściwie będzie robiła nasza aplikacja. Czy poszedłbyś do sklepu ze sprzętem, nakupowałbyś wszelkich materiałów, a dopiero potem zacząłbyś się zastanawiać, co z nich zrobić? To samo pytanie należy zadać sobie w trakcie tworzenia aplikacji. Gdy zaczniesz stosować te metody, szybko staną się one Twoją drugą naturą.

Czym jest projekt aplikacji

Po zakończeniu procesu oceny aplikacji naturalnym krokiem jest przejście do projektowania. Ale tak jak w przypadku etapu oceny, dobrze jest najpierw opracować jakiś plan. Planem będzie szkic kreślony ołówkiem na papierze albo projekt we Flashu, składający się z podstawowych składników. Na rysunku 11.1 widzimy przykładowy szkic.



Rysunek 11.1. *Przykładowy szkic aplikacji sporządzony z użyciem podstawowych składników dostępnych we Flashu*

Piszemy szkielet skryptu

Po sporządzeniu całościowego szkicu rozwijanej aplikacji możemy przejść do realizowanych przez nią funkcji. Jest to moment, w którym zaczniemy pisać pseudokod w postaci funkcji, zmiennych, zdarzeń oraz wszelkich innych konstrukcji, które będziemy chcieli zastosować w aplikacji.

```
// Szkielet skryptu

function loadUser(id:uint):void
{
    // Odwołaj się do serwera,
    // przekazując id jako identyfikator użytkownika
}

function handleServerResponse(e:Event):void
{
    // Przejmij odpowiedź serwera,
    // Wywołaj odpowiednie funkcje aplikacji
}

function drawInterface():void
{
    // Wyświetl interfejs użytkownika
}

function redraw():void
{
    // Ponownie wyświetl interfejs po pobraniu informacji,
    // aby dane były aktualne
}

// Przypisanie procedur obsługi zdarzeń
```

Jak widać, szkielet nie tworzy całej aplikacji, przedstawia tylko jej budowę. Dzięki niemu szybko określisz sposób działania programu oraz jego zadania. Pisanie pseudokodu jest tym ważniejsze, im większy jest projekt, nad którym się pracuje, ale niezależnie od jego wielkości dobrą praktyką jest ocena, szkicowanie i planowanie.

Zakończenie etapu planowania

Ostatnie czynności związane z planowaniem aplikacji wcale nie muszą być ostateczne. Możesz albo powtórzyć procedurę planowania, dokonując niezbędnych lub przydatnych uzupełnień, albo rozpocząć proces tworzenia aplikacji. Czasem o wyborze decyduje klient lub dyrektor do spraw tworzenia projektów, o ile taki jest.

W tym momencie procesu planowania wiemy już, co aplikacja będzie robić, kto będzie z niej korzystał oraz w jaki sposób ją utworzymy. Stworzyliśmy również jej szkielet i rozpoczęliśmy programowanie. Następny krok należy do Ciebie jako do dewelopera. Możesz albo kontynuować proces projektowania i spróbować doprowadzić go końca, albo przejść do pisania kodu.

Zazwyczaj będziesz rozwijał plan aplikacji do etapu, w którym będziesz mógł rozpocząć programowanie, ponieważ najprawdopodobniej w trakcie implementacji wprowadzisz wiele zmian do samego planu. Jednak przy odpowiedniej dozie planowania możesz ograniczyć konieczność wprowadzania tego rodzaju zmian do minimum.

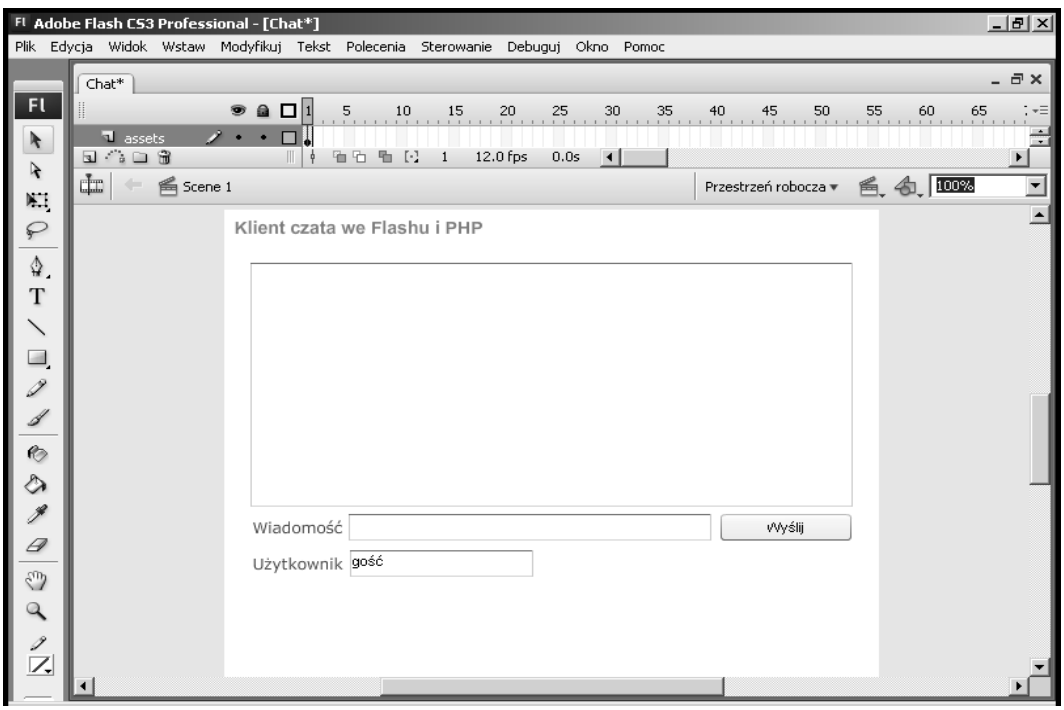
Piszemy klienta czata

Gdy już poznałeś nieco najwydajniejsze metody tworzenia oprogramowania, możesz przystąpić do pisania swojej pierwszej w pełni działającej aplikacji. Będzie to klient czata we Flashu, który będzie wykorzystywał PHP do komunikacji z bazą danych MySQL. Po utworzeniu aplikacji podstawowej będziemy dodawać do niej różne opcje, ucząc się w ten sposób najlepszych metod tworzenia skalowalnego oprogramowania.

Aplikację skalowalną rozwija się, mając na względzie to, że na początku mogą z niej korzystać dwaj użytkownicy, ale z czasem mogą jej używać miliony ludzi. Chodzi o to, żeby się z tym liczyć i rozwijać oprogramowanie w sposób pozwalający na poszerzanie jego możliwości.

Część aplikacji, którą tworzymy we Flashu

Rozpocznijmy tworzenie aplikacji od sporządzenia projektu. Jej projekt jest już gotowy, ale nie wahaj się go zmieniać. Tak naprawdę to zachęcam Cię do rozbudowywania wszystkich przykładów, bo właśnie to jest sposób na przyswojenie sobie nowych technik i stanie się lepszym deweloperem. Interfejs czata możemy zobaczyć na rysunku 11.2.



Rysunek 11.2. *Projekt czata utworzony we Flashu z gotowych składników*

Zasadniczo interfejs składa się ze składnika typu *TextArea*, który służy do wyświetlania wiadomości czatowych, dwóch składników typu *TextInput*, w których wpisuje się wiadomość oraz imię czy pseudonim, oraz przycisku typu *Button* do wysyłania komunikatów.

W tabeli 11.2 przedstawiłem nazwy obiektów reprezentujące poszczególne składniki projektu:

Tabela 11.2. *Nazwy obiektów reprezentujące poszczególne składniki projektu*

Składnik	Nazwa obiektu	Zastosowanie
TextArea	`messagesTxt`	Wyświetla wiadomości pojawiające się w czacie
TextInput	`newMsgTxt`	Nowa wiadomość
TextInput	`usernameTxt`	Imię osoby korzystającej z czata
Button	`sendBtn`	Wysyła nową wiadomość do serwera

W trakcie pisania we Flashu skryptu naszej aplikacji, odwołując się do poszczególnych obiektów, będziemy korzystać z ich nazw.

Po ukończeniu projektu interfejsu użytkownika — w naszym przypadku po jego załadowaniu — możemy przystąpić do pisania kodu.

Na początek utwórzmy blok deklaracji zmiennych globalnych. Pierwsza zmienna zawiera bezwzględną ścieżkę do pliku „.php”. Dwie dalsze — reprezentują skrypty PHP, odpowiedzialne za wysyłanie i pobieranie wiadomości do i z bazy danych. Następnie mamy dwie zmienne logiczne, dzięki którym mamy pewność, że nie zostaną utworzone dwie wiadomości naraz. Ostatni zbiór zmiennych jest nam potrzebny do utworzenia obiektu czasomierza oraz niszczyciela pamięci podręcznej.

```
var phpPath:String = "http://localhost/helion/rozdzial11/chatClient/";
var phpSendMessage:String = phpPath + "message.php";
var phpLoadMessages:String = phpPath + "getMessages.php";

var loadingMessages:Boolean = false;
var sendingMessage:Boolean = false;

var timer:Timer;
var cacheBuster:String = "?cb=1";
```

Na początku napiszemy funkcję `init()`, czyli funkcję inicjalizującą. Jest ona wywoływana tylko jeden raz, podczas uruchamiania aplikacji. Jej zadaniem jest przypisanie funkcji obsługi czasomierza, z którego korzystamy w trakcie pobierania wiadomości, oraz pierwsze wywołanie niszczyciela pamięci podręcznej. O obiekcie tym powiemy kilka słów nieco dalej. Ostatnim zadaniem funkcji `init()` jest wywołanie funkcji `loadMessages()` i wypełnienie listy wiadomości:

```
function init():void
{
    // Uruchom czasomierz na potrzeby ładowania obrazów
    timer = new Timer(5000, 0);
    timer.addEventListener(TimerEvent.TIMER, timerHandler);
    timer.start();
```

```
    cacheBuster =  getCacheBuster();
    loadMessages(); //pierwsze wywołanie
}
```

Dostępna w środowisku Flash klasa `Timer` ma wiele fantastycznych zastosowań. W naszym przykładzie korzystamy ze zdarzenia `TIMER`, które generowane jest za każdym razem, gdy licznik spadnie do zera. Zarówno czas odliczania, jak i liczbę powtórzeń przekazuje się do konstruktora jako parametry. Ustalamy odliczanie na `5000`, czyli pięć sekund, a liczbę powtórzeń na `0`, co mówi ActionScriptowi, że ma powtarzać odliczanie w nieskończoność. Po utworzeniu czasomierza przypisujemy mu procedurę obsługi zdarzeń i niezwłocznie uruchamiamy odliczanie. Procedura obsługi czasomierza wywołuje funkcję `loadMessages()`.

Można by skrócić ten fragment kodu poprzez wstawienie funkcji `loadMessages()` zamiast procedury obsługi `timerHandler()`, jednak dzięki zastosowanemu podejściu mamy możliwość dodania w przyszłości nowych udogodnień. Pisanie właściwych procedur obsługi, aby kod w przyszłości można było zmieniać, jest ze wszech miar dobrą praktyką, a jej stosowanie sprawia dodatkowo, że kod łatwiej się czyta:

```
function timerHandler(e:TimerEvent):void
{
    loadMessages();
}
```

Funkcja `loadMessages()` odwołuje się do serwera i obsługuje jego odpowiedź. Odwołanie do serwera jest bardzo podobne do odwołań w innych przykładach, jednak pojawia się tu kilka nowych elementów. Jednym z nich jest zmienna `cacheBuster`, dzięki której mamy pewność, że nie pobieramy wyników znajdujących się w pamięci podręcznej.

Blokujemy przechowywanie danych dynamicznych w pamięci podręcznej

Właśnie zadeklarowaliśmy zmienną, która pozwala na wyłączenie umieszczania wyników w pamięci podręcznej, ale o co właściwie chodzi? O przechowywaniu danych pochodzących z serwera w pamięci podręcznej mówimy wówczas, gdy dane pozyskiwane dynamicznie przechowuje się w celu szybszego ładowania podczas następnych odwołań. W pierwszej chwili wydawać się może, że to świetny pomysł; zresztą w większości sytuacji to po prostu jest świetny pomysł. Jednak podczas pobierania danych czata, które wciąż się zmieniają, możemy z dużym prawdopodobieństwem założyć, że powinny być one zawsze świeże. Osiągamy to, dodając zmienną o nazwie `cacheBuster`, która oszukuje przeglądarkę, że każde odwołanie do tego samego pliku jest w istocie innym odwołaniem[1].

Poniżej znajduje się bardzo prosty przykład przekazania niszczyciela pamięci podręcznej i właściwego ciągu znaków do serwera:

```
function getRandom(length:uint):void
{
    return Math.round(Math.random() * (length-1)) + 1;
}
```

[1] Opisane rozwiązanie nie jest specjalnie zgodne ze standardem http, choć niewątpliwie działa. Bardziej stosownym podejściem jest użycie nagłówków: „Cache-control: no-cache” i „Pragma: no-cache” (dla zachowania zgodności z http 1.0). Nagłówki te można wysłać z poziomu skryptu PHP funkcją header(). Zaletą takiego podejścia jest między innymi to, że strony takie nie są w ogóle zapisywane w pamięci podręcznej — *przyp. red.*

```
var rand:String = "?cb=" + getRandom(8);
var php:String = "http://localhost/helion/rozdzial11/getMessages.php.php" + rand;

trace("URL: " + phpFile); //getMessages.php?cb=65378426
```

Tłumiki pamięci podręcznej wydłużają czas ładowania i sprawiają, że każdy plik z tłumikiem będzie ładowany za każdym razem, gdy do serwera zostanie wysłane żądanie jego pobrania.

Następnie w ramach procesu wysyłania wiadomości sprawdzamy, czy zawiera ona przynajmniej trzy znaki. Jeżeli jej długość jest mniejsza, wyświetlany jest komunikat o błędzie. Istnieją dwa sposoby zapisu tego sprawdzenia:

Pierwszy sposób to:

```
if (newMsgTxt.text.length > 2)
{
}
```

Drugi sposób zapisu jest łatwiejszy do zrozumienia:

```
if (newMsgTxt.text.length >= 3)
{
}
```

Oba zapisy oznaczają to samo, jednak drugi jest o wiele czytelniejszy z punktu widzenia logiki.

Teraz, skoro wiadomość ma odpowiednią długość, możemy przejść do dalszej części procesu wysyłania jej do serwera. Teraz tworzymy wywołanie serwera:

```
var variables:URLVariables = new URLVariables();
variables.user = usernameTxt.text;
variables.msg = newMsgTxt.text;
var urlRequest:URLRequest = new URLRequest(phpSendMessage + getCacheBuster());
urlRequest.method = URLRequestMethod.POST;
urlRequest.data = variables;

var urlLoader:URLLoader = new URLLoader();
urlLoader.addEventListener(Event.COMPLETE, sendMessageHandler);
urlLoader.load(urlRequest);
```

Odwołując się do serwera, należy nadać odpowiednie wartości właściwościom obiektu klasy `URLVariables`, aby przesłać wprowadzoną nazwę użytkownika oraz wiadomość. Wywołujemy również funkcję oszukującą pamięć podręczną, aby dane zawsze były aktualne.

Na końcu funkcji — zamiast czekać, aż lista wiadomości zostanie zaktualizowana z serwera — od razu dodajemy wiadomość użytkownika. Dzięki temu aplikacja lepiej reaguje na polecenia i nie odnosi się wrażenia, że wiadomości są ignorowane.

```
addMessage(usernameTxt.text, newMsgTxt.text);
```

Funkcja przyjmuje dwa argumenty: imię lub pseudonim użytkownika oraz wiadomość; obie wartości pochodzą z odpowiednich składników. Potem otrzymane informacje umieszczamy w ciągu znaków w postaci kodu HTML, który przypisujemy do zmiennej `messagesTxt` klasy

TextArea. Nazwę użytkownika umieszczamy między znacznikami pogrubienia w celu wyróżnienia imienia. Tak naprawdę, w polach edycyjnych udostępniających HTML możesz stosować wiele najczęściej używanych znaczników:

```
function addMessage(user:String, msg:String):void
{
   messagesTxt.htmlText += "<b>" + user + "</b>" + ": " + msg + "\n";
}
```

Tabela 11.3 przedstawia dostępne znaczniki HTML:

Tabela 11.3. *Dostępne znaczniki HTML*

Nazwa	Znacznik HTML
Znacznik zakotwiczenia	<a>
Znacznik pogrubienia	<b>
Znacznik przerwania wiersza	
Znacznik wyboru czcionki	<font>
Znacznik obrazu	<img>
Znacznik kursywy	<i>
Znacznik elementu listy	
Znacznik akapitu	<p>
Znacznik sekcji	
Znacznik podkreślenia	<u>

Wywołanie funkcji sendMessageHandler() następuje za każdym razem, gdy wiadomość zostaje pomyślnie wysłana do serwera. Jedynym istotnym fragmentem tej funkcji jest jej ostatni wiersz, który czyści pole tekstowe zawierające wiadomość i pozwala użytkownikowi wpisać nowy komunikat. Moglibyśmy usunąć wysłaną wiadomość już w funkcji sendMessage(), ale zastosowane rozwiązanie daje nam pewność, że pozostanie ona nietknięta aż do czasu dodania jej do listy.

```
function sendMessageHandler(e:Event):void
{
   ...
   newMsgTxt.text = "";
}
```

Skrypt PHP poinformuje o błędzie, jeżeli nie powiodło się ładowanie strony, natomiast nie poinformuje, jeżeli nie powiedzie się realizacja zapytania SQL. Oczywiście moglibyśmy wyposażyć nasz przykład w lepszą obsługę błędów.

Gdy już mamy kod wysyłający i obsługujący odwołania do serwera, możemy zająć się funkcją zarządzającą wiadomościami i wyświetlającą je w polu typu TextArea.

Funkcję loadMessages() wywołujemy z dwóch miejsc. Najpierw z funkcji init(), której przyjrzeliśmy się wcześniej, a następnie z procedury obsługi zdarzenia generowanego przez czasomierz.

Na początek sprawdzamy, czy nie nastąpiło równoległe wywołanie funkcji `loadMessages()`. Dzięki temu mamy pewność, że nie zasypiemy serwera wiadomościami, przez co mógłby przestać odpowiadać. Jeżeli ładowanie właśnie się odbywa, to po prostu opuszczamy funkcję, kończąc całą operację.

Jeżeli nie odbywa się równoległe ładowanie, ustawiamy wartość zmiennej `loadingMessages`. Przypomina to zamknięcie drzwi po wejściu do pokoju. Większość zadań wykonywanych przez funkcję `loadMessages()` przypomina wysyłanie.

Ogólnie mówiąc, najpierw określamy plik PHP, który powinien zostać załadowany, oraz procedurę obsługi zdarzenia, które zostanie wygenerowane, gdy dane nadejdą z serwera:

```
function loadMessages():void
{
   if (loadingMessages) return;
   loadingMessages  = true;

   var urlRequest:URLRequest = new URLRequest(phpLoadMessages + getCacheBuster());
   var urlLoader:URLLoader = new URLLoader();

   urlLoader.addEventListener(Event.COMPLETE, loadMessagesHandler);
   urlLoader.load(urlRequest);
}
```

Obsługa odpowiedzi w formacie XML

Funkcja pobierająca odpowiedź przetwarza dane w formacie XML i przekazuje wiadomość do wyświetlenia:

```
function loadMessagesHandler(e:Event):void
{
   var loader:URLLoader = URLLoader(e.target);
   var xml:XML = new XML(loader.data);

   messagesTxt.htmlText = "";

   for each(var item in xml..message)
   {
      addMessage(item.name, item.msg);
   }

   cacheBuster = getCacheBuster();
   loadingMessages = false;
}
```

Należy pamiętać, że wielkość liter elementów odpowiedzi XML ma znaczenie. Standardem są małe litery lub wielkie litery na początku wyrazów w przypadku wielu słów.

Po pomyślnym załadowaniu dokumentu XML następuje pętla `for each`, tak samo jak w przykładzie ładowania pliku w formacie XML w rozdziale 4. Działanie pętli opiera się na elementach `message`, znajdujących się w odpowiedzi. Przykładowy dokument nadesłany przez serwer mógłby wyglądać tak:

```
<messages>
   <message id='29'>
      <name>guest1</name>
      <msg>Flash daje sporo zabawy</msg>
   </message>
   <message id='30'>
      <name>guest2</name>
      <msg>PHP i Flash jest jeszcze fajniejszy</msg>
   </message>
   <message id='32'>
      <name>guest1</name>
      <msg>Dzięki nim tyle możesz zrobić</msg>
   </message>
   <message id='33'>
      <name>guest2</name>
      <msg>No pewnie, popatrz na tego czata!</msg>
   </message>
</messages>
```

Ostatnią czynnością wykonywaną w ramach funkcji `loadMessagesHandler()` jest utworzenie niszczyciela pamięci podręcznej i nadanie zmiennej `loadingMessages` wartości `false`. Dzięki temu możliwe będzie pobranie kolejnych wiadomości.

O niszczycielach pamięci podręcznej mówiliśmy już wcześniej. Warto jednak dodać, że istnieje wiele sposobów tworzenia unikalnych ciągów znaków. Ciągle zmieniającą się daną jest na przykład aktualna data, a w ActionScripcie istnieje metoda `getTime()`, która zwraca liczbę milisekund, jakie upłynęły od 1 stycznia 1970 roku. Korzystamy z metody pobierania czasu, ponieważ czas wciąż płynie, a jego wartość nigdy się nie powtarza. Dzięki temu za każdym razem, gdy skorzystamy z tej metody, otrzymujemy unikalny ciąg znaków.

W tym momencie kod w ActionScripcie jest już gotowy. Poniżej zamieszczam go w całości, abyś łatwiej mógł się w nim odnaleźć:

```
var phpPath:String = "http://localhost/helion/rozdzial11/chatClient/";
var phpSendMessage:String = phpPath + "message.php";
var phpLoadMessages:String = phpPath + "getMessages.php";

var loadingMessages:Boolean = false;
var sendingMessage:Boolean = false;
var chatMessages:Array = new Array();

var timer:Timer;
var cacheBuster:String = "?cb=1";

function init():void
{
   // Uruchom czasomierz na potrzeby ładowania obrazów
   timer = new Timer(5000, 0);
   timer.addEventListener(TimerEvent.TIMER, timerHandler);

   timer.start();

   cacheBuster =  getCacheBuster();

   loadMessages(); // pierwsze wywołanie
}
```

```
function sendMessage(e:MouseEvent):void
{
   if (usernameTxt.text == "")
   {
      trace("Wymagane jest imię lub pseudonim użytkownika");
      return;
   }

   if (newMsgTxt.text.length >= 3)
   {
      var variables:URLVariables = new URLVariables();
      variables.user = usernameTxt.text;
      variables.msg = newMsgTxt.text;

      var urlRequest:URLRequest = new URLRequest(phpSendMessage + getCacheBuster());
      urlRequest.method = URLRequestMethod.POST;
      urlRequest.data = variables;

      var urlLoader:URLLoader = new URLLoader();
      urlLoader.addEventListener(Event.COMPLETE, sendMessageHandler);
      urlLoader.load(urlRequest);

      // Aby wiadomość była wyświetlana
      addMessage(usernameTxt.text, newMsgTxt.text);
   }
}

function sendMessageHandler(e:Event):void
{
   var loader:URLLoader = URLLoader(e.target);
   var variables:URLVariables = new URLVariables(loader.data);
   trace("Odpowiedź: " + variables.resp);

   // Wyczyść pole wiadomości
   newMsgTxt.text = "";
}

function loadMessages():void
{
   if (loadingMessages) return;
   loadingMessages  = true;

   var urlRequest:URLRequest = new URLRequest(phpLoadMessages + getCacheBuster());
   var urlLoader:URLLoader = new URLLoader();

   urlLoader.addEventListener(Event.COMPLETE, loadMessagesHandler);
   urlLoader.load(urlRequest);
}

function loadMessagesHandler(e:Event):void
{
   var loader:URLLoader = URLLoader(e.target);
   var xml:XML = new XML(loader.data);

   loadingMessages = false;
   messagesTxt.htmlText = "";

   for each(var item in xml..message)
```

```
    {
        addMessage(item.name, item.msg);
    }

    cacheBuster = getCacheBuster();
}

function getCacheBuster():String
{
    var date:Date = new Date();
    cacheBuster = "?cb=" + date.getTime();
    return cacheBuster;
}

function addMessage(user:String, msg:String):void
{
    messagesTxt.htmlText += "<b>" + user + "</b>" + ": " + msg + "\n";
}

function timerHandler(e:TimerEvent):void
{
    trace("Obsługa czasomierza");
    loadMessages();
}

sendBtn.addEventListener(MouseEvent.CLICK, sendMessage);

init();
```

Część aplikacji w PHP

W tym momencie ukończyliśmy już część kodu naszej aplikacji w ActionScripcie. Teraz zajmiemy się kodem w PHP, wywoływanym w ActionScripcie.

Kod w PHP dzielimy na trzy pliki, jak widać w tabeli 11.4.

Tabela 11.4. *Podział skryptów PHP*

Plik z kodem w PHP	Zadania
getMessages.php	Pobiera wszystkie wiadomości z ostatnich 15 minut
messages.php	Zapisuje nową wiadomość w bazie danych
dbconn.php	Połączenie z bazą danych, do którego dostęp mają pozostałe skrypty

Pierwszym skryptem, nad którym będziemy pracować, jest *getMessages.php*. Na początku tego skryptu dołączamy plik zawierający połączenie z bazą danych, któremu przyjrzymy się za chwilę. Potem następuje odwołanie do MySQL-a, za pomocą którego odpytujemy bazę danych i pobieramy wiadomości z ostatnich 15 minut:

```
$sql = "SELECT * FROM flashChat WHERE dateAdded > " . (time() - (60 * 15));
$result = mysql_query($sql);
```

Warunek ostatnich 15 minut uzyskujemy, wprowadzając odpowiedni wiersz. Funkcja `time()` służy do pobierania wartości zmiennej czasu rzeczywistego w systemie UNIX. Od otrzymanej wartości odejmujemy `60` i mnożymy ją przez `15`. Liczba 60 oznacza liczbę sekund w minucie, a liczba 15 określa liczbę minut, o jaką nam chodzi. Moglibyśmy zapisać to proste równanie matematyczne, pomijając mnożenie. Pominięcie go sprawi, że kod będzie mniej czytelny, jednak zwiększy szybkość działania aplikacji:

```
time() - 900
```

Ciąg znaków z zapytaniem SQL zostaje przekazany do odpowiedniej funkcji, odwołującej się do serwera — `mysql_query()`, której wynik przypisujemy zmiennej `$result`. Po pomyślnej realizacji odwołania do bazy danych MySQL tworzymy pętlę, wewnątrz której przekształcamy dane na wiadomości. Na rysunku 11.3 w przeglądarce zapytań bazy MySQL — dostępnej za darmo pod adresem internetowym *http://www.mysql.com* — widzimy zapytanie.

Rysunek 11.3. *Wynik zapytania SQL widoczny w programie MySQL Query Browser*

Pętlę tworzymy za pomocą słowa kluczowego `while`. Wykonuje się ona, dopóki z bazy danych nie otrzymamy poprawnego wiersza. Moglibyśmy skorzystać z pętli `for` i użyć funkcji `mysql_num_rows()`, aby sprawdzić, ile wierszy zostało zwróconych przez serwer.

Funkcja `mysql_fetch_array()` pobiera jeden wiersz z tabeli bazy w postaci tablicy asocjacyjnej, którą przypisujemy zmiennej `$row`. Pętla została napisana w taki sposób, że generuje węzły XML, zawierające dane wiadomości, o których wspominaliśmy już w tym rozdziale:

```
while ($row = mysql_fetch_array($result))
{
   $xmlData .= "   <message id='" . $row['id'] . "'>\n";
   $xmlData .= "      <name>" . $row['username'] . "</name>\n";
   $xmlData .= "      <msg>" . $row['message'] . "</msg>\n";
   $xmlData .= "   </message>\n";
}
```

Właśnie uporaliśmy się z odwołaniem do serwera MySQL oraz z pętlą generującą wiadomość w formacie XML, którą zwracamy do skryptu we Flashu. Przesyłanie danych z powrotem do ActionScripta jest bardzo proste; umieszczamy dane jako argument wyrażenia `print`:

```
print $xmlData;
```

Za wyrażeniem `print` kryje się działanie całego skryptu *getMessages.php*, który prezentuję poniżej w całości:

```
<?php

include 'dbconn.php';

$sql = "SELECT * FROM flashChat WHERE dateAdded > " . (time() - (60 * 15));
$result = mysql_query($sql);

$xmlData = "<messages>\n";

while ($row = mysql_fetch_array($result))
{
   $xmlData .=  "   <message id='" . $row['id'] . "'>\n";
   $xmlData .=  "      <name>" . $row['username'] . "</name>\n";
   $xmlData .=  "      <msg>" . $row['message'] . "</msg>\n";
   $xmlData .=  "   </message>\n";
}

$xmlData .= "</messages>";
print $xmlData;

?>
```

Teraz zajmijmy się skryptem *messages.php*, który wstawia nowe wiadomości do bazy danych MySQL. Początek pliku wygląda tak samo jak poprzednio — dołączamy plik *dbconn.php*, który obsługuje połączenie z serwerem MySQL oraz logowanie:

```
include 'dbconn.php';
```

Następnie sprawdzamy, czy ciąg znaków przysłany z ActionScripta przypadkiem nie ma zerowej długości, co oznaczałoby, że jest pusty. W PHP długość ciągu otrzymujemy jako wynik funkcji `strlen()`, podając ciąg jako jej argument:

```
if (strlen($_POST['msg']) > 0)
{
   ...
}
```

Gdy przekonamy się, że wiadomość rzeczywiście ma poprawną długość, deklarujemy trzy zmienne; pierwsza zawierała nazwę użytkownika, druga — samą wiadomość, a trzecia — czas w formacie uniksowym:

```
$username = $_POST['user'];
$message = $_POST['msg'];
$date = time();
```

Potem za pomocą funkcji `mysql_query()` wykonujemy właściwe odwołanie do serwera MySQL. Funkcja ta przyjmuje jako argument zapytanie SQL, a zwraca albo identyfikator zasobu, albo błąd.

Samo wywołanie MySQL-a bardzo przypomina podobne wywołania z poprzednich przykładów. W bazie danych istnieje tabela `flashChat`, która ma cztery kolumny. Jednak w jednej z nich znajduje się pole `auto_increment`, a zatem nie trzeba nadawać mu wartości w ramach wyrażenia `INSERT`.

```
mysql_query("INSERT INTO flashChat (username, message, dateAdded)
  VALUES (
    '" . $username . "',
    '" . $message . "',
    '" . $date . "'
  )");
```

W bardziej dopracowanej aplikacji dodalibyśmy warstwę bezpieczeństwa, kontrolującą informacje przesyłane z zewnętrznego źródła. Nie jest ważne, czy chodzi o Flasha, o przeglądarkę, czy o usługę zewnętrzną; poprawność danych zawsze należy sprawdzać.

Użyliśmy nazwy `dateadded`, ponieważ `date` w SQL-u jest słowem zarezerwowanym i przeważnie spowoduje wygenerowanie błędu. W żadnym języku programowania nigdy nie należy stosować słów zarezerwowanych do innych celów niż cele wyznaczone przez twórców języka.

Pod koniec skryptu wysyłamy odpowiedź do Flasha, informując go, że wiadomość została dodana i że użytkownik może dodać kolejne komunikaty. Skrypt *messages.php* nie jest skomplikowany, niemniej spełnia swoje zadanie. Poniżej zamieszczam go w całości, abyś łatwiej mógł się w nim odnaleźć:

```
<?php

include 'dbconn.php';

if (strlen($_POST['msg']) > 0)
{
   $username = $_POST['user'];
   $message = $_POST['msg'];
   $date = time();

   mysql_query("INSERT INTO flashChat (username, message, dateAdded)
     VALUES (
       '" . $username . "',
       '" . $message . "',
       '" . $date . "'
     )");

   print "resp=MESSAGE_ADDED";
}

?>
```

Za pomocą PHP łączymy się z bazą danych

Ostatnim plikiem PHP w naszym czacie jest *dbconn.php*. Jego zadaniem jest obsługa połączenia z bazą danych oraz udostępnienie łącznika pomiędzy aplikacją a tabelami znajdującymi się w MySQL-u.

Skrypt jest niewielki, jednak istnieje wobec niego pewien bardzo istotny wymóg — musi być napisany w ścisłej zgodzie z najsurowszymi zasadami bezpieczeństwa. W wielu miejscach mówiliśmy już, jak ważne jest bezpieczeństwo. Zapoznałeś się już z wieloma przykładami, również tutaj nie pomijamy tej kwestii, aby uprościć sobie życie.

W pierwszej części skryptu deklarujemy zmienne do obsługi bazy danych, które będą zawierać informacje dotyczące połączenia. Bardzo często zdarza się, że w bardziej rozbudowanych projektach istnieje osobny plik ze zmiennymi konfiguracyjnymi. Taki plik powinien być dołączany na samym początku, aby można było się do niego odwoływać w dalszej części kodu.

Jednak, jako że ta aplikacja jest dość mała, zadeklarujemy zmienną reprezentującą połączenie już w pliku *dbconn.php*.

```
$host = "localhost";
$user = "użytkownik";
$pass = "hasło";
$database = "baza_danych";
```

Pierwsza zmienna często przyjmuje wartość `localhost`, ale może zawierać adres IP zdalnego serwera, jeżeli zdarzy się, że baza danych została uruchomiona na innej maszynie niż serwer WWW z PHP. W mniejszych systemach raczej nie spotkasz się ze zdalną instalacją MySQL-a, ale w przypadku większych aplikacji zdarza się to bardzo często.

Trzy kolejne zmienne reprezentują nazwę użytkownika, hasło oraz nazwę bazy danych, z którą mamy się połączyć. Jeśli jeszcze nie posiadasz wymaganych informacji, to może Ci ich udzielić administrator systemu lub zdalnego serwera.

MySQL w systemie Windows domyślnie instaluje się, tworząc konto **root**. W instalacji w systemie UNIX, którą opisywałem w rozdziale 1., użytkownikiem jest mysql, w przypadku instalacji z paczek sytuacja może wyglądać podobnie. W każdym przypadku konto niezabezpieczone hasłem stwarza duże zagrożenie, zatem hasło należy niezwłocznie nadać.

Po zdefiniowaniu odpowiednich zmiennych możemy połączyć się z MySQL-em. W PHP dokonujemy tego za pomocą funkcji `mysql_connect()`, przyjmującej trzy argumenty: nazwę serwera, nazwę użytkownika oraz hasło, które właśnie zdefiniowaliśmy:

```
$link = mysql_connect($host, $user, $pass);
```

Funkcja `mysql_connect()` zwraca identyfikator zasobu, który będziemy przechowywać w zmiennej `$link`. Odwołujemy się do niej już przy wyborze bazy danych.

Wybór bazy danych polega po prostu na podaniu nazwy bazy, z którą chcemy się połączyć, i przekazaniu identyfikatora połączenia, otrzymanego podczas nawiązywania komunikacji z serwerem:

```
mysql_select_db($database, $link);
```

Ostatnią, niemniej bardzo istotną czynnością jest usunięcie zmiennych zawierających informacje o połączeniu z bazą MySQL. Niszczymy czy usuwamy zmienną, podając ją jako parametr funkcji `unset()`, która usuwa zmienną z pamięci:

```
unset($host);
unset($user);
unset($pass);
```

```
unset($database);
unset($link);
```

Jest to czynność bardzo istotna, bo dzięki niej w dalszej części aplikacji nie będzie można się do nich odwołać. Jest to ważne zwłaszcza wówczas, gdy w ramach aplikacji korzystamy z oprogramowania napisanego przez innych.

Lepszą metodą na osiągnięcie opisanego celu jest umieszczenie całości kodu w klasie. Wówczas praca z nią przypomina pracę z zamkniętym składnikiem we Flashu. Dzięki temu dostępne jest tylko to, co ma być dostępne; reszta pozostaje ukryta.

Poniżej znajduje się przykład implementacji połączenia z bazą danych z zastosowaniem klasy:

```
<?php

// Prosta klasa reprezentująca połączenie z bazą danych MySQL

class MysqlConnection
{

   public $link;

   private $host = "localhost";
   private $user = "użytkownik";
   private $pass = "hasło";
   private $database = "baza_danych";

   function mysqlConnect() {}

   public function connect()
   {
      $this->link = mysql_connect(
        $this->host,
        $this->user,
        $this->pass
      );
      mysql_select_db($this->database, $this->link);
   }

   public function setConnectionDetails($h='', $u='', $p='', $d='')
   {
      $this->host = $h;
      $this->user = $u;
      $this->pass = $p;
      $this->database = $d;
   }

   public function getLink()
   {
      return $this->link;
   }
}

$sql = new MysqlConnection();
$sql->connect();

?>
```

W pierwszej chwili wydaje się, że ten kod niewiele odbiega od wcześniejszego przykładu, jednak niezwykle istotną różnicą jest sposób deklaracji zmiennych:

```
private $host = "localhost";
private $user = "użytkownik";
private $pass = "hasło";
private $database = "baza_danych";
```

Jak dowiedzieliśmy się w rozdziale 10., właściwości klasy w PHP mogą być dostępne publicznie lub prywatnie. W naszym przykładzie zmienne z informacjami na temat połączenia deklarujemy jako prywatne, co blokuje dostęp do nich spoza klasy. Dzięki temu nie będzie możliwe ich przypadkowe ujawnienie, a dodatkowo z takiego podejścia płynie jeszcze jedna korzyść. Powiedzmy, że piszemy nowy projekt i chcemy połączyć się z bazą danych. Dokonujemy tego bardzo prosto, jak widać poniżej:

```
<?php

include 'MySQLConnection.php';

$mysqlConn = new MySQLConnection();
$mysqlConn->setConnectionDetails('serwer','użytkownik','hasło','baza_danych');
$mysqlConn->connect();

$query = 'SEKECT * FROM tabela';
$result = mysql_query($query, $mysqlConn->getLink());

?>
```

Zauważ, że korzystamy z własnej klasy realizującej połączenie. Podajemy szczegóły dotyczące połączenia, a następnie przekazujemy ciąg znaków z zapytaniem do funkcji `mysql_query()`. W kodzie nie są dostępne informacje na temat połączenia, zatem nikt ich nie pozna.

Podczas łączenia się z bazą danych na aktywnym serwerze dobrze jest wyłączyć wyświetlanie komunikatów o błędach, lub przynajmniej zawiesić ich wyświetlanie.

Poniżej znajduje się kod oryginalnego pliku z połączeniem, zamieszczony w całości:

```
<?php

$host = "localhost";
$user = "użytkownik";
$pass = "hasło";
$database = "baza_danych";

$link = mysql_connect($host, $user, $pass);
mysql_select_db($database, $link);

unset($host);
unset($user);
unset($pass);
unset($database);
unset($link);

?>
```

Tworzymy odpowiednią tabelę w bazie danych

Skrypty w PHP i w ActionScripcie są już gotowe. Ale gdybyśmy teraz spróbowali przetestować aplikację, to nie będzie działać, ponieważ nie utworzyliśmy tabeli SQL, do której odwołuje się skrypt PHP w celu wysyłania i pobierania wiadomości.

Składnia zapytania SQL nie jest trudna, jednak samo zapytanie należy tworzyć bardzo ostrożnie. Szybkość korzystania ze źle utworzonej tabeli będzie systematycznie spadać, kiedy zacznie wzrastać ilość przechowywanych w niej danych. W zapytaniu SQL tworzymy tabelę `flashChat` i dodajemy wiersze, z których będziemy korzystać w PHP. Przyjrzyj się polu `id`, do którego nie odwołujemy się ze skryptu PHP; jest ono wykorzystywane wewnątrz tabeli do indeksowania i przypisywania kluczy. O polu `id` możemy również myśleć jako o kluczu do tajemnicy, która spowija miejsce przechowywania konkretnych informacji w tej ogromnej tabeli.

Kolumny, z których korzystamy w PHP, to `username` (nazwa użytkownika), `message` (wiadomość) oraz `dateAdded` (data dodania). Pole `message` jest najważniejsze, ponieważ zostało zdefiniowane jako `TEXT`, co oznacza, że jego długość jest dowolna. W związku z tym wiadomość może być w zasadzie dowolnej długości. Pole to moglibyśmy zdefiniować jako `varchar` — co określiłoby maksymalną długość — na przykład w taki sposób:

```
message varchar(150) NOT NULL default ''
```

Takie zdefiniowanie kolumny `message` sprawiłoby, że wiadomości z czata dłuższe niż 150 znaków zostałyby ucięte lub skrócone. W podobnych sytuacjach często korzysta się z typu TEXT, jednak może to mieć negatywny wpływ na wydajność, gdy tabela zrobi się większa.

```
CREATE TABLE flashChat (
   id int(11) not null auto_increment,
   username varchar(20) NOT NULL default '',
   message text NOT NULL,
   dateAdded int(11) NOT NULL default 0,
   PRIMARY KEY (id)
) ENGINE=MyISAM;
```

Musieliśmy napisać wiele kodu, ale nasz flashowy czat korzystający z PHP jest już gotowy. Poświęć czas na dokładne przyjrzenie się kodowi naszej aplikacji i dodanie do niej nowych opcji. Poniżej przedstawiam kilka pomysłów.

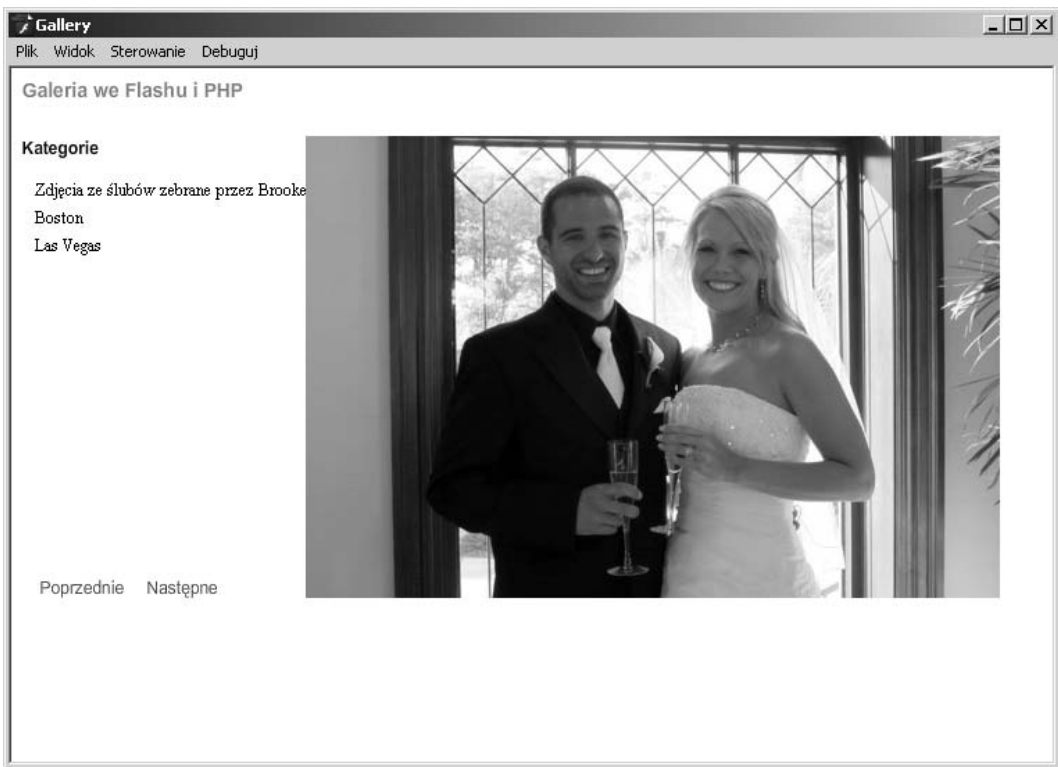
Pierwszą i najbardziej oczywistą rzeczą, jaką można dodać, jest zwiększenie poziomu bezpieczeństwa komunikacji między częścią aplikacji we Flashu i w PHP. Możesz też dodać panel moderatora lub prosty skrypt moderujący. Taki skrypt stanowi dodatek do niniejszej książki.

Powinieneś już dość dobrze wiedzieć, jak tworzyć kompletne aplikacje we Flashu, PHP i MySQL-u. W następnym podrozdziale napiszemy we Flashu i PHP w pełni działającą galerię fotograficzną z kategoriami i panelem nawigacyjnym.

Tworzymy galerię fotograficzną, korzystającą ze skryptu PHP

Czy jest coś fajniejszego niż galeria we Flashu? A co z galerią dynamiczną, w której skrypt w PHP wciąż aktualizuje odpowiednie pliki XML? W tym podrozdziale opiszę, jak krok po kroku napisać właśnie taką galerię. Zaczniemy od ActionScripta, a potem przejdziemy do części w PHP. Na koniec ocenimy aplikację i zastanowimy się nad możliwymi kierunkami jej rozwoju.

Jak dowiedzieliśmy się na początku tego rozdziału, sposobem na utworzenie dobrej aplikacji jest rozpoczęcie od projektu i oceny jeszcze przed przystąpieniem do programowania. Poświęć chwilę na przyjrzenie się zrzutowi przedstawiającemu gotową aplikację, która jest dostępna w kodach źródłowych do książki, dostępnych na serwerze FTP. Na rysunku 11.4 widzimy gotową aplikację.



Rysunek 11.4. *Ukończona aplikacja we Flashu i PHP wraz z załadowaną zawartością*

Gotowy skrypt we Flashu automatycznie wypełni listę kategorii, będzie dynamicznie ładował obrazy i umożliwi oglądanie poprzedniego i następnego obrazka.

Piszemy kod w ActionScripcie

Gdy już wiemy, co będzie robić aplikacja, możemy zacząć definiować odpowiednie zmienne:

```
var phpPath:String = "http://localhost/helion/rozdzial11/photoGallery/";
var phpFile:String = phpPath + "gallery.php";

var images:Array = new Array();

var imageHolder:MovieClip;
var categoryHolder:MovieClip;
```

Pierwsze dwie zmienne określają skrypt w PHP, generujący informacje o kategoriach i obrazach. Zmienna `images` służy do przechowywania zwracanych przez skrypt PHP informacji o obrazie, z których korzystamy, ładując obraz. Ostatnie dwie zmienne w tym fragmencie przechowują klipy filmowe głównego obrazu oraz panelu nawigacyjnego. Obie zmienne otrzymują wartości w trakcie działania aplikacji, po pobraniu informacji na temat obrazów i kategorii.

Kolejny zestaw zmiennych jest potrzebny zwłaszcza do poruszania się pomiędzy obrazami i kategoriami:

```
var currentID:uint;
var imageDir:String = "photos/";

var currentImage:uint = 0;
var cacheBuster:String = "?cb=1";
```

Zmienna `currentId` przechowuje identyfikator właśnie oglądanego obrazu. Będziemy z niej korzystać w funkcjach przenoszących do poprzedniego i następnego zdjęcia. Zmienna `imageDir` określa katalog, w którym znajdują się katalogi reprezentujące poszczególne kategorie. Ostatnia zmienna to niszczyciel pamięci podręcznej, dzięki któremu — jak wyjaśniłem na początku rozdziału — pobierane są zawsze aktualne dane.

Po zdefiniowaniu wszystkich koniecznych zmiennych możemy przejść do serca aplikacji, które stanowią funkcje.

Funkcja `init()` tworzy dwa klipy filmowe. Klipy filmowe tworzymy, pozycjonujemy oraz dołączamy do listy wyświetlania w sposób dynamiczny. Osiągamy to, korzystając z metody `addChild()` i przekazując referencję do klipów. Również w funkcji `init()` generujemy łańcuch pozwalający pominąć pamięć podręczną. Na koniec wywołujemy funkcję `load`
`↪Categories()`. Funkcję tę należy wywołać tylko na początku działania aplikacji, bo później zarówno obiekty, jak i kategorie mogą nie istnieć lub zostać zduplikowane.

```
function init()
{
   imageHolder = new MovieClip();
   imageHolder.x = 212;
   imageHolder.y = 49;
   addChild(imageHolder);

   categoryHolder = new MovieClip();
   categoryHolder.x = 15;
   categoryHolder.y = 50;
```

```
    addChild(categoryHolder);

    cacheBuster = getCacheBuster();

    loadCategories();
}
```

Na zakończenie inicjalizacji w funkcji `init()` wywołujemy funkcję `loadCategories()`. Funkcja ta za pośrednictwem obiektu klasy `URLRequest`, po odpowiednim ustawieniu pola `action`, które mówi skryptowi na serwerze, że ma podać listę kategorii, wywołuje skrypt PHP. Jest to ważne, ponieważ skrypt *gallery.php* obsługuje zarówno kategorie, jak i wybór obrazu. W zasadzie funkcja `loadCategories()` bardzo przypomina pozostałe funkcje ładujące w tej książce.

```
function loadCategories():void
{
    var action:String = "action=cat";
    var urlRequest:URLRequest = new URLRequest(phpFile + getCacheBuster() + "&" + action);

    var urlLoader:URLLoader = new URLLoader();
    urlLoader.addEventListener(Event.COMPLETE, drawCategories);
    urlLoader.load(urlRequest);
}
```

Zaraz po przysłaniu odpowiedzi przez serwer wywołujemy funkcję `drawCategories()`. Dane pobieramy w postaci obiektu XML i przetwarzamy wewnątrz pętli `for each`. Aby lepiej zrozumieć działanie pętli, spójrzmy na prosty wynik, zwrócony przez skrypt w PHP:

```
<category id="2" name="Krajobrazy" copyright="inny" />
```

Tekst kategorii, wyświetlany w panelu `Stage`, to pole tekstowe utworzone dynamicznie wewnątrz pętli `for each`. Moglibyśmy dołączyć klip filmowy z biblioteki, jednak takie podejście sprawiłoby, że aplikacja byłaby bardziej rozbita na części, i pozbawiłoby nas niektórych możliwości formatowania danych.

```
function drawCategories(e:Event):void
{
    ...
    for each(var item in xml..category)
    {
        ...
    }
}
```

Zanim przejdziemy do kodu realizującego zadania specyficzne dla naszej aplikacji, przyjrzyjmy się, w jaki sposób tworzone jest pole tekstowe:

```
var txt:TextField = new TextField();
txt.selectable = false;
txt.width = 200;
txt.text = "Przykładowy tekst";
```

W pierwszym wierszu tworzymy obiekt klasy `TextField`, przypisując go zmiennej `txt`. W następnej linii upewniamy się, że tekstu nie będzie można zaznaczyć za pomocą myszki.

Nie zawsze warto ustawiać właściwość `selectable` na `false`. Użytkownicy często chcą skopiować treść tekstu, zwłaszcza gdy jest on długi.

We fragmencie kodu po pierwszej linijce ustawiamy szerokość pola tekstowego na 200 pikseli, aby dostosować ją do długości tekstu. W ostatnim wierszu po prostu przypisujemy tekst, który będzie widoczny w tym polu.

Po utworzeniu pola tekstowego dodajemy odbiornik zdarzeń, który powoduje pobranie listy kategorii po naciśnięciu tego pola.

Funkcje anonimowe

Bezpośrednio do wywołania metody `addEventListener()` dołączamy funkcję anonimową. Do funkcji anonimowej nie da się odwołać poprzez nazwę, ponieważ ona po prostu jej nie ma. Funkcje anonimowe stosuje się zamiast zwyczajnych funkcji w sytuacji, gdy wykonywane przez nią zadanie jest proste i nie wymaga dużej ilości kodu. Patrząc realistycznie, funkcje te stosuje się, aby kod był zwięźlejszy, lub jeśli chcemy mieć dostęp do zmiennej, której zasięg ogranicza się do wywoływanej metody.

Niżej przedstawiam przykład funkcji anonimowej, podobnej do funkcji umieszczonej wewnątrz funkcji `drawCategories()`:

```
txtContainer.addEventListener(MouseEvent.CLICK,
  function(e:Event):void
  {
     trace("Tu funkcja anonimowa, nie mam nazwy");
  }
);
```

Może dostrzegasz już jeden z potencjalnych powodów niestosowania funkcji anonimowych (poza tym, że nie można z nich korzystać w innych funkcjach) — zdecydowanie obniżają one czytelność kodu. Jest tak dlatego, że są one ukryte głęboko w wywołaniu — w naszym wypadku metody `addEventListener()`. Poza tym funkcji anonimowej nie można usunąć, co może doprowadzić do wycieków pamięci.

Ostatnim zadaniem, jakie spełnia funkcja `drawCategories()`, jest dodanie pola tekstowego do obiektu `Stage` za pomocą metody `addChild()`; to samo uczynimy w przypadku kategorii:

```
txtContainer.addChild(txt);
categoryHolder.addChild(txtContainer);
```

Niżej przedstawiam kod funkcji `drawCategories()` w całości, abyś łatwiej mógł się w nim odnaleźć:

```
function drawCategories(e:Event):void
{
   var loader:URLLoader = URLLoader(e.target);
   var xml:XML = new XML(loader.data);

   for each(var item in xml..category)
   {
      var txtContainer:MovieClip = new MovieClip();
      var txt:TextField = new TextField();
```

```
        txt.selectable = false;
        txt.width = 200;
        txt.text = item.attribute('name');
        txt.y = uint(item.attribute('id') + 4) * 2;
        txt.name = "text_" + item.attribute('id');
        txtContainer.addEventListener(MouseEvent.CLICK,
          function(e:Event):void
          {
             loadImages(e.target.name.substring(5));
          }
        );
        txtContainer.addChild(txt);
        categoryHolder.addChild(txtContainer);

    }
}
```

Następną funkcją, którą się zajmiemy, jest `loadImages()`. Jej zadaniem jest pobieranie informacji o obrazach ze skryptu PHP. Informacje zwracane tu przez serwer są prawie identyczne jak w przypadku funkcji obsługującej kategorie. Zmiennej `action` zostaje przypisany ciąg `photos`, definiujemy również zmienną `id`, aby skrypt wiedział, który obraz ma pobrać.

```
function loadImages(id:uint):void
{
   var action:String = "action=photos&id=" + id;
   var urlRequest:URLRequest = new URLRequest(phpFile + getCacheBuster() + "&" + action);
   var urlLoader:URLLoader = new URLLoader();
   urlLoader.addEventListener(Event.COMPLETE, imagesLoaded);
   urlLoader.load(urlRequest);
   currentID = id;
}
```

Po wywołaniu pliku PHP odsyłana jest odpowiedź, a wówczas następuje odwołanie do funkcji `imagesLoaded()`. Informacje o obrazie odsyłane są w postaci dokumentu XML, a następnie przetwarzane w pętli `for each`.

Wewnątrz pętli przetwarzamy każdy element `photo` dokumentu XML, a następnie tworzymy obiekt, który dodajemy, „dokładamy” do tablicy `images`:

```
function imagesLoaded(e:Event):void
{
   ...
   for each(var item in xml..photo)
   {
      images.push({name:'', src:item.attribute('src')});
   }
   ...
}
```

A oto definicja obiektu; dalej przedstawiam jej bardziej czytelną wersję:

```
{ name:'', src:itemattribute('src') }
```

A to alternatywny sposób deklarowania obiektu:

```
var obj:Object = new Object();
obj.name = '';
obj.src = itemattribute('src');
```

Ostatnim zadaniem funkcji `imagesLoaded()` jest nadanie odpowiedniej wartości zmiennej `currentImage` oraz wywołanie funkcji `displayImage()` wraz z przekazaniem jej argumentów ścieżki do obrazu. Ścieżkę do obrazu pobieramy z tablicy `images`, podając wartość zmiennej `currentImage` jako indeks.

```
function imagesLoaded(e:Event):void
{
   ...
   currentImage = 0;
   displayImage(images[currentImage].src);
}
```

Poniżej przedstawiam kod funkcji `loadedImages()` w całości:

```
function imagesLoaded(e:Event):void
{
   var loader:URLLoader = URLLoader(e.target);
   var xml:XML = new XML(loader.data);
   images = new Array();
   for each(var item in xml..photo)
   {
      images.push({name:'', src:item.attribute('src')});
   }
   currentImage = 0;
   displayImage(images[currentImage].src);
}
```

Po załadowaniu kategorii i obrazów możemy wyświetlić zdjęcie. Robimy to, tworząc na podstawie ścieżki do obrazu, identyfikatora bieżącej kategorii oraz nazwy zdjęcia obiekt klasy `URLRequest`. Obiekt ładujący tworzymy bezpośrednio wewnątrz wywołania `addChild()`, które troszczy się o wyświetlenie obrazu po całkowitym załadowaniu. Na komputerze lokalnym nie zauważysz, że ładowanie trwa, ale w sieci upływ czasu może być dostrzegalny.

```
function displayImage(src:String):void
{
   var loader:Loader = new Loader();
   loader.load(new URLRequest(imageDir + currentID + "/" + src));
   imageHolder.addChild(loader);
}
```

Dobrym zwyczajem jest stosowanie preloaderów, gdy mamy do czynienia z ładowaniem danych. Dzięki temu użytkownik wie, że coś się dzieje.

Poruszanie się po galerii fotograficznej

Część odpowiadającą za poruszanie się w naszej aplikacji zbudujemy przy użyciu dwóch klipów filmowych, umieszczonych w panelu `Stage`. Każdemu klipowi przyporządkowujemy odbiornik zdarzeń, wywołujący procedurę obsługi ładującą poprzedni lub następny obraz.

Poruszanie się między obrazami

Jak z pewnością zauważyłeś, funkcja `nextImage()` zawiera kilka sprawdzeń. Dzięki nim wiemy, czy wartość `currentImage` nie jest większa od całkowitej liczby obrazów, co doprowadziłoby do błędu podczas ładowania. To samo sprawdzenie przeprowadzamy w funkcji `prevImage()`, oczywiście z tą różnicą, że badamy, czy wartość `currentImage` nie jest mniejsza od zera:

```
function nextImage(e:MouseEvent):void
{
   currentImage++;
   if (currentImage > images.length-1)
   {
      currentImage = 0;
   }
   displayImage(images[currentImage].src);
}

function prevImage(e:MouseEvent):void
{
   currentImage--;
   if (currentImage <= 0)
   {
      currentImage = images.length-1;
   }
   displayImage(images[currentImage].src);
}
```

Funkcje wyświetlające poprzedni i następny obraz spowodują błąd, jeżeli nie wybrano kategorii.

Ostatnia funkcja kodu w ActionScripcie generuje niszczyciela pamięci podręcznej, dzięki któremu mamy pewność, że odwołania do serwera nie są w niej przechowywane. Funkcja tworząca niszczyciela jest taka sama, jak w czacie we Flashu, który napisaliśmy w tym rozdziale.

W końcowym fragmencie kodu aplikacji wywołujemy funkcję `init()`, która wykonuje się na samym początku, a następnie dodajemy odbiorniki zdarzeń, wywołujące odpowiednie funkcje, do przycisków nawigacyjnych:

```
function getCacheBuster():String
{
   var date:Date = new Date();
   cacheBuster = "?cb=" + date.getTime();
   return cacheBuster;
}

init();

prevMC.addEventListener(MouseEvent.CLICK, prevImage);
nextMC.addEventListener(MouseEvent.CLICK, nextImage);
```

Poniżej znajduje się kompletny kod galerii fotograficznej w ActionScripcie:

```
var phpPath:String = "http://localhost/helion/rozdzial11/photoGallery/";
var phpFile:String = phpPath + "gallery.php";

var images:Array = new Array();

var imageHolder:MovieClip;
var categoryHolder:MovieClip;

var currentID:uint;
var imageDir:String = "photos/";

var currentImage:uint = 0;

var cacheBuster:String = "?cb=1";

function init()
{
   imageHolder = new MovieClip();
   imageHolder.x = 212;
   imageHolder.y = 49;
   addChild(imageHolder);

   categoryHolder = new MovieClip();
   categoryHolder.x = 15;
   categoryHolder.y = 50;
   addChild(categoryHolder);

   cacheBuster = getCacheBuster();

   loadCategories();
}

function loadCategories():void
{
   var action:String = "action=cat";
   var urlRequest:URLRequest = new URLRequest(phpFile + getCacheBuster() + "&" + action);

   var urlLoader:URLLoader = new URLLoader();
   urlLoader.addEventListener(Event.COMPLETE, drawCategories);
   urlLoader.load(urlRequest);
}

function drawCategories(e:Event):void
{
   var loader:URLLoader = URLLoader(e.target);
   var xml:XML = new XML(loader.data);

   for each(var item in xml..category)
   {
      var txtContainer:MovieClip = new MovieClip();

      var txt:TextField = new TextField();
      txt.selectable = false;
      txt.width = 200;
      txt.text = item.attribute('name');
      txt.y = uint(item.attribute('id') + 4) * 2;
```

```
            txt.name = "text_" + item.attribute('id');
            txtContainer.addEventListener(MouseEvent.CLICK,
              function(e:Event):void
              {
                 loadImages(e.target.name.substring(5));
              }
            );

            txtContainer.buttonMode = true;

            txtContainer.addChild(txt);
            categoryHolder.addChild(txtContainer);
         }
      }

      function loadImages(id:uint):void
      {
         trace("Ładowanie obrazów: " + id);
         var action:String = "action=photos&id=" + id;
         var urlRequest:URLRequest = new URLRequest(phpFile + getCacheBuster() + "&" + action);

         var urlLoader:URLLoader = new URLLoader();
         urlLoader.addEventListener(Event.COMPLETE, imagesLoaded);
         urlLoader.load(urlRequest);
         currentID = id;
      }

      function imagesLoaded(e:Event):void
      {
         var loader:URLLoader = URLLoader(e.target);
         var xml:XML = new XML(loader.data);
         images = new Array();

         for each(var item in xml..photo)
         {
            images.push({name:'', src:item.attribute('src')});
         }

         currentImage = 0;
         displayImage(images[currentImage].src);
      }

      function displayImage(src:String):void
      {
         trace("Ładowanie obrazu: " + src);

         var loader:Loader = new Loader();
         loader.load(new URLRequest(imageDir + currentID + "/" + src));
         imageHolder.addChild(loader);
      }

      function nextImage(e:MouseEvent):void
      {
         currentImage++;
         if (currentImage > images.length-1)
         {
            currentImage = 0;
```

```
    }
    displayImage(images[currentImage].src);
}

function prevImage(e:MouseEvent):void
{
    currentImage--;
    if (currentImage <= 0)
    {
        currentImage = images.length-1;
    }
    displayImage(images[currentImage].src);
}

function getCacheBuster():String
{
    var date:Date = new Date();
    cacheBuster = "?cb=" + date.getTime();
    return cacheBuster;
}

init();

prevMC.addEventListener(MouseEvent.CLICK, prevImage);
nextMC.addEventListener(MouseEvent.CLICK, nextImage);
```

Skrypty galerii fotograficznej w PHP

Kod PHP naszej aplikacji umieściliśmy w trzech plikach. Pierwszym z nich jest *categories.php*, w którym znajduje się statyczna reprezentacja kategorii wysyłanych do Flasha.

Pierwsza część kodu opisuje kategorie za pomocą tablicy wielowymiarowej. Pojedyncza kategoria składa się z nazwy (`name`), identyfikatora (`id`) oraz informacji o prawach autorskich (`copyright`):

```
$categories = array(
  array("Boston", 1, "M. Keefe"),
  array("Krajobrazy", 2, "Ktoś inny"),
  array("Las Vegas", 3, "M. Keefe"),
  array("Śluby", 4, "Ktoś inny")
);
```

Na początku funkcji `getCategories()` tworzymy zmienną globalną `$categories`. Następnie tworzymy pętlę, w której konstruujemy dokument XML z informacjami, które zostaną zwrócone do Flasha:

```
function getCategories()
{
    global $categories;

    $xml = "<categories>\n";

    for ($i=0; $i < count($categories); $i++)
    {
        $xml .= "<category id=\"" .
        $categories[$i][1] . "\" name=\"" .
```

```
      $categories[$i][0] . "\" copyright=\"" .
      $categories[$i][2] . "\" />\n";
   }

   $xml .= "</categories>";

   return $xml;
}
```

Liczba przebiegów pętli zależy od długości tablicy `$categories`:

```
count($categories)
```

Wewnątrz pętli znajduje się tylko kod tworzący dokument XML, podobnie jak w przypadku odpowiadającego mu kodu w ActionScripcie, któremu już miałeś okazję się przyjrzeć:

```
$xml .= "   <category id=\"" .
  $categories[$i][1] . "\" name=\"" .
  $categories[$i][0] . "\" copyright=\"" .
  $categories[$i][2] . "\" />\n";
```

Ostatnią czynnością wykonywaną w tej funkcji jest zwrócenie danych w postaci dokumentu XML do części aplikacji we Flashu:

```
return $xml;
```

Poniżej przedstawiam kod pliku *categories.php* w całości:

```
<?php

$categories = array(
  array("Boston", 1, "M. Keefe"),
  array("Krajobrazy", 2, "Ktoś inny"),
  array("Las Vegas", 3, "M. Keefe"),
  array("Śluby", 4, "Ktoś inny")
);

function getCategories()
{
   global $categories;

   $xml = "<categories>\n";

   for ($i=0; $i < count($categories); $i++)
   {
      $xml .= "<category id=\"" .
      $categories[$i][1] . "\" name=\"" .
      $categories[$i][0] . "\" copyright=\"" .
      $categories[$i][2] . "\" />\n";
   }

   $xml .= "</categories>";

   return $xml;
}

?>
```

Kolejnym plikiem do napisania jest *getPhotos.php*, którego zadaniem jest otwarcie katalogu ze zdjęciami i zwrócenie dokumentu w formacie XML ze ścieżkami do poszczególnych obrazów.

Sercem tego skryptu jest funkcja `getPhotosFromID()`, przyjmująca jeden argument — identyfikator zdjęcia. Zanim przejdziemy dalej, upewnijmy się, że otrzymaliśmy prawidłowy identyfikator. W tym celu wprowadźmy proste wyrażenie warunkowe. Jeżeli identyfikator jest poprawny, możemy przejść do otwarcia katalogu, a następnie do wnętrza pętli `while`.

```
?php

$photo_dir = "photos/";

function getPhotosFromID($id=null)
{
   global $photo_dir;

   if ($id == null)
   {
      print "Nie podano identyfikatora";
      return false;
   }

   $xml = "<photos id=\"" . $id . "\">";

   $dir = opendir($photo_dir . $id);
   while (false !== ($file = readdir($dir)))
   {
      if ($file != "." && $file != ".." && $file != ".DS_Store")
      {
         $xml .= "<photo name=\"" . "" . "\" src=\"" . $file . "\" />\n";
      }
   }
   closedir($dir);

   $xml .= "</photos>";

   return $xml;
}

?>
```

W pętli `while` przechodzimy kolejno przez wszystkie pliki w katalogu aż do momentu, gdy wskaźnik do pliku będzie miał wartość `false`, co oznacza, że nie odnaleziono poprawnego pliku:

```
while (false !== ($file = readdir($dir)))
{
   ...
}
```

Wprowadzamy wyrażenie warunkowe, aby wykluczyć z listy nazwy . oraz .., które wskazują odpowiednio na katalog bieżący oraz katalog nadrzędny. Gdybyśmy nie zastosowali tego sprawdzenia, to w dokumencie XML pojawiłyby się co najmniej dwa błędne wpisy albo — co gorsza — mogłoby dojść do zawieszenia programu, ponieważ pętla `while` mogłaby się wykonywać w nieskończoność.

Po zebraniu wszystkich nazw plików w danym katalogu zamykamy go, zwalniając tym samym cenne zasoby. Ma to znaczenie szczególnie, gdy plik może być jednocześnie używany przez inny proces.

```
closedir($dir);
```

Ostatnią czynnością wykonywaną w tym skrypcie jest odesłanie dokumentu XML do części aplikacji napisanej w ActionScripcie w celu dalszego przetwarzania.

Po utworzeniu dokumentów obsługujących kategorie i pliki możemy zająć się skryptem *gallery.php*, który obsługuje odwołania z Flasha i odsyła poprawnie skonstruowany dokument XML, sporządzony na podstawie żądania:

```
<?php

include 'categories.php';
include 'getPhotos.php';

header('Content-type: text/xml');

if ($_GET['action'] == 'cat')
{
   print getCategories();
}
else if ($_GET['action'] == 'photos')
{
   print getPhotosFromID($_GET['id']);
}

?>
```

Skrypt rozpoczynamy, dołączając dwa poprzednio napisane pliki. Potem wykonujemy odwołanie do funkcji `header()`, aby wymusić zwrócenie poprawnie skonstruowanego dokumentu XML. Funkcji `header()` można używać do podania praktycznie dowolnego typu. Ustalamy nagłówek na samym początku i odtąd wynik jest zgodny z danym formatem. Na przykład moglibyśmy wyeksportować zawartość jako plik „.png”:

```
header("Content-type: image/png");
```

Musimy sprawdzić w naszej aplikacji, czy korzystamy z zawartości odpowiedniego rodzaju. Jeżeli typ jest nieprawidłowy, mogą pojawić się błędy, a czasem nawet aplikacja może się „wysypać”.

W końcowym bloku kodu określamy, jakiego rodzaju zawartość chcemy otrzymać. Mamy do dyspozycji dwie możliwości — listę kategorii oraz listę fotografii. Żądanie, zawarte w adresie URL, otrzymujemy w skrypcie PHP w zmiennej `$_GET['action']`:

```
http://localhost/helion/rozdzial11/photoGallery/gallery.php?cb=1192408716823&action=cat
```

W tym momencie galeria we Flashu z silnikiem w PHP jest już gotowa. Możesz oczywiście ją poszerzyć, dodając podkategorie, możliwość przenoszenia obrazów, a nawet tytuły i opisy do każdego zdjęcia.

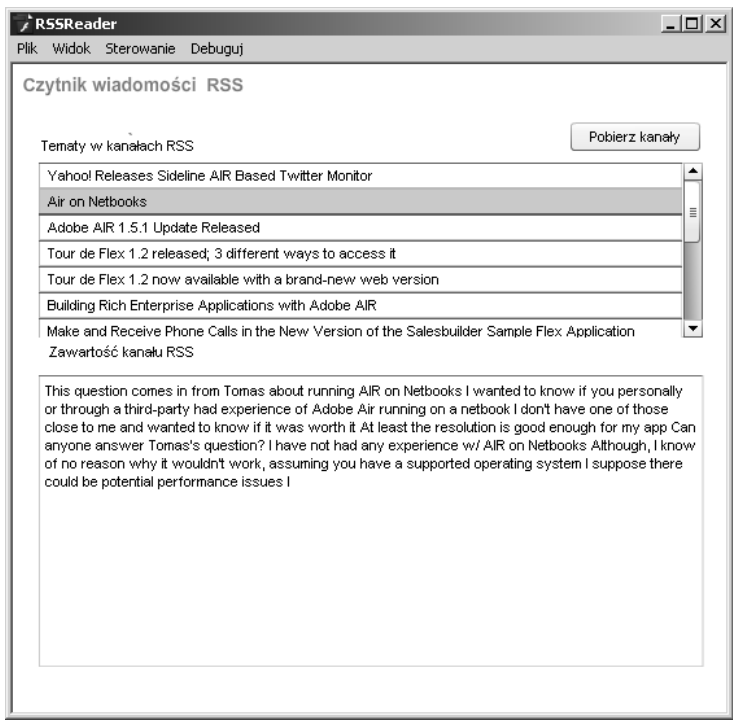
To dobra rzecz w ActionScripcie: możesz tworzyć nowe aplikacje na bazie przykładów z książki albo po prostu ich używać.

Korzystamy z PHP do napisania czytnika wiadomości RSS

Programy odczytujące wiadomości RSS są bardzo popularne i pisze się je praktycznie na wszelkie urządzenia, które mają możliwość wyświetlania stron WWW. Możesz znaleźć je wszędzie: w przeglądarce na biurku czy w telefonie w kieszeni.

RSS to zbiór kanałów WWW, służących do publikacji często zmienianych treści, takich jak wpisy, wiadomości, nagłówki, podkasty czy rozrywka. Można również myśleć o RSS jako o roznosicielu gazet, który przynosi Ci wiadomości codzienne; jedyna różnica polega na tym, że nie ma ograniczeń co do liczby kanałów, które możesz subskrybować.

Na rysunku 11.5 znajduje się widok czytnika wiadomości RSS, nad którym będziemy pracować.



Rysunek 11.5. *Czytnik wiadomości RSS z mechanizmem dostarczania wiadomości w PHP*

Aplikacja utworzona została na bazie wcześniej przygotowanych składników, umieszczonych w pliku startowym. Trzy elementy, którymi będziemy się zajmować, to składniki *List*, *TextArea* i *Submit*. Każdemu z nich przypisaliśmy nazwę obiektu, którą posługujemy się w kodzie.

Importujemy klasy

Większości klas dostarczonych z Flashem nie trzeba importować. Jednak istnieją pewne wyjątki, a jednym z nich jest klasa ListEvent:

```
import fl.events.ListEvent;
```

Po zaimportowaniu klasy ListEvent możemy zająć się deklaracją zmiennych. Jedyną zmienną globalną potrzebną w naszej aplikacji jest nazwa skryptu PHP:

```
var phpPath:String = "http://localhost/helion/rozdzial11/rssReader/";
var phpFile:String = phpPath + "rss.php";
```

Wywołujemy plik PHP

Funkcja, która odwołuje się do skryptu PHP zwracającego dokument XML, bardzo przypomina odpowiednie funkcje w poprzednich przykładach. Tworzymy obiekt klasy URLRequest oraz obiekt klasy URLLoader, nadajemy im odpowiednie wartości, a następnie dodajemy do tego ostatniego odbiornik zdarzeń, reagujący na zdarzenie COMPLETE:

```
function loadFeeds():void
{
   var urlRequest:URLRequest = new URLRequest(phpFile);
   var urlLoader:URLLoader = new URLLoader();
   urlLoader.addEventListener(Event.COMPLETE, feedHandler);
   urlLoader.load(urlRequest);
}
```

Funkcja feedHandler() przetwarza odpowiedź odesłaną przez skrypt PHP. W naszej aplikacji skrypt PHP zwraca dokument XML, na podstawie którego wypełniamy składnik klasy List. Wpisy RSS umieszczamy w składniku klasy List za pomocą metody addItem().

Funkcja feedHandler() przyjmuje jako argument obiekt. Aby można było go dodać, musi on mieć przynajmniej właściwość label, ale zazwyczaj będziesz dodawał również właściwość data:

```
function feedHandler(e:Event):void
{
   ...
   for each(var item in xml..entry)
   {
      topicsList.addItem({label:item..name, data:item..desc});

      topicsList.addEventListener(ListEvent.ITEM_CLICK, listClickhandler);
   }
}
```

Po kliknięciu na pozycję na liście ma zostać załadowana treść wiadomości, zatem musimy napisać funkcję obsługującą to zdarzenie. Zdarzenie typu ListEvent jest przekazywane z funkcji obsługi wydarzenia ITEM_CLICK, które generuje właściwość item. We właściwości item przechowujemy właściwość data. W naszym przykładzie dane stanowi treść wiadomości RSS, zatem możemy po prostu przekazać ją do składnika feedBody klasy TextArea.

```
function listClickhandler(e:ListEvent):void
{
  feedBody.htmlText = e.item.data;
}
```

Ostatnią funkcją w czytniku wiadomości RSS jest procedura obsługi naciśnięcia przycisku, wywoływana za każdym razem, gdy użytkownik naciśnie przycisk klasy `Button`. W funkcji wywołujemy po prostu funkcję `loadFeeds()`.

```
function submitHandler(e:Event):void
{
   loadFeeds();
}
```

Jak widzisz, skrypt Flasha jest bardzo prosty. Dzięki XML-owi aplikacje WWW rzeczywiście tworzy się szybciej, a nasz czytnik jest tego doskonałym przykładem.

Niżej przedstawiam kod w ActionScripcie w całości, abyś łatwiej mógł się w nim odnaleźć:

```
import fl.events.ListEvent;

var phpPath:String = "http://localhost/helion/rozdzial11/rssReader/";
var phpFile:String = phpPath + "rss.php";

function loadFeeds():void
{
   var urlRequest:URLRequest = new URLRequest(phpFile);
   var urlLoader:URLLoader = new URLLoader();
   urlLoader.addEventListener(Event.COMPLETE, feedHandler);
   urlLoader.load(urlRequest);
}

function feedHandler(e:Event):void
{
   var loader:URLLoader = URLLoader(e.target);
   var xml:XML = new XML(loader.data);

   for each(var item in xml..entry)
   {
      topicsList.addItem({label:item..name, data:item..desc});
      topicsList.addEventListener(ListEvent.ITEM_CLICK, listClickhandler);
   }
}

function listClickhandler(e:ListEvent):void
{
  feedBody.htmlText = e.item.data;
}

function submitHandler(e:Event):void
{
   loadFeeds();
}

loadBtn.addEventListener(MouseEvent.CLICK, submitHandler);
```

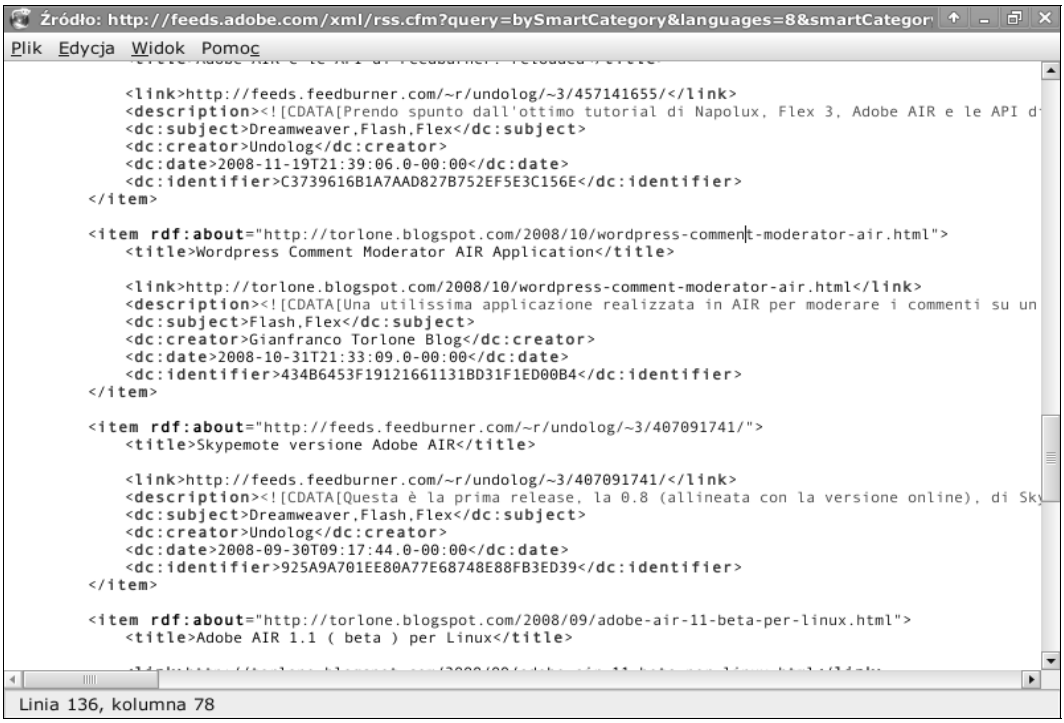
Po napisaniu kodu czytnika w ActionScripcie możemy przejść do kodu w PHP.

Kanał RSS, z którego korzystamy w naszym przykładzie — spójrz na rysunek 11.6 — znajduje się na stronie WWW firmy Adobe i zawiera najnowsze wiadomości oraz informacje o środowisku Adobe AIR:

```
<?php

$rssFeed = "http://weblogs.macromedia.com/mxna/xml/rss.cfm?" .
  "query=bySmartCategory&languages=1&smartCategoryId=28&" .
  "smartCategoryKey=F2DFD9E0-FBB6-4C2D-2AFE6AFD941FDDB1";

?>
```



Rysunek 11.6. *Tak może wyglądać kanał RSS w przeglądarce bez zainstalowanego czytnika*

Dokument XML utworzony po pomyślnym pobraniu kanału przechowujemy w zmiennej `$feed`. Kanał wczytujemy za pomocą biblioteki SimpleXML, udostępnianej wraz z PHP 5. Nie jest to jedyna biblioteka do przetwarzania dokumentów XML dla PHP, ale jest najwydajniejsza i najprostsza w użyciu.

```
$feed = "";
$xml = simplexml_load_file($rssFeed);
```

Teraz zajmiemy się pętlą `foreach`, w której utworzymy dokument XML; zostanie on odesłany do Flasha:

```
$feed .= "<items>\n";

foreach ($xml->item as $item)
{
   $desc = $item->description;

   $desc = preg_replace('/[...\[\]]/', '', $desc);
```

```
    $feed .= "   <entry>\n";
    $feed .= "      <name>" . $item->title . "</name>\n";
    $feed .= "      <desc><![CDATA[" . $desc ."]]></desc>\n";
    $feed .= "   </entry>\n";
}

$feed .= "</items>\n";
```

W pętli analizujemy każdy element dokumentu XML i przebiegamy każdy węzeł `item`.

Jak widzisz, opis przekazujemy do zmiennej `$desc`. Powód jest prosty. Opis należy oczyścić, zanim go zwrócimy. Oczyszczanie przeprowadzamy za pomocą funkcji `preg_replace()`, która na podstawie wyrażenia regularnego stanowiącego jej argument usunie niezacytowane lub niepoprawne znaki:

```
$desc = preg_replace('/[...\[\]]/', '', $desc);
```

W niniejszej książce wyrażenia regularne nie zostały szczegółowo opisane, jednak istnieje bardzo dobry przewodnik po tych wyrażeniach, który możesz znaleźć pod adresem: *http://pl.php.net/manual/pl/reference.pcre.pattern.syntax.php*.

W końcowym fragmencie skryptu PHP tworzymy nagłówek, a następnie zwracamy dokument XML do Flasha:

```
header('Content-type: text/xml');
print '<?xml version="1.0" encoding="UTF-8"?>' ."\n";
print $feed;
```

Jak widzisz, do napisania czytnika wiadomości RSS nie potrzebujemy pisać w PHP długiego kodu, a to dzięki temu, że biblioteka SimpleXML jest tak wspaniała. Przykład moglibyśmy rozszerzyć, dodając ściąganie większej ilości wiadomości z kanału RSS. Na przykład moglibyśmy wyświetlać tytuły poszczególnych wpisów, datę, a nawet adres URL oryginalnej wiadomości.

Poniżej znajduje się skrypt PHP w całości, abyś łatwiej mógł się w nim odnaleźć:

```
<?php

$rssFeed = "http://weblogs.macromedia.com/mxna/xml/rss.cfm?" .
  "query=bySmartCategory&languages=1&smartCategoryId=28&" .
  "smartCategoryKey=F2DFD9E0-FBB6-4C2D-2AFE6AFD941FDDB1";

$feed = "";
$xml = simplexml_load_file($rssFeed);

$feed .= "<items>\n";

foreach ($xml->item as $item)
{
   $desc = $item->description;

   $desc = preg_replace('/[...\[\]]/', '', $desc);

   $feed .= "   <entry>\n";
   $feed .= "      <name>" . $item->title . "</name>\n";
   $feed .= "      <desc><![CDATA[" . $desc ."]]></desc>\n";
```

```
    $feed .= "   </entry>\n";
}

$feed .= "</items>\n";

header('Content-type: text/xml');
print '<?xml version="1.0" encoding="UTF-8"?>' ."\n";
print $feed;

?>
```

Tworzymy dynamiczny baner przy użyciu PHP, Flasha i MySQL-a

Wielu projektantów tworzy ogłoszenia na strony internetowe, od drobnych ogłoszeń, znajdujących się gdzieś na stronie, aż po wielkie ogłoszenia, które same stanowią stronę WWW. Najczęstszą postacią tego rodzaju ogłoszeń jest baner, którego rozmiar to zazwyczaj 468×60 pikseli, jak na rysunku 11.7. Tego rodzaju banery są zwykle wyposażone w skrypt ładujący odpowiednią stronę internetową, gdy użytkownik na nie kliknie. A co, gdybyśmy mogli śledzić te kliknięcia? Albo jeszcze lepiej: Czemu nie zrobić dynamicznego banera, który wczytuje losową reklamę, i nie wymaga od właściciela wprowadzania zmian nigdzie indziej poza plikiem XML oraz katalogiem z obrazami?

Rysunek 11.7.
Baner w działaniu

W tym podrozdziale zajmiemy się właśnie tworzeniem dynamicznego banera reklamowego we Flashu, do którego potem za pomocą kilku linijek kodu w PHP dodamy możliwość śledzenia kliknięć. Przykład nie wymaga żadnych plików startowych, ponieważ baner będzie ładował dowolny obraz, a aplikacja powstanie w całości w ActionSrcipcie.

Na początku inicjalizujemy zmienne, z których korzystamy:

```
var phpPath:String = "http://localhost/helion/rozdzial11/bannerAd/";
var phpFile:String = phpPath + "ads.php";

var imageHolder:MovieClip;
var cacheBuster:String = "?cb=1";
var adURL:String;
```

Po zdefiniowaniu zmiennych możemy przystąpić do pisania funkcji. Pierwsza z nich tworzy miejsce na obrazek, dodaje do niego odbiornik zdarzeń i wywołuje funkcję `loadImage()`:

```
imageHolder = new MovieClip();
imageHolder.x = 0;
imageHolder.y = 0;
imageHolder.addEventListener(MouseEvent.CLICK, loadAdURL);
imageHolder.buttonMode = true;
```

```
addChild(imageHolder);

cacheBuster = getCacheBuster();

loadImage();
```

Funkcja `loadImage()` ładuje dokument XML, w którym znajdują się dane dotyczące ogłoszenia, a następnie przypisuje do ładowanego obiektu odpowiednią funkcję obsługi zdarzeń, wywoływaną, gdy dokument został już w całości pobrany:

```
function loadImage():void
{
   var urlRequest:URLRequest = new URLRequest(phpFile + getCacheBuster());
   var urlLoader:URLLoader = new URLLoader();
   urlLoader.addEventListener(Event.COMPLETE, imageLoaded);
   urlLoader.load(urlRequest);
}
```

Gdy dokument XML zostanie załadowany, następuje wywołanie funkcji `imageLoaded()`. Funkcja ta pobiera dane zawarte w dokumencie, wyłuskuje z nich informacje o obrazku, a w końcu go ładuje. Poniżej przedstawiam krótki opis poszczególnych czynności.

W ten sposób pobieramy dane i tworzymy obiekt klasy XML:

```
function imageLoaded(e:Event):void
{
   var urlLoader:URLLoader = URLLoader(e.target);
   var xml:XML = new XML(urlLoader.data);
   ...
```

W kolejnym fragmencie pobieramy informacje o obrazku i umieszczamy je w zmiennych lokalnych:

```
var url:String = xml..banner.attribute('url');
var name:String = xml..banner.attribute('name');
var image:String = xml..banner.attribute('src');
var directory:String = xml..banner.attribute('dir');

adURL = url;
```

Na koniec ładujemy obrazek i dodajemy go do listy wyświetlania:

```
var loader:Loader = new Loader();
loader.load(new URLRequest(directory + image));
imageHolder.addChild(loader);
```

Otwieramy okno przeglądarki

Właśnie napisaliśmy kod ładujący i wyświetlający obrazek. Teraz dodamy odbiornik zdarzeń, który — gdy użytkownik kliknie — wywoła odpowiednią funkcję. Do otwarcia w przeglądarce nowego okna ze wskazaną stroną używamy funkcji `navigateToURL()`:

```
function loadAdURL(e:MouseEvent):void
{
   navigateToURL(new URLRequest(adURL));
}
```

W ostatniej części skryptu znajduje się wywołanie funkcji `init()`, która go uruchamia:

```
init();
```

Poniżej znajduje się całość kodu w ActionScripcie, abyś łatwiej mógł się w nim odnaleźć:

```
var phpPath:String = "http://localhost/helion/rozdzial11/bannerAd/";
var phpFile:String = phpPath + "ads.php";

var imageHolder:MovieClip;
var cacheBuster:String = "?cb=1";
var adURL:String;

function init()
{
   imageHolder = new MovieClip();
   imageHolder.x = 0;
   imageHolder.y = 0;
   imageHolder.addEventListener(MouseEvent.CLICK, loadAdURL);
   imageHolder.buttonMode = true;
   addChild(imageHolder);

   cacheBuster = getCacheBuster();

   loadImage();
}

function loadImage():void
{
   var urlRequest:URLRequest = new URLRequest(phpFile + getCacheBuster());
   var urlLoader:URLLoader = new URLLoader();
   urlLoader.addEventListener(Event.COMPLETE, imageLoaded);
   urlLoader.load(urlRequest);
}

function imageLoaded(e:Event):void
{
   var urlLoader:URLLoader = URLLoader(e.target);
   var xml:XML = new XML(urlLoader.data);

   var url:String = xml..banner.attribute('url');
   var name:String = xml..banner.attribute('name');
   var image:String = xml..banner.attribute('src');
   var directory:String = xml..banner.attribute('dir');

   adURL = url;

   var loader:Loader = new Loader();
   loader.load(new URLRequest(directory + image));
   imageHolder.addChild(loader);
}

function loadAdURL(e:MouseEvent):void
{
   navigateToURL(new URLRequest(adURL));
}

function getCacheBuster():String
```

```
{
   var date:Date = new Date();
   cacheBuster = "?cb=" + date.getTime();
   return cacheBuster;
}

init();
```

Piszemy kod w PHP

Mamy już cały kod w ActionScripcie, przystąpmy zatem do pisania części w PHP. W pliku *ads.php* znajduje się definicja dwóch zmiennych globalnych oraz jedna funkcja.

Pierwsza zmienna globalna zawiera katalog, gdzie znajdują się obrazki z reklamami. Druga zmienna to tablica, zawierająca dane poszczególnych reklam:

```
$adImageDir = "./adImages/";
$bannerAds = array(array('Nazwa namera', 'randomimage1.jpg', 'http://localhost/'),
array('Nazwa namera', 'randomimage2.jpg', 'http://localhost/'),);
```

Funkcja `getBannerAd()` deklaruje obydwie zmienne jako globalne, są one zatem dostępne we wnętrzu funkcji cały czas.

Wybór losowy

Pojedyncze ogłoszenie jest wybierane z tablicy za pomocą wartości losowej. Wartość tę generujemy za pomocą funkcji `mt_rand()`; uwzględniamy przy tym długość tablicy:

```
$random = (mt_rand() % count($bannerAds));
```

Dokument XML generujemy, zwracając poszczególne wiersze, które zostaną przetworze przez część banera napisaną w ActionScripcie.

```
function getBannerAd()
{
   ...
   $random  = (mt_rand() % count($bannerAds));

   $xml .= "<banner id=\"" . 0 .
     "\" dir=\"" . $adImageDir .
     "\" url=\"" . $bannerAds[$random][2] .
     "\" name=\"" . $bannerAds[$random][0] .
     "\" src=\"" . $bannerAds[$random][1] . "\" />\n";

   $xml .= "</banners>";

   return $xml;
}

print getBannerAd();
```

Ukończyliśmy pisanie skryptu PHP, obsługującego ogłoszenia reklamowe. Jak widzisz, nie było trzeba wiele kodu, aby napisać tę aplikację. Ten prosty przykład łatwo poszerzyć, wprowadzając kategorie lub nawet grupy obrazków, które zmieniają się w trakcie wyświetlania filmu na stronie internetowej.

Poniżej przedstawiam kod w całości, abyś łatwo mógł się w nim odnaleźć:

```
<?php

$adImageDir = "./adImages/";

$bannerAds = array(
  array('Nazwa namera', 'randomimage1.jpg', 'http://localhost/'),
  array('Nazwa namera', 'randomimage2.jpg', 'http://localhost/'),
  array('Nazwa namera', 'randomimage3.jpg', 'http://localhost/'),
  array('Nazwa namera', 'randomimage4.jpg', 'http://localhost/'),
  array('Nazwa namera', 'randomimage5.jpg', 'http://localhost/'),
  array('Nazwa namera', 'randomimage6.jpg', 'http://localhost/'),
  array('Nazwa namera', 'randomimage7.jpg', 'http://localhost/'),
  array('Nazwa namera', 'randomimage8.jpg', 'http://localhost/')
);

function getBannerAd()
{
   global $bannerAds, $adImageDir;

   $xml = "<banners>\n";

   $random  = (mt_rand() % count($bannerAds));

   $xml .= "<banner id=\"" . 0 .
     "\" dir=\"" . $adImageDir .
     "\" url=\"" . $bannerAds[$random][2] .
     "\" name=\"" . $bannerAds[$random][0] .
     "\" src=\"" . $bannerAds[$random][1] . "\" />\n";

   $xml .= "</banners>";

   return $xml;
}

print getBannerAd();

?>
```

Właśnie napisałeś we Flashu i w PHP w pełni działającą przeglądarkę ogłoszeń. Techniki przyswojone w tym podrozdziale łatwo możesz wykorzystać we własnych projektach. Naprawdę gorąco zachęcam Cię, abyś poszerzył ten przykład i wyposażył go w większe możliwości.

Przykład ten można również uprościć, ładując statyczny dokument XML; jednak takie podejście utrudniłoby i ograniczyło możliwość wprowadzania zmian. Jako że aplikacja jest częściowo napisana w PHP, możesz dodać do niej warstwę obsługi bazy danych i zwracać do Flasha pobrane informacje o obrazach; zawarte w niej dane można by zmieniać za pomocą innej aplikacji.

Piszemy część licznika odwiedzin w PHP

Licznik odwiedzin stosuje się, aby ustalić, jaka liczba gości odwiedza stronę. Zazwyczaj licznik odwiedzin pozostaje widoczny dla gości w postaci tekstu lub grafiki. W niektórych witrynach wykorzystuje się inny sposób monitorowania dla celów statystycznych, a dane z monitoringu nie są publicznie dostępne. Ważną atrakcją, a zarazem możliwością licznika odwiedzin jest reprezentacja graficzna.

Do przechowywania danych licznika możemy użyć zwykłego pliku tekstowego albo bazy danych. W naszym przykładzie skorzystamy z bazy danych, a uczynimy tak ze względu na szybkość (baza danych przetwarza informacje znaczenie szybciej) oraz kwestie uprawnień do plików. Czasami serwery blokują pliki, co oznacza, że nie można ich otworzyć. To sprawiłoby, że nasz licznik przestałby działać, a przecież nie tego oczekujemy.

Mechanizm licznika odwiedzin

Mechanizm kryjący się za licznikiem odwiedzin jest dość prosty. Najpierw z bazy danych pobieramy bieżącą wartość licznika i dodajemy do niej jeden:

```
$oldcount = $row['amount'];
$newCount = $oldCount + 1;
```

Po uzyskaniu nowej wartości wpisujemy ją z powrotem do tabeli w bazie danych. Czynimy to, aktualizując istniejący wiersz poprzez wpisanie do kolumny `amount` (liczba) wartości zmiennej `$newCount`:

```
mysql_query("UPDATE counter SET amount=" . $mewCount);
```

Na koniec zwracamy nową wartość do Flasha, aby ją wyświetlić:

```
return "resp=" . $newCount;
```

To cały kod w PHP, konieczny do utworzenia licznika odwiedzin. Poniżej przedstawiam skrypt w całości:

```
<?php

include 'dnConn.php'

$query = "SELECT amount FROM counter";
$result = mysql_query($query);
$row = mysql_fetch_array($result);
$oldcount = $row['amount'];
$newCount = $oldCount + 1;

mysql_query("UPDATE counter SET amount=" . $mewCount);

return "resp=" . $newCount;

?>
```

Piszemy część licznika odwiedzin we Flashu

Po napisaniu części licznika w PHP możemy przejść do części aplikacji we Flashu. Składa się ona w stu procentach z kodu w ActionScripcie.

Na początku licznik musi odwołać się do pliku PHP, a istnieją ku temu dwa powody. Po pierwsze skrypt PHP musi pobrać i zaktualizować wartość licznika, a po drugie zwrócić do Flasha jego wartość, która zostanie umieszczona w dynamicznym polu tekstowym.

W pierwszej części definiujemy zmienną `phpFile`, która zawiera adres URL skryptu PHP, znajdującego się na serwerze, do którego będziemy się odwoływać:

```
var phpFile:String = "http://localhost/helion/rozdzial11/hitCounter.php";
```

Pierwszą funkcją, którą napiszemy, będzie `loadHitCounter()`, która odwołuje się do serwera i dodaje odpowiednią funkcję obsługi zdarzeń:

```
function loadHitCounter():void
{
   var urlRequest:URLRequest = new URLRequest(phpFile);
   var urlLoader:URLLoader = new URLLoader();
   urlLoader.addEventListener(Event.COMPLETE, handleServeResponse);
   urlLoader.load(urlRequest);
}
```

Po pobraniu odpowiedzi wywoływana jest funkcja `handleServerResp()`, której przekazujemy pobrane dane. Dane te przekazujemy następnie do obiektu klasy `URLVariables` w celu uzyskania wartości właściwości `resp`. To w niej znajduje się bieżąca wartość licznika.

```
function handleServerResp(e:Event):void
{
   var loader:URLLoader = URLLoader(e.target);
   var variables:URLVariables = new URLVariables(loader.data);
   var count:uint = variables.resp;
   ...
}
```

Na koniec wartość licznika umieszczamy w tworzonym dynamicznie polu tekstowym, które w tym przykładzie nie jest w żaden sposób formatowane, ale przecież łatwo możesz to zrobić sam:

```
   var txt:TextField = new TextField();
   txt.selectable = false;
   txt.width = 200;
   txt.text = count + "odwiedzin";
}
```

Na samym końcu skryptu we Flashu znajduje się wywołanie funkcji `loadHitCounter()`, która uruchamia całą aplikację:

```
loadhitCounter();
```

Podsumowanie

W tym rozdziale poznałeś etapy tworzenia i projektowania aplikacji. Gdy już dowiedziałeś się, jak należy rozwijać oprogramowanie, napisałeś we Flashu i w PHP klienta czata.

W następnym podrozdziale dowiedziałeś się, jak napisać opartą na Flashu galerię ze zdjęciami, pozwalającą na dynamiczne dodawanie zdjęć i kategorii dzięki zastosowaniu dokumentu XML.

Ostatni podrozdział został poświęcony tworzeniu innych aplikacji z zastosowaniem Flasha, PHP i MySQL-a, abyś lepiej zrozumiał prezentowane techniki pracy.

W tym momencie powinieneś już wiedzieć, jak tworzyć wydajne aplikacje, robiące użytek z danych udostępnianych dynamicznie w celu łatwiejszej aktualizacji oraz zwiększenia możliwości samego programu.