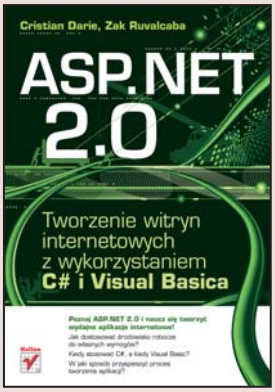
ASP.NET 2.0. Tworzenie witryn internetowych z wykorzystaniem C# i Visual Basica



Autor: Cristian Darie, Zak Ruvalcaba
Tłumaczenie: Ireneusz Jakóbik, Maciej Jezierski
ISBN: 978-83-246-0999-4
Tytuł oryginału: Build Your Own ASP.NET 2.0 Web Site Using C# & VB
Format: B5, stron: około 608
Przykłady na ftp: 1991 kB

Poznaj ASP.NET 2.0 i naucz się tworzyć wydajne aplikacje internetowe!

- Jak dostosować środowisko robocze do własnych wymogów?
- Kiedy stosować C#, a kiedy Visual Basic?
- W jaki sposób przyspieszyć proces tworzenia aplikacji?

Czas statycznych witryn WWW dawno już przeminął. Dziś, aby przyciągnąć odwiedzających, należy regularnie aktualizować stronę, zamieszczając na niej aktualne treści. Serwisy i aplikacje internetowe pobierające artykuły z baz danych, łatwe do modyfikowania i zabezpieczone przed niepowołanym dostępem to teraźniejszość i przyszłość sieci. ASP to wykorzystywana od dłuższego czasu technologia tworzenia dynamicznych witryn WWW, która po zaprezentowaniu platformy .NET zyskała zupełnie nowe oblicze. Tworzenie serwisów i aplikacji internetowych z wykorzystaniem ASP.NET 2.0 przebiega błyskawicznie. Możliwość użycia języka C# lub Visual Basic oraz bezpłatnych i łatwych w konfiguracji narzędzi sprawiają, że ASP.NET 2.0 zyskuje coraz większą popularność.

Po lekturze książki „ASP.NET 2.0. Tworzenie witryn internetowych z wykorzystaniem C# i Visual Basica" ta technologia przestanie mieć przed Tobą tajemnice. Czytając ją, dowiesz się, jak zainstalować i skonfigurować narzędzia do pracy, podejmiesz decyzję o wyborze języka programowania, którego użyjesz do tworzenia aplikacji sieciowych, i poznasz ten język. Nauczysz się korzystać z kontrolek .NET, budować aplikacje, łączyć je z bazami danych i implementować mechanizmy uwierzytelniania użytkowników. Przeczytasz także o generowaniu przesyłek e-mail z poziomu aplikacji oraz o operacjach na plikach.

- Instalacja i konfiguracja serwera i narzędzi produkcyjnych
- Podstawy ASP.NET
- Programowanie w C# i Visual Basic
- Tworzenie stron internetowych z wykorzystaniem kontrolek
- Zastosowanie kaskadowych arkuszy stylów
- Sprawdzanie poprawności danych wprowadzanych w formularzach
- Projektowanie i tworzenie baz danych
- Manipulowanie danymi za pomocą języka SQL
- Korzystanie z technologii ADO.NET
- Wyświetlanie danych na stronie za pomocą kontrolek
- Zabezpieczanie aplikacji

Platforma .NET to przyszłość programowania. Już dziś poznaj jej elementy

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

Spis treści

Rozdział 4.

Tworzenie stron internetowych ASP.NET

Jeśli kiedykolwiek budowałeś modele z klocków lego, jesteś dobrze przygotowany do tworzenia prawdziwych stron ASP.NET. Technologia ta oferuje wiele technik umożliwiających programistom niezależne tworzenie różnych części stron internetowych, a następnie składanie ich w całość.

Zawartość, którą tworzymy podczas pracy z ASP.NET, prawie nigdy nie jest statyczna. W czasie projektowania myślimy w kategoriach szablonów posiadających miejsca na zawartość, która zostanie wygenerowana dynamicznie w czasie działania aplikacji.

W tym rozdziale omówimy wiele obiektów i technik, które nadają stronom internetowym ASP.NET wygląd, włączając w to:

- formatki internetowe,
- kontrolki HTML serwera,
- kontrolki internetowe serwera,
- kontrolki internetowe użytkownika,
- strony wzorcowe,
- obsługę nawigacji strony,
- stylizowanie stron i kontrolek za pomocą CSS.

Jeśli ta lista trochę Cię przestraszyła — nie przejmuj się. Wszystko to jest znacznie łatwiejsze do zrozumienia, niż by się wydawało.

Formatki internetowe

Jak wiesz, podczas uczenia się nowej technologii zawsze trzeba opanować nową terminologię. Jednak w ASP.NET nawet najprostsze terminy używane do opisu podstaw stron internetowych zostały zmienione, żeby odzwierciedlić procesy, które w nich zachodzą.

Terminem używanym do opisania strony internetowej ASP.NET jest **formatka internetowa** (ang. *web form*). Jest to główny obiekt w programowaniu ASP.NET. Spotkałeś się już z formatkami internetowymi — to pliki *.aspx*, z którymi dotąd pracowałeś w tej książce. Na pierwszy rzut oka formatki internetowe wyglądają jak strony HTML, ale poza statyczną zawartością HTML znajdują się w nich również elementy prezentacyjne ASP.NET oraz kod wykonywany po stronie serwera, który generuje dynamiczną zawartość i wykonuje żądane funkcje.

Każda formatka internetowa posiada znacznik `<form runat="server">`, zawierający elementy specyficzne dla ASP.NET, z których składa się strona. Na stronie może znajdować się tylko jedna formatka. Podstawowa struktura formatki internetowej znajduje się poniżej:

```
<html>
   <head>\
      <script runat="server" language="język programowania">
         ...tu znajduje się kod
      </script>
   </head>
   <body>
      <form runat="server">
         ...tu znajdują się elementy interfejsu użytkownika...
      </form>
   </body>
</html>
```

Do odwoływania się do formatki internetowej i wykonywania na niej operacji z poziomu języka programowania wykorzystujemy klasę `System.Web.UI.Page`. Być może przypominasz ją sobie z przykładu z chowaniem kodu w rozdziale 3. W pliku chowającym kod musimy jawnie korzystać z tej klasy. W sytuacjach, w których nie korzystamy z plików chowających kod (np. cały kod piszemy w pliku *.aspx*), klasa `Page` także jest używana, tyle tylko, że niejawnie.

Wewnątrz formatki możemy korzystać z różnorodnych elementów interfejsu użytkownika począwszy od statycznego kodu HTML, aż po elementy, których wartości i właściwości mogą zostać wygenerowane lub zmieniane albo przed pierwszym wczytaniem strony, albo po wysłaniu formularza. Elementy te — w terminologii ASP.NET nazywane kontrolkami — umożliwiają wielokrotne używanie w różnych formatkach internetowych wspólnych funkcjonalności, takich jak nagłówek strony, kalendarz, zawartość koszyka na zakupy czy ramka z cytatem dnia. W ASP.NET jest kilka typów kontrolek:

- kontrolki HTML serwera,
- kontrolki internetowe serwera,
- kontrolki internetowe użytkownika,
- strony wzorcowe.

Pomiędzy powyższymi typami kontrolek są znaczne różnice techniczne, jednak łączy je łatwość integracji i możliwość wielokrotnego wykorzystywania w witrynach internetowych. Przyjrzyjmy się wszystkim powyższym typom kontrolek po kolei.

Kontrolki HTML serwera

Kontrolki HTML serwera są na pozór podobne do zwykłych znaczników HTML, ale zawierają atrybut `runat="server"`. Pozwala on ASP.NET na sterowanie tymi kontrolkami, dzięki czemu możemy odnosić się do nich z poziomu języka programowania. Jeśli na przykład na stronie mamy znacznik `<a>` i chcemy dynamicznie, za pomocą kodu VB lub C#, zmienić adres, do którego się odnosi, wykorzystamy atrybut `runat="server"`.

Dla większości często używanych elementów HTML są odpowiednie kontrolki HTML po stronie serwera. Stworzenie takiej kontrolki jest łatwe: wystarczy na końcu zwykłego znacznika HTML dodać atrybut `runat="server"`. Pełna lista bieżących klas kontrolek HTML i powiązanych z nimi znaczników znajduje się tabeli 4.1.

Tabela 4.1. *Klasy kontrolek HTML*

Klasa	Powiązany znacznik
HtmlAnchor	<a runat="server">
HtmlButton	<button runat="server">
HtmlForm	<form runat="server">
HtmlImage	<img runat="server">
HtmlInputButton	<input type="submit" runat="server">
	<input type="reset" runat="server">
	<input type="button" runat="server">
HtmlInputCheckBox	<input type="checkbox" runat="server">
HtmlInputFile	<input type="file" runat="server">
HtmlInputHidden	<input type="hidden" runat="server">
HtmlInputImage	<input type="image" runat="server">
HtmlInputRadioButton	<input type="radio" runat="server">
HtmlInputText	<input type="text" runat="server">
	<input type="password" runat="server">
HtmlSelect	<select runat="server">
HtmlTable	<table runat="server">
HtmlTableRow	<tr runat="server">
HtmlTableCell	<td runat="server">
	<th runat="server">
HtmlTextArea	<textarea runat="server">
HtmlGenericControl	
	Wszystkie pozostałe znaczniki HTML

Bardziej szczegółowy opis tych klas znajduje się w „dodatku A".

Wszystkie klasy kontrolek HTML serwera są zawarte w przestrzeni nazw `System.Web.UI.HtmlControls`. Ponieważ są one przetwarzane przez ASP.NET po stronie serwera, mamy dostęp z poziomu kodu do ich właściwości w każdym miejscu strony. Jeśli znasz JavaScript, HTML i CSS, wiesz, że zmienianie tekstu w znacznikach HTML, a nawet modyfikowanie zawartych w nich stylów może być kłopotliwe i podatne na błędy. Kontrolki HTML serwera pozwalają rozwiązać te problemy, umożliwiając łatwą modyfikację strony za pomocą wybranego języka ASP.NET, np. VB lub C#.

Korzystanie z kontrolek HTML serwera

Nic nie wyjaśnia teorii lepiej niż działający przykład. Stwórzmy prostą ankietę korzystającą z następujących kontrolek:

- `HtmlForm`,
- `HtmlButton`,
- `HtmlInputText`,
- `HtmlSelect`.

Rozpoczniemy od stworzenia nowego pliku o nazwie *Survey.aspx*. Umieść go w katalogu *Learning*, który założyłeś w rozdziale 1. Poniższy kod tworzy wizualny interfejs ankiety:

Plik: Survey.aspx (fragment)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Transitional//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-transitional.dtd">
<html>
   <head>
      <title>Korzystanie z kontrolek HTML serwera w ASP.NET</title>
      <!-- Tu będzie znajdował się kod -->
   </head>
   <body>
   <form runat="server">
         <h2>Weź udział w ankiecie!</h2>
         <!-- Wyświetl imię użytkownika -->
         <p>
            Imię:<br />
            <input type="text" id="name" runat="server" />
         </p>
         <!-- Wyświetl e-mail -->
         <p>
            E-mail:<br />
            <input type="text" id="email" runat="server" />
         </p>
         <!-- Wyświetl listę technologii -->
         <p>
            Których technologii po stronie serwera używasz?<br />
            <select id="serverModel" runat="server" multiple="true">
               <option>ASP.NET</option>
               <option>PHP</option>
               <option>JSP</option>
               <option>CGI</option>
```

```
            <option>ColdFusion</option>
         </select>
      </p>
      <!-- Wyświetl .NET opcje opinii o net -->
      <p>
         Czy wciąż lubisz .NET?<br />
         <select id="likeDotNet" runat="server">
            <option>Tak</option>
            <option>Nie</option>
         </select>
      </p>
      <!-- Wyświetl przycisk potwierdzający -->
      <p>
         <button id="confirmButton" OnServerClick="Click"
               runat="server">Zatwierdź</button>
      </p>
      <!-- Etykieta potwierdzenia -->
      <p>
         <asp:Label id="feedbackLabel" runat="server" />
      </p>
   </form>
</body>
</html>
```

Z tego, co dotąd dowiedziałeś się o kontrolkach HTML, powinieneś wywnioskować, z jakimi klasami będziemy mieli do czynienia na tej stronie. Wszystko, co zrobiliśmy, to umieszczenie kilku kontrolek `HtmlInputText`, kontrolki `HtmlButton` oraz `HtmlSelect` wewnątrz wymaganej kontrolki `HtmlForm`. Dodaliśmy także kontrolkę `Label`, którą wykorzystamy do wyświetlenia wyników użytkownikowi.

Kontrolki HTML serwera w działaniu

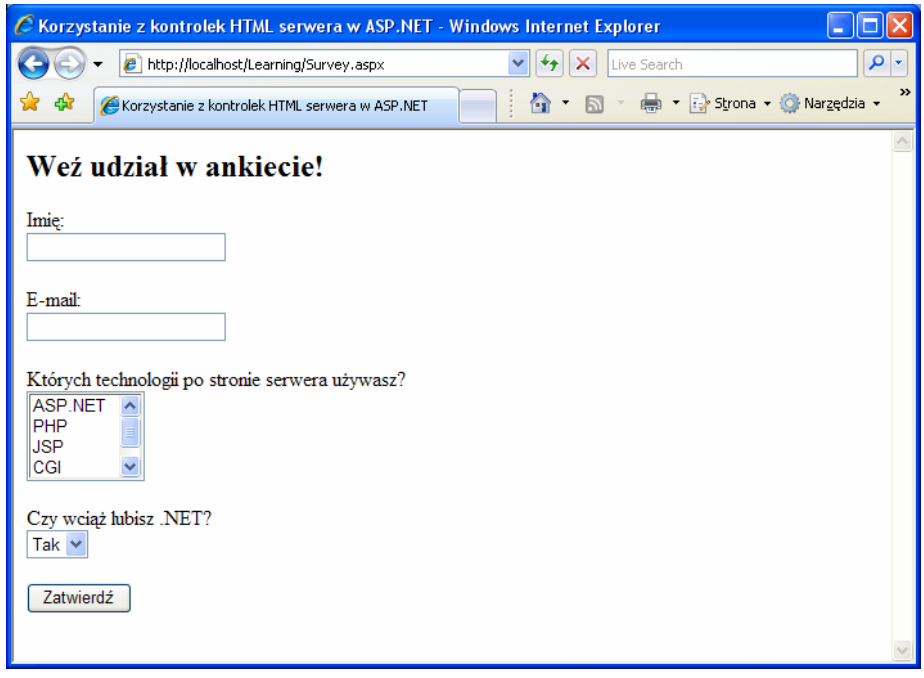
Zapamiętaj, że kontrolki HTML serwera są w istocie znacznikami HTML z atrybutem `runat="server"`. W większości przypadków będziesz musiał także przypisać im identyfikatory, które umożliwią Ci używanie ich w kodzie.

Kiedy wszystko będzie już gotowe, formatka internetowa *Survey.aspx* będzie przypominała rysunek 4.1.

Kiedy użytkownik naciśnie przycisk, wyświetlimy przesłane odpowiedzi w przeglądarce. W prawdziwej aplikacji najprawdopodobniej zapisalibyśmy informacje w bazie danych i być może pokazalibyśmy wyniki w postaci wykresu. W każdym przypadku będziemy odnosić się do właściwości kontrolek HTML w następujący sposób:

Visual Basic *Plik: Survey.aspx (fragment)*

```
<script runat="server" language="VB">
   Sub Click(ByVal s As Object, ByVal e As EventArgs)
      Dim i As Integer
      feedbackLabel.Text = "Nazywasz się: " & name.Value & "<br />"
      feedbackLabel.Text += "Twój adres e-mail: " & email.Value & _
            "<br />"
```



Rysunek 4.1. *Prosta formatka wykorzystująca kontrolki HTML serwera*

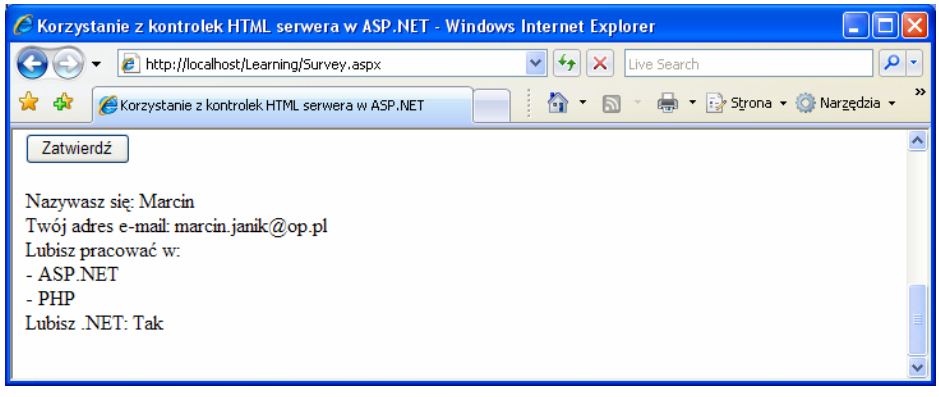
```
        feedbackLabel.Text += "Lubisz pracować w:<br />"
        For i = 0 To serverModel.Items.Count - 1
           If serverModel.Items(i).Selected Then
              feedbackLabel.Text += " - " & _
                    serverModel.Items(i).Text & "<br />"
           End If
        Next i
        feedbackLabel.Text += "Lubisz .NET: " & likeDotNet.Value
    End Sub
</script>
```

C# *Plik: Survey.aspx (fragment)*

```
<script runat="server" language="C#">
   void Click(Object s, EventArgs e)
   {
      feedbackLabel.Text = "Nazywasz się: " + name.Value + "<br />";
      feedbackLabel.Text += "Twój adres e-mail: " + email.Value +
            "<br />";
      feedbackLabel.Text += "Lubisz pracować w:<br />";
      for (int i = 0; i <= serverModel.Items.Count - 1; i++)
      {
         if (serverModel.Items[i].Selected)
         {
            feedbackLabel.Text += " - " + serverModel.Items[i].Text +
                  "<br />";
         }
      }
      feedbackLabel.Text += "Lubisz .NET: " + likeDotNet.Value;
   }
</script>
```

Podobnie jak w przykładach z poprzednich rozdziałów, rozpoczniemy od umieszczenia kodu VB lub C# w bloku kodu skryptu strony serwera wewnątrz części `<head>` strony. Następnie stworzymy nową funkcję obsługi zdarzenia `Click`, która będzie przyjmowała dwa parametry zwyczajowe parametry. W końcu użyjemy kontrolki `Label` do wyświetlenia na stronie odpowiedzi użytkownika.

Po napisaniu kodu zapisz swoją pracę i przetestuj wyniki w przeglądarce. Wpisz jakieś informacje i naciśnij przycisk. Żeby wybrać kilka opcji z listy `serverModel`, kiedy będziesz klikał opcje, naciśnij i przytrzymaj *Ctrl*. Informacje, które wprowadziłeś, pojawią się u dołu strony, tak jak na rysunku 4.2.



Rysunek 4.2. *Podgląd wyników ankiety*

Konkludując, praca z kontrolkami HTML serwera jest naprawdę prosta. Wszystko, czego potrzebujesz, to przypisanie każdej kontrolce identyfikatora oraz dodanie atrybutu `runat="server"`. Następnie możesz w prosty sposób odwoływać się i przeprowadzać na nich operacje wykorzystując kod VB lub C# po stronie serwera.

Kontrolki internetowe serwera

Kontrolki internetowe serwera mogą być postrzegane jako bardziej zaawansowana wersja kontrolek. Służą one do generowania zawartości za Ciebie — nie masz wpływu na to, jaki HTML zostanie wygenerowany. O ile dobra znajomość HTML-a jest przydatna, o tyle nie jest ona konieczna dla tych, którzy korzystają z kontrolek internetowych serwera.

Spójrzmy na przykład. Możemy użyć kontrolki serwera `Label` do umieszczenia tekstu wewnątrz formatki internetowej. Żeby zmienić tekst w kontrolce `Label` z poziomu kodu C# lub VB, po prostu ustawiamy jej właściwość `Text`, tak jak poniżej:

Visual Basic

```
myLabel.Text = "Myszka Miki"
```

Podobnie, żeby dodać pole tekstowe do formatki, używamy kontrolki internetowej serwera `TextBox`. Ponownie możemy odczytać bądź zapisać tekst za pomocą właściwości `Text`:

C#

```
username = usernameTextBox.Text;
```

Mimo że stosujemy kontrolkę `TextBox`, ASP.NET tak naprawdę używa elementu `input`, jednak nie musimy się już przejmować jego szczegółami. Dzięki kontrolkom internetowym serwera Microsoft po prostu ponownie wymyślił HTML.

W przeciwieństwie do kontrolek HTML serwera, kontrolki internetowe serwera nie przekładają się bezpośrednio na tworzone przez nie elementy HTML. Na przykład, do wygenerowania elementu `select` możemy użyć jednej z dwóch kontrolek internetowych serwera: `DropDownList` lub `ListBox`.

Kontrolki internetowe serwera stosują ten sam podstawowy wzór co znaczniki HTML, ale nazwa znacznika jest poprzedzona przez `asp:` i zapisana w notacji języka Pascal. Notacja języka Pascal polega na zapisywaniu pierwszych liter każdego wyrazu wielką literą (np. `TextBox`). Identyfikatory obiektów są zazwyczaj zapisywane za pomocą notacji wielbłądziej, w której wszystkie pierwsze litery poszczególnych wyrazów, z wyjątkiem pierwszego, są zapisywane wielką literą (np. `usernameTextBox`).

Rozpatrzmy poniższy element HTML `input`, który tworzy pole tekstowe:

```
<input type="text" name="usernameTextBox" size="30" />
```

Odpowiadającą mu kontrolką internetową serwera jest kontrolka `TextBox`, która wygląda następująco:

```
<asp:TextBox id="usernameTextBox" runat="server" Colums="30">
</asp:textBox>
```

Pamiętaj o tym, że w przeciwieństwie do zwykłych elementów HTML, które mógłbyś wykorzystać w formatkach internetowych, kontrolki internetowe serwera są najpierw przetwarzane przez ASP.NET i zamieniane na HTML. Efektem ubocznym tego podejścia jest to, że musisz pamiętać o tym, żeby zawsze wstawiać znaczniki zamykające (`</asp:TextBox>` w powyższym kodzie). Parsery HTML-a w przeglądarkach nie są rygorystyczne co do poprawności kodu, ale ASP.NET jest. Pamiętaj o tym, że jeśli niczego nie ma pomiędzy znacznikami otwierającymi i zamykającymi kontrolkę internetową serwera, możesz użyć skrótowej składni (`/>`). Tak więc kontrolka `TextBox` mogłaby wyglądać następująco:

```
<asp:TextBox id="usernameTextBox" runat="server" Colums="30" />
```

Podsumowując, kluczowe rzeczy, o których należy pamiętać podczas pracy z kontrolkami internetowymi serwera, to:

- Żeby prawidłowo działać, kontrolki internetowe serwera muszą zostać umieszczone wewnątrz znacznika `<form runat="server">`.
- Internetowe kontrolki serwera wymagają atrybutu `runat="server"`.
- Internetowe kontrolki serwera wstawiamy do formatki za pomocą przedrostka `asp:`.

Internetowych kontrolek serwera jest więcej niż kontrolek HTML. Niektóre z nich oferują zaawansowane funkcje, których po prostu nie można zrealizować za pomocą samego HTML-a, a inne z kolei tworzą dosyć złożony HTML za Ciebie. Z wieloma kontrolkami internetowymi serwera oraz sposobem ich działania zapoznamy się w kolejnych rozdziałach.

Więcej o kontrolkach internetowych serwera, włączając w to właściwości metody i zdarzenia, dowiesz się w „dodatku A".

Standardowe kontrolki internetowe serwera

Zbiór standardowych kontrolek internetowych serwera dostarczany z ASP.NET na wiele sposobów odzwierciedla kontrolki HTML serwera. Jednak kontrolki internetowe serwera udostępniają wiele udoskonaleń i rozszerzeń, takich jak obsługa zdarzeń czy stanu widoku, spójniejszy zestaw właściwości i metod oraz więcej wbudowanych funkcjonalności. W tym punkcie przyjrzymy się kilku kontrolkom, z których najprawdopodobniej będziesz korzystał w codziennej pracy.

Pamiętaj o tym, żeby korzystać z dokumentacji .NET Framework 2.0 SDK za każdym razem, kiedy będziesz chciał dowiedzieć się więcej o jakiejś klasie (lub kontrolce). Dokumentacja dostępna jest w menu *Start/Programy/Microsoft .NET Framework SDK 2.0/Documentation*. Żeby znaleźć klasę, po prostu wyszukaj jej nazwę. Jeśli jest wiele klas o tej samej nazwie, ale znajdują się one w różnych przestrzeniach nazw, będziesz mógł wybrać odpowiednią z okienka *Index Results*. Na przykład są trzy klasy o nazwie `Label` umieszczone w przestrzeniach nazw `System.Web.UI.MobileControls`, `System.Web.UI.WebControls` oraz `System.Windows.Form`, co widać na rysunku 4.3. Najprawdopodobniej będziesz zainteresowany wersją klasy umieszczoną w przestrzeni nazw `WebControls`.

Label

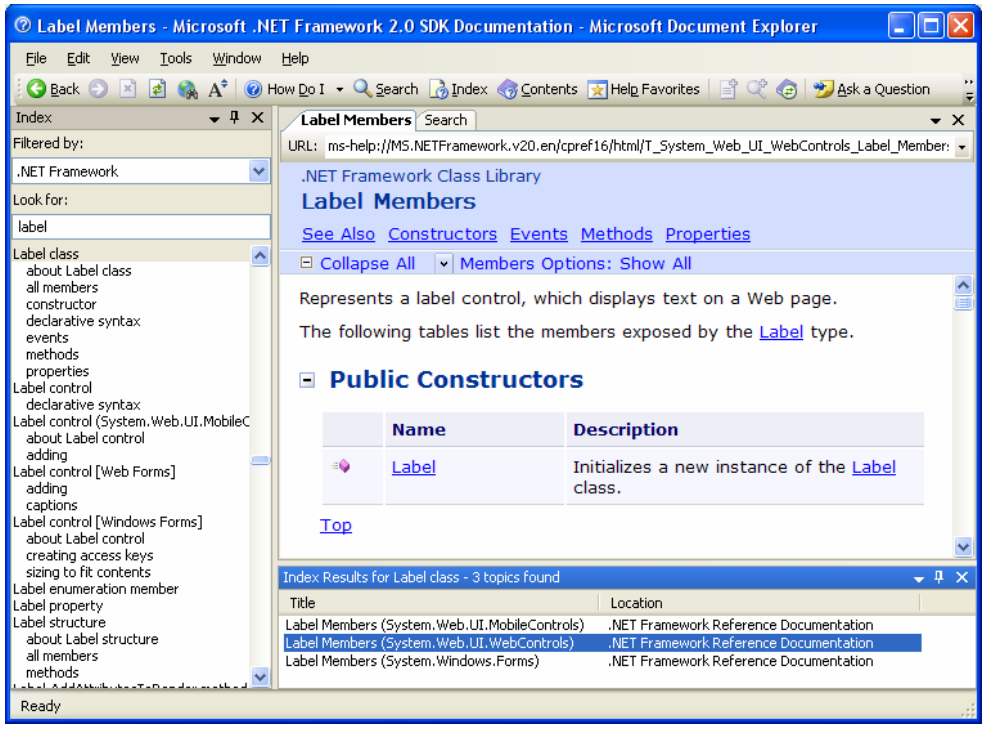
Najprostszym sposobem wyświetlenia statycznego tekstu na stronie jest po prostu wpisanie go bezpośrednio do sekcji body strony bez żadnych znaczników. Jeśli jednak chcesz zmieniać wyświetlany na stronie tekst za pomocą kodu ASP.NET, możesz umieścić go wewnątrz kontrolki `Label`. Poniżej znajduje się typowy przykład:

```
<asp:Label id="messageLabel" Text="" runat="server" />
```

Poniższy kod ustawia właściwość `Text` kontrolki `Label`, tak żeby wyświetlała tekst „Witaj świecie!":

Visual Basic

```
Public Sub Page_Load()
   messageLabel.Text = "Witaj świecie!"
End Sub
```



Rysunek 4.3. *Dokumentacja kontrolki Label*

C#

```
public void Page_Load()
{
    messageLabel.Text = "Witaj swiecie";
}
```

W kodzie funkcji `Page_Load` możemy zobaczyć, że kiedy strona po raz pierwszy jest wczytywana, właściwość `Text` kontrolki `Label` z identyfikatorem `message` zostanie ustawiona na „Witaj świecie”.

Literal

Jest to najprawdopodobniej najprostsza kontrolka w ASP.NET. Jeśli ustawisz jej właściwość `Text`, spowoduje to po prostu wstawienie tekstu do wynikowego kodu HTML bez żadnych zmian. W przeciwieństwie do `Label`, która ma podobna funkcjonalność, `Literal` nie opakowuje tekstu w znaczniki `<span>` umożliwiające ustawienie informacji o stylu.

TextBox

Kontrolka `TextBox` jest wykorzystywana do tworzenia pola tekstowego, w którym użytkownik może zapisywać bądź z którego może czytać zwykły tekst. Właściwość `TextMode` umożliwia ustawienie sposobu wyświetlania tekstu, tak żeby znajdował się w jednej linijce, w

wielu linijkach lub był ukrywany podczas wprowadzania (na przykład tak jak w polach haseł HTML). Poniższy kod demonstruje, w jaki sposób moglibyśmy użyć jej w prostej stronie logowania:

```
<p>
   Użytkownik: <asp:TextBox id="userTextBox" TextMode="SingleLine"
      Columns="30" runat="server" />
</p>
<p>
   Hasło: <asp:TextBox id="passwordTextBox"
      TextMode="Password" Columns="30" runat="server" />
</p>
<p>
   Komentarz: <asp:TextBox id="commentsTextBox"
      TextMode="MultiLine" Columns="30" rows="10"
      runat="server" />
</p>
```

W każdej z powyższych instancji `TextBox` atrybut `TextMode` ustawia rodzaj pola tekstowego, które ma zostać stworzone.

HiddenField

`HiddenField` jest prostą kontrolką, która tworzy element `input` z atrybutem `type` równym `hidden`. Możemy ustawić jej jedyną ważną właściwość: `Value`.

Button

Domyślnie kontrolka `Button` tworzy element `input` z atrybutem `type` o wartości `submit`. Po naciśnięciu przycisku formularz, który go zawiera, jest wysyłany do serwera w celu przetworzenia i wywoływane są zdarzenia `Click` oraz `Command`.

Poniższy kod wyświetla kontrolkę `Button` i `Label`:

```
<asp:Button id="submitButton" Text="Wyślij" runat="server"
   OnClick="WriteText" />
<aps:Label id="messageLabel" runat="server" />
```

Zauważ atrybut `OnClick` w kontrolce. Po naciśnięciu przycisku wywoływane jest zdarzenie `Click`, co powoduje wywołanie procedury `WriteText`. Będzie ona zawierała kod, który wykonuje funkcje przeznaczone dla omawianego przycisku, na przykład wyświetlenie komunikatu:

Visual Basic

```
Public Sub WriteText(s As Object, e As EventArgs)
   messageLabel.Text = "Witaj swiecie!"
End Sub
```

C#

```
public void WriteText(Object s, EventArgs e)
{
   messageLabel.Text = "Witaj swiecie!";
}
```

To ważne, żeby zrozumieć, iż zdarzenia są związane z większością kontrolek internetowych serwera, a podstawowe techniki korzystania z nich są takie same, jak technika, której używaliśmy ze zdarzeniem `Click` kontrolki `Button`. Wszystkie kontrolki implementują standardowy zestaw zdarzeń, ponieważ dziedziczą po klasie bazowej `WebControl`.

ImageButton

Kontrolka `ImageButton` jest podobna do kontrolki `Button`, ale zamiast z systemowego przycisku korzysta z podanego przez nas obrazka. Spójrz na przykład:

```
<asp:ImageButton id="myImgButton" ImageUrl="myButton.gif
   runat="server" OnClick="WriteText" />
<asp:Label id="messageLabel" runat="server" />
```

Zdarzenie `Click` kontrolki `ImageButton` odbiera koordynaty punktu, w którym został kliknięty obrazek:

Visual Basic

```
Public WriteText(s As Object, e As ImageClickEventArgs)
   messsageLabel.Text = "Koordynaty: " & e.X & "," & e.Y
End Sub
```

C#

```
public void WriteText(Object s, ImageClickEventArgs e)
{
   messageLabel.Text = "Koordynaty: " + e.X + "," + e.Yl
}
```

LinkButton

Kontrolka `LinkButton` tworzy hiperłącze, po którego kliknięciu wywoływane jest zdarzenie `Click`. Z punktu widzenia kodu ASP.NET, kontrolki `LinkButton` mogą być traktowane w większości przypadków jak przyciski.

```
<asp:LinkButton id="myLinkButton" Text="Naciśnij tu"
   runat="server" />
```

HyperLink

Kontrolka `HyperLink` tworzy na stronie hiperłącze o adresie ustalanym przez właściwość `NavigateUrl`. W przeciwieństwie do kontrolki `LinkButton` udostępniającej funkcjonalności, takie jak zdarzenia `Click` czy sprawdzanie poprawności, kontrolki `HyperLink` są przeznaczone do nawigacji z jednej strony do innej.

```
<asp:HyperLink id="myLink" NavigateUrl="http://www.sitepoint.com/"
   ImageUrl="splogo.gif" runat="server">SitePoint</asp:HyperLink>
```

Jeśli jest podany, atrybut `ImageUrl` powoduje, że kontrolka wyświetla określony obrazek. W takim przypadku tekst podany wewnątrz kontrolki będzie funkcjonował jako alternatywny tekst obrazka.

CheckBox

Kontrolkę `CheckBox` możesz wykorzystać do wyświetlenia pola wyboru, które ma tylko dwa stany — zaznaczone lub niezaznaczone.

```
<asp:CheckBox id="questionCheck" Text="Tak, lubię .NET!"
   Checked="True" runat="server" />
```

Głównym zdarzeniem związanym z kontrolką `CheckBox` jest `CheckChanged`, które może zostać obsłużone przez atrybut `OnCheckChanged`. Właściwość `Checked` ma wartość `True`, jeśli pole wyboru jest zaznaczone, i `False`, jeśli nie jest.

RadioButton

Kontrolka `RadioButton` jest podobna do `CheckBox` z tym wyjątkiem, że kontrolki tego typu mogą być grupowane razem, dzięki czemu będą reprezentować zbiór opcji, z których tylko jedna może zostać wybrana. Do grupowania służy właściwość `GroupName`.

```
<asp:RadioButton id="warszawa" GropupName="City" Text="Warszawa"
   runat="server" /><.br />
<asp:RadioButton id="katowice" GropupName="City" Text="Katowice"
   runat="server" /><.br />
<asp:RadioButton id="kielce" GropupName="City" Text="Kielce"
   runat="server" /><.br />
<asp:RadioButton id="krakow" GropupName="City" Text="Kraków"
   runat="server" /><.br />
```

Podobnie jak w kontrolce `CheckBox`, głównym zdarzeniem związanym z kontrolkami `RadioButton` jest `CheckChanged`, które może zostać obsłużone przez atrybut `OnCheckChanged`.

Inną kontrolką, którą możemy wykorzystać do wyświetlenia pól wielokrotnego wyboru, jest `RadioButtonList`, opisana w dalszej części rozdziału.

Image

Kontrolka `Image` tworzy obrazek, który może być zmieniany dynamicznie z poziomu kodu. Odpowiada ona znacznikowi `<img>` w HTML-u. Oto przykład:

```
<asp:Image id="myImage" ImageUrl="mygif.gif" runat="server"
   AlternateText="opis" />
```

ImageMap

Kontrolka `ImageMap` generuje HTML wyświetlający obrazki posiadające klikalne obszary, nazywane **aktywnymi miejscami**. Każde z takich miejsc inaczej reaguje na kliknięcie przez użytkownika.

Obszary są definiowane za pomocą trzech kontrolek, określających aktywne miejsca o różnych kształtach: `CircleHotspot` (okrąg), `RectangleHotspot` (prostokąt) oraz `PolygonHotspot` (wielokąt). Poniżej znajduje się przykład definiujący kontrolkę `ImageMap` z dwoma okrągłymi aktywnymi miejscami:

```
<asp:ImageMap id="myImageMap" tunat="server" ImageUrl="image.jpg">
   <asp:CircleHotSpot AlternateText="Przycisk1"
      Radius="20" X="50" Y="50" />
   <asp:CircleHotSpot AlternateText="Przycisk2"
      Radius="20" X="100" Y="50" />
</asp:ImageMap>
```

Żeby skonfigurować działania wynikające z kliknięcia aktywnych miejsc przez użytkownika, musimy ustawić właściwość `HotSpotMode` kontrolki `ImageMap` i (lub) poszczególnych obiektów aktywnych miejsc, używając wartości przedstawionych w tabeli 4.2. Jeśli właściwość `HotSpotMode` jest ustawiona zarówno w kontrolce `ImageMap`, jak i dla miejsca aktywnego, ta ostatnia właściwość nadpisze tę ustawioną w bardziej ogólnej kontrolce `ImageMap`.

Tabela 4.2. *Wartości HotSpotMode*

wartość HotPotMode	Zachowanie po kliknięciu aktywnego miejsca
`Inactive`	brak
`Navigate`	Użytkownik jest przenoszony do określonego adresu URL.
`NotSet`	Jeśli jest ustawiona dla kontrolki `HotSpot`, zachowanie jest dziedziczone po rodzicielskiej kontrolce `ImageMap`. Jeśli w rodzicielskiej kontrolce `ImageMap` określonej domyślnej wartości, ustawiana jest wartość `Navigate`.
	Jeśli jest ustawiona dla `ImageMap`, wartość ta jest równoważna z `Navigate`.
`PostBack`	Miejsce aktywne wywołuje zdarzenie `Click`, które może zostać obsłużone po stronie serwera i pozwala wykonać jakąś operacje w odpowiedzi na działanie użytkownika.

Dokumentacja Microsoft .NET Framework 2.0 SDK dla klasy `ImageMap` zawiera szczegółowe przykłady korzystania z wartości `HotSpotMode`.

PlaceHolder

Kontrolka `PlaceHolder` umożliwia dynamiczne — za pomocą kodu — dodawanie elementów w określonych miejscach na stronie:

```
<asp:PlaceHolder id="placeHolder" runat="server"  />
```

Poniższy kod dodaje nową kontrolkę `HtmlButton` wewnątrz kontrolki `PlaceHolder`:

Visual Basic

```
Public Sub Page_Load()
   Dim myButton As HtmlButton = New HtmlButton()
   myButton.InnerText = "Nowy przycisk"
   placeHolder.Controls.Add(myButton)
EndSub
```

C#

```
public void Page_Load()
{
   HtmlButton myButton = new HtmlButton();
   myButton.InnerText = "Nowy przycisk";
   placeHolder.Controls.Add(myButton);
}
```

Panel

Kontrolka `Panel` działa podobnie do elementu `div` w HTML, ponieważ umożliwia traktowanie zawartych w niej elementów jako grupy i wykonywanie na nich zbiorczych operacji. Na przykład panel można wyświetlać lub chować za pomocą zdarzenia `Click` kontrolki:

```
<asp:Panel id="myPanel" runat="server">
   <p>Użytkownik: <asp:TextBox id="usernameTextBox" Colums="30"
      runat="server" /></p>
   <p>Hasło: <asp:TextBox id="passwordTextBox"
      TextMode="Password" Colums="30" runat="server" />
   </p>
</asp:Panel>
<asp:Button id="hideButton" Text="Ukryj panel" OnClick="HidePanel"
   runat="server" />
```

Powyższy kod umieszcza w kontrolce `Panel` dwie kontrolki `TextBox`. Kontrolka `Button` znajduje się poza kontrolką `Panel`. Procedura `HidePanel` będzie sterowała widocznością kontrolki `Panel` poprzez ustawianie jej właściwości `Visible` na `False`:

Visual Basic

```
Public Sub HidePanel(s As Object, e As EventArgs)
   myPanel.Visible = False
End Sub
```

C#

```
public void HidePanel(Object s, EventArgs e)
{
   myPanel.Visible = false;
}
```

W powyższym przykładzie, kiedy użytkownik kliknie przycisk, wywoływane jest zdarzenie `Click`, które wykonuje procedurę `HidePanel` ustawiającą właściwość kontrolki Panel na `False`.

Kontrolki List

W tym punkcie zapoznamy się z kontrolkami ASP.NET wyświetlającymi proste listy elementów: `ListBox`, `DropDownList`, `CheckBoxList`, `RadioButtonList` oraz `BulletedList`.

DropDownList

Kontrolka `DropDownList` jest podobna do elementu HTML `select`. Umożliwia ona wybór jednego elementu z listy za pomocą rozwijanego menu.

```
<asp:DropDownList id="ddlFavColor" runat="server">
   <asp:ListItem Text="Czerwony" value="red" />
   <asp:ListItem Text="Zielony" value="green" />
   <asp:ListItem Text="Niebieski" value="blue" />
</asp:DropDownList>
```

Najbardziej przydatnym zdarzeniem udostępnianym przez tę kontrolkę jest `SelectedIndexChanged`. Zdarzenie to jest udostępniane przez inne kontrolki list, takie jak `CheckBoxList` i `RadioButtonList`, i umożliwia programiście łatwą komunikację z kontrolką. Kontrolki te mogą być również powiązane bazą danych, dzięki czemu można do rozwijanego menu wstawiać dynamiczną zawartość.

ListBox

Kontrolka `ListBox` odpowiada elementowi HTML `select` z ustawionym atrybutem `multiple` lub `size` (`size` musi mieć wartość 2 lub większą). Jeśli ustawisz atrybut `SelectionMode` na `multiple`, użytkownik będzie mógł wybrać więcej niż jeden element z listy, tak jak w poniższym przykładzie:

```
<asp:ListBox id="listTechnologies" runat="server"
      SelectionMode="Multiple">
   <asp:ListItem Text="ASP.NET" Value="aspnet" />
   <asp:ListItem Text="JSP" Value="jsp" />
   <asp:ListItem Text="PHP" Value="php" />
   <asp:ListItem Text="CGI" Value="cgi" />
   <asp:ListItem Text="ColdFusion" Value="cf" />
</asp:ListBox>
```

RadioButtonList

Podobnie jak kontrolka `RadioButton`, kontrolka `RadioButtonList` reprezentuje pola wielokrotnego wyboru, z tą różnicą, że ta druga reprezentuje ich listę i używa bardziej zwięzłej składni. Oto przykład:

```
<asp:RadioButton id="favouriteColor" runat="server">
   <asp:ListItem Text="Czerwony" Value="red" />
   <asp:ListItem Text="Zielony" Value="green" />
   <asp:ListItem Text="Niebieski" Value="blue" />
</asp:RadioButtonList>\
```

CheckBoxList

Jak się pewnie domyślasz, `CheckBoxList` reprezentuje grupę pól wyboru. Jest odpowiednikiem użycia kilku kontrolek `CheckBox` obok siebie:

```
<asp:CheckBoxList id="favoutiteFood" runat="server">
   <asp:ListItem Text="Pizza" Value="pizza" />
   <asp:ListItem Text="Tacos" Value="tacos" />
   <asp:ListItem Text="Pasta" Value="pasta" />
<.asp:CheckListBox>
```

BulletedList

Kontrolka `BulletedList` wyświetla wypunktowane bądź ponumerowane listy za pomocą znaczników `<ul>` (lista wypunktowana) lub `<ol>` (lista numerowana). W przeciwieństwie do innych kontrolek list, `BulletedList` nie umożliwia wyboru elementów i w związku z tym nie obsługuje zdarzenia `SelectedIndexChanged`.

Pierwszą właściwością, którą będziesz chciał ustawić, jest `DisplayMode`. Możesz jej przypisać wartość `Text` (domyślna) lub `HyperLink` (elementy będą wyświetlane jako łącza). Jeśli `DisplayMode` ma wartość `HyperLink`, możesz użyć zdarzenia `Click` do reagowania na kliknięcie jednego z elementów przez użytkownika.

Inną ważną właściwością jest `BulletStyle`. Określa ona styl wypunktowania. Może mieć następujące wartości: `Numbered` (1, 2, 3, …), `LowerAlpha` (a, b, c, …), `UpperAlpha` (A, B, C, …), `LowerRoman` (i, ii, iii, …), `UpperRoman` (I, II, III, …), `Circle`, `Disc`, `Square` i `CustomImage`. Jeśli styl zostanie ustawiony na `CustomImage`, będziesz musiał ustawić także właściwość `BulletImageUrl` określającą obrazek, który będzie używany w wypunktowaniu. Jeśli styl jest jedną z list numerowanych, możesz ustawić właściwość `FirstBulletNumber`, która określa początkową liczbę lub literę numerowania.

Zaawansowane kontrolki

Kontrolki te są zaawansowane w następujących znaczeniach: sposobu, w jaki się ich używa, generowanego przez nie kodu HTML oraz pracy wykonywanej za Ciebie. Niektóre z tych kontrolek nie są dostępne w starszych wersjach ASP.NET. Wiele z nich poznamy w dalszej części książki (w tym rozdziale opiszemy tylko niektóre).

Calendar

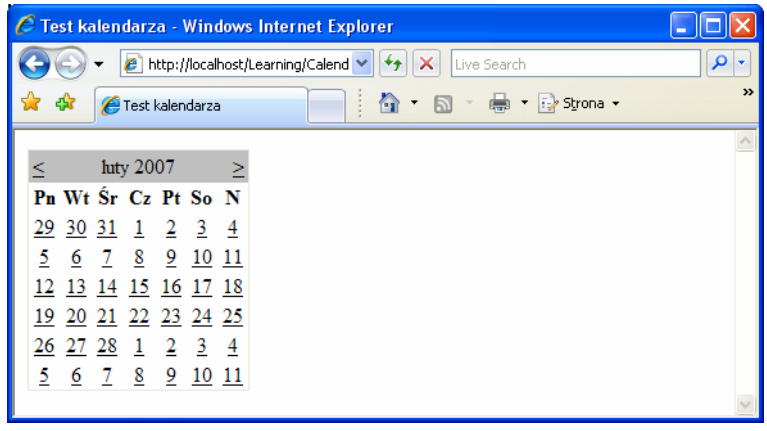
`Calendar` jest świetnym przykładem tego, że kontrolek ASP.NET można wielokrotnie używać. Generuje ona intuicyjny kalendarz, który użytkownik może klikać i wybierać dni, tygodnie, miesiące itd.

Kontrolka `Calendar` wymaga tylko niewielkiego dostosowania. Można ją stworzyć na stronie w następujący sposób:

Plik: Calendar.aspx (fragment)

```
<!DOCTYPE HTML PUBLIC "-//W3C//DTD HTML 4.01//EN"
      "http://www.w3.org/TR/html4/strict.dtd">
<html>
   <head>
      <title>Test kalendarza</title>
   </head>
   <body>
      <form runat="server">
         <asp:Calendar id="myCalendar" runat="server" />
      </form>
   </body>
</html>
```

Jeśli zapiszesz tę stronę w katalogu *Learning*, a następnie ją wczytasz, wynik będzie podobny do rysunku 4.4.



Rysunek 4.4. *Wyświetlanie domyślnego kalendarza*

Kontrolka `Calendar` zawiera szeroki wachlarz właściwości, metod i zdarzeń, włączając te przedstawione w tabeli 4.3.

Przyjrzyjmy się przykładowi, który używa niektórych z tych właściwości, zdarzeń oraz metod do stworzenia kontrolki `Calendar` umożliwiającej użytkownikowi wybieranie dni, tygodni i miesięcy. Zmień kontrolkę `Calendar` w pliku *Calendar.aspx* i dodaj do niej etykietę w następujący sposób:

Plik: Calendar.aspx (fragment)

```
<asp:Calendar id="myCalendar" runat="server" DayNameFormat="Short"
      FirstDayOfWeek="Sunday" NextPrevFormat="FullMonth"
      SelectionMode="DayWeekMonth" SelectWeekText="Select Week"
      SelectMonthText="Select Month" TitleFormat="Month"
      OnSelectionChanged="SelectionChanged" />
<h2>Wybrałeś następujące daty:</h2>
<asp:Label ID="myLabel" runat="server" />
```

Teraz dodaj znacznik `<script runat="server">` do sekcji `head` formatki internetowej i dołącz funkcję obsługi zdarzenia `SelectionChanged`, do której będzie odwoływał się kalendarz:

Visual Basic *Plik: Calendar.aspx (fragment)*

```
<script runat="server" language="VB">
   Sub SelectionChanged(ByVal s As Object, ByVal e As EventArgs)
      myLabel.Text = ""
      For Each d As DateTime In myCalendar.SelectedDates
         myLabel.Text &= d.ToString("D") & "<br />"
      Next
   End Sub
</script>
```

Tabela 4.3. *Niektóre właściwości kontrolki Calendar*

Właściwość	Opis
`DayNameFormat`	Ta właściwość ustawia format nazw dni. Możliwymi wartościami są `FirstLetter`, `FirstTwoLetters` i `Short`. Domyślną wartością jest `Short` — wyświetlane są trzyliterowe skróty.
`FirstDayOfWeek`	Ta właściwość ustawia dzień, od którego zaczyna się tydzień. Domyślna wartość jest określana przez ustawienia regionalne serwera, ale jeśli chcesz, możesz ją zmienić na `Sunday` (niedziela) lub `Monday` (poniedziałek).
`NextPrevFormat`	Ta właściwość steruje formatem łączy następnego i poprzedniego miesiąca. Domyślnie jest ustawiona na `CustomText`, ale można jej przypisać `ShortMonth` lub `FullMonth`.
`SelectedDate`	Ta właściwość zawiera wartość typu `DateTime` określającą podświetlony dzień. Będziesz jej często używał do określania, który dzień został wybrany przez użytkownika.
`SelectionMode`	Ta właściwość określa, czy można wybierać dni, tygodnie lub miesiące. Możliwymi wartościami są: `Day`, `DayWeek`, `DayWeekMonth` oraz `None`, a domyślną jest `Day`. Jeśli wybrana jest wartość `Day`, użytkownik może wybrać tylko dzień; jeśli `DayWeek`, użytkownik może wybrać dzień i tydzień, itd.
`SelectedMonthText`	Ta właściwość określa tekst łącza wyświetlanego w celu umożliwienia użytkownikowi wybrania całego miesiąca z kalendarza.
`SelectedWeekText`	Ta właściwość określa tekst łącza wyświetlanego w celu umożliwienia użytkownikowi wybrania całego tygodnia z kalendarza.
`ShowDayHeader`	Jeśli ta właściwość ma wartość `True`, wyświetlane są nazwy dni tygodnia. Domyślną wartością jest `True`.
`ShowGridLines`	Jeśli ta właściwość ma wartość `True`, wyświetlane są nazwy dni tygodnia. Domyślną wartością jest `True`.
`ShowNextPrevMonth`	Jeśli ta właściwość ma wartość `True`, wyświetlane są łącza do następnego i poprzedniego miesiąca. Domyślną wartością jest `True`.
`ShowTitle`	Jeśli ta właściwość ma wartość `True`, wyświetlany jest tytuł kalendarza. Domyślną wartością jest `False`.
`TitleFormat`	Ta właściwość określa, w jaki sposób wyświetlane są nazwy miesięcy w pasku tytułu. Możliwymi wartościami są: `Month` (miesiąc) i `MonthYear` (miesiąc i rok). Domyślną wartością jest `MonthYear`.
`TodaysDate`	Ta właściwość typu `DateTime` określa bieżącą datę kalendarza. Domyślnie wartość ta nie jest podświetlana w kontrolce `Calendar`.
`VisibleDate`	Ta właściwość typu `DateTime` określa, który miesiąc jest wyświetlany.

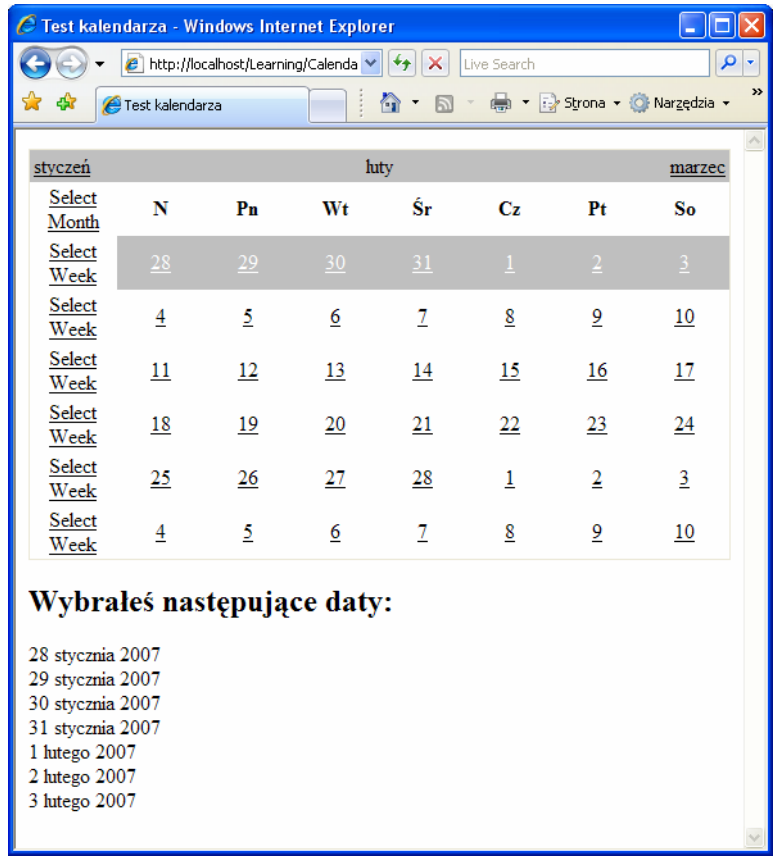
C# *Plik: Calendar.aspx (fragment)*

```
<script runat="server" language="C#">
   void SelectionChanged(Object s, EventArgs e)
   {
      myLabel.Text = "";
      foreach (DateTime d in myCalendar.SelectedDates)
      {
         myLabel.Text += d.ToString("D") + "<br />";
      }
   }
```

```
</script>
```

Zapisz swoją pracę i przetestuj w przeglądarce. Spróbuj wybrać dzień, tydzień lub miesiąc. Wybrane daty zostaną podświetlone, tak jak na rysunku 4.5.

Rysunek 4.5. *Korzystanie z kontrolki Calendar*



W `SelectionChanged` sprawdzamy każdą datę wybraną przez użytkownika i dodajemy ją do etykiety umieszczonej na stronie.

AdRotator

Kontrolka `AdRotator` umożliwia losowe wyświetlanie listy banerów reklamowych w aplikacji internetowej. Jest jednak czymś więcej niż tylko substytutem stworzenia skryptu losującego od zera. Ponieważ kontrolka `AdRotator` pobiera zawartość z pliku XML, administracja i aktualizacja plików z banerami reklamowymi oraz zmiana ich właściwości jest bardzo prosta. Ponadto plik XML umożliwia sterowanie rysunkiem banera, łączem, adresem docelowym łącza oraz częstotliwością pojawiania się banera w stosunku do innych banerów.

Korzyści wynikające z korzystania z tej kontrolki nie kończą się tylko na tym. Ponieważ większość właściwości kontrolki `AddRorator` znajduje się w pliku XML, jeśli chcesz, mo-

żesz je współużytkować w internecie, dzięki czemu inni sprzedawcy lub partnerzy Twojej firmy mogą korzystać z Twoich banerów reklamowych na swoich stronach.

Podstawy XML-a

W swojej istocie XML jest po prostu formatem tekstowym do przesyłania lub przechowywania danych. Nie zawiera on informacji, w jaki sposób dane powinny być prezentowane. XML jest bardzo prosty dla początkujących, ponieważ bardzo przypomina HTML: obydwa zawierają dużą liczbę znaczników wewnątrz pojedynczych ostrych nawiasów (< oraz >). Największą różnicą pomiędzy XML-em a HTML-em jest to, że XML nie ma stałych znaczników i umożliwia tworzenie własnych do opisu danych, które chcemy reprezentować.

Przyjrzyj się poniższemu elementowi HTML:

```
<p>Gwiezdne Wojny Epizod I: Mroczne Widmo</p>
```

W tym przykładzie zawartość pomiędzy znacznikami jest opisana jako paragraf. Jest to wystarczające, jeśli wszystkim, co chcemy zrobić, jest wyświetlenie tekstu „Gwiezdne Wojny Epizod I: Mroczne Widmo" na stronie internetowej. Co będzie jednak, gdybyśmy chcieli mieć dostęp do tych słów jak do danych?

Podobnie jak w przypadku HTML-a, celem XML-a jest opisanie zawartości dokumentu. Jednak w przeciwieństwie do HTML-a, XML nie opisuje, w jaki sposób jakaś zawartość ma być wyświetlana — opisuje on, **czym jest ta zawartość**. Za pomocą XML-a autor strony internetowej może oznaczyć zawartość dokumentu, opisując ją w kategoriach jej znaczenia jako danych.

Możemy wykorzystać XML-a do oznaczenia słów „Gwiezdne Wojny Epizod I: Mroczne Widmo" w sposób, który lepiej odzwierciedla znaczenie tej zawartości jako danych:

```
<film>
   <title>Gwiezdne Wojny Epizod I: Mroczne Widmo</title>
</film>
```

Wybrane przez nas nazwy znaczników najlepiej opisują zawartość elementu. Definiujemy także w miarę potrzeb własne nazwy atrybutów. Na przykład w powyższym przykładzie mógłbyś zechcieć rozróżniać pomiędzy wersją VHS i DVD czy zapisywać, jak się nazywa reżyser filmu. Można to zrobić, dodając atrybuty i elementy w sposób pokazany poniżej:

```
<film format="DVD">
   <title>Gwiezdne Wojny Epizod I: Mroczne Widmo</title>
   <director>George Lucas</director>
</film>
```

Jeśli chcesz to przetestować, stwórz plik o nazwie *ads.xml* w katalogu *Learning* i umieść w nim zawartość przedstawioną poniżej. Spróbuj stworzyć własne banery albo wykorzystać te dostarczone z kodem do książki.

Plik: Ads.xml (fragment)

```
<?xml version="1.0" encoding="Windows-1250"?>
<Advertisements>
   <Ad>
      <ImageUrl>workatdorknozzle.gif</ImageUrl>
      <NavigateUrl>http://www.dorknozzle.com</NavigateUrl>
      <TargetUrl>_blank</TargetUrl>
      <AlternateText>Praca w Dorknozzle.com!</AlternateText>
      <Keyword>Witryna działu HR</Keyword>
      <Impressions>2</Impressions>
```

```
    </Ad>
    <Ad>
       <ImageUrl>getthenewsletter.gif</ImageUrl>
       <NavigateUrl>http://www.dorknozzle.com</NavigateUrl>
       <TargetUrl>_blank</TargetUrl>
       <AlternateText>Pobierz gazetkę Nozzle!</AlternateText>
       <Keyword>Witryna działu sprzedaży</Keyword>
       <Impressions>1</Impressions>
    </Ad>
</Advertisements>
```

Jak możesz zobaczyć, element `Advertisements` jest głównym węzłem i zgodnie ze specyfikacją XML pojawia się tylko raz. Dla każdej poszczególnej reklamy po prostu dodajemy element potomny `Ad`. Na przykład powyższy plik z reklamami zawiera szczegóły dwóch banerów.

Jak już prawdopodobnie zauważyłeś, plik *.xml* umożliwia określenie właściwości dla każdego banera poprzez wpisanie odpowiedniego elementu wewnątrz poszczególnych elementów `Ad`.

`ImageUrl`
: Adres URL obrazka wyświetlanego jako baner reklamowy.

`NavigateURL`
: Strona, do której zostanie przeniesiony użytkownik po kliknięciu banera.

`AlternateText`
: Tekst, który będzie wyświetlany, jeśli przeglądarka nie obsługuje obrazków.

`Keyword`
: Słowo kluczowe używane do określenia kategorii banera.
Jeśli wykorzystasz właściwość `KeywordFilter` kontrolki `AdRotator`, możesz określić kategorię banerów do wyświetlania.

`Impressions`
: Częstotliwość, z jaką będą wyświetlane poszczególne banery w odniesieniu do innych banerów.

  Im większa jest ta liczba, tym częściej określony baner będzie wyświetlany w przeglądarce. Liczba dla tego elementu może przyjmować wartości od 1 do 2 048 000 000. Jeśli je przekracza, strona wywoła wyjątek.

Wszystkie elementy poza `ImageURL` są opcjonalne. Ponadto jeśli określisz `Ad` bez `NavigateURL`, baner będzie wyświetlany bez hiperłącza.

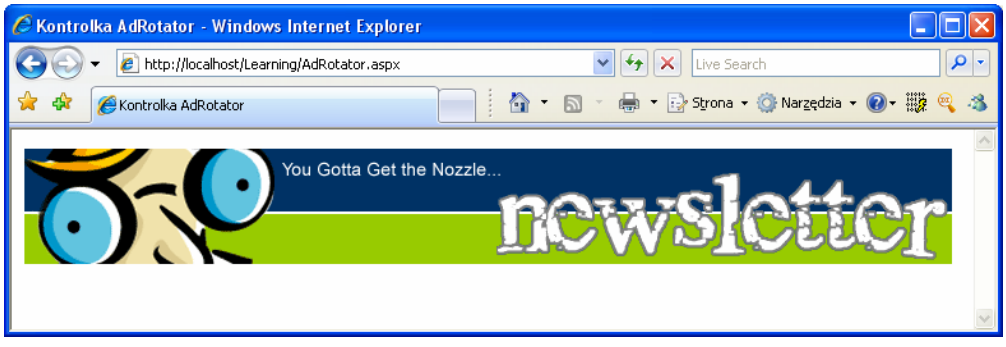
Żeby skorzystać z pliku *Ads.xml*, stwórz nową stronę ASP.NET o nazwie *AdRotator.aspx* zawierającą poniższy kod:

Plik AdRotator.aspx (fragment)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
   <head>
      <title>AdRotator Control</title>
   </head>
   <body>
```

```
        <form runat="server">
            <asp:AdRotator id="adRotator" runat="server"
                  AdvertisementFile="Ads.xml" />
        </form>
    </body>
</html>
```

Żeby zobaczyć obrazki reklamowe, będziesz musiał umieścić pliki *workatdorknozzle.gif* oraz *getthenewsletter.gif* w katalogu *Learning*. Zapisz swoją pracę i przetestuj w przeglądarce. Wynik powinien wyglądać podobnie do przedstawionego na rysunku 4.6.



Rysunek 4.6. *Wyświetlanie reklam za pomocą AdRotator.aspx*

Po kilkukrotnym przeładowaniu strony będziesz mógł zauważyć, że pierwszy baner pojawia się częściej niż drugi. Dzieje się tak, ponieważ wartość `Impressions` w pierwszym elemencie `Ad` jest dwukrotnie większa od wartości w drugim elemencie, dlatego też pierwszy element będzie pojawiał się dwa razy częściej.

TreeView

Kontrolka `TreeView` ma ogromne możliwości i pozwala na wyświetlanie złożonej hierarchicznej struktury elementów. Zazwyczaj używamy jej do wyświetlenia struktury katalogu bądź hierarchii nawigacji, ale może być również wykorzystywana do pokazania drzewa genealogicznego, struktury organizacyjnej korporacji czy innej struktury.

`TreeView` może pobierać dane z różnych źródeł. Różne źródła danych przedstawimy w dalszej części książki. Na razie skupimy się na klasie `SiteMapDataSource`, zawierającej hierarchiczną mapę witryny. Domyślnie mapa ta jest wczytywana z pliku o nazwie *Web.sitemap*, umieszczonego w głównym katalogu projektu. Jest to plik XML, wyglądający podobnie do poniższego:

Plik: Web.sitemap

```
<?xml version="1.0" encoding="Windows-1250"?>
<siteMap
      xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0">
   <siteMapNode title="Strona główna" url="~/Default.aspx"
         description="Strona główna">
      <siteMapNode title="Demo kontrolki TreeView"
url="~/TreeViewDemo.aspx"
            description="Przykład kontrolki TreeView" />
```

```
      <siteMapNode title="Zdarzenie Click" url="~/ClickEvent.aspx"
            description="Przykład zdarzenia Click" />
      <siteMapNode title="Pętle"   url="~/Loops.aspx"
            description="Przykład pętli" />
   </siteMapNode>
</siteMap>
```

Ograniczenia Web.sitemap

Ważnym ograniczeniem, o którym należy pamiętać podczas pracy z plikami *Web.sitemap*, jest to, że muszą zawierać tylko jeden węzeł `SiteMapNode` będący bezpośrednim potomkiem głównego elementu `siteMap`.

W powyższym przykładzie element `siteMapNode` z tytułem *Strona główna* jest pojedynczy. Jeśli dodalibyśmy obok niego (a nie wewnątrz niego) kolejny element `siteMapNode`, plik *Web.sitemap* byłby nieprawidłowy.

Żeby korzystać z tego pliku, musisz dodać do strony kontrolkę `SiteMapDataSource` oraz `TreeView`, która pobiera z niej dane:

Plik: TreeViewDemo.aspx (fragment)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
   <head>
      <title>Demo kontrolki TreeView</title>
   </head>
   <body>
      <form runat="server">
         <asp:SiteMapDataSource ID="mySiteMapDataSource"
               runat="server" />
         <asp:TreeView ID="myTreeView" runat="server"
               DataSourceID="mySiteMapDataSource" />
      </form>
   </body>
</html>
```

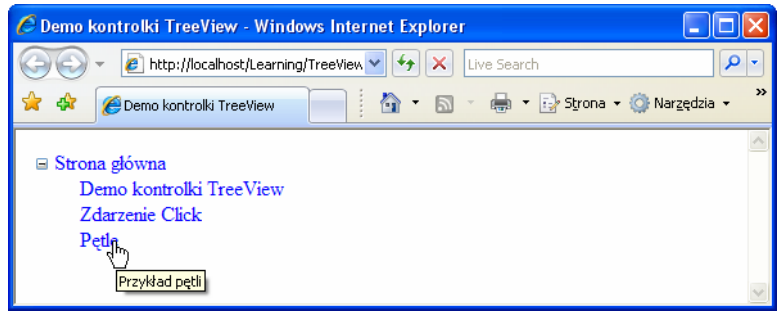
Zauważ, że `SiteMapDataSource` jest kontrolką, która nie generuje żadnego kodu HTML na stronie. Jest wiele podobnych do niej kontrolek będących źródłami danych — szczegółowo zgłębimy ten temat w dalszej części książki.

Po połączeniu z przykładowym plikiem *Web.sitemap* powyższa formatka internetowa wygeneruje wynik przedstawiony na rysunku 4.7.

Jak możesz zobaczyć, kontrolka `TreeView` wygenerowała drzewko za nas. Można nawet rozwijać i zwijać gałąź *Strona główna*.

W wielu przypadkach nie będziemy chcieli pokazywać głównej gałęzi. Możemy ją ukryć, ustawiając właściwość `ShowStartingNode` kontrolki `SiteMapDataSource` na `false`.

Rysunek 4.7. *Prosta kontrolka TreeView*

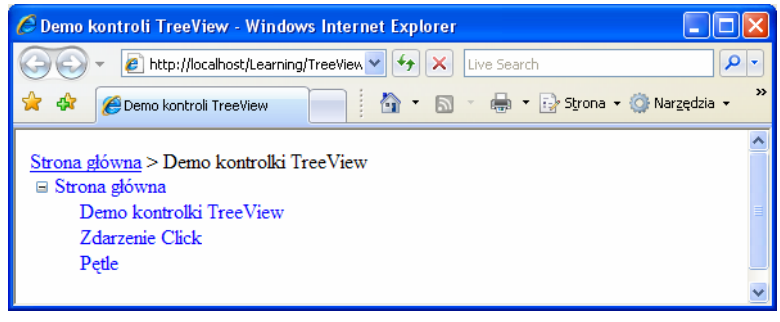



```
<asp:SiteMapDataSource ID="mySiteMapDataSource" runat="server"
      ShowStartNode="false" />
```

SiteMapPath

Kontrolka `SiteMapPath` udostępnia funkcjonalność generującą **ścieżkę nawigacji** strony. Ścieżka nawigacji ułatwia użytkownikom orientację, przedstawiając w prosty sposób informację, w jakim miejscu witryny się znajdują, oraz wyświetlając podręczne łącza do rodzicielskich gałęzi bieżącej lokacji. Przykład ścieżki nawigacji jest przedstawiony na rysunku 4.8.

Rysunek 4.8. *Ścieżka nawigacji stworzona za pomocą kontrolki SiteMapPath*



Kontrolka `SiteMapPath` do wyświetlenia bieżącego miejsca witryny, w którym znajduje się użytkownik, automatycznie wykorzysta każdą kontrolkę `SiteMapDataSource` znajdującą się w formatce internetowej. Żeby zobaczyć wynik przedstawiony na rysunku 4.8, mógłbyś na przykład po prostu dodać poniższy kod do formatki z poprzedniego przykładu:

```
<asp:SiteMapPath id="mySiteMapPath" runat="server"
      PathSeparator=" > ">
</asp:SiteMapPath>
```

Jeśli uruchomisz teraz przykład, zobaczysz, że ścieżka nawigacji pojawi dokładnie tak jak na rysunku 4.8.

Pamiętaj o tym, że kontrolka `SiteMapPath` pokazuje tylko gałąź odpowiadającą istniejącej stronie witryny, jeśli więc nie będzie pliku *Default.aspx*, główna gałąź nie pokaże się. Podobnie, jeśli strona, którą wczytujesz, nie nazywałaby się *TreeViewDemo.aspx*, kontrolka `SiteMapPath` niczego nie wyświetli.

Menu

Kontrolka `Menu` jest podobna do `TreeView`, ponieważ wyświetla hierarchiczne dane ze źródła danych. Sposób, w jaki działają obydwie kontrolki, także jest bardzo podobny. Najważniejszymi różnicami jest ich wygląd oraz to, że `Menu` obsługuje szablony, dzięki czemu można ją lepiej dostosować, a także wyświetla tylko dwa poziomy elementów(menu i podmenu).

MultiView

Kontrolka `MultiView` jest podobna do kontrolki `Panel`, ponieważ nie generuje sama interfejsu, ale zawiera inne kontrolki. `MultiView` może przechowywać więcej stron danych (nazywanych widokami) i pozwala na wyświetlanie jednej z nich w danym czasie. Możesz zmienić aktywny widok (ten, który jest prezentowany użytkownikowi), ustawiając wartość właściwości `ActiveViewIndex`. Pierwsza strona odpowiada `ActiveViewIndex` o wartości `0`, druga — `1`, trzecia — `3`, itd.

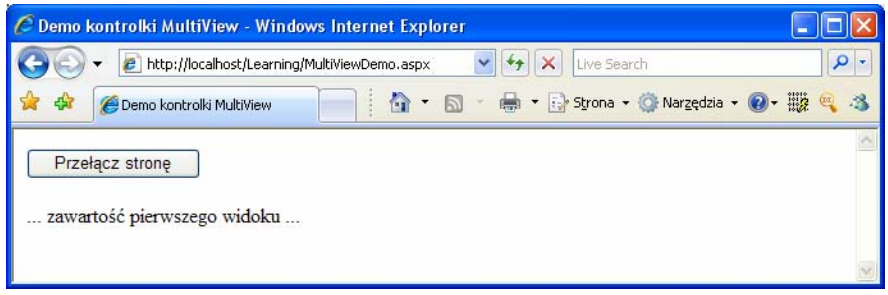
Zawartość każdego szablonu określana jest wewnątrz potomnych elementów `View`. Przyjrzyj się poniższemu fragmentowi kodu tworzącemu kontrolkę `Button` oraz kontrolkę `MultiView`:

Plik: MultiViewDemo.aspx (fragment)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
   <head>
      <title>Demo kontrolki MultiView</title>
   </head>
   <body>
      <form runat="server">
         <p>
            <asp:Button id="myButton" Text="Przełącz stronę"
                  runat="server" OnClick="SwitchPage" />
         </p>
         <asp:MultiView ID="myMultiView" runat="server"
               ActiveViewIndex="0">
            <asp:View ID="firstView" runat="server">
               <p>... zawartość pierwszego widoku ...</p>
            </asp:View>
            <asp:View ID="secondView" runat="server">
               <p>... zawartość drugiego widoku ...</p>
            </asp:View>
         </asp:MultiView>
      </form>
   </body>
</html>
```

Jak możesz zobaczyć, domyślną wartością `ActiveIndex` jest `0`, dlatego też po pierwszym uruchomieniu powyższego kodu `MultiView` wyświetli pierwszy szablon przedstawiony na rysunku 4.9.

Kliknięcie przycisku spowoduje wyświetlenie drugiego szablonu. Poniżej znajduje się funkcja obsługi `SwitchPage`:



Rysunek 4.9. *Wykorzystanie kontrolki MultiView*

Visual Basic *Plik: MultiViewDemo.aspx (fragment)*

```
<script runat="server" language="VB">
   Sub SwitchPage(s as Object, e as EventArgs)
      myMultiView.ActiveViewIndex = _
         (myMultiView.ActiveViewIndex + 1) Mod 2
   End Sub
</script>
```

C# *Plik: MultiViewDemo.aspx (fragment)*

```
<script runat="server" language="C#">
   public void SwitchPage(Object s, EventArgs e)
   {
      myMultiView.ActiveViewIndex =
          (myMultiView.ActiveViewIndex + 1) % 2;
   }
</script>
```

Ta prosta procedura do ustawienia `AcvtiveIndex` na `1`, kiedy bieżącą wartością jest `0` i na odwrót, używa operatora reszty dzielenia.

Kontrolka `MultiView` ma wiele użytecznych funkcji, dlatego też powinieneś przeczytać dokumentację, jeśli będziesz korzystał z niej w środowisku produkcyjnym.

Wizard

Kontrolka `Wizard` (kreator) jest bardziej zaawansowaną wersją kontrolki `MultiView`. Umożliwia ona wyświetlanie jednej lub więcej stron za jednym razem i posiada również dodatkowe wbudowane funkcjonalności, takie jak przyciski nawigacyjne czy panel boczny wyświetlający poszczególne kroki kreatora.

FileUpload

Kontrolka `FileUpload` umożliwia użytkownikom wysyłanie plików na stronę. Sposobu korzystania z tej kontrolki nauczysz się w rozdziale 14.

Kontrolki internetowe użytkownika

W miarę jak będziesz tworzył rzeczywiste projekty, często spotkasz się z fragmentami interfejsu użytkownika, które pojawiają się w wielu miejscach — nagłówki lub stopki, łącza nawigacyjne czy okienka do logowania są tylko przykładami. Umieszczanie ich wyglądu oraz zachowania we własnych kontrolkach umożliwi Ci ponowne użycie tych składników w taki sam sposób, w jaki wielokrotnie korzystasz z wbudowanych kontrolek ASP.NET.

Tworzenie własnych kontrolek internetowych serwera wiąże się z pisaniem zaawansowanego kodu VB lub C# i wykracza poza zakres tej książki, ale dobrze wiedzieć, że jest to możliwe. Tworzenie dopasowanych kontrolek internetowych serwera ma sens, kiedy musisz zrobić bardziej złożone kontrolki, które udostępniają duże możliwości sterowania nimi i cechują się wysoką wydajnością, albo jeśli chcesz stworzyć kontrolki, które łatwo można zintegrować z wieloma projektami.

Ci z nas, którzy nie są zaawansowanymi programistami, mogą rozwijać własne kontrolki poprzez tworzenie **kontrolek internetowych użytkownika**. One również mają duże możliwości i da się je wielokrotnie wykorzystywać w danym projekcie. Tak jak inne kontrolki, mogą one udostępniać właściwości, zdarzenia oraz metody i są łatwe do zaimplementowania.

Kontrolka internetowa użytkownika jest reprezentowana przez klasę, dziedziczącą po `System.Web.UI.UserControl` i zawierającą podstawową funkcjonalność, którą musisz rozszerzyć, żeby stworzyć swoje własne kontrolki. Główną wadą kontrolek internetowych użytkownika jest to, że są one silnie związane z projektem, w którym są zaimplementowane. W związku z tym rozpowszechnianie czy umieszczanie ich w innym projekcie jest znacznie trudniejsze niż w przypadku internetowych kontrolek serwera.

Kontrolki internetowe użytkownika są implementowane bardzo podobnie jak zwyczajne formatki internetowe — zawierają inne kontrolki, znaczniki HTML oraz kod strony serwera. Rozszerzeniem pliku kontrolki internetowej użytkownika jest *.ascx*.

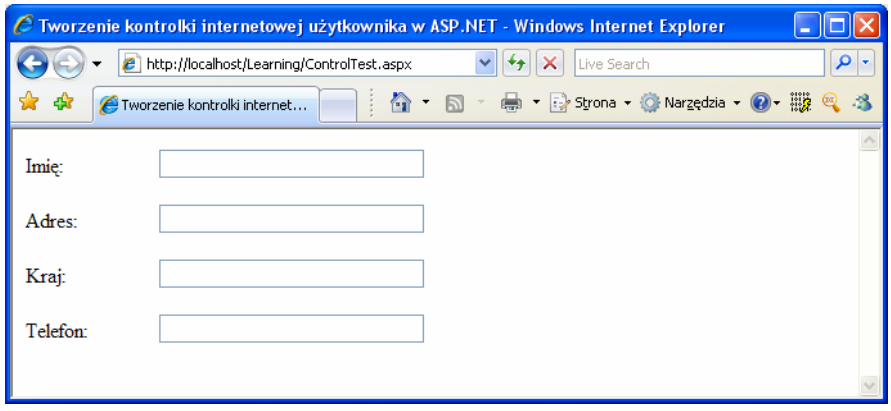
Tworzenie kontrolki internetowej użytkownika

Prześledźmy prosty przykład tworzenia kontrolki internetowej użytkownika. Przyjmijmy, że na witrynie internetowej posiadasz wiele formularzy składających się z par kontrolek `Label` i `TextBox`, tak jak na rysunku 4.10.

Wszystkie etykiety muszą mieć stałą szerokość 100 pikseli, a pola tekstowe muszą przyjmować maksymalnie 20 znaków.

Zamiast dodawać wiele etykiet i pól tekstowych do formatki, a następnie ustawiać ich właściwości, ułatwmy sobie życie tworząc kontrolkę internetową użytkownika, która zawiera kontrolkę `Label` o określonej szerokości i pole tekstowe przyjmujące 20 znaków. Dzięki temu będziesz mógł używać takiej kontrolki za każdym razem, kiedy będzie potrzebna w projekcie.

W katalogu *Learning* stwórz nowy plik o nazwie *SmartBox.ascx*. Następnie umieść w nim części składowe kontrolki — kontrolkę `Label` i `TextBox` — tak jak poniżej:



Rysunek 4.10. *Prosty formularz*

Plik: SmartBox.ascx (fragment)

```
<p>
   <asp:Label ID="myLabel" runat="server" Text="" Width="100" />
   <asp:TextBox ID="myTextBox" runat="server" Text="" Width="200"
         MaxLength="20" />
</p>
```

Szerokość etykiet w Firefoksie

Niestety, ustawienie właściwości `Width` kontrolki `Label` nie gwarantuje, że będzie ona miała tę samą szerokość we wszystkich przeglądarkach. Na przykład bieżąca wersja Firefoksa nie wyświetli powyższej etykiety tak samo jak Internet Explorer.

Żeby obejść ten problem, powinieneś wykorzystać arkusz stylów CSS oraz właściwość `CssClass`. Przyjrzymy się jej w dalszej części rozdziału.

W rozdziale 3. krótko opisaliśmy właściwości, ale nie powiedzieliśmy, w jaki sposób stworzyć własne właściwości w klasie. Dotąd pracowałeś z wieloma właściwościami wbudowanymi w kontrolki. Na przykład widziałeś mnóstwo przykładów kodu, w którym ustawiana jest właściwość `Text` kontrolki `Label`.

Ponieważ kontrolka internetowa użytkownika jest klasą, może mieć także metody, właściwości itp. Nasza kontrolka *SmartBox.ascx* rozszerza klasę `System.Web.UI.UserControl` poprzez dodanie dwóch właściwości:

- `LabelText` jest właściwością tylko do zapisu, umożliwiającą formatką ustawienie tekstu etykiety w kontrolce;
- `Text` jest właściwością tylko do odczytu, która zwraca tekst wpisany w polu tekstowym przez użytkownika.

Dodajmy do skryptu strony serwera element, który umożliwi nam sterowanie tymi kontrolkami — jedną o nazwie `Text` dla tekstu w polu tekstowym i jedną o nazwie `LabelText` dla tekstu w etykiecie:

Visual Basic *Plik: SmartBox.ascx (fragment)*

```
<script runat="server" language="VB">
   Public WriteOnly Property LabelText() As String
      Set(ByVal value As String)
         myLabel.Text = value
      End Set
   End Property

   Public ReadOnly Property Text() As String
      Get
         Text = myTextBox.Text
      End Get
   End Property
</script>
```

C# *Plik: SmartBox.ascx (fragment)*

```
<script runat="server" language="C#">
   public string LabelText
   {
      set
      {
         myLabel.Text = value;
      }
   }
   public string Text
   {
      get
      {
         return myTextBox.Text;
      }
   }
</script>
```

Podobnie jak formatki internetowe, kontrolki internetowe użytkownika mogą pracować z plikami chowającymi kod, ale ponieważ chcemy uprościć przykłady, nie będziemy ich stosować. Z bardziej złożonymi kontrolkami internetowymi użytkownika spotkasz się w następnym rozdziale.

Kiedy korzystasz z kontrolki `SmartBox`, możesz w poniższy sposób ustawić jej etykietę i pobrać tekst wpisany przez użytkownika:

Visual Basic

```
mySmartBox.LabelText = "Adres:"
userAddress = mySmartBox.Text
```

C#

```
mySmartBox.LabelText = "Adres:";
userAddress = mySmartBox.Text;
```

Przyjrzyjmy się, jak zaimplementowaliśmy tę funkcjonalność. W .NET właściwości mogą być tylko do odczytu, tylko do zapisu bądź do zapisu i odczytu. W wielu przypadkach będziesz chciał, żeby właściwości mogły być zarówno odczytywane, jak i zapisywane. Jednak w tym

przypadku chcemy, żeby tekst wewnętrznej etykiety dało się zapisać, a tekst z pola tekstowego odczytywać.

Żeby w VB zdefiniować właściwość tylko do zapisu, musimy użyć modyfikatora `WriteOnly`. Właściwości tylko do zapisu muszą definiować specjalny blok kodu rozpoczynający się od słowa kluczowego `Set`. Taki blok kodu, nazywany **akcesorem**, wygląda jak procedura przyjmująca parametr zawierający wartość, która ma być ustawiana. Wartość jest wykorzystywana w kodzie do przeprowadzenia żądanych operacji — w przypadku właściwości `LabelText` będzie to ustawienie właściwości `Text` etykiety, tak jak poniżej:

Visual Basic *Plik: SmartBox.ascx (fragment)*

```
Public WriteOnly Property LabelText() As String
   Set(ByVal value As String)
      myLabel.Text = value
   End Set
End Property
```

Zakładając, że formatka korzysta z obiektu `SmartBox` o nazwie `mySmartbox`, możemy ustawić właściwość tekst etykiety w poniższy sposób:

Visual Basic

```
mySmartBox.LabelText = "Adres:"
```

Kiedy wykonywany jest powyższy kod, wywoływany jest akcesor `Set` właściwości `LabelText` z parametrem *`value`* o wartości `Adres:`. Akcesor użyje tej wartości do ustawienia właściwości `Text` kontrolki `Label`.

Innym akcesorem, którego możesz użyć podczas definiowania właściwości, jest `Get`. Umożliwia on zamiast zapisywania, odczytywanie wartości. Oczywiście, nie możesz dodać akcesora `Get` do wartości z modyfikatorem `WriteOnly`, ale jest on wymagany dla właściwości z modyfikatorem `ReadOnly`, takiej jak `Text`:

Visual Basic *Plik: SmartBox.ascx (fragment)*

```
Public ReadOnly Property Text() As String
   Get
      Text = myTextBox.Text
   End Get
End Property
```

Właściwość `Text` jest tylko do odczytu, ale nie musi tak być. Jeśli chciałbyś umożliwić formatkom wykorzystywanie kontrolki do ustawiania domyślnego tekstu w polu tekstowym, musiałbyś dodać akcesor `Set` i usunąć modyfikator `ReadOnly`.

Kiedy definiujesz właściwości w C#, nie musisz ustawiać żadnych specjalnych modyfikatorów, takich jak `ReadOnly` czy `WriteOnly`, dla właściwości tylko do zapisu i tylko do odczytu. Właściwość, która ma tylko akcesor `get`, będzie domyślnie uznawana jako tylko do odczytu.

C# *Plik: SmartBox.ascx (fragment)*

```
public string Text
{
   get
   {
      return myTextBox.Text;
   }
}
```

Podobnie, właściwość, która ma tylko akcesor `set`, będzie uznawana za tylko do zapisu:

C# *Plik: SmartBox.ascx (fragment)*

```
public string LabelText
{
   set
   {
      myLabel.Text = value;
   }
}
```

Korzystanie z kontrolek internetowych użytkownika

Po stworzeniu kontrolki użytkownika można się do nich odnosić ze strony ASP.NET za pomocą dyrektywy `Register`:

```
<%@ Register TagPrefix="prefix" TagName="name"'
      Src="source.ascx" %>
```

Dyrektywa `Register` wymaga trzech atrybutów:

`TagPrefix`
: Przedrostek dla kontrolki użytkownika umożliwiający grupowanie powiązanych kontrolek razem i uniknięcie konfliktu nazw.

`TagName`
: Nazwa znacznika kontrolki, która będzie używana podczas dodawania kontrolki do strony ASP.NET.

`Src`
: Ścieżka do pliku *.ascx* opisującego kontrolkę użytkownika.

Po zarejestrowaniu kontrolki, jej instancje tworzymy za pomocą znacznika *<TagPrefix:TagName>*.

Wypróbujmy przykład korzystający z kontrolki `SmartBox`. Stwórz nowy plik o nazwie *ControlTest. aspx* w katalogu *Learning* i umieść w nim następującą zawartość:

Plik: ControlTest.aspx (fragment)

```
<%@ Register TagPrefix="sp" TagName="SmartBox"
      Src="SmartBox.ascx" %>
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
```

```
   <head>
      <title>Tworzenie kontrolki internetowej użytkownika w ASP.NET</title>
   </head>
   <body>
      <form id="Form1" runat="server">
         <sp:SmartBox id="nameSb" runat="server" LabelText="Imię:" />
         <sp:SmartBox id="addressSb" runat="server"
               LabelText="Adres:" />
         <sp:SmartBox id="countrySb" runat="server"
               LabelText="Kraj:" />
         <sp:SmartBox id="phoneSb" runat="server"
               LabelText="Telefon:" />
      </form>
   </body>
</html>
```

Wczytanie tej strony spowoduje wygenerowanie wyniku, który widzieliśmy na rysunku 4.10.

Oczywiście jest to bardzo prosty przykład, ale łatwo możemy go rozbudować do innych celów. W powyższym kodzie możesz zobaczyć, że ustawiamy właściwość `LabelText` bezpośrednio w znaczniku kontrolki. Zamiast tego moglibyśmy odwoływać się do właściwości z poziomu kodu, tak jak poniżej:

Visual Basic *Plik: ControlTest.aspx (fragment)*

```
<script runat="server" language="VB">
   Protected Sub Page_Load()
      nameSb.LabelText = "Imię:"
      addressSb.LabelText = "Adres:"
      countrySb.LabelText = "Kraj:"
      phoneSb.LabelText = "Telefon:"
   End Sub
</script>
```

C# *Plik: ControlTest.aspx (fragment)*

```
<script runat="server" language="C#">
   protected void Page_Load()
   {
      nameSb.LabelText = "Imię:";
      countrySb.LabelText = "Adres:";
      addressSb.LabelText = "Kraj:";
      phoneSb.LabelText = "Telefon:";
   }
</script>
```

Strony wzorcowe

Strony wzorcowe są nową funkcjonalnością w ASP.NET, która wiele zmienia w sposobie składania formatek internetowych. Strony wzorcowe są podobne do kontrolek internetowych użytkownika, ponieważ składają się z HTML-a i innych kontrolek, mogą być rozszerzane o zdarzenia, metody i właściwości, a także nie mogą być wczytywane bezpośrednio przez użytkowni-

ków — zamiast tego są używane jak klocki, które można wykorzystać podczas projektowania struktury formatki internetowej.

Strona wzorcowa jest szablonem strony, który może być zastosowany do nadania wielu formatkom internetowym spójnego wyglądu. Na przykład może ona określać standardową strukturę zawierającą nagłówek, stopkę i inne elementy, które mają być wyświetlane na wielu formatkach internetowych aplikacji.

Pliki stron wzorcowych zawsze mają rozszerzenie *.master* i, podobnie do formatek internetowych i kontrolek internetowych użytkownika, obsługują pliki ukrywające kod. Wszystkie strony wzorcowe dziedziczą po klasie `System.Web.UI.MasterPage`.

Tworzenie struktury witryny za pomocą stron wzorcowych i kontrolek internetowych użytkownika daje duże możliwości łatwego modyfikowania i rozszerzania witryny. Jeśli witryna korzysta z tych funkcjonalności w dobrze zaplanowany sposób, modyfikowanie poszczególnych elementów strony lub jej funkcjonalności może być bardzo łatwe, ponieważ aktualizowanie strony wzorcowej lub kontrolki internetowej użytkownika wywołuje natychmiastowy efekt we wszystkich formatkach internetowych, które z nich korzystają.

Jak już wspomnieliśmy, strona wzorcowa jest tworzona za pomocą HTML-a i kontrolek, włączając w to specjalną kontrolkę `ContentPlaceHolder`. Jest ona zamarkowanym miejscem, które może być wypełnione zawartością odpowiednią do potrzeb każdej formatki internetowej korzystającej ze strony wzorcowej. Podczas tworzenia strony wzorcowej dołączamy do niej całą podstawową strukturę przyszłych stron, które będą z niej korzystać, włączając w to znaczniki `<html>`, `<head>` oraz `<body>`, i umożliwiamy stronie internetowej określenie zawartości wyświetlanej w zamarkowanych miejscach.

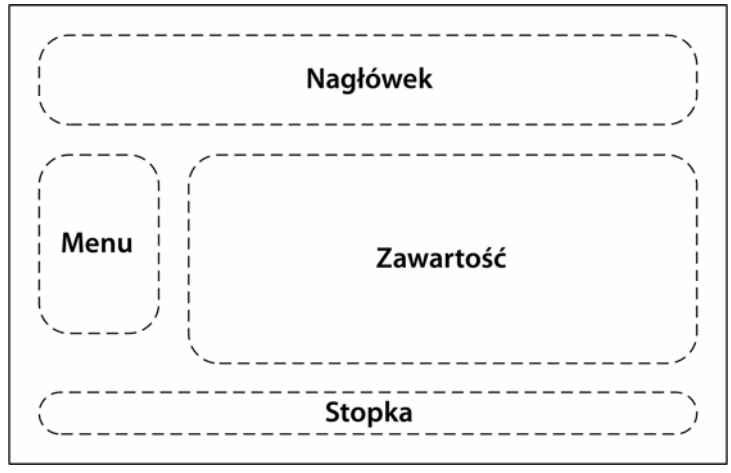
Zobaczmy na prostym przykładzie, w jaki sposób to działa. Przypuśćmy, że mamy witrynę z wieloma stronami zawierającymi standardowy nagłówek, stopkę i menu nawigacyjne, ułożone tak jak ramki na rysunku 4.11.

Jeśli wszystkie strony w witrynie mają ten sam nagłówek, stopkę i menu nawigacyjne, można umieścić te komponenty w stronie wzorcowej i tworzyć formatki internetowe, które na każdej stronie dopasowują tylko obszary zawartości. Zaczniemy tworzyć taką stronę w rozdziale 5., ale przyjrzyjmy się krótkiemu przykładowi.

Dla uproszczenia nie dołączymy menu — dołączymy tylko nagłówek, stopkę i miejsce na zawartość. W folderze *Learning* stwórz nowy plik o nazwie *FrontPages.master* i wpisz do niego poniższy kod:

Plik: FrontPages.master (fragment)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
```



Rysunek 4.11. *Prosty układ witryny internetowej*

```
   <head>
      <title>Strona główna</title>
   </head>
   <body>
      <form id="myForm" runat="server">
         <h1>Witamy w SuperSite Inc!</h1>
         <asp:ContentPlaceHolder id="FrontPageContent"
               runat="server" />
         <p>Copyright 2006</p>
      </form>
   </body>
</html>
```

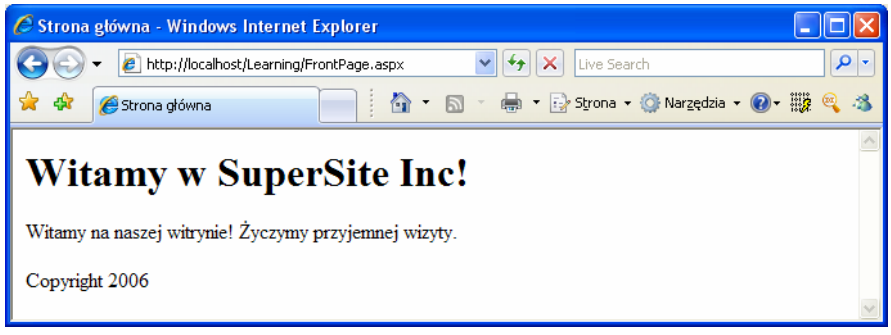
Strona wzorcowa wygląda niemalże jak formatka internetowa, z wyjątkiem jednej ważnej różnicy: ma pustą kontrolkę `ContentPlaceHolder`. Jeśli chcesz stworzyć formatkę internetową opartą na tej stronie wzorcowej, musisz tylko odnieść się do niej za pomocą dyrektywy `Page` i dodać kontrolkę `Content` z zawartością, którą chcesz umieścić.

Spróbujmy to zrobić. Stwórz formatkę internetową o nazwie *FronPage.aspx* i dodaj do niej poniższy kod:

Plik: FrontPage.master (fragment)

```
<%@ Page MasterPageFile="FrontPages.master" %>
<asp:Content id="myContent" runat="server"
      ContentPlaceHolderID="FrontPageContent">
   <p>
      Witamy na naszej witrynie! Życzymy przyjemnej wizyty.
   </p>
</asp:Content>
```

Wszystko gotowe! Po wczytaniu *FrontPage.aspx* przeglądarka wygeneruje wynik widoczny na rysunku 4.12.



Rysunek 4.12. *Korzystanie ze strony wzorcowej*

Chociaż powyższy przykład jest bardzo prosty, łatwo można zobaczyć otwierające się możliwości: możesz bez problemu stworzyć wiele formatek internetowych opartych na tym szablonie. W naszym przypadku strona wzorcowa zawiera tylko pojedynczą kontrolkę `ContentPlaceHolder`, ale mogłaby mieć ich więcej. Ponadto strona wzorcowa może definiować zawartość, domyślnie wyświetlaną wewnątrz kontrolki `ContentPlaceHolder` na stronach, których formatki nie mają elementu `Content` dla tej kontrolki.

Korzystanie z kaskadowych arkuszy stylów (CSS)

To oczywiste, że kontrolki ułatwiają ponowne używanie funkcjonalności w wielu miejscach. Nie sposób wyobrazić sobie łatwiejszego sposobu dodawania kalendarzy do wielu formatek internetowych niż wykorzystanie kontrolki internetowej serwera `Calendar`.

Jednak kontrolki nie rozwiązują problemu określania i zarządzania wizualnymi elementami witryny internetowej. Nowoczesne witryny internetowe, żeby zachować świeży wygląd, wymagają ciągłej aktualizacji. Ręczna edycja setek stron po to tylko, żeby zmienić na przykład kolor ramki, nie jest zbyt przyjemna, nie wspominając już o tym, że po takiej operacji trzeba sprawdzić, czy zmiany są spójne. Bywa to nawet jeszcze bardziej nieprzyjemne, jeśli klient chce poważniejszej aktualizacji, takiej jak poprzestawianie elementów na stronie.

Dobrą wiadomością jest to, że praca utrzymaniowa może być dużo łatwiejsza dzięki planowaniu na przyszłość, prawidłowemu zastosowaniu kilku podstawowych zasad i wydajnemu wykorzystaniu narzędzi, które oferuje HTML i ASP.NET.

Podstawowym narzędziem do tworzenia wizualnych stylów wielokrotnego użytku jest **CSS** (kaskadowy arkusz stylów). Początkowo HTML był projektowany do udostępniania prostej zawartości tekstowej i nie zwracano zbyt wielkiej uwagi na to, jak poszczególne elementy wyglądały w przeglądarce. HTML pozostawiał tę decyzję każdej przeglądarce, a ona z kolei dopasowywała wygląd do ograniczeń i możliwości komputera użytkownika. Chociaż możemy zmienić czcionkę, rozmiary kolory itp. za pomocą znaczników HTML, może to doprowadzić do skomplikowania kodu i spowodować, że później będzie bardzo trudno zmienić styl stron.

CSS umożliwia twórcom witryn internetowych tworzenie pojedynczych zestawów stylów w jednym miejscu i zastosowanie ich we wszystkich stronach witryny. Strony korzystające

z takiego arkusza stylów będą wyświetlać te same czcionki, kolory i rozmiary, dzięki czemu witryna będzie miała spójny wygląd. Bez względu na to, czy witryna zawiera trzy czy trzysta stron, zmiana stylu w arkuszu spowoduje, że natychmiast zmieni się wygląd wszystkich stron, które z niego korzystają.

Przyjrzyj się tematom i skórkom

ASP.NET 2.0 udostępnia dodatkowe możliwości dla twórców elementów wizualnych wielokrotnego użycia — **tematy** i **skórki**. Dowiesz się o nich więcej w rozdziale 5.

Typy stylów i arkuszy stylów

Są trzy różne sposoby powiązania stylów z elementami na stronie:

Zewnętrzne arkusz stylów

Dzięki umieszczeniu **reguł stylów** w zewnętrznym arkuszu stylów każda strona, która będzie z nich korzystać, musi być połączona tylko z jednym plikiem. Powoduje to, że aktualizacja wyglądu strony staje się dziecinnie prosta.

Żeby skorzystać z zewnętrznego arkusza stylów w formatce internetowej, należy umieścić w jej elemencie `head` następujący znacznik:

```
<link rel="stylesheet" type="text/css" href="plik.css" />
```

W powyższym przykładzie *plik.css* byłby plikiem tekstowym zawierającym reguły CSS, podobnym do poniższego:

```
a
{
   background: #ff9;
   color: #00f;
   text-decoration: underline;
}
```

Osadzony arkusz stylów

Reguły stylów możesz umieścić wewnątrz znaczników `<style type="text/css">` w sekcji `head` strony.

```
<style type="text/css">
   a
   {
      background: #ff9;
      color: #00f;
      text-decoration: underline;
   }
</style>
```

Problemem podczas korzystania z osadzonych stylów jest to, że nie można ich użyć ponownie w innej stronie, co znacznie utrudnia wprowadzanie zmian dotyczących całej witryny.

Wplatane reguły stylów

Wplatane style umożliwiają nadanie stylu pojedynczemu elementowi za pomocą atrybutu `style`. Na przykład za pomocą poniższego oznaczenia moglibyśmy nadać akapitowi czerwoną ramkę:

```
<p style="border-style: groove; color: red;">
   Copyright 2006
</p>
```

Jeśli dana reguła jest używana w osadzonym lub zewnętrznym arkuszu stylów, jej pierwsza część musi określać elementy, których ma dotyczyć ta reguła. Robimy to za pomocą **selektora**. W ASP.NET zazwyczaj używamy dwóch typów selektorów:

Selektor typu elementu

Selektor typu elementu jest stosowany dla każdej instancji określonego elementu. Na przykład jeśli chcielibyśmy zmienić kolor wszystkich nagłówków drugiego poziomu w dokumencie, użylibyśmy selektora dla elementu typu `<h2>`:

```
h2
{
   color: #369;
}
```

Klasy

Bezsprzecznie najpopularniejszym sposobem wykorzystania stylów w stronie jest nadanie każdemu elementowi atrybutu `class`, a następnie określenie stylu dla elementów posiadających daną wartość tego atrybutu. Na przykład poniższe oznaczenie wyświetla akapit, którego atrybut `class` ma wartość `fineprint`:

```
<p class="fineprint">
   Copyright 2006
</p>
```

Teraz, wiedząc, że wszystkie elementy z atrybutem `class` równym `fineprint` mają być wyświetlone drobnym drukiem, stworzymy regułę stylu, która zmniejszy tekst w tym akapicie oraz we wszystkich elementach z atrybutem `class="fineprint"`:

```
.fineprint
{
   font-family: Arial;
   fon-size: x-small;
}
```

Bez względu na to, czy tworzysz zewnętrzne arkusze, osadzone arkusze czy wplatane reguły stylów, deklaracje stylów używają tej samej składni.

Masz już podstawowe pojęcie o niektórych koncepcjach zawartych w CSS, przyjrzyjmy się więc różnym typom stylów, które mogą być używane w aplikacjach ASP.NET.

Właściwości stylów

Za pomocą stylów możesz zmieniać wiele różnych typów właściwości. Poniżej znajduje się lista najbardziej powszechnych typów właściwości:

Czcionka
Ta kategoria udostępnia formatowanie elementów tekstowych, włączając w to krój czcionki, rozmiary, ozdabianie, grubość, kolory itp.

Tło
Ta kategoria umożliwia dostosowanie tła obiektów i tekstu. Wartości w tej kategorii pozwalają zmieniać tło, włączając w to wybór, czy jako tło ma być używany kolor, czy rysunek i czy rysunek tła ma być powtarzany.

Blok
Ta kategoria umożliwia zmianę odległości pomiędzy akapitami, liniami tekstu, słowami i literami.

Pudełko
Ta kategoria umożliwia dopasowanie tabel. W zależności od potrzeb możesz zmieniać obramowanie, marginesy wewnętrzne, odległości i kolory w tabeli, wierszu lub komórce.

Ramki
Tak kategoria umożliwia wyświetlanie wokół elementów strony ramek o różnych kolorach, stylach i grubości.

Lista
Ta kategoria umożliwia dopasowanie sposobu, w jaki tworzone są numerowane i nienumerowane listy.

Pozycjonowanie
Zmiana pozycjonowania umożliwia dowolne przesuwanie i umiejscawianie znaczników i kontrolek na stronie.

Powyższe kategorie przedstawiają zarys aspektów projektowych, które mogą być zmieniane za pomocą CSS. W miarę postępów w lekturze tej książki wiele z tych typów właściwości stylów zostanie wyjaśnionych.

Właściwość CssClass

Po zdefiniowaniu klasy w arkuszu stylów (zewnętrznym bądź wewnętrznym) będziesz chciał ją powiązać z elementami w formatkach internetowych. Możesz je powiązać z kontrolkami internetowymi serwera w ASP.NET za pomocą właściwości `CssClass`. W większości przypadków wartość, jaką nadasz właściwości `CssClass`, będzie używana jako wartość atrybutu `class` elementu tworzonego przez kontrolkę.

Przyjrzyjmy się przykładowi. Najpierw w katalogu *Learning* stwórz plik o nazwie *Styles.css* i skopiuj do niego poniższy kod:

Plik: Styles.css

```
.title
{
   font-family: Arial, Helvetica, sans-serif;
   font-size: 19px
}
.dropdownmenu
{
```

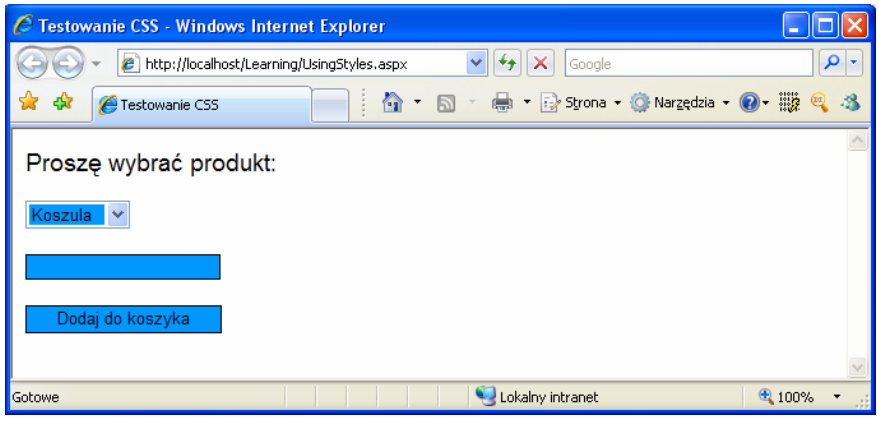
```
   font-family: Arial;
   background-color: #0099FF;
}
.textbox
{
   font-family: Arial;
   background-color: #0099FF;
   border: 1px solid
}
.button
{
   font-family: Arial;
   background-color: #0099FF;
   border: 1px solid
}
```

Następnie stwórz plik o nazwie *UsingStyles.aspx* zawierający poniższy kod:

Plik: UsingStyles.aspx (fragment)

```
<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.0 Strict//EN"
      "http://www.w3.org/TR/xhtml1/DTD/xhtml1-strict.dtd">
<html>
   <head>
      <title>Testowanie CSS</title>
      <link href="Styles.css" type="text/css" rel="stylesheet" />
   </head>
   <body>
      <form runat="server">
         <p class="title">Proszę wybrać produkt:</p>
         <p>
            <asp:DropDownList id="productsList"
                  CssClass="dropdownmenu" runat="server">
               <asp:ListItem Text="Koszula" selected="true" />
               <asp:ListItem Text="Kapelusz" />
               <asp:Listitem Text="Majtki" />
               <asp:ListItem Text="Skarpetki" />
            </asp:DropDownList>
         </p>
         <p>
            <asp:TextBox id="quantityTextBox" CssClass="textbox"
                  runat="server" />
         </p>
         <p>
            <asp:Button id="addToCartButton" CssClass="button"
                  Text="Dodaj do koszyka" runat="server"   />
         </p>
      </form>
   </body>
</html>
```

Po wczytaniu tej strony powinien wyświetlić się wynik, widoczny na rysunku 4.13.



Rysunek 4.13. *CSS w działaniu*

W następnym rozdziale nauczymy się korzystać z programu Visual Web Developer do tworzenia definicji CSS za pomocą prostego wizualnego interfejsu.

Podsumowanie

W tym rozdziale omówiliśmy formatki internetowe, kontrolki HTML serwera, kontrolki internetowe serwera, kontrolki internetowe użytkownika, strony wzorcowe oraz arkusze stylów. Wszystkie te elementy mogą być połączone w celu stworzenia struktur o dużych możliwościach dla witryn internetowych.

W następnym rozdziale zaczniemy tworzyć „prawdziwe" aplikacje internetowe, przekładając na praktykę większość teorii, którą dotąd poznałeś, a także korzystając z profesjonalnych środowisk programistycznych, które wykonają część pracy za Ciebie.