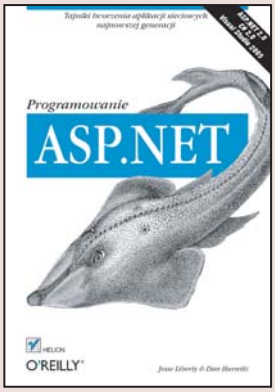
ASP.NET. Programowanie



Autorzy: Jesse Liberty, Dan Hurwitz
ISBN: 83-246-0361-1
Tytuł oryginału: Programming ASP.NET
Format: B5, stron: 920

ASP.NET to jedna z najpopularniejszych obecnie technologii tworzenia dynamicznych witryn WWW i aplikacji internetowych. Autor tej technologii — Microsoft — udostępnił twórcom doskonałe narzędzia programistyczne oraz bogaty zbiór bibliotek i kontrolek. Dzięki możliwościom pakietu Visual Studio 2005 i platformy programistycznej .NET 2.0 przygotowanie nawet najbardziej rozbudowanej aplikacji sieciowej przebiega błyskawicznie. Natomiast za pomocą gotowych elementów można zbudować formularze, zrealizować połączenia z bazą danych i zabezpieczyć aplikację przed nieautoryzowanym dostępem tak łatwo, że programista może skupić się wyłącznie na projektowaniu i tworzeniu logiki aplikacji.

Książka „ASP.NET. Programowanie" to podręcznik opisujący zasady tworzenia aplikacji i witryn internetowych przy użyciu języka C# 2.0, środowiska programistycznego Visual Studio 2005 i bibliotek .NET 2.0. Przedstawia środowisko Visual Studio 2005 i szczegółowo omawia koncepcję budowania aplikacji za pomocą kontrolek.
Po przeczytaniu tej książki stworzysz witryny internetowe w technologii ASP.NET 2.0, implementując w nich mechanizmy uwierzytelniania i personalizacji, dostępu do baz danych oraz usług sieciowych. Dowiesz się, jak budować własne usługi sieciowe i z nich korzystać, a także jak optymalizować wydajność aplikacji i wdrażać je, posługując się mechanizmami udostępnianymi przez platformę .NET 2.0.

- Interfejs użytkownika środowiska Visual Studio 2005
- Obsługa zdarzeń za pomocą kontrolek
- Przegląd kontrolek platformy .NET 2.0
- Tworzenie szkieletu witryny internetowej
- Wyszukiwanie błędów w kodzie, ich usuwanie i obsługa
- Weryfikacja danych z formularzy
- Dostęp do baz danych za pomocą ADO.NET
- Uwierzytelnianie użytkowników i personalizacja witryny
- Mechanizmy nawigacyjne
- Tworzenie i wykorzystywanie usług sieciowych
- Wdrażanie aplikacji

Stwórz wydajne i bezpieczne witryny internetowe, korzystając z ASP.NET 2.0

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

Spis treści

ROZDZIAŁ 4.

Kontrolki podstawowe

Informacje dotyczące kontrolek przedstawiliśmy już w rozdziale 3. Chociaż pokrótce wspomniano zarówno o serwerowych, jak i o klasycznych kontrolkach HTML, to jednak większość informacji dotyczyła kontrolek serwerowych ASP.NET, serca technologii ASP.NET.

Jak już o tym wcześniej wspomnieliśmy, kontrolki serwerowe są nazywane w różny sposób: „kontrolki ASP", „kontrolki ASP.NET", „kontrolki serwerowe ASP.NET", „kontrolki sieciowe" i „kontrolki serwera sieciowego". W tej książce używamy określenia „kontrolki serwerowe ASP.NET" lub „kontrolki serwerowe". Kiedy odwołujemy się do „kontrolek serwerowych", to z kontekstu powinno wynikać, czy mamy na myśli jedynie kontrolki serwerowe ASP.NET, czy również kontrolki serwerowe HTML.

Zagadnienia wspólne dla wszystkich kontrolek serwerowych ASP.NET zostały już przedstawione i obejmowały: zdarzenia, składnię, programowy dostęp do kontrolek w trakcie działania aplikacji (za pomocą właściwości `ID`) oraz użycie VS2005 do budowania witryny internetowej z wykorzystaniem kontrolek, jednakże w przypadku żadnej z kontrolek nie zagłębialiśmy się w szczegóły.

Ten rozdział dostarczy licznych szczegółów dotyczących wielu podstawowych kontrolek ASP.NET, między innymi: `TextBox`, `Button`, `CheckBox`, kontrolek `RadioButton`, list, tabel i grafiki. Dokonamy analizy wspólnych dla wielu kontrolek funkcji i właściwości oraz przyjrzymy się określonym szczegółom podstawowych kontrolek serwerowych ASP.NET dostępnych na platformie .NET.

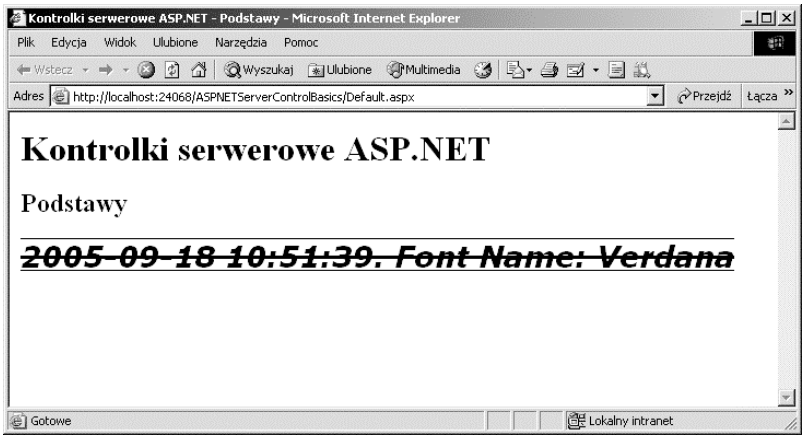
W kolejnym rozdziale przedstawimy wiele zaawansowanych kontrolek serwerowych będących częścią ASP.NET, na przykład kontrolek wyświetlania, `Wizard`, `FileUpload`, `AdRotator` i `Calendar`. W innych rozdziałach skupimy się na kontrolkach danych, kontrolkach sprawdzania poprawności, kontrolkach logowania i bezpieczeństwa itd.

Podstawy

W tym podrozdziale pokażemy dwa różne sposoby tworzenia prostej witryny internetowej. W pierwszym przypadku skorzystamy z prostego edytora tekstowego (*Notatnik*), natomiast za drugim razem wykorzystamy VS2005. Celem tego ćwiczenia jest zaprezentowanie, w jaki sposób można utworzyć witrynę internetową za pomocą dowolnego edytora tekstowego oraz jak znacznie łatwiej użyć do tego celu pakietu VS2005.

Będzie to jedyny przypadek w całej książce, gdy witryna internetowa zostanie utworzona bez użycia VS2005.

Niezależnie od wykorzystanej techniki, końcowa strona internetowa będzie podobna do przedstawionej na rysunku 4.1. Utworzona strona będzie prezentowała pewne właściwości, zdarzenia i metody, która są wspólne dla wszystkich kontrolek serwerowych ASP.NET.



Rysunek 4.1. Kontrolki serwerowe ASP.NET: Podstawy

W celu utworzenia tej strony internetowej bez korzystania z zalet VS2005, otwieramy program *Notatnik* lub inny ulubiony edytor tekstowy, w którym można utworzyć zwykły plik tekstowy (nie jest nim na przykład Microsoft Word). W pliku umieszczamy kod przedstawiony na listingu 4.1.

Listing 4.1. Plik ASPNETServerControlBasics-TextEditor.aspx

```
<%@ Page Language="C#" %>
<script runat="server">
   void lblTime_Init(object sender, EventArgs e)
   {
      lblTime.Font.Name = "Verdana";
      lblTime.Font.Size = 20;
      lblTime.Font.Underline = true;
      lblTime.Font.Bold = true;
      lblTime.Font.Italic = true;
      lblTime.Font.Overline = true;
      lblTime.Font.Strikeout = true;
      lblTime.Text = DateTime.Now.ToString() +
         ". Font Name: " +
         lblTime.Font.Name;
   }
</script>

<html>
   <body>
      <form id="form1" runat="server">
         <h2>Podstawy</h2>
         <asp:Label ID="lblTime" runat="server" OnInit="lblTime_Init" />
      </form>
   </body>
</html>
```

Zapisujemy plik pod nazwą *ASPNETServerControlBasics-TextEditor.aspx* w dowolnym katalogu, powiedzmy, *c:\websites*.

W celu łatwego przetworzenia strony przez ASP.NET na serwerze, musimy uzyskać do niej dostęp w przeglądarce przez *localhost*. Tak więc, zachodzi potrzeba utworzenia katalogu wirtualnego dla katalogu zawierającego plik strony internetowej.

`localhost` oznacza dla przeglądarki adres odnajdujący serwer WWW IIS na komputerze lokalnym.

Brak zainstalowanego serwera IIS na komputerze uniemożliwi zobaczenie tej strony, uruchomionej w przeglądarce spoza VS2005.

VS2005 nie wymaga instalacji IIS, ponieważ w domyślnym trybie jest w stanie samodzielnie zapewniać obsługę swoich stron. Istnieje możliwość wymuszenia na VS2005 korzystania z trybu IIS w celu uzyskania dostępu do stron internetowych. W tym celu, podczas tworzenia nowej witryny w oknie dialogowym *New Web Site*, trzeba zaznaczyć opcję *HTTP* z menu *Location*. Wówczas VS2005 automatycznie utworzy niezbędny katalog wirtualny.

Pełny opis różnych sposobów dostępu do witryn internetowych w VS2005 został przedstawiony w rozdziale 2.

Należy otworzyć *Zarządzanie komputerem*, klikając prawym przyciskiem myszy ikonę *Mój komputer* i wybierając opcję *Zarządzanie komputerem*. Opcjonalnie można również przejść do *Panelu Sterowania/Narzędzia administracyjne/Zarządzanie komputerem*. Trzeba rozwinąć węzeł *Aplikacje i usługi/Internetowe usługi informacyjne* i kliknąć prawym przyciskiem myszy pozycję *Domyślna witryna sieci Web*. Z rozwijanego menu wybieramy *Nowy/Katalog wirtualny* i podążamy za wskazówkami kreatora, używając `website` jako aliasu. Oczywiście, wskazujemy również położenie katalogu, w którym został umieszczony plik *.aspx*.

Następnie, po otworzeniu przeglądarki internetowej, wpisujemy następujący adres URL:

```
http://localhost/websites/ASPNETServerControlBasics-TextEditor.aspx
```

Po krótkiej chwili oczekiwania, w trakcie której ASP.NET przetworzy i zwróci stronę, przeglądarka wyświetli wygenerowany kod HTML.

Teraz można utworzyć dokładnie taka samą stronę, ale za pomocą VS2005. Otwieramy więc IDE i tworzymy nową witrynę internetową o nazwie *ASPNETServerControlBasic*. Wprowadzamy kilka elementów nagłówkowych HTML, a następnie przeciągamy i upuszczamy na stronie kontrolkę `Label`. Ukończona strona *default.aspx* będzie posiadała kod podobny do przedstawionego na listingu 4.2. Zwróćmy uwagę, jak połączenie techniki *przeciągnij i upuść* oraz listy *IntelliSense* powoduje, że proces ten jest znacznie mniej uciążliwy i bardziej odporny na powstawanie błędów.

Listing 4.2. Zawartość pliku Default.aspx w projekcie ASPNETServerControlBasics

```
<%@ Page Language="C#" AutoEventWireup="true"
   CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolki serwerowe ASP.NET - Podstawy</title>
</head>
```

```
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolki serwerowe ASP.NET</h1>
      <h2>Podstawy</h2>
       <asp:Label ID="lblTime" runat="server" OnInit="lblTime_Init"></asp:Label>
   </div>
   </form>
</body>
</html>
```

Instrukcje krok po kroku, w jaki sposób utworzyć tę stronę internetową za pomocą VS2005, technologii *przeciągnij i upuść* oraz *IntelliSense* znajdują się w ramce „Od ASPX do technologii *przeciągnij i upuść*".

Kiedy kontrolki zostaną już umieszczone na swoich miejscach, zajmiemy się utworzeniem obsługi zdarzeń dla zdarzenia `Init` kontrolki `Label lblTime`. Jeżeli nie otwarto widoku *Design view*, to należy teraz do niego przejść i zaznaczyć kontrolkę `lblTime`. W górnej części okna *Properties* pojawi się symbol zaznaczonej błyskawicy.

Kliknięcie tej błyskawicy spowoduje wyświetlenie wszystkich zdarzeń dostępnych dla danej kontrolki. Dwukrotnie klikamy komórkę obok zdarzenia `Init`. W obszarze roboczym zostanie otwarty plik ukrytego kodu *default.aspx.cs*, zawierający szkielet kodu w miejscu obsługi zdarzenia `lblTime_Init`. Umieszczony w nawiasie kursor pozwala na wpisywanie kodu C# programisty. Wprowadzamy te wiersze kodu przedstawionego na listingu 4.3, które zapisano pogrubioną czcionką.

Listing 4.3. Kod obsługi zdarzenia lblTime

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
   }


protected void lblTime_Init(object sender, EventArgs e)
   {
      lblTime.Font.Name = "Verdana";
      lblTime.Font.Size = 20;
      lblTime.Font.Underline = true;
      lblTime.Font.Bold = true;
      lblTime.Font.Italic = true;
      lblTime.Font.Overline = true;
      lblTime.Font.Strikeout = true;
      lblTime.Text = DateTime.Now.ToString() +
         ". Nazwa czcionki: " +
         lblTime.Font.Name;
   }
}
```

Od ASPX do technologii przeciągnij i upuść

W trakcie programowania z wykorzystaniem VS2005, jeśli tylko staje się to możliwe, warto używać listy *IntelliSense* oraz technologii *przeciągnij i upuść* zamiast ręcznego wpisywania kodu. Z drugiej jednak strony, uznajemy, że zamiast podawać za każdym razem żmudne instrukcje krok po kroku, dokładnie wskazujące sposób budowania każdej witryny, znacznie efektywniej i elastyczniej dla Czytelnika, a przy tym mniej podatne na błędy będzie pokazanie ukończonej aplikacji w formie zrzutu ekranu i/lub strony *.asapx*.

Praca Czytelnika nie powinna polegać na przepisywaniu całej strony *.aspx*, ale na odczytywaniu przedstawionej na niej zawartości i odtworzeniu tej zawartości na swoim komputerze przy użyciu najodpowiedniejszej techniki: *przeciągnij i upuść* lub ręcznej edycji kodu. Zaprezentujemy teraz, w jaki sposób można utworzyć aplikację spoglądając na rysunek 4.1 oraz odczytując listing 4.2.

Pierwszym zauważalnym elementem jest nazwa witryny internetowej (pokazana w adresie URL na rysunku 4.1) *ASPNETServerControlBasics*. Tworzymy więc nową witrynę internetową i nadajemy jej nazwę *ASPNETServerControlBasics*. Następnie otwieramy stronę *default.aspx* tej witryny.

Należy zwrócić uwagę, analizując listing 4.2, że tytuł został ustalony (wiersz w listingu: *Kontrolki serwerowe ASP.NET — podstawy*). Wprowadzamy więc ten tytuł w pliku *default.aspx*, dokonując bezpośredniej edycji pliku.

Następnie zwracamy uwagę na dwa elementy nagłówków HTML: `h1` i `h2`. One również zostały wpisane bezpośrednio w widoku *Source view* pliku *default.aspx*, ponieważ pasek narzędziowy nie zawiera żadnych kontrolek nagłówków HTML. W tym miejscu jednakże po wpisaniu otwierającego znaku nawiasu z pomocą przychodzi lista *IntelliSense*, która wyświetla rozwijaną listę wszystkich elementów możliwych do prawidłowego umieszczenia wewnątrz tej pary nawiasów. W trakcie wpisywania kolejnych znaków, lista dynamicznie prezentuje dostępne możliwości.

Kolejny krok to przeciągnięcie na stronę kontrolki `Label` z sekcji *Standard* paska narzędziowego. Można to zrobić zarówno w widoku *Design view*, jak i *Source view*. Zaletą korzystania z tej pierwszej możliwości jest trochę łatwiejsze generowanie obsługi zdarzeń.

Po umieszczeniu wskaźnika myszy na kontrolce `Label`, ponownie zarówno w widoku *Design view*, jak i *Source view*, okno *Properties* wyświetli bieżące właściwości kontrolki. Zmieniamy wartość właściwości `ID` z domyślnej `Label1` na `lblTime`. Usuwamy zawartość właściwości `Text`.

Alternatywną możliwością jest bezpośrednie wpisanie w widoku *Source view* deklaracji kontrolek bez używania okna *Properties*. *IntelliSense* ponownie okazuje się pomocne, wyświetlając listę wszystkich możliwych argumentów (właściwości) i zdarzeń. Jeżeli dokonamy zmian w dowolnym widoku lub oknie *Properties*, wówczas pozostałe widoki i okna natychmiast uwzględnią wprowadzone zmiany.

Zaletą przedstawienia Czytelnikowi pliku *.aspx* jest to, że w ten sposób staje się jasne, jak powinien wyglądać ukończony kod. Dzięki temu można budować kod albo ręcznie, albo za pomocą technologii *przeciągnij i upuść* i/lub okna *Properties* — wybór należy do programisty. W rzeczywistości, jeżeli Czytelnik nie ma ochoty na wykonywanie czynności w żaden z przedstawionych sposobów, to może pobrać kompletne kody źródłowe z witryny wydawnictwa Helion poświęconej tej książce, jak to zostało opisane we wstępie.

Uruchamiamy stronę, naciskając klawisz *F5* bądź wybierając z menu *Debug* opcję *Start*. Uzyskany wynik powinien być podobny do tego pokazanego na rysunku 4.1.

Te dwa przykłady demonstrują zastosowanie kontrolki `Label`, obsługę zdarzenia oraz właściwości ustawione kontrolce.

Przedstawiona prosta strona internetowa zawiera statyczny tekst oraz kontrolkę serwerową `Label`, której przypisano identyfikator `id` o wartości `lblTime`. Identyfikator pozwala na odwołanie się do tej kontrolki z dowolnego miejsca w kodzie.

Bardziej interesujący jest atrybut `onInit`, który definiuje obsługę zdarzeń dla zdarzenia `Init`. Zdarzenie `Init`, będące elementem klasy `Control`, zostaje wywołane w trakcie inicjalizacji kontrolki. Jest to pierwszy krok w cyklu życia każdej kontrolki. Wszystkie `WebControls`, jako że wywodzą się z klasy `Control`, posiadają zdarzenie `Init`.

W listingach 4.1 i 4.3, zdarzenie `Init` jest obsługiwane przez metodę o nazwie `lblTime_Init`, zdefiniowaną, odpowiednio, w bloku kodu na początku pliku *.aspx* lub w pliku ukrytego kodu. Metoda `lblTime_Init` ustawia kilka właściwości czcionki etykiety (`Name`, `Size` itd.) oraz ustawia wartość właściwości `Text`. Wartość właściwości `Text` jest połączeniem aktualnej daty i czasu, właściwym ciągiem znakowym oraz nazwą użytej czcionki. Ponieważ `DateTime.Now` jest typem `DateTime`, to musi zostać skonwertowany w kodzie C# do postaci ciągu znakowego.

Wynik działania tego kodu, pokazany na rysunku 4.1, nie jest ładny, ale kształcący. Na rysunku pokazano, w jaki sposób można stosować kilka atrybutów tekstowych — pogrubienie, kursywa, podkreślenie i skreślenie — do formatowania tekstu na etykiecie.

Czcionki (obiekt `Font`) wymagają specjalnej wzmianki, ponieważ zawierają podwłaściwości, które zostały przedstawione w tabeli 4.1. Kiedy podwłaściwości są używane w HTML, dostęp do nich w kodzie jest deklaracyjny w formie:

```
Font-Italic
```

Tabela 4.1. Podwłaściwości obiektu Font

Podwłaściwość	Typ	Wartości	Opis
`Bold`	Boolean	`true`, `false`	Powoduje pogrubienie tekstu. Wartością domyślną jest `false`.
`Italic`	Boolean	`true`, `false`	Tekst zostaje napisany kursywą. Wartością domyślną jest `false`.
`Name`	String	`Verdana`, `Curier` itd.	Automatycznie uaktualnia pierwszy element właściwości `Names`. Czcionka musi być zainstalowana i dostępna dla przeglądarki klienta.
`Names`	String	`Times` itd.	Uporządkowana tablica nazw czcionek. Właściwość `Name` zostaje automatycznie uaktualniona pierwszym elementem tablicy.
`Strikeout`	Boolean	`true`, `false`	Umieszcza linię biegnącą poprzez tekst (skreślenie). Wartością domyślną jest `false`.
`Underline`	Boolean	`true`, `false`	Umieszcza linię pod tekstem (podkreślenie). Wartością domyślną jest `false`.
`Overline`	Boolean	`true`, `false`	Umieszcza linię nad tekstem. Wartością domyślną jest `false`. Nie jest generowana w przeglądarkach typu *downlevel*.
`Size`	`FontUnit` lub String	`Small`, `Smaller`, `Large`, `Larger` lub liczba całkowita oznaczająca wielkość w punktach.	Stosuje nazwy wielkości lub liczby całkowite oznaczające wielkość czcionki. Nazwy wielkości działają deklaracyjnie tylko jako atrybuty kontrolki.

W blokach kodu podwłaściwości są dostępne programowo w taki sposób:

```
Font.Italic
```

W przypadku używania liczby punktów zamiast nazw wielkości dla określenia wielkości czcionki, warto zwrócić uwagę, że wersja C# `FontUnit` dostarcza operatora konwersji, który pobiera liczbę całkowitą `int`, a tworzy `FontUnit`. Dlatego też można skorzystać z następującego zapisu:

```
lblTime.Font.Size = 20;
```

Można to zrobić bez wyraźnego posiadania egzemplarza obiektu `FontUnit` (co jest wymagane, przykładowo, w VS2005).

Przypominamy, że w tym rozdziale analizujemy różne kontrolki dostępne dla programisty. Wyjaśnimy, w jaki sposób używać każdej z nich oraz przedstawiamy przykłady ich wykorzystania.

Kontrolka Label

Kontrolka `Label` jest używana do wyświetlania tekstu. Właściwość `Text` kontrolki `Label` zawiera wyświetlany przez nią ciąg tekstowy. Właściwość `Text` jest jedyną właściwością kontrolki `Label`, która nie jest dziedziczona z klas `Control` lub `WebControl`. Kontrolka `Label` nie posiada zdarzeń lub metod, które nie wywodzą się z klas `Control` lub `WebControl`.

W poprzednich rozdziałach opisaliśmy kontrolkę `Label` w kilku przykładowych kodach. Właściwości `Text` i `Font` kontrolki `Label` mogą zostać ustawione programowo (jak przedstawiono to w listingach 4.1 i 4.3) lub deklaracyjnie.

Kontrolka TextBox

Kontrolka `TextBox` może być wykorzystywana zarówno do wprowadzania danych przez użytkownika, jak i jedynie do wyświetlania tekstu. Możliwości konfiguracyjne kontrolki obejmują przedstawienie jej w postaci jednego wiersza lub kilku wierszy, a nawet do przyjmowania haseł. Jeżeli zostanie zastosowana w postaci wielowierszowej, słowa będą automatycznie przenoszone do nowego wiersza, dopóki właściwość `Wrap` nie zostanie ustawiona jako `false`. Tekst zawarty w kontrolce może przekraczać wielkość kontrolki wyświetlanej na ekranie. Kontrolki `TextBox`, `DropDownList`, `Label` i inne kontrolki tekstowe implementują interfejs `ITextControl`, który jest nowością w ASP.NET 2.0. Wspomniany interfejs posiada pojedynczą właściwość `Text`, która jest wizualną zawartością kontrolki.

W tabeli 4.2 zostały przedstawione najczęściej wykorzystywane właściwości, które można określać dla kontrolki `TextBox`. Jeżeli jakikolwiek z tych argumentów zostanie pominięty w kontrolce, wówczas zostanie zastosowana jego domyślna wartość.

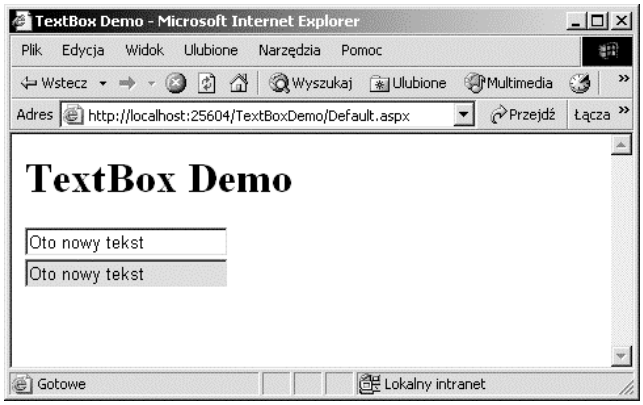
Oprócz zdarzeń dziedziczonych z klasy `WebControl`, takich jak `Init`, `Load` i `PreRender`, kontrolka `TextBox` wywołuje również zdarzenie `TextChanged`, gdy zawartość pola tekstowego zostanie zmieniona, a kontrolka utraci aktywność. Nie jest to zdarzenie powodujące ponowne wysłanie żądania do serwera, chyba że wartość właściwości `AutoPostBack` wynosi `true`.

Tabela 4.2. Niektóre właściwości określone dla kontrolki TextBox

Nazwa	Typ	Pobierz	Ustaw	Wartości	Opis
`AutoPostBack`	Boolean	x	x	`true`, `false`	Określa, czy zmiana zawartości kontrolki przez użytkownika powoduje ponowne wysłanie żądania do serwera. W przypadku wartości `false`, ponowne komunikowanie się z serwerem nie wystąpi, dopóki strona nie zostanie odświeżona albo przyciskiem, albo przez inną kontrolkę z właściwością `AutoPostBack` ustawioną na `true`. Wartością domyślną jest `false`.
`Columns`	Int32	x	x	`0`, `1`, `2` itd.	Szerokość pola tekstowego w znakach. Wartość domyślna — `0` — wskazuje, że właściwość nie jest ustawiona.
`MaxLength`	Int32	x	x	`0`, `1`, `2` itd.	Maksymalna dozwolona liczba znaków. Jeżeli wartość `MaxLength` jest większa od `Columns`, wówczas jedynie fragment ciągu znaków zostanie wyświetlony bez używania klawiszy *Home*, *End* i strzałek. Wartością domyślną jest `0`, oznaczające brak ograniczenia w liczbie znaków możliwych do wprowadzenia do pola tekstowego.
`ReadOnly`	Boolean	x	x	`true`, `false`	Wartość `true` oznacza, że zawartość kontrolki nie może zostać zmieniona przez użytkownika, chociaż wciąż może zostać zmieniona programowo. Wartością domyślną jest `false`.
`Rows`	Int32	x	x	`0`, `1`, `2` itd.	Liczba wierszy tekstu w wielowierszowym polu tekstowym. Wartością domyślną jest `0`, co oznacza brak ograniczenia w liczbie wierszy.
`Text`	String	x	x		Zawartość kontrolki `TextBox`.
`TextMode`	TextBox-Mode	x	x	`SingleLine`, `MultiLine`, `Password`	`SingleLine` jest wartością domyślną i wyświetla pojedynczy wiersz tekstu. `MultiLine` pozwala na wyświetlanie wielu wierszy tekstu oraz pionowego paska przewijania, nawet dla `Rows = 1`. Po osiągnięciu szerokości pola, tekst jest automatycznie przenoszony do nowego wiersza. Klawisz *Enter* wprowadza znak powrotu karetki/nowego wiersza. Kliknięcie myszą lub klawisz *Tab* powodują wycofanie aktywności z pola i inicjalizację ponownego zapytania do serwera, jeśli `AutoPostBack` ma wartość `true`. `Password` wyświetla gwiazdki jako zawartość pola, a przy odświeżeniu strony czyści pole. Wartość nie uwzględnia wielkości liter.
`Validation-Group`	String	x	x		Określa, której grupy sprawdzania poprawności (jeżeli w ogóle), ta kontrolka jest elementem. Sprawdzanie poprawności zostanie omówione w rozdziale 8.
`Wrap`	Boolean	x	x	`true`, `false`	Wskazuje, czy tekst wewnątrz wielowierszowego pola tekstowego powinien być przenoszony do nowego wiersza. W przypadku wartości `false`, pole tekstowe będzie posiadało poziomy suwak przewijania. Wartością domyślną jest `true`.

Kiedy kontrolka `TextBox` zostaje zadeklarowana w pliku z treścią (*.aspx* lub *.ascx*), wówczas metoda obsługi zdarzenia `TextChanged` zostaje określona atrybutem `OnTextChanged`. `TextChanged` jest domyślną obsługą zdarzenia, utworzoną przez VS2005 po dwukrotnym kliknięciu kontrolki `TextBox` w widoku *Design view*. Obsługa zdarzenia jest przekazywana standardowym argumentem `EventArgs`.

Poniższy przykład zademonstruje podstawowe wykorzystanie kontrolki `TextBox` z uwzględnieniem obsługi zdarzenia `TextChanged`. Na omawianej, przykładowej stronie znajdują się dwa pola tekstowe: pierwsze służące do wprowadzania tekstu, natomiast drugie w postaci kontrolki tylko do odczytu, wyświetla zawartość pierwszego pola. Ukończona strona, po zmianie tekstu w polu wejściowym, powinna być podobna do pokazanej na rysunku 4.2.



Rysunek 4.2. Aplikacja TextBoxDemo

W VS2005 tworzymy nową witrynę internetową i nadajemy jej nazwę *TextBoxDemo*. Następnie na stronę przeciągamy dwie kontrolki `TextBox`. Właściwość `ID` pierwszej kontrolki powinna zostać ustawiona jako `txtInput`, natomiast drugiej — na `txtEcho`. W przypadku pola `txtInput` właściwość `AutoPostBack` ustawiamy jako `true`, tak więc formularz automatycznie odświeży zawartość strony, jeżeli zawartość kontrolki ulegnie zmianie. Kontrolce `txtEcho` ustawiamy wartość `LightGray` właściwości `BackColor`, a właściwość `ReadOnly` otrzymuje wartość `true`.

Zawartość pliku przedstawionego na listingu 4.4 obejmuje również atrybut `OnTextChanged`, którego dodanie zostanie omówione w kolejnym akapicie.

Listing 4.4. Plik Default.aspx aplikacji TextBoxDemo

```
<%@ Page Language="C#" AutoEventWireup="true"
   CodeFile="Default.aspx.cs" Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Demo kontrolki TextBox</title>
</head>
```

```
<body>
   <form id="form1" runat="server">
   <div>
      <h1>TextBox Demo</h1>
      <asp:TextBox ID="txtInput" runat="server"
         AutoPostBack="true"
         OnTextChanged="txtInput_TextChanged" >
      Wprowadź dowolny tekst
      </asp:TextBox>
      <br />
      <asp:TextBox ID="txtEcho" runat="server"
         BackColor="LightGray"
         ReadOnly="True">
      </asp:TextBox>
   </div>
   </form>
</body>
</html>
```

W celu łatwego utworzenia domyślnej obsługi zdarzenia pola `txtInput` wystarczy przejść do widoku *Design view* i dwukrotnie kliknąć kontrolkę `TextBox`. VS2005 doda do deklaracji `txtInput` w pliku z treścią atrybut `OnTextChanged` i otworzy plik ukrytego kodu, który będzie zawierał szkielet obsługi zdarzenia. W tym miejscu musimy wpisać pogrubiony wiersz z kodu przedstawionego na listingu 4.5.

Listing 4.5. Obsługa zdarzenia TextChanged w aplikacji TextBoxDemo

```
protected void txtInput_TextChanged(object sender, EventArgs e)
   {
      txtEcho.Text = txtInput.Text;
   }
```

Przy pierwszym uruchomieniu aplikacji pole tekstowe `txtInput` będzie zawierało komunikat „Wprowadź dowolny tekst". Kiedy zawartość kontrolki zostanie zmieniona i użytkownik opuści pole tekstowe, wtedy nastąpi wywołanie zdarzenia `TextChanged`. Określona w atrybucie `OnTextChanged` obsługa zdarzeń `txtInput_TextChanged` zostanie uruchomiona i wypełni pole tekstowe `txtEcho`.

Kontrolka HiddenField

Ukryte pola są częstym trikiem programistów HTML, pozwalającym na przenoszenie informacji między stronami, jeśli te informacje nie powinny być widoczne dla użytkownika.

Łatwiejszym i bardziej eleganckim sposobem realizacji tego zadania jest użycie jednego z mechanizmu stanów dostarczanych przez platformę .NET (pełna analiza stanów zostanie przedstawiona w rozdziale 6.), jednakże czasami jest to niemożliwe, być może z powodu wydajności, przepustowości łącza bądź kwestii bezpieczeństwa (w rzeczywistości, wydajność aplikacji i przepustowość łącza są jak dwie strony tej samej monety).

W klasycznych stronach HTML czasami stosuje się poniższy fragment kodu w celu implementacji ukrytego pola:

```
<input type="hidden" value="foo">
```

ASP.NET używa ukrytego pola w implementacji stanu widoku (*view state*). Można się o tym przekonać, analizując kod źródłowy, który został wygenerowany w przeglądarce. W celu wyświetlenia kodu źródłowego w przeglądarce Internet Explorer należy wybrać polecenie *Źródło* z menu *Widok* (inne przeglądarki posiadają analogiczne opcje). W kodzie źródłowym można zauważyć fragmenty podobne do poniższego, w którym atrybut `value` zawiera wszystkie informacje zapisane w stanie widoku.

```
<input type="hidden" name="__VIEWSTATE"
   value="/wEPDwUJLOCHlBR...YfL+BDX7xhMw=" />
```

W celu skorzystania z zalet przetwarzania po stronie serwera, można dokonać konwersji kontrolki na kontrolkę serwerową HTML, dodając atrybuty `id` i `runat`:

```
<input type="hidden" value="foo" id="myHiddenControl" runat="server">
```

Kontrolka ASP.NET `HiddenField` jest najlepszym rozwiązaniem ze wszystkich przedstawionych opcji (zakładamy, że istnieje jakiś istotny powód niepozwalający na użycie możliwości stanów ASP.NET) ponieważ zapewnia następujące możliwości:

- spójność programową;
- łatwy dostęp do wartości `Value`, która przechowuje wartość obsługiwaną przez kontrolkę;
- właściwość `ClientID` dziedziczoną z klasy `Control`, która zapewnia samej kontrolce atrybut `id`;
- dostęp do zdarzenia `ValueChanged` (domyślne zdarzenie kontrolki `HiddenField` w VS2005).

Kontrolka `HiddenField` jest nowością w ASP.NET 2.0.

Zdarzenie `ValueChanged` jest wywoływane w trakcie odświeżenia strony, gdy wartość `Value` kontrolki różni się od poprzedniego odświeżenia strony. Zdarzenie nie powoduje wysłania żądania do serwera — w przeciwieństwie do większości kontrolek typu *non-postback*, kontrolka `HiddenField` nie posiada właściwości `AutoPostBack` do wymuszenia wysłania natychmiastowego żądania do serwera. Podobnie jak w przypadku wszystkich kontrolek typu *non-postback* (przedstawionych w poprzednim rozdziale), zdarzenie będzie buforowane, dopóki formularz nie zostanie odświeżony przez inną kontrolkę. Wówczas zdarzenie zostanie obsłużone przez serwer.

Wymienione funkcje zostaną przedstawione na listingu 4.6, prezentującego plik o nazwie *HiddenFieldDemo*. Plik z treścią został również umieszczony w tym listingu. Oprócz wyświetlania kodu HTML, w którym uwzględniono nagłówek `<h2>`, informujący o odświeżeniu strony, formularz zawiera kontrolkę `TextBox`. Kontrolka ta służy do wprowadzenia nowej wartości dla kontrolki `HiddenField`. Na stronie znajduje się również przycisk HTML (zastosowany do uruchamiania funkcji po stronie klienta, bez powodowania ponownego wysłania żądania do serwera) oraz przycisk ASP.NET (wymuszający ponowne wysłanie żądania do serwera). Kontrolka `Label` wyświetla zawartość ukrytego pola.

Listing 4.6. Plik Default.aspx witryny internetowej HiddenFieldDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
   Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka HiddenField</title>
</head>

<script language=javascript>
   function ChangeHiddenValue()
   {
      alert('Wprowadzanie ChangeHiddenValue');

      var hdnId = '<%=hdnSecretValue.ClientID%>'
      var hdn = document.getElementById(hdnId);

      var txt = document.getElementById('txtSecretValue');

      hdn.value = txt.value;
      alert('Wartość została zmieniona');
   }
</script>

<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka HiddenField</h1>
      <h2>Strona została odświeżona o <% =DateTime.Now.ToString() %>.</h2>
      <asp:HiddenField ID="hdnSecretValue" runat="server"
         OnValueChanged="hdnSecretValue_ValueChanged" />
      Wprowadź tajną wartość:
      <asp:TextBox ID="txtSecretValue" runat="server" />
      <br />
      <br />
      <input type=button value="Zmiana ukrytej wartości"
         onclick="ChangeHiddenValue()" />
      <asp:Button ID="btnPost" runat="server" Text="Odśwież" />
      <br />
      <br />
      <asp:Label ID="lblMessage" runat="server" Text=""/>
   </div>
   </form>
</body>
</html>
```

Przycisk HTML wywołuje funkcję `ChangeHiddenValue`, która jest zawarta wewnątrz bloku skryptu JavaScript, zaznaczonego w listingu pogrubionymi wierszami. Funkcja posiada dwie metody `alert` pomocne w trakcie usuwania błędów. Mogą one zostać pominięte lub umieszczone w komentarzach.

Funkcja `ChangeHiddenValue` pokazuje dwie równoznaczne drogi pobrania odniesienia do kontrolki na stronie. Każdy z tych sposobów korzysta z metody JavaScript `getElementById`, która zwraca odniesienie do pierwszego znalezionego na stronie elementu, posiadającego podany atrybut `ID`.

W przypadku pierwszej z przedstawionych technik, kontrolka `HiddenField` otrzymuje odniesienie dzięki swojej właściwości `ClientID`. Odkąd ta właściwość wywodzi się z klasy `Control`, stała się dostępna dla wszystkich kontrolek serwerowych ASP.NET. Odpowiedni fragment kodu, wykonywany po stronie serwera, jest wywoływany z wewnątrz funkcji JavaScript przez umieszczenie go między znacznikami `<%= %>`.

W przypadku drugiej z technik, atrybut `ID` kontrolki `TextBox` jest przekazywany do metody `getElementById`.

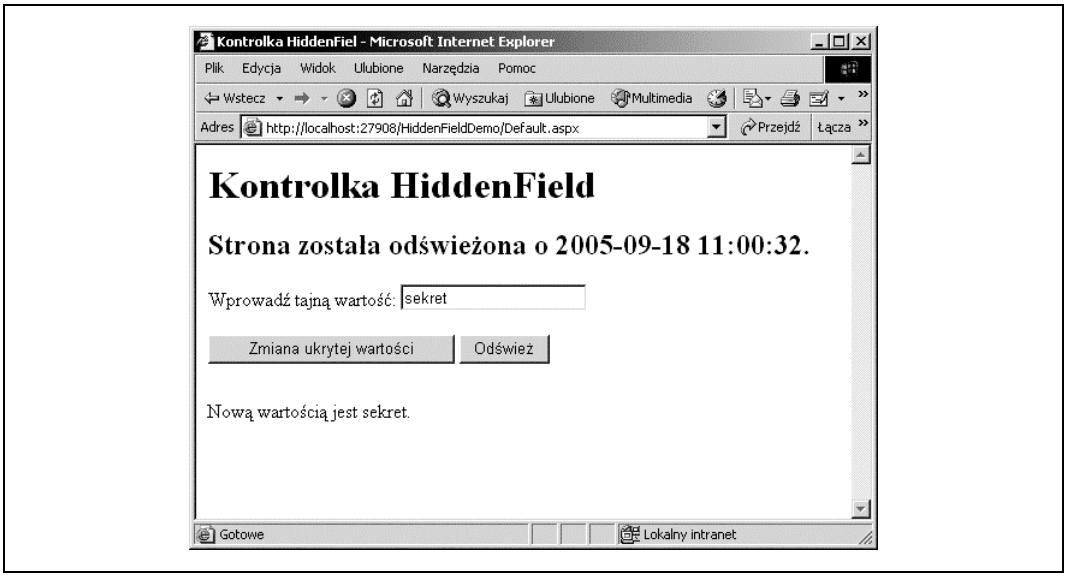
Zdarzenie `HiddenField ValueChanged` jest obsługiwane przez metodę po stronie serwera `hdnSecretValue_ValueChanged`, jak jest to wskazane przez pogrubiony na listingu 4.6 atrybut `OnValueChanged`. Obsługa zdarzenia umieszczona w pliku ukrytego kodu *default.aspx.cs* została przedstawiona na listingu 4.7. Pozostała część pliku ukrytego kodu nie została w tym miejscu zaprezentowana, ponieważ jest standardowym kodem umieszczanym przez VS2005.

Listing 4.7. Fragment pliku default.aspx.cs witryny HiddenFieldDemo

```
protected void hdnSecretValue_ValueChanged(object sender, EventArgs e)
{
   HiddenField hdn = (HiddenField)sender;
   lblMessage.Text = "Nowa wartość wynosi " + hdn.Value + ".";
}
```

Pierwszy wiersz w `hdnSecretValue_ValueChanged` otrzymuje odniesienie do kontrolki, która wywołała zdarzenie. Jest ono przechowywane w argumencie `sender` i zgłasza `sender` jako `HiddenField`. Następnie właściwość `Value` obiektu `HiddenField` jest używana do ustawienia właściwości `Text` etykiety `lblMessage`.

Po wprowadzeniu tajnej wartości i naciśnięciu przycisku *Odśwież*, zobaczymy stronę internetową podobną do pokazanej na rysunku 4.3. Przycisk *Odśwież* nie posiada obsługi zdarzenia, ponieważ nie potrzebujemy jej do wykonywania żadnej funkcji, innej niż powodującej ponowne zapytanie serwera.



Rysunek 4.3. Aplikacja HiddenFieldDemo

Kontrolki Button

Kontrolki *Button* przekazują formularz z powrotem do serwera, rozpoczynając tym samym proces przetwarzania po stronie serwera. Istnieją trzy typy kontrolek ASP.NET `Button`, wszystkie należą do przestrzeni nazw `System.Web.UI.WebControls`:

`Button`
: Standardowy przycisk.

`LinkButton`
: Kontrolka `LinkButton` jest czymś w rodzaju połączenia standardowego przycisku oraz kontrolki `HyperLink` (zostanie opisana w kolejnym podrozdziale). Kontrolka `LinkButton` jest wyświetlana użytkownikowi jako łącze (na przykład kolorowy i podkreślony tekst).

`ImageButton`
: Kontrolka `ImageButton` wykonuje te same funkcje, co standardowy przycisk. Od standardowego przycisku różni się tym, że w przeglądarce jest wyświetlana jako element graficzny. Kontrolka `ImageButton` nie posiada właściwości `Text`, ale zawiera atrybut `AlternateText`, który wyświetla odpowiedni tekst w przeglądarkach tekstowych.

  Dodatkowo, obsługa zdarzeń stosuje argument zdarzenia `ImageClickEventArgs`, który jest odmienny od obsługi zdarzeń kontrolek `Button` i `LinkButton`. Ten argument zdarzenia posiada dwa pola (nie zostały zaprezentowane w przedstawionym przykładzie) zawierające współrzędne X i Y miejsca, w którym użytkownik kliknął obrazek. Wspomniane pola mogą zostać wykorzystane przez programistę do implementacji jego własnej funkcjonalności mapy graficznej.

Oprócz wszystkich właściwości, metod i zdarzeń dziedziczonych z klasy `WebControl`, wszystkie trzy przyciski zapewniają obsługę dwóch następujących zdarzeń:

`Click`
: To zdarzenie jest wywoływane po kliknięciu kontrolki, gdy do przycisku nie zostanie przypisana żadna nazwa polecenia (to znaczy do właściwości `CommandName` kontrolki `Button` nie przypisano żadnej wartości). Metoda jest przekazywana argumentem typu `EventArgs`.

`Command`
: To zdarzenie jest wywoływane po kliknięciu kontrolki, gdy do przycisku przypisano nazwę polecenia (to znaczy do właściwości `CommandName` kontrolki `Button` przypisano nazwę polecenia). Zdarzenie jest przekazywane argumentem typu `CommandEventArgs`, które posiada dwa następujące elementy:

  `CommandName`
  : Nazwa polecenia.

  `CommandArgument`
  : Opcjonalny argument polecenia.

Wszystkie trzy typy kontrolki `Button` implementują interfejs `IButtonControl`, który jest nowością w ASP.NET 2.0. Interfejs wymaga zdarzeń `Click` i `Command` oraz takich właściwości jak `Text` i `CausesValidation`, jak również innych, które zostaną wkrótce opisane. To dzięki interfejsowi `IButtonControl` kontrolka działa jak przycisk.

Kolejny przykład *ButtonDemo* tworzy stronę internetową zawierającą kontrolki `Button`, `LinkButton` i `ImageButton`. Każdy z przycisków wykonuje to samo zadanie: przekazuje kontrolę innej stronie internetowej. Plik z treścią został przedstawiony na listingu 4.8.

Listing 4.8. Strona Default.aspx witryny internetowej ButtonDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Przyciski</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolki Button</h1>
      <asp:Button ID="btnLink" runat="server"
         Text="Łącze do strony docelowej"
         ToolTip="Kliknij tutaj, aby przejść do strony docelowej."
         OnClick="btnLink_Click" />
      <asp:ImageButton ID="imgLink" runat="server"
         AlternateText="Łącze do strony docelowej"
         ImageUrl="Dan at Vernal Pool.jpg"
         ToolTip="Kliknij tutaj, aby przejść do strony docelowej."
         OnClick="imgLink_Click" />
      <asp:LinkButton ID="lnkLink" runat="server"
         ToolTip="Kliknij tutaj, aby przejść do strony docelowej."
         Font-Name="Comic Sans MS Bold"
         Font-Size="16pt"
         OnClick="btnLink_Click">
      LinkButton do strony docelowej
      </asp:LinkButton>
   </div>
   </form>
</body>
</html>
```

Obsługa zdarzeń `Click` umieszczona w pliku ukrytego kodu została przestawiona na listingu 4.9.

Listing 4.9. Obsługa zdarzeń Click dla witryny ButtonDemo

```
protected void btnLink_Click(object sender, EventArgs e)
{
   Response.Redirect("//localhost/websites/StronaDocelowa.aspx");
}
protected void imgLink_Click(object sender, ImageClickEventArgs e)
{
   Response.Redirect("//localhost/websites/StronaDocelowa.aspx");
}
```

Położenia plików

Niezależnie od tego, że położenie plików jest wymagane przez ASP.NET jako argument metody `Redirect` czy jako właściwość, na przykład `ImageButton.ImageURL`, istnieją cztery sposoby przedstawienia adresu URL:

Względne
: Położenie jest określane w odniesieniu do katalogu głównego aplikacji. Adres rozpoczyna się od znaku kropki (`.`) lub samą nazwą, ale nie znakiem ukośnika (`/`).

Względem aplikacji
: Położenie jest względem katalogu głównego aplikacji. Wykorzystywany jest operator ~ (tylda), który oznacza katalog główny aplikacji, jak to przedstawiono w poniższym przykładzie:

```
BackImageUrl="~/images/Sunflower Bkgrd.jpg"
```

: Powyższy zapis powoduje odniesienie do pliku w katalogu *images*, który znajduje się w katalogu głównym.

: Zaletą używania adresowania względnego lub względem aplikacji jest łatwiejsze programowanie. Pełna analiza zagadnień związanych z programowaniem zostanie przedstawiona w rozdziale 19.

Absolutne
: Ścieżka do pliku zapisanego na komputerze lokalnym rozpoczyna się od ukośnika (`/`), wskazując katalog na bieżącym dysku twardym lub inny napęd, inny napęd oraz ścieżkę dostępu.

: Jeżeli aplikacja została zaprogramowana na komputerze z zupełnie inną strukturą katalogów, wprowadzony tu kod może wymagać zmian. W ten sposób można uniknąć wystąpienia błędów.

Pełna ścieżka
: Może być to jeden z kilku typów. Format *Universal Naming Convention* (UNC) określa położenie wszystkich możliwych zasobów w sieci i przyjmuje następującą formę:

```
\\nazwa-serwera\sciezka-do-wspoldzielonego-zasobu
```

: Może być to adres URL do strony w internecie, na przykład:

```
http://www.DowolnaDomena.pl
```

: Może być to również usługa serwera obecnego na komputerze lokalnym, na przykład:

```
//localhost/witryny_www/StronaDocelowa.aspx
```

Ukończona strona została pokazana na rysunku 4.4.

Aby zapewnić prawidłowe działanie kodu przedstawionego w witrynie *ButtonDemo*, musimy posiadać docelową stronę internetową, do której prowadzi łącze. Może to być dowolny prawidłowy plik *.html*, *.asp* lub *.aspx*. W powyższym przykładzie strona docelowa została zdefiniowana jako *StronaDocelowa.aspx* umieszczona w katalogu wirtualnym *websites*. Oprócz tego będzie konieczny plik graficzny dla kontrolki `ImageButton`. W naszym przykładzie wykorzystaliśmy plik *Dan at vernal pool.jpg* umieszczony w katalogu witryny internetowej, aczkolwiek może zostać użyty dowolny plik graficzny.

Rysunek 4.4. Witryna internetowa ButtonDemo

Duża różnica między kontrolką `LinkButton` a standardową kontrolką `Button` polega na tym, że dostępne funkcje kontrolki `LinkButton` są implementowane za pomocą skryptu po stronie klienta. Stanie się to wyraźnie oczywiste, jeżeli spojrzymy na kod źródłowy wygenerowany przez przeglądarkę wyświetlającą stronę internetową *ButtonDemo*. Fragment tego kodu został przedstawiony na listingu 4.10. Należy pamiętać, że ten kod źródłowy został wygenerowany przez ASP.NET, a nie wpisany przez programistę.

Listing 4.10. Fragment kodu źródłowego strony ButtonDemo

```
<script type="text/javascript">
<!--
var theForm = document.forms['form1'];
function __doPostBack(eventTarget, eventArgument) {
   if (theForm.onsubmit == null || theForm.onsubmit()) {
      theForm.__EVENTTARGET.value = eventTarget;
      theForm.__EVENTARGUMENT.value = eventArgument;
      theForm.submit();
   }
}
// --->
</script>

<input type="submit" name="btnLink" value="Łącze do strony docelowej"
   id="btnLink" title="Kliknij tutaj, aby przejść do strony docelowej." />

<input type="image" name="imgLink" id="imgLink"
   title="Kliknij tutaj, aby przejść do strony docelowej."
   src="Dan%2Oat%2OVernal%2OPool.jpg" alt="Łącze do strony docelowej"
   style="border-width:Opx;"/>

<a id="lnkLink" title="Kliknij tutaj, aby przejść do strony docelowej."
   href="javascript:__doPostBack('lnkLink','')"
   style="font-family:Comic Sans MS Bold;font-size:16pt;">
   Łącze do strony docelowej</a>
```

Kontrolka HyperLink

Kontrolka `HyperLink` jest podobna do kontrolki `LinkButton`, choć dzieli je zasadnicza różnica. Otóż kontrolka `HyperLink` natychmiast wywołuje docelowy adresu URL, bez ponownego odświeżania formularza, podczas gdy kontrolka `LinkButton` to odświeżanie wykonuje. Jeżeli zostanie wybrana obsługa zdarzenia kontrolki `LinkButton`, nastąpi przejście do docelowego adresu URL. Kontrolka `HyperLink` zachowuje się w sposób podobny do kontrolki HTML.

Kontrolka `HyperLink` posiada cztery określone parametry:

`ImageUrl`
: Ścieżka do pliku graficznego przeznaczonego do wyświetlenia zamiast tekstu. W przypadku zastosowania tego atrybutu, kontrolka będzie wyświetlana użytkownikowi dokładnie tak samo jak kontrolka `ImageButton`. Oczywiście, kontrolka `ImageButton` wciąż odświeża stronę, podczas gdy `HyperLink` jedynie ją wywołuje.

`NavigateUrl`
: Docelowy adres URL.

`Text`
: Ciąg znakowy, który zostanie wyświetlony w przeglądarce jako łącze. Jeżeli ustawiono jednocześnie właściwości `Text` i `ImageUrl`, wówczas zastosowana będzie właściwość `ImageUrl`. Tekst zostanie wyświetlony, jeśli plik graficzny będzie niedostępny.

  Jeżeli przeglądarka obsługuje mechanizm podpowiedzi, a właściwość `ToolTip` (dziedziczona z klasy `WebControl`) nie została ustawiona, wtedy wartość `Text` zostanie wyświetlona jako podpowiedź. Jeśli właściwość `ToolTip` została ustawiona, wtedy jako podpowiedź zostanie wyświetlony ciąg znakowy `ToolTip`.

`Target`
: Definiuje docelowe okno lub ramkę, do którego zostanie wczytana strona docelowa. Wartość rozróżnia wielkość liter i musi rozpoczynać się znakiem z zakresu `a-z`, za wyjątkiem wartości specjalnych przedstawionych w tabeli 4.3. Wszystkie wartości specjalne rozpoczynają się od znaku podkreślenia.

Tabela 4.3. Wartości specjalne atrybutu Target

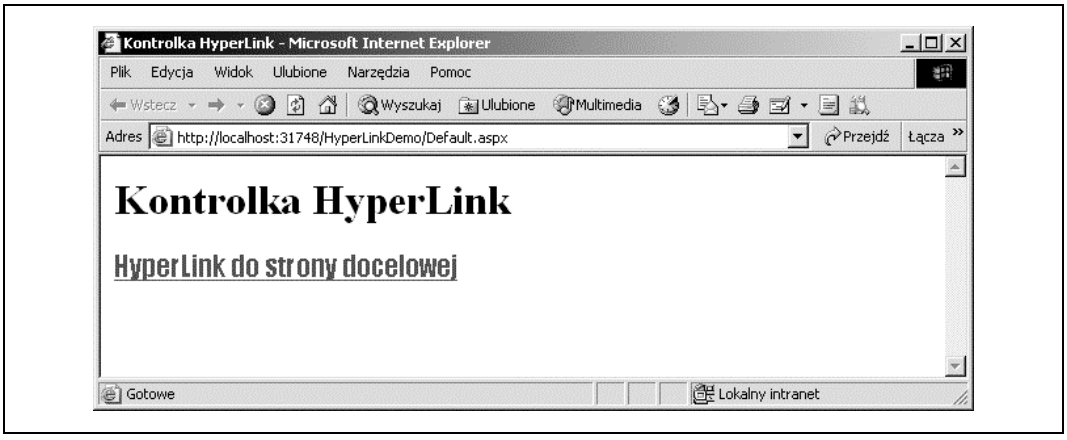
Wartość docelowa	Opis
`_blank`	Generuje zawartość w nowym nienazwanym oknie bez ramek.
`_new`	Nieudokumentowana, ale zachowuje się tak samo jak wartość `_blank`.
`_parent`	Generuje zawartość w oknie nadrzędnym albo bloku `frameset` okna albo ramce z łączem. Jeżeli element potomny jest oknem lub ramką najwyższego poziomu, wówczas zachowuje się tak samo jak `_self`.
`_self`	Generuje zawartość w bieżącej ramce lub aktywnym oknie. Jest to wartość domyślna.
`_top`	Generuje zawartość w bieżącym pełnym oknie bez ramek.

Poniżej prezentujemy przykładowy plik *HyperLinkDemo* pokazuje w działaniu kontrolkę `HyperLink`. Plik z treścią został przedstawiony na listingu 4.11. Kontrolka `HyperLink` nie wysyła żądania do serwera, tak więc ten przykład nie zawiera pliku ukrytego kodu.

Listing 4.11. Plik Default.aspx z aplikacji HyperLinkDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka HyperLink</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka HyperLink</h1>
      <asp:HyperLink ID="hypLink" runat="server"
         NavigateUrl="//localhost/websites/StronaDocelowa.aspx"
         Target="_self"
         Font-Names="Impact"
         Font-Size="16"
         ToolTip="Kliknij tutaj, aby przejść do strony docelowej.">
      HyperLink do strony docelowej
      </asp:HyperLink>
   </div>
   </form>
</body>
</html>
```

Kiedy strona *HyperLinkDemo* zostanie uruchomiona, będzie wyglądała podobnie do pokazanej na rysunku 4.5. Aby zapewnić prawidłowe działanie przedstawionego kodu, musimy posiadać stronę internetową o nazwie *StronaDocelowa.aspx*, umieszczoną w katalogu fizycznym, który odpowiada wirtualnemu katalogowi *websites* na komputerze lokalnym.



Rysunek 4.5. Aplikacja HyperLinkDemo

Kontrolka `HyperLink` jest generowana w przeglądarce klienta jako znacznik kotwiczący HTML (to znaczy `<a>`). Możemy to sprawdzić, analizując w przeglądarce kod źródłowy wyświetlonej strony.

Zaznaczanie wartości

Kilka kontrolek serwerowych ASP.NET pozwala użytkownikowi na zaznaczenie jednej lub wielu wartości:

`CheckBox`
: Pozwala na zaznaczenie danych typu Boolean.

`CheckBoxList`
: Grupa kontrolek `CheckBox`, które mogą być dynamicznie tworzone i łączone ze źródłami danych.

`RadioButton`
: Pozwala na wybranie tylko jednej opcji z grupy.

`RadioButtonList`
: Grupa kontrolek `RadioButton`, które mogą być dynamicznie tworzone i łączone ze źródłami danych.

`ListBox`
: Pozwala na zaznaczenie jednego bądź większej liczby elementów z zdefiniowanej wcześniej listy.

`DropDownList`
: Podobna do kontrolki `ListBox`, ale pozwala na zaznaczenie tylko jednego elementu.

`BulletedList`
: Formatowanie za pomocą znaku wypunktowania, może być zwykłym tekstem lub łączem.

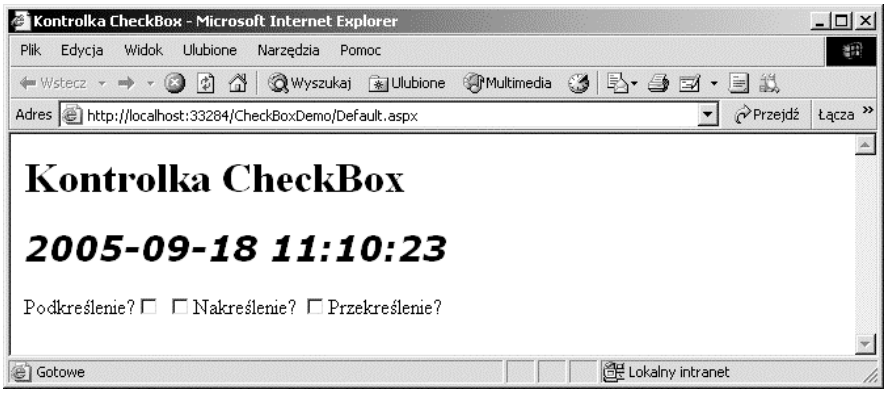
Wszystkie wymienione kontrolki wywodzą się z klasy `WebControl`. Kontrolka `RadioButton` wywodzi się później z klasy `CheckBox`, a wszystkie kontrolki list wywodzą się z klasy abstrakcyjnej `ListControl`. Każda z wymienionych kontrolek zostanie szczegółowo przedstawiona w kolejnych podrozdziałach.

Kontrolka CheckBox

Kontrolka `CheckBox` umożliwia użytkownikowi zaznaczenie danych typu Boolean (na przykład *Tak/Nie* lub *Prawda/Fałsz*). Jeżeli ułożono razem kilka pól wyboru (nie należy ich mylić z kontrolką `CheckBoxList`, która zostanie omówiona w kolejnym podrozdziale), wówczas można zaznaczyć wiele opcji. Żadna z opcji nie wyklucza innej.

Kontrolki `CheckBox` i `RadioButton` implementują interfejs `ICheckBoxControl`, który jest nowością w ASP.NET 2.0. Interfejs udostępnia jedną właściwość o nazwie `Checked` oraz jedno zdarzenie `CheckedChanged`. Poniżej zostały one opisane.

Aplikacja *CheckBoxDemo*, pokazana na rysunku 4.6, demonstruje użycie trzech niezależnych kontrolek `CheckBox` do sterowania sposobem pojawienia się kontrolki `Label`. Klikając dowolne z przedstawionych w przykładzie pól wyboru — *Podkreślenie*, *Nakreślenie* lub *Przekreślenie* — spowoduje nałożenie odpowiedniego atrybutu czcionki na ciąg tekstowy zawarty w kontrolce `Label`.



Rysunek 4.6. Aplikacja CheckBoxDemo

Zawartość pliku z treścią znajduje się na listingu 4.12.

Listing 4.12. Plik Default.aspx z aplikacji CheckBoxDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka CheckBox</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka CheckBox</h1>
      <asp:Label ID="lblTime" runat="server"
         OnInit="lblTime_Init" />
      <br />
      <br />
      <asp:CheckBox ID="chkUnderLine" runat="server"
         AutoPostBack="True"
         Text="Podkreślenie?"
         TextAlign="Left"
         OnCheckedChanged="chkUnderLine_CheckedChanged" />
      <asp:CheckBox ID="chkOverLine" runat="server"
         AutoPostBack="True"
         Text="Nakreślenie?"
         OnCheckedChanged="chkOverLine_CheckedChanged" />
      <asp:CheckBox ID="chkStrikeout" runat="server"
         AutoPostBack="True"
         Text="Przekreślenie?"
         OnCheckedChanged="chkStrikeout_CheckedChanged" />
   </div>
   </form>
</body>
</html>
```

Każda z kontrolek serwerowych ASP.NET w tym przykładzie — etykieta oraz trzy pola wyboru — uwzględniają obsługę zdarzeń. Zdarzenie `Init` kontrolki `Label` obsługuje format i zawartość etykiety przy każdym odświeżeniu strony. Kontrolki `CheckBox` posiadają domyślną obsługę zdarzeń `CheckedChanged`. Zdarzenie przekazuje standardowy argument `EventArgs`, który nie posiada żadnych właściwości.

Wszystkie wymienione metody obsługi zdarzeń zostały zawarte w pliku ukrytego kodu, przedstawionym na listingu 4.13.

Listing 4.13. Obsługa zdarzeń w pliku ukrytego kodu Default.aspx.cs aplikacji CheckBoxDemo

```
protected void lblTime_Init(object sender, EventArgs e)
{
   lblTime.Font.Name = "Verdana";
   lblTime.Font.Size = 20;
   lblTime.Font.Bold = true;
   lblTime.Font.Italic = true;
   lblTime.Text = DateTime.Now.ToString();
}
protected void chkUnderLine_CheckedChanged(object sender, EventArgs e)
{
   if (chkUnderLine.Checked)
      lblTime.Font.Underline = true;
   else
      lblTime.Font.Underline = false;
}
protected void chkOverLine_CheckedChanged(object sender, EventArgs e)
{
   if (chkOverLine.Checked)
      lblTime.Font.Overline = true;
   else
      lblTime.Font.Overline = false;
}
protected void chkStrikeout_CheckedChanged(object sender, EventArgs e)
{
   if (chkStrikeout.Checked)
      lblTime.Font.Strikeout = true;
   else
      lblTime.Font.Strikeout = false;
}
```

Podobnie jak wszystkie kontrolki wywodzące się z klasy `WebControl`, również kontrolki `CheckBox` posiadają właściwość `ID`. Jednak jak pokazuje przykładowy kod w listingu 4.12, istnieje również kilka innych właściwości i metod, które nie są dziedziczone z klasy `WebControl`. Wspomniane elementy zostały przedstawione w tabeli 4.4. Trzeba dodać, że niektóre z tych właściwości, na przykład `AutoPostBack` i `Text`, są często spotykane dla kilku innych kontrolek.

Kontrolka RadioButton

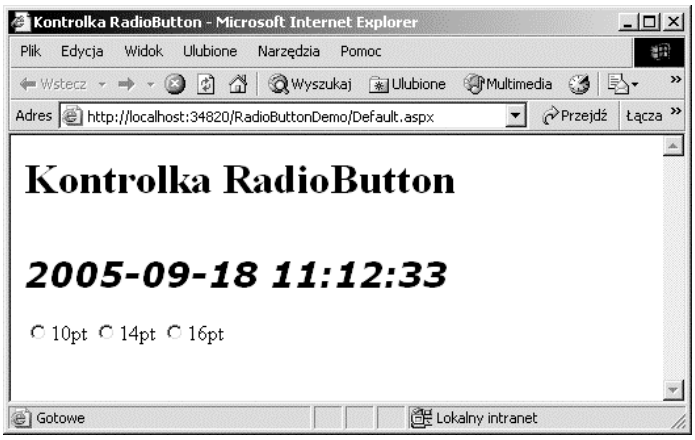
Kontrolka `RadioButton` jest bardzo podobna i faktycznie wywodzi się z kontrolki `CheckBox`. Zasadnicza różnica między tymi dwoma klasami polega na tym, że kontrolki `RadioButton` są zwykle zgrupowane za pomocą właściwości `GroupName`. Tylko jedna kontrolka `RadioButton` z tej grupy może pozostawać zaznaczona (jej właściwość `Checked` otrzymuje wartość `true`). Zmiana właściwości `Checked` na `true` jednej kontrolki `RadioButton` z danej grupy powoduje zmianę właściwości `Checked` pozostałych kontrolek na `false`. Dodatkowo, przyciski opcji są zwykle wyświetlane jako okrągłe, w przeciwieństwie do kwadratowych pól wyboru.

Tabela 4.4. Elementy klasy CheckBox, które nie są dziedziczone z klasy WebControl

Nazwa	Typ	Pobierz	Ustaw	Wartości	Opis
`AutoPostBack`	Boolean	x	x	`true`, `false`	Określa, czy zmiana zawartości kontrolki przez użytkownika spowoduje automatyczne wysłanie ponownego żądania do serwera. W przypadku wartości `false`, ponowne wysłanie żądania nie wystąpi, dopóki strona nie zostanie odświeżona albo przyciskiem, albo przez inną kontrolkę z ustawionym `AutoPostBack` jako `true`.
`Checked`	Boolean	x	x	`true`, `false`	Wskazuje, czy kontrolka `CheckBox` jest zaznaczona. Wartością domyślną jest `false`.
`Text`	String	x	x		Etykieta tekstowa przypisana kontrolce `CheckBox`.
`TextAlign`	TextAlign	x	x	`Left`, `Right`	Wskazuje, czy etykieta tekstowa znajdzie się po lewej czy po prawej stronie kontrolki `CheckBox`. Wartością domyślną jest `Right`.
`CheckedChanged`	Event			EventArgs	Wywołanie zdarzenia następuje po zmianie właściwości `Changed`. Dopóki wartość `AutoPostBack` nie zostanie ustawione jako `true`, to zdarzenie nie spowoduje natychmiastowego wysłania ponownego żądania do serwera.

Kolejny przykład — *RadioButtonDemo* — zawiera trzy kontrolki `RadioButton` ustawiające wielkość etykiety. Każdy z przycisków opcji w aplikacji *RadioButtonDemo* jest elementem grupy `grpSize`.

Zawartość pliku z treścią dla tego przykładu została przedstawiona na listingu 4.14, a obsługi zdarzeń w pliku ukrytego kodu zaprezentowano na listingu 4.15. Wynik uruchomienia tej aplikacji został pokazany na rysunku 4.7.



Rysunek 4.7. Aplikacja RadioButtonDemo

Listing 4.14. Plik Default.aspx witryny internetowej RadioButtonDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka RadioButton</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka RadioButton</h1>
      <br />
      <asp:Label ID="lblTime" runat="server"
         OnInit="lblTime_Init"></asp:Label>
      <br />
      <br />
      <asp:RadioButton ID="rdoSize10" runat="server"
         GroupName="grpSize"
         AutoPostBack="True"
         Text="10pt"
         OnCheckedChanged="grpSize_CheckedChanged" />
      <asp:RadioButton ID="rdoSize14" runat="server"
         GroupName="grpSize"
         AutoPostBack="True"
         Text="14pt"
         OnCheckedChanged="grpSize_CheckedChanged" />
      <asp:RadioButton ID="rdoSize16" runat="server"
         GroupName="grpSize"
         AutoPostBack="True"
         Text="16pt"
         OnCheckedChanged="grpSize_CheckedChanged" />
   </div>
   </form>
</body>
</html>
```

Listing 4.15. Kod obsługi zdarzeń aplikacji RadioButoonDemo w pliku ukrytego kodu Default.aspx.cs

```
protected void grpSize_CheckedChanged(object sender, EventArgs e)
{
   if (rdoSize10.Checked)
      lblTime.Font.Size = 10;
   else if (rdoSize14.Checked)
      lblTime.Font.Size = 14;
   else lblTime.Font.Size = 16;
}
protected void lblTime_Init(object sender, EventArgs e)
{
   lblTime.Font.Name = "Verdana";
   lblTime.Font.Size = 20;
   lblTime.Font.Bold = true;
   lblTime.Font.Italic = true;
   lblTime.Text = DateTime.Now.ToString();
}
```

Wywodzące się z klasy `CheckBox` zdarzenie `CheckedChanged` jest obsługiwane przez obsługę zdarzenia `onCheckedChanged`, która z kolei wykorzystuje metodę `grpSize_CheckedChanged`. Metoda ta stanowi blok instrukcji warunkowej `if…else`, a efektem jej działania jest zmiana

wielkości tekstu w zależności od wybranej opcji. W praktyce, prawdopodobnie lepszym rozwiązaniem byłoby użycie polecenia C# `switch`, co znacznie ułatwiłoby dodawanie w przyszłości kolejnych przycisków opcji.

Zaznaczanie z listy

ASP.NET dostarcza pięć kontrolek serwerowych, pozwalających na zaznaczanie jednego lub wielu elementów z listy:

- `BulletedList`
- `CheckBoxList`
- `DropDownList`
- `ListBox`
- `RadioButtonList`

Wszystkie wymienione kontrolki wywodzą się z klasy `ListControl` i mają ze sobą wiele wspólnego.

- `ListItem` (informacje wyświetlane przez listę) funkcjonuje dokładnie w ten sam sposób, co właściwości `Value` i `Text` dla wszystkich kontrolek `ListControl`.
- Właściwość `Items` kontrolki zawiera zestaw wszystkich `ListItem`.
- `ListItem` mogą zostać dodane do zbioru `Item` albo statycznie, na przykład deklaracyjnie w pliku z treścią, programowo dzięki metodzie `Add` albo ze źródła danych.
- Kreator *Data Source Configuration Wizard* lub edytor *ListItem Collection Editor* są łatwo dostępne poprzez kliknięcie tagu inteligentnego (ang. *smart tag*) kontrolki — małej ikony w prawym górnym rogu kontrolki.
- Właściwości `SelectedIndex` i `SelectedItem` kontrolki wskazują na zaznaczone elementy z najniższym indeksem. W przypadku kontrolek umożliwiających pojedyncze zaznaczenia, na przykład `DropDownList`, `RadioButtonList` i `ListBox` (jeżeli właściwość `SelectionMode` posiada wartość `ListSelectionMode.Single`, co jest domyślnym ustawieniem), zaznaczony indeks jest z definicji najniższym indeksem. Dla kontrolek pozwalających na wielokrotne zaznaczenia, na przykład `CheckBoxList` i `ListBox` z właściwością `SelectionMode` ustawioną jako `ListSelectionMode.Multiple`, te właściwości będą odnosiły się do zaznaczonych elementów z najniższym indeksem.
- Właściwość `SelectedValue` kontrolki otrzymuje lub określa wartość zaznaczonego elementu.
- Właściwość `AppendDataBoundItems` kontrolki (nowość w ASP.NET 2.0) pozwala elementom dodanym przez łączenie danych (co zostanie opisane w rozdziale 9.) na dodanie ich do zbioru `Item` zamiast zastąpienia zbioru `Item`, co jest domyślnym zachowaniem.
- Wszystkie pięć kontrolek zgłasza i odpowiada na zdarzenie `SelectedIndexChanged`.

Kontrolki `ListBox` i `DropDownList` różnią się od innych kontrolek list (`BulletedList`, `CheckBoxList` i `RadioButtonList`) tym, że są wyświetlane użytkownikowi jako pojedyncza kontrolka (pole listy lub rozwijana lista) zamiast zbioru łączy, przycisków bądź pól wyboru. Kontrolki `ListBox` i `DropDownList` mogą posiadać dłuższe listy, ponieważ zawierają paski przewijania.

Różnice pomiędzy pięcioma kontrolkami list zostały podsumowane w tabeli 4.5.

Tabela 4.5. Różnice pomiędzy pięcioma kontrolkami list

Charakterystyka	BulletedList	CheckBoxList	RadioButtonList	DropDownList	ListBox
Tylko pojedyncze zaznaczenia.	x		x	x	
Możliwość zaznaczenia jednocześnie więcej niż jednego elementu.		x			x
Wyświetla całą listę.	x	x	x		
Wyświetla w danym momencie pojedynczy element wraz z przyciskiem służącym do pokazania całej listy, używającej w razie konieczności pionowego paska przewijania.				x	
Wyświetla wiele elementów, używając w razie konieczności pionowego paska przewijania.					x
Najlepszy w przypadku krótkich list.	x	x	x		
Najlepszy w przypadku długich list.				x	x

W kolejnych podrozdziałach zostaną opisane kontrolki i obiekty powiązane z zaznaczaniem elementów z listy.

Obiekt ListItem

Pięć kontrolek serwerowych pozwalających na zaznaczanie elementów z list wywodzi się z klasy `ListControl`. Kontrolka `ListControl` składa się ze zbioru obiektów `ListItem`. Każdy z obiektów `ListItem` posiada cztery właściwości, szczegółowo przedstawione w tabeli 4.6.

Tabela 4.6. Właściwości obiektu ListItem

Nazwa	Typ	Pobierz	Ustaw	Opis
`Enabled`	Boolean	x	x	Jeżeli posiada ustawioną wartość `false`, wówczas pozwala elementowi na pozostawanie nieaktywnym i niewidocznym w trakcie wyświetlania listy, jednak wciąż obecnym w zbiorze `Item`.
`Selected`	Boolean	x	x	Wartość wskazująca, że element został zaznaczony.
`Text`	String	x	x	Ciąg tekstowy wyświetlany dla `ListItem`.
`Value`	String	x	x	Wartość przypisana `ListItem`. Wartość ta nie jest wyświetlana, ale dostępna programowo.

W trakcie pracy z listami, wyświetlanie użytkownikowi wartości różnych od tych przekazywanych do kodu jest dość powszechnym zjawiskiem. Przykładowo, w przypadku prezentowania użytkownikom zestawienia stanów w USA, lista może wyświetlać ich nazwy, takie jak `Massachusetts`, natomiast kiedy element zostanie wybrany, program będzie przekazywał zaznaczony element jako `ma`. Wówczas `Massachusetts` będzie właściwością `Text` obiektu `ListItem`, podczas gdy `ma` będzie właściwością `Value`.

Właściwość `Text` może zostać określona na jeden z dwóch sposobów:

Wewnętrzna zawartość HTML
: Tekst zawarty pomiędzy otwierającym a zamykającym znacznikiem dowolnej kontrolki.

Atrybut `Text`
: Atrybut wewnątrz otwierającego znacznika kontrolki `ListItem`.

Można tu zastosować zarówno zamykający znacznik bez wewnętrznego HTML, jak i domknąć znacznik otwierający. Wszystkie przedstawione poniżej trzy wiersze są równoznaczne:

```
<asp:ListItem>Element 7</asp:ListItem>
<asp:ListItem text="Element 7"></asp:ListItem>
<asp:ListItem text="Element 7" />
```

Jeżeli określono właściwość `Text` oraz wewnętrzny HTML, wówczas zostanie wyświetlony wewnętrzny kod HTML. Przykładowo, warto zastanowić się nad poniższym wierszem:

```
<asp:ListItem Text="Element 7">Element 8</asp:ListItem>
```

Jeśli taki wiersz zostanie uwzględniony w kodzie, na stronie internetowej zostanie wyświetlony tekst „Element 8”.

Właściwość `Value` może zostać ustawiona podobnie jak właściwość `Text`. Tak więc, można dokonać modyfikacji wierszy kodu przedstawionych powyżej, aby została ustawiona również wartość:

```
<asp:ListItem value="7">Element 7</asp:ListItem>
<asp:ListItem text="Element 7" value="7"></asp:ListItem>
<asp:ListItem text="Element 7" value="7" />
```

Kontrolka CheckBoxList

Kontrolka `CheckBoxList` jest kontrolką nadrzędną („rodzicem”) zawierającą zbiór elementów `CheckBox`. Jest bardzo podobna do grupy kontrolek `CheckBox`, przedstawionych we wcześniejszym przykładzie *CheckBoxDemo* na rysunku 4.6. Wyjątkiem jest fakt, że wszystkie pola wyboru „potomne” są obsługiwane jako grupa. Kontrolka `CheckBoxList` wywodzi się z klasy `ListControl`, a nie bezpośrednio z klasy `WebControl`.

W trakcie tworzenia serii pól wyboru danych z bazy danych, kontrolka `CheckBoxList` jest lepiej dopasowana niż pojedyncze pola wyboru, chociaż każda z tych kontrolek może łączyć dane. Łączenie danych zostanie przeanalizowane w rozdziale 9.

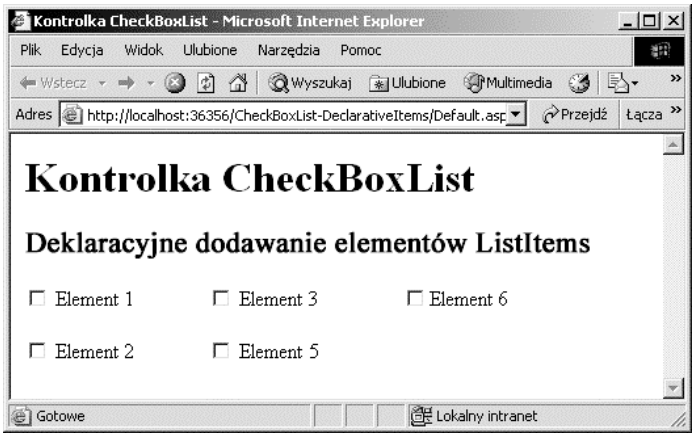
Istnieją trzy sposoby dodawania elementów do zbioru `Item` kontrolki `CheckBoxList`:

- deklaracyjne, za pomocą znacznika kontrolki `<asp:ListItem>`;
- programowe z tablicy;
- dynamiczne ze źródła danych, na przykład z bazy danych.

Deklaracyjne dodawanie elementów

Witryna internetowa *CheckBoxList-DeclarativeItems*, którą pokazano na rysunku 4.8, przedstawia wiele właściwości kontrolki `CheckBoxList`. Lista elementów została dodana deklaracyjnie w pliku z treścią *Default.aspx*. Atrybuty w deklaracji kontrolki `CheckBoxList` odpowiadają właściwościom klasy `CheckBoxList`, określają wygląd i zachowanie kontrolki.

Plik z treścią tej witryny internetowej został przedstawiony na listingu 4.16. W przypadku tej aplikacji nie występują zdarzenia, stąd brak obsługi zdarzeń i pliku ukrytego kodu.



Rysunek 4.8. Witryna internetowa CheckBoxList-DeclarativeItems

Listing 4.16. Plik Default.aspx witryny CheckBoxList-DeclarativeItems

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka CheckBoxList</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka CheckBoxList</h1>
      <h2>Deklaracyjne dodawanie elementów ListItems</h2>
      <asp:CheckBoxList ID="cblItems" runat="server"
            AutoPostBack="True"
            CellPadding="5"
            CellSpacing="10"
            RepeatColumns="3">
         <asp:ListItem> Element 1 </asp:ListItem>
         <asp:ListItem> Element 2 </asp:ListItem>
         <asp:ListItem> Element 3 </asp:ListItem>
         <asp:ListItem> Element 5 </asp:ListItem>
         <asp:ListItem> Element 6 </asp:ListItem>
      </asp:CheckBoxList>
   </div>
   </form>
</body>
</html>
```

W kodzie na listingu 4.16 zastosowano wartości domyślne dla tych właściwości, które je posiadają, jak to zostało wskazane w tabeli 4.7. Zmieniając właściwości `RepeatDirection`, `RepeatLayout` i `TextAlign` na, odpowiednio, `Horizontal`, `Flow` i `Left` uzyskaliśmy wynik pokazany na rysunku 4.9.

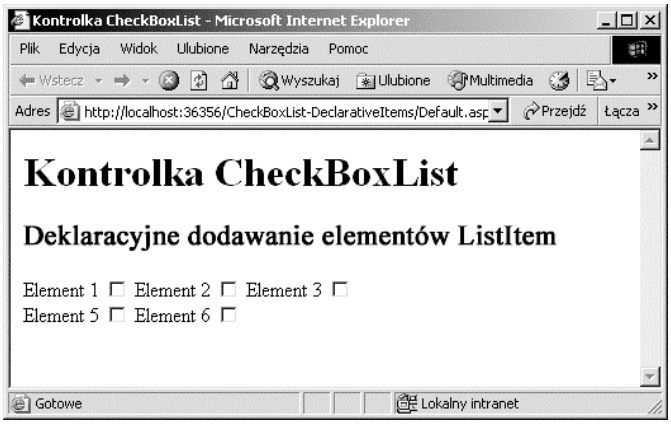
Tabela 4.7. Właściwości kontrolki CheckBoxList

Nazwa	Typ	Pobierz	Ustaw	Wartości	Opis
`AutoPostBack`	Boolean	x	x	`true`, `false`	Określa, czy zmiana zawartości kontrolki przez użytkownika powoduje automatyczne wysłanie ponownego żądania do serwera. W przypadku wartości `false`, ponowne przekazanie żądania do serwera nie nastąpi, dopóki strona nie zostanie odświeżona albo przyciskiem, albo przez inną kontrolkę z właściwością `AutoPostBack` ustawioną na `true`. Wartością domyślną jest `false`.
`CellPadding`	Integer	x	x	Liczby całkowite	Odległość wyrażona w pikselach między krawędzią a zawartością komórki. Wartością domyślną jest `-1`, co wskazuje, że właściwość nie jest ustawiona.
`CellSpacing`	Integer	x	x	Liczby całkowite	Odległość wyrażona w pikselach między krawędzią a zawartością komórki. Wartością domyślną jest `-1`, co wskazuje, że właściwość nie jest ustawiona.
`DataSource`	Object	x	x		Źródło, które wypełnia kontrolkę.
`RepeatColumns`	Integer	x	x	Liczby całkowite	Liczba wyświetlanych kolumn.

`RepeatDirection`	Repeat-Direction	x	x	`Horizontal`, `Vertical`	`Horizontal`
					Powoduje, że elementy zostaną wczytane od lewej do prawej strony, następnie z góry do dołu.
					`Vertical`
					Powoduje, że elementy zostaną wczytane z góry do dołu, następnie od lewej do prawej strony.
					Wartością domyślną jest `Vertical`.

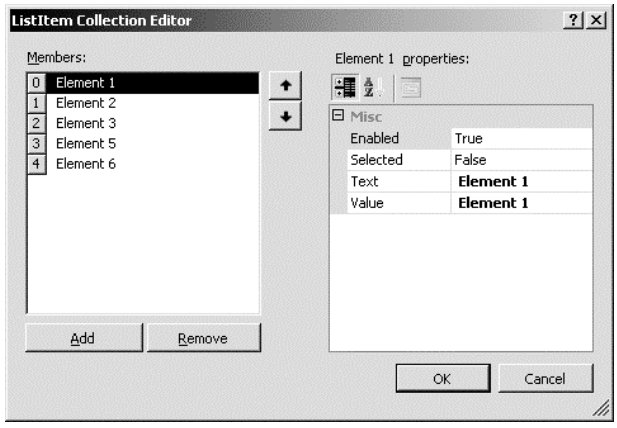
`RepeatLayout`	Repeat-Layout	x	x	`Flow`, `Table`	`Flow`
					Powoduje, że elementy zostaną wyświetlone bez struktury tabeli.
					`Table`
					Powoduje, że elementy zostaną wyświetlone w strukturze tabeli.
					Wartością domyślną jest `table`.

`Selected`	Boolean	x	x	`true`, `false`	Wartość wskazująca, że element został zaznaczony. Wartością domyślną jest `false`.
`TextAlign`	TextAlign	x	x	`Left`, `Right`	Wskazuje, czy etykieta tekstowa znajdzie się po lewej czy po prawej stronie pól wyboru. Wartością domyślną jest `Right`.



Rysunek 4.9. Statyczna modyfikacja witryny CheckBoxList-DeclarativeItems z wykorzystaniem niestandardowych wartości właściwości

Element `ListItem` może zostać wpisany ręcznie w pliku z zawartością (*IntelliSense* pozwoli zminimalizować ilość wpisywanego tekstu) lub można skorzystać z edytora *Collection Editor*. W celu użycia edytora *Collection Editor*, należy zaznaczyć kontrolkę `CheckBoxList` w widoku *Design view*, a następnie kliknąć tag inteligentny (małą ikonę w prawym górnym rogu kontrolki w widoku *Design view*) i wybrać z pojawiającego się menu opcję *Edit Items…*. Pojawi się okno dialogowe pokazane na rysunku 4.10, którego używamy w celu dodania lub usunięcia elementów `ListItem` bądź zmiany ich właściwości.

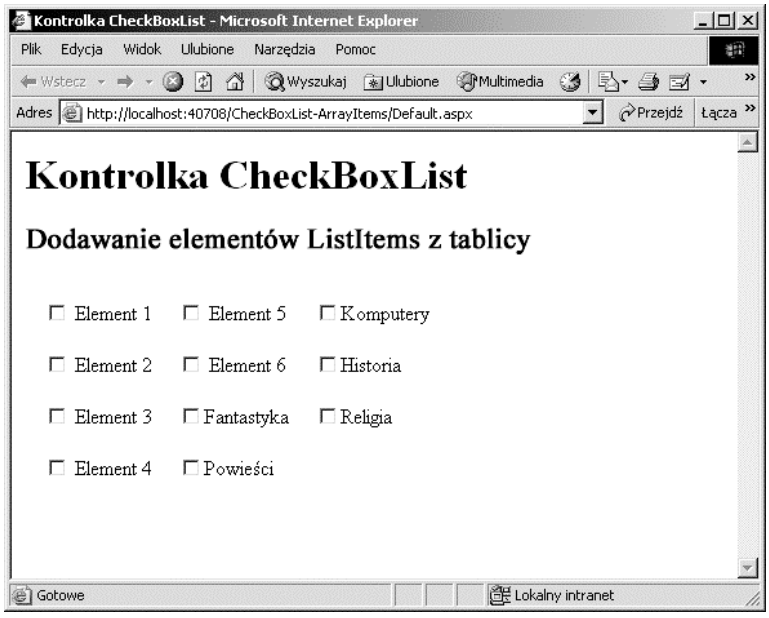


Rysunek 4.10. Okno dialogowe ListItem Editor

Programowe dodawanie elementów z tablicy

Zdarzają się sytuacje, w których podczas kompilacji kodu nie wiadomo, jakie pola wyboru zostaną utworzone. Na przykład, programista może zechcieć, aby aplikacja zapełniała listę w zależności od wartości innych kontrolek na stronie. W takich przypadkach zachodzi potrzeba uzyskania możliwości programowego dodawania elementów do zbioru `Item`.

W kolejnym przykładzie *CheckBoxList-ArrayItems*, pokazanym na rysunku 4.11, obiekty `ListItem` zostaną dodane zarówno w sposób programowy, jak również — w celach demonstracyjnych — zostaną zakodowane wewnątrz znaczników `CheckBoxList`.



Rysunek 4.11. Witryna internetowa CheckBoxList-ArrayItems

Zawartość pliku z treścią dla pokazanej witryny została przedstawiona na listingu 4.17, natomiast plik ukrytego kodu zaprezentowano na listingu 4.18.

Listing 4.17. Plik Default.aspx witryny CheckBoxList-ArrayItems

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka CheckBoxList</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka CheckBoxList</h1>
      <h2>Dodawanie elementów ListItems z tablicy</h2>
      <asp:CheckBoxList ID="cblGenre" runat="server"
            AutoPostBack="True"
            CellPadding="5"
            CellSpacing="10"
            RepeatColumns="3"
            OnInit="cblGenre_Init">
         <asp:ListItem> Element 1 </asp:ListItem>
         <asp:ListItem> Element 2 </asp:ListItem>
```

```
            <asp:ListItem> Element 3 </asp:ListItem>
            <asp:ListItem> Element 4 </asp:ListItem>
            <asp:ListItem> Element 5 </asp:ListItem>
            <asp:ListItem> Element 6 </asp:ListItem>
         </asp:CheckBoxList>
      </div>
      </form>
   </body>
   </html>
```

Listing 4.18. Obsługa zdarzenia w pliku ukrytego kodu witryny CheckBoxList-ArrayItems

```
protected void cblGenre_Init(object sender, EventArgs e)
{
   // Tworzenie tablicy elementów do dodania.
   string[] Genre = { "Fantastyka", "Powieści", "Komputery", "Historia", "Religia" };

   for (int i = 0; i < Genre.Length; i++)
   {
      this.cblGenre.Items.Add(new ListItem(Genre[i]));
   }
}
```

Pozostała część pliku ukrytego kodu z tego przykładu zawiera standardowy kod umieszczany przez VS2005.

Do znacznika kontrolki można dodać atrybut, który implementuje obsługę zdarzeń w trakcie inicjalizacji kontrolki:

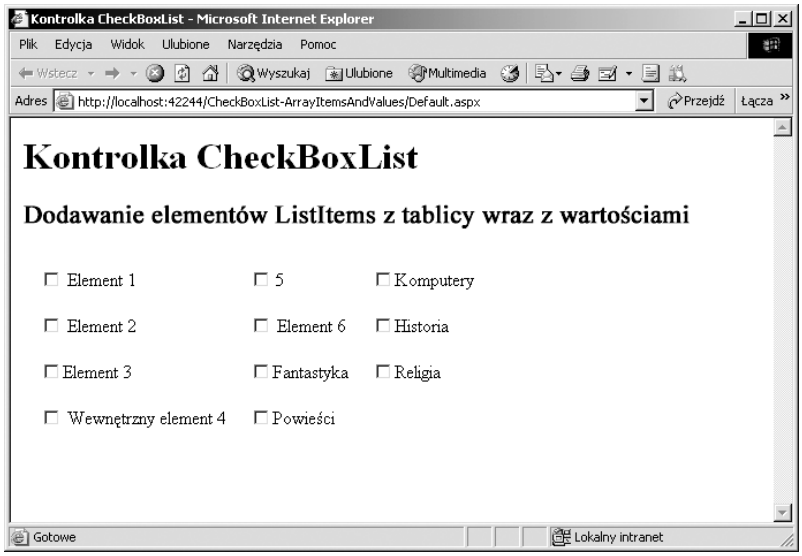
```
onInit="cblGenre_Init"
```

Następnie dodajemy do pliku ukrytego kodu *Default.aspx.cs* metodę `cblGenre_Init`, wywoływaną przez `onInit`. Metoda tworzy tablicę gatunków, które zostaną dodane do listy pól wyboru. W kolejnym kroku następuje przeglądanie tablicy za pomocą pętli `for`, wywołującej metodę `Add` na każdym elemencie. Powoduje to dodanie nowego obiektu `ListItem` do zbioru `Item` kontrolki `CheckBoxList`.

Kod przedstawiony na listingach 4.17 i 4.18 może zostać zmodyfikowany przez dodanie właściwości `Value` do niektórych elementów `ListItem`, utworzonych w deklaracji `CheckBoxList`, jak również we wszystkich obiektach `ListItem` utworzonych w procedurze zdarzenia `cblGenre_Init`. Zostało to zademonstrowane w nowej witrynie *CheckBoxList-ArrayItemsAndValues* skopiowanej z *CheckBoxList-ArrayItems* i odpowiednio zmodyfikowanej. Ukończona strona została pokazana na rysunku 4.12. Plik z treścią *default.aspx* nowej witryny jest przedstawiony na listingu 4.19. Wiersze, które odróżniają ten listing od listingu 4.17 zostały pogrubione.

Listing 4.19. Plik Default.aspx witryny CheckBoxList-ArrayItemsAndValues

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka CheckBoxList</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka CheckBoxList</h1>
```



Rysunek 4.12. Witryna internetowa CheckBoxList-ArrayItemsAndValues

```
        <h2>Dodawanie elementów ListItems z tablicy wraz z wartościami</h2>
        <asp:CheckBoxList ID="cblGenre" runat="server"
              AutoPostBack="True"
              CellPadding="5"
              CellSpacing="10"
              RepeatColumns="3"
              OnInit="cblGenre_Init">
           <asp:ListItem value="1"> Element 1 </asp:ListItem>
           <asp:ListItem text="Element 2" value="2"></asp:ListItem>
           <asp:ListItem text="Element 3" />
           <asp:ListItem text="Element 4"> Wewnętrzny element 4 </asp:ListItem>
           <asp:ListItem value="5"></asp:ListItem>
           <asp:ListItem> Element 6 </asp:ListItem>
        </asp:CheckBoxList>
     </div>
     </form>
</body>
</html>
```

W przedstawionym na listingu 4.20 kodzie obsłudze zdarzenia `Init` w pliku ukrytego kodu zostały pogrubione wiersze różniące się od poprzedniego przykładu.

Listing 4.20. Obsługa zdarzenia w pliku ukrytego kodu witryny CheckBoxList-ArrayItemsAndValues

```
protected void cblGenre_Init(object sender, EventArgs e)
{
   // Tworzenie tablicy elementów do dodania.
   string[] Genre = { "Fantastyka", "Powieści", "Komputery", "Historia", "Religia" };
   string[] Code = {"sf", "nvl", "cmp", "his", "rel" };

   for (int i = 0; i < Genre.Length; i++)
   {
      // Dodajemy zarówno właściwość Text jak i Value.
      this.cblGenre.Items.Add(new ListItem(Genre[i], Code[i]));
   }
}
```

W zdarzeniu `cblGenre_Init` przedstawionym na listingu 4.20, w miejscu, gdzie poprzednio znajdował się element tablicy zawierający właściwości `Text`, teraz znajdują się dwa elementy. Pierwszy dostarcza właściwości `Text`, natomiast drugi — właściwości `Value`. Można więc teraz użyć przeciążonej metody `Add`, przekazując jej pojedynczy argument składający się z obiektu `ListItem`:

```
this.cblGenre.Items.Add(new ListItem(Genre[i], Code[i]));
```

Obiekt może *przeciążyć* swoje metody, co oznacza, że może zadeklarować dwie lub więcej metod o tej samej nazwie. Kompilator rozróżnia te metody na podstawie liczby i typu dostarczonych parametrów.

Przykładowo, klasa `ListItemCollection` przeciąża metodę `Add`. Jedna wersja pobiera ciąg znaków, podczas gdy druga wersja pobiera obiekt `ListItem`.

Na koniec, w tworzeniu statycznej `ListItem` użyliśmy kilku różnych metod tworzenia właściwości `Value` i `Text`, włączając wystąpienie brakującej właściwości `Text` (`Element 5`), brakującej właściwości `Value` (`Element 3`, `Element 4`, `Element 6`) oraz rozbieżną od wewnętrznej zawartości HTML właściwość `Text` (`Element 4`). Różnice między rysunkami 4.11 i 4.12 są zauważalne w przypadku elementów 4. i 5.

Możemy zauważyć, że jeżeli brakuje właściwości `Value`, wówczas zostaje wyświetlona właściwość `Text`. Natomiast jeśli brakuje właściwości `Text`, to zostaje wyświetlona właściwość `Value`. Jeżeli właściwość `Text` różni się od wewnętrznej zawartości HTML, wówczas zostanie wyświetlona wewnętrzna zawartość HTML.

Dodawanie elementów ze źródeł danych

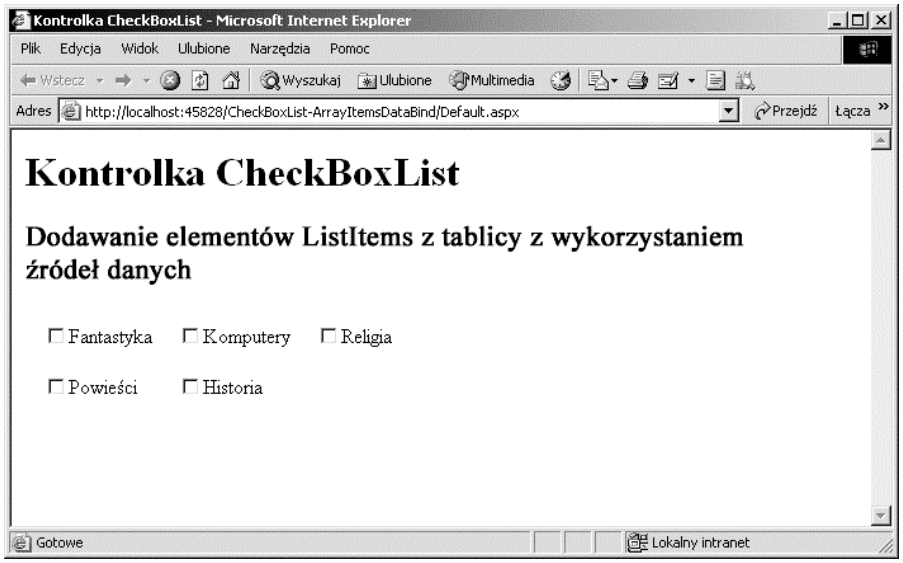
Z prawdziwą potęgą programowego dodawania elementów mamy do czynienia, gdy do zapełnienia elementów kontrolki `CheckBoxList` wykorzystujemy źródła danych. Oczywiście, najwydajniejszym źródłem danym jest baza danych. Zagadnienia z tym związane zostaną omówione w rozdziale 9. Teraz jednakże pokażemy, w jaki sposób można użyć utworzonej tablicy w celu zademonstrowania łączenia ze źródłem danych.

Należy skopiować poprzedni przykład do nowej witryny internetowej i nadać jej nazwę *CheckBoxList-ArrayItemsDataBind*. Trzeba jeszcze jedynie zmodyfikować metodę obsługi zdarzenia `cblGenre_Init` w pliku ukrytego kodu. Zastępujemy więc pętlę `for` w zdarzeniu `cblGenre_Init` przedstawionym na listingu 4.18 dwoma wierszami kodu, które określają źródło danych, a następnie wykonują połączenie do tego źródła. Nowa metoda wraz z wymienionymi wierszami (pogrubione) została przedstawiona na listingu 4.21.

Listing 4.21. Zmodyfikowany plik ukrytego kodu witryny CheckBoxList-ArrayItemsDataBind

```
protected void cblGenre_Init(object sender, EventArgs e)
{
    // Tworzenie tablicy elementów do dodania.
    string[] Genre = { "Fantastyka", "Powieści", "Komputery", "Historia", "Religia" };

    cblGenre.DataSource = Genre;
    cblGenre.DataBind();
}
```

Można by oczekiwać, że wyniki nie będą się różniły od tych przedstawionych na rysunku 4.12, ale nie w tym przypadku. Zamiast tego, otrzymamy witrynę pokazaną na rysunku 4.13.



Rysunek 4.13. Elementy kontrolki CheckBoxList dodane za pomocą DataBind()

W poprzednim przykładzie, elementy `ListItem` zostały dodane za pomocą z pętli `for` przez metodę zdarzenia `Init` po utworzeniu kontrolki. W tym przykładzie istniejące już obiekty `ListItem` zostały zastąpione nowym źródłem danych, ponieważ zbiór `ListControl.Items` został zainicjalizowany przez źródło danych, a więc poprzednio zdefiniowane obiekty `ListItem` zostały utracone.

Opisane powyżej zachowanie jest domyślne w trakcie łączenia danych z kontrolką `ListControl`. Ewentualnie można ustawić właściwość `AppendDateBoundItems` (nowość w ASP.NET 2.0) kontrolki jako `true`, co spowoduje, że dołączane elementy danych zostaną dodane do istniejącego zbioru `Items`, zamiast zastąpić ten zbiór.

Obsługa zaznaczeń użytkownika

Kiedy użytkownik zaznaczy bądź usunie zaznaczenie jednego z pól wyboru kontrolki `CheckBoxList`, zostanie wywołane zdarzenie `SelectedIndexChanged`. Zdarzenie przekazuje argument typu `EventArgs`, które nie zawiera żadnych właściwości. Poprzez ustawianie atrybutu dla tej obsługi zdarzenia oraz umieszczenie kodu w metodzie obsługi zdarzenia, można odpowiadać na kliknięcia użytkownika jednego z pól wyboru. Jeżeli właściwość `AutoPostBack` jest ustawiona jako `true`, wówczas odpowiedź nastąpi natychmiastowo. W przeciwnym przypadku, odpowiedź pojawi się dopiero wtedy, gdy formularz zostanie ponownie przekazany do serwera.

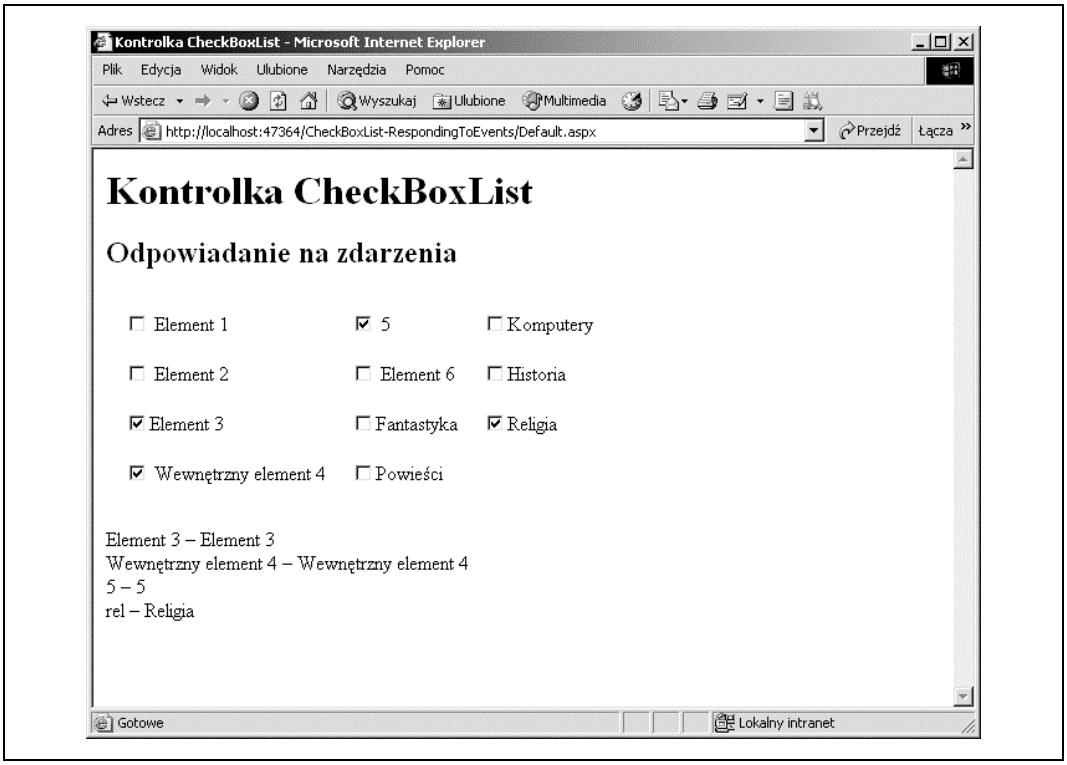
Aby zobaczyć działanie tego mechanizmu, skopiujemy poprzedni przykład do nowej witryny *CheckBoxList-RespondingToEvents*. Do witryny dodajemy pogrubione wiersze kodu przedstawionego na listingu 4.22 pliku z treścią oraz przedstawionego na listingu 4.23 pliku ukrytego kodu. Ukończona witryna internetowa wraz z kilkoma polami wyboru została pokazana na rysunku 4.14.

Listing 4.22. Plik Default.aspx witryny CheckBoxList-RespondingToEvents

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka CheckBoxList</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka CheckBoxList</h1>
      <h2>Odpowiadanie na zdarzenia</h2>
      <asp:CheckBoxList ID="cblGenre" runat="server"
            AutoPostBack="True"
            CellPadding="5"
            CellSpacing="10"
            RepeatColumns="3"
            OnInit="cblGenre_Init"
            OnSelectedIndexChanged="cblGenre_SelectedIndexChanged">
         <asp:ListItem value="1"> Element 1 </asp:ListItem>
         <asp:ListItem text="Element 2" value="2"></asp:ListItem>
         <asp:ListItem text="Element 3" />
         <asp:ListItem text="Element 4"> Wewnętrzny element 4 </asp:ListItem>
         <asp:ListItem value="5"></asp:ListItem>
         <asp:ListItem> Element 6 </asp:ListItem>
      </asp:CheckBoxList>
      <asp:Label ID="lblGenre" runat="server" Text="Label"></asp:Label>
   </div>
   </form>
</body>
</html>
```

Listing 4.23. Zmodyfikowana obsługa zdarzenia w pliku ukrytego kodu witryny CheckBoxList-RespondingToEvents

```
protected void cblGenre_Init(object sender, EventArgs e)
{
   string[] Genre = { "Fantastyka", "Powieści", "Komputery", "Historia", "Religia" };
   string[] Code = {"sf", "nvl", "cmp", "his", "rel" };

   for (int i = 0; i < Genre.Length; i++)
   {
      // Dodajemy zarówno właściwość Text, jak i Value.
      this.cblGenre.Items.Add(new ListItem(Genre[i], Code[i]));
   }
}

protected void cblGenre_SelectedIndexChanged(object sender, EventArgs e)
{
   StringBuilder sb = new StringBuilder();
   foreach (ListItem li in cblGenre.Items)
   {
      if (li.Selected == true)
      {
         sb.Append("<br/>" + li.Value + " - " + li.Text);
      }
   }
```



Rysunek 4.14. Witryna internetowa CheckBoxList-RespondingToEvents po zaznaczeniu kilku pól wyboru

```
    if (sb.Length == 0)
       lblGenre.Text = "Brak zaznaczonej kategorii.";
    else
       lblGenre.Text = sb.ToString();

}
```

Warto zwrócić uwagę, w jaki sposób klasa `StringBuilder` jest wykorzystywana w metodzie `cblGenre_SelectedIndexChanged` do utworzenia ciągu znaków, zamiast łączenia każdej wartości ciągu znaków z poprzednią wartością, jak w poniższym wierszu kodu C#:

```
str += "<br/>" + li.Value + " = " + li.Text;
```

Ciągi znakowe są niezmienne. Gdy piszemy:

```
String pierwszyCiag = "Witaj";
String pierwszyCiag += " świecie";
```

to wygląda jak połączenie drugiej części ciągu znakowego z `pierwszyCiag`. natomiast faktycznie wykonywanym działaniem jest zaalokowaniem i przypisanie drugiego ciągu znaków do odniesienia ciągu znaków oraz zniszczenie pierwszego ciągu znaków. Jeżeli wykonujemy dużo tego typu działań (na przykład w pętli), wówczas jest to bardzo nieefektywne działanie, ponieważ tworzenie i niszczenie ciągów znakowych są czasochłonnymi operacjami.

Klasa `StringBuilder` dostarcza efektywniejszego sposobu konstruowania ciągów znakowych, ponieważ nie wymaga tworzenia nowego ciągu znaków przy każdej modyfikacji.

Kod, który w stosunku do poprzednich przykładów jest kodem dodatkowym, został na listingach 4.22 i 4.23 zapisany pogrubioną czcionką i przedstawia odpowiedź na zdarzenie `SelectedIndexChanged`.

W kodzie na listingach 4.22 i 4.23 dodaliśmy atrybut `OnSelectedIndexChanged` służący do identyfikacji obsługi zdarzeń dla zdarzenia `SelectedIndexChanged`. Podobnie jak wszystkie obsługi zdarzeń, także i ten atrybut tworzy się przez umieszczenie na początku nazwy zdarzenia przedrostku „`On`". Następnie dodajemy do formularza kontrolkę `Label` (`lblGenre`) wyświetlającą zaznaczone elementy.

Obsługa zdarzeń prowadzi do metody o nazwie `cblGenre_SelectedIndexChanged` w pliku ukrytego kodu. W tej metodzie obsługi zdarzeń przechodzimy kolejno przez wszystkie elementy `ListItem` kontrolki `CheckBoxList`. Dla każdego elementu `ListItem` możemy wówczas sprawdzić, czy właściwość `Selected` została ustawiona jako `true`. Jeżeli tak jest, wtedy do konstruowanego przez nas ciągu znakowego HTML dodajemy właściwość `Value` tego elementu, używając w tym celu klasy `StringBuilder`. Na końcu jest sprawdzana długość `StringBuilder`. Jeżeli wynosi ona zero, wówczas zostaje wyświetlony odpowiedni komunikat. W przeciwnym przypadku zostaje wyświetlony ciąg znakowy `StringBuilder`, zawierający zaznaczone wartości.

Kontrolka RadioButtonList

Kontrolka `RadioButtonList` jest bardzo podobna do kontrolki `CheckBoxList`. Obie wywodzą się z klasy `ListControl` i współużytkują wszystkie te same właściwości, zdarzenia i metody. Jedyną różnicą między tymi dwiema kontrolkami (poza okrągłym kształtem kontrolki `RadioButtonList` i kwadratowym kształtem kontrolki `CheckBoxList`) jest fakt, że kontrolka `RadioButtonList` może posiadać w danym momencie tylko jeden zaznaczony element. Kiedy jeden z elementów zostanie zaznaczony, pozostałe zaznaczenia elementów z danej grupy są usuwane.

Kontrolki `RadioButtonList` i `CheckBoxList` współużytkują dwie właściwości dziedziczone z klasy `ListControl`, które zostały przedstawione w tabeli 4.8.

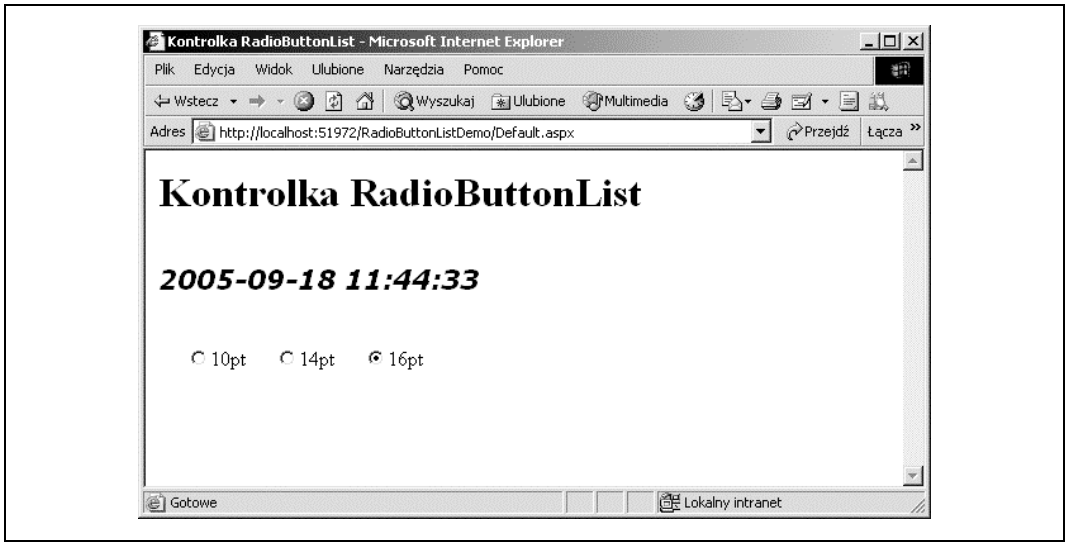
Tabela 4.8. Właściwości zaznaczenia dziedziczone z klasy ListControl

Nazwa	Typ	Pobierz	Ustaw	Opis
`SelectedIndex`	Integer	x	x	Najmniejszy indeks zaznaczonych elementów z listy. Jeżeli jego wartość wynosi `-1`, wówczas nic nie zostało zaznaczone.
`SelectedItem`	ListItem	x		Zwraca zaznaczony element z najmniejszym indeksem.

W celu zademonstrowania dużej użyteczności tych właściwości, skopiujemy witrynę internetową *RadioButtonDemo* wykorzystywaną do przedstawienia przycisków opcji do nowej witryny o nazwie *RadioButtonListDemo*. Istniejące w witrynie trzy przyciski opcji, które wpływały na wielkość czcionki, należy zastąpić pojedynczą kontrolką `RadioButtonList` o nazwie `rblSize`, jak to przedstawiono na listingu 4.24. Ukończona strona po zaznaczeniu wielkości czcionki została pokazana na rysunku 4.15. Wygląda podobnie jak wersja z oddzielnymi przyciskami opcji, ale teraz jest łatwiejsza do zapełnienia jej z źródła danych.

Listing 4.24. Plik Default.aspx witryny internetowej RadioButtonListDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka RadioButtonList</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka RadioButtonList</h1>
      <br />
      <asp:Label ID="lblTime" runat="server"
         OnInit="lblTime_Init"></asp:Label>
      <br />
      <br />
      <asp:RadioButtonList
         id="rblSize" runat="server"
         AutoPostBack="True"
         cellSpacing="20"
         repeatColumns="3"
         repeatDirection="horizontal"
         RepeatLayout="table"
         textAlign="right"
         OnSelectedIndexChanged="rblSize_SelectedIndexChanged">

         <asp:ListItem text="10pt" value="10"/>
         <asp:ListItem text="14pt" value="14"/>
         <asp:ListItem text="16pt" value="16"/>
      </asp:RadioButtonList>
   </div>
   </form>
</body>
</html>
```



Rysunek 4.15. Aplikacja RadioButtonListDemo

Obsługa zdarzeń dla tej witryny umieszczona w pliku ukrytego kodu została przedstawiona na listingu 4.25.

Listing 4.25. Obsługi zdarzeń aplikacji RadioButoonListDemo zawarte w pliku ukrytego kodu Default.aspx.cs

```
protected void lblTime_Init(object sender, EventArgs e)
{
   lblTime.Font.Name = "Verdana";
   lblTime.Font.Size = 2O;
   lblTime.Font.Bold = true;
   lblTime.Font.Italic = true;
   lblTime.Text = DateTime.Now.ToString();
}
protected void rblSize_SelectedIndexChanged(object sender, EventArgs e)
{
   //  Sprawdź, czy dokonano jakiegokolwiek zaznaczenia.
   if (rblSize.SelectedIndex != -1)
  {
      int size = Convert.ToInt32(rblSize.SelectedItem.Value);
      lblTime.Font.Size = size;
  }
}
```

W aplikacji *RadioButonListDemo* pierwotne oddzielne przyciski opcji zostały zastąpione przez kontrolkę `RadioButtonList`. Każdy obiekt `ListItem` posiada właściwości `Text` i `Value`. Obsługa zdarzeń `rblSize_SelectedIndexChanged` ustawia właściwość `Font.Size`, wymagając do tego celu liczby całkowitej, którą pobiera z właściwości `SelectedItem.Value` listy przycisków opcji.

```
int size = Convert.ToInt32(rblSize.SelectedItem.Value);
lblTime.Font.Size = size;
```

Przedstawione rozwiązanie funkcjonuje prawidłowo, ponieważ C# zapewnia wyraźną konwersję operatora, konwertując `Int32` na nowy egzemplarz `FontUnit`.

Metoda obsługi zdarzeń korzysta ze wspomnianych wcześniej właściwości `SelectedIndex` i `SelectedItem`. Właściwość `SelectedIndex` przestawia najmniejszą liczbę całkowitą, będącą wartością indeksu wszystkich zaznaczonych elementów. Natomiast właściwość `SelectedItem` zwraca właściwość `Text` elementu wskazanego przez `SelectedIndex`. Ponieważ kontrolka `RadioButtonList` z definicji może posiadać nie więcej niż jeden zaznaczony element, dlatego też `SelectedIndex` i `SelectedItem` będą informować, który z elementów został zaznaczony. Właściwości te stają się bardziej niejednoznaczne, gdy zostaną zastosowane w kontrolce `CheckBoxList` lub innej kontrolce `ListControl`, pozwalającej na wiele zaznaczeń.

Aplikacja *RadioButtonListDemo* sprawdza, czy co najmniej jedna wartość została zaznaczona. Jeżeli żaden z elementów nie został zaznaczony, wówczas właściwość `SelectedIndex` jest równa `−1`. Natomiast, jeśli jakikolwiek element został zaznaczony, to ustawiamy dla niego właściwość `Font.Size`, konwertując właściwość `SelectedItem.Value` do postaci liczby całkowitej. Zwróćmy uwagę na następujące dwa wiersze kodu C# na listingu 4.25:

```
int size = Convert.ToInt32(rblSize.SelectedItem.Value);
lblTime.Font.Size = size;
```

Mogą one zostać napisane jako pojedynczy wiersz:

```
lblTime.Font.Size = Convert.ToInt32(rblSize.SelectedItem.Value);
```

Mimo takich możliwości należy wspomnieć, że często dłuższe wiersze kodu rozbija się na większą liczbę krótszych linii. Taka praktyka zapewnia większą czytelność kodu źródłowego i ułatwia usuwanie błędów aplikacji.

Kontrolka DropDownList

Kontrolka `DropDownList` wyświetla w danym momencie pojedynczy element wraz z przyciskiem pozwalającym na rozwinięcie listy i wyświetlenie większej liczby możliwych opcji wyboru. Tylko jeden element może zostać zaznaczony.

Kolejny przykład *DropDownListDemo* zademonstruje użycie kontrolki `DropDownList`. W obsłudze zdarzenia `Page_Load` zostanie użyta dwuwymiarowa tablica przechowująca właściwości `Text` i `Value`. Wspomniana tablica jest następnie używana w celu dodania obiektów `ListItem` do zbioru `Item`. Ukończona aplikacja została pokazana na rysunku 4.16. Plik z treścią dla tej strony został przedstawiony na listingu 4.26, natomiast zawartość pliku ukrytego kodu jest na listingu 4.27.

Rysunek 4.16. Aplikacja DropDownListDemo

Listing 4.26. Plik Default.aspx witryny internetowej DropDownListDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka DropDownList</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
```

```
      <h1>Kontrolka DropDownList</h1>

      <asp:DropDownList ID="ddl" runat="server"
         AutoPostBack="True"
         OnSelectedIndexChanged="ddl_SelectedIndexChanged">
      </asp:DropDownList>
      <br />
      <asp:Label ID="lblDdl" runat="server" ></asp:Label>

   </div>
   </form>
</body>
</html>
```

Listing 4.27. Obsługa zdarzeń aplikacji DropDownListDemo zawartych w pliku ukrytego kodu

```
protected void Page_Load(object sender, EventArgs e)
{
   if (!IsPostBack)
   {
      // Tworzymy dwuwymiarową tablicą dla listy.
      // Pierwszy wymiar zawiera tytuł książki.
      // Drugi wymiar zawiera numer ISBN.
      string[,] books = {
         {"Programming C#","0596001177"},
         {"Programming Visual Basic .NET","0596004389"},
         {"Programming .NET Windows Applications","0596003218"},
         {"Programming ASP.NET","0596001711"},
         {"WebClasses From Scratch","078972126O"},
         {"Teach Yourself C++ in 21 Days","067232072X"},
         {"Teach Yourself C++ in 10 Minutes","067231603X"},
         {"XML & Java From Scratch","0789724766"},
         {"Complete Idiot's Guide to a Career in Computer Programming","0789719959"},
         {"XML Web Documents From Scratch","0789723166"},
         {"Clouds To Code","1861000952"},
         {"C++: An Introduction to Programming","157576O614"},
         {"C++ Unleashed","0672312395"}
      };

      // W tym miejscu zapełniamy listę.
      for (int i = 0; i < books.GetLength(0); i++)
      {
         // Dodajemy zarówno wartości Text, jak i Value.
         ddl.Items.Add(new ListItem(books[i, 0], books[i, 1]));
      }
   }
}

protected void ddl_SelectedIndexChanged(object sender, EventArgs e)
{
   // Sprawdzamy, czy cokolwiek zostało zaznaczone.
   if (ddl.SelectedIndex != -1)
   {
      lblDdl.Text = ddl.SelectedItem.Text + " ---> ISBN: " +
         ddl.SelectedValue;
   }
}
```

Na listingu 4.26 dodano kontrolkę `DropDownList` z atrybutem `ID` o wartości `ddl`. Ta kontrolka zostaje następnie zapełniona za pomocą metody obsługi zdarzenia `Page_Load`, kiedy strona jest wczytana po raz pierwszy.

Aby uniemożliwić uruchamiane tego kodu przy każdym wczytywaniu strony, trzeba sprawdzić, czy wartość właściwości `IsPostBack` jest ustawiona jako `true`. Właściwość `IsPostBack` posiada wartość `false` przy pierwszym wczytaniu strony, ale przy każdym przekazaniu formularza z powrotem do serwera przyjmuje wartość `true`, co jest wynikiem działalności użytkownika na kontrolce. W wielu aplikacjach zawartość kontrolek jest wypełniana danymi z bazy danych, co może być dość kosztowną operacją. Korzystanie z bazy danych jedynie wtedy, gdy jest to niezbędne, powoduje, że ta implementacja jest efektywniejsza. W przykładzie *CheckBoxList-ArrayItemsAndValues* do wypełnienia kontrolki właściwościami `Text` i `Value` używaliśmy dwóch tablic. W bieżącym przykładzie wykorzystamy jedną dwuwymiarową tablicę, która wykona to samo zadanie. Podobnie jak we wcześniejszym przykładzie, zostanie wywołana metoda `Items.Add`, która dodaje do kontrolki elementy `ListItem`. W rozdziale 9. zostanie omówiony sposób wypełniania kontrolki `ListControl` danymi pochodzącymi z bazy danych.

Podobnie jak w innych kontrolkach `ListControl`, atrybut `OnSelectedIndexChanged` wskazuje metodę obsługi zdarzenia — `ddl_SelectedIndexChanged`. W tej metodzie, podobnie jak w przypadku kontrolki `RadioButtonList`, pierwszą czynnością jest sprawdzenie, czy którykolwiek element został zaznaczony. Wykonujemy to zadanie, testując, czy wartość właściwości `SelectedIndex` jest równa `-1`. Jeżeli jakikolwiek element został zaznaczony, wówczas w kontrolce `Label` o nazwie `lblDdl` wyświetlamy połączenie właściwości `SelectedItem.Text` i `SelectedValue`.

Kontrolka ListBox

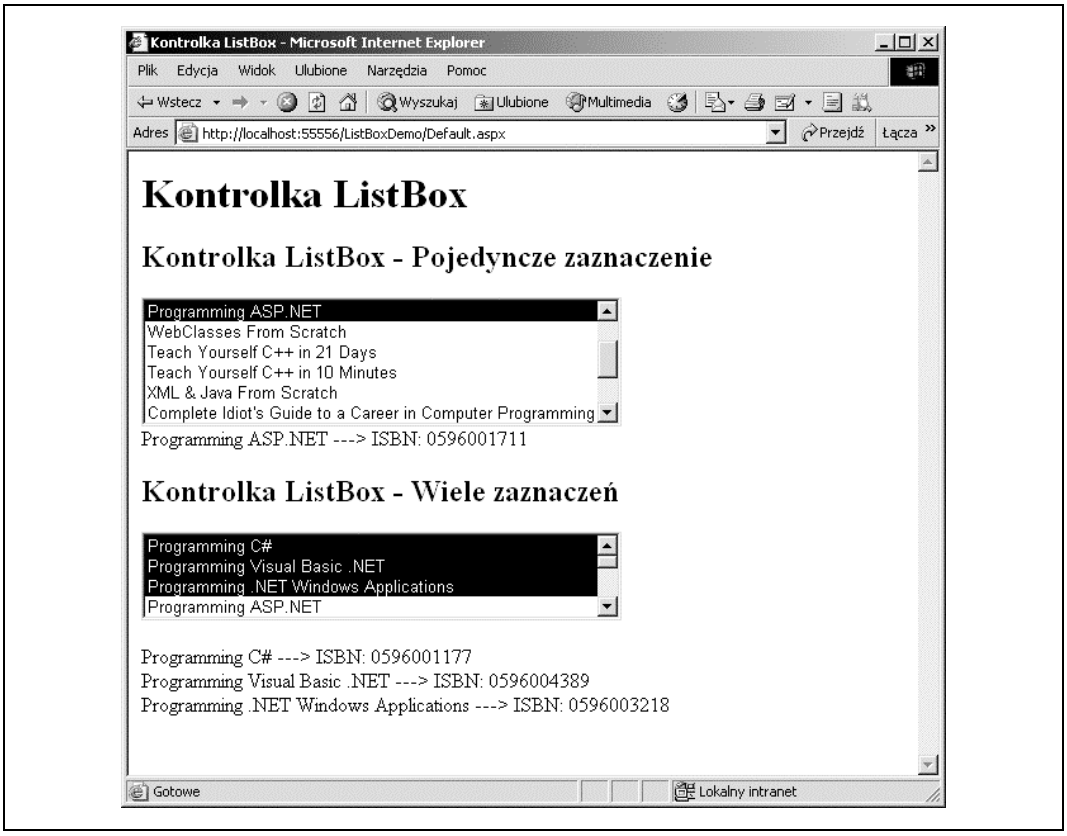
Kontrolki `ListBox` są bardzo podobne do kontrolek `DropDownList`. Różnica między tymi kontrolkami polega na tym, że w przypadku `ListBox` wszystkie elementy listy są od razu widoczne. Jeżeli zachodzi taka konieczność, zostaje wyświetlony pionowy pasek przewijania. Zmiana właściwości `SelectionMode` z domyślnej wartości `Single` na `Multiple` powoduje, że w kontrolce `ListBox` można dokonać zaznaczenia wielu elementów.

Kolejny przykład *ListBoxDemo*, pokazany na rysunku 4.17, przedstawia dwie różne kontrolki `ListBox`. Pierwsza z nich pozwala na dokonanie jednego zaznaczenia, podczas gdy druga zezwala na dokonanie wielu zaznaczeń. Jak będziemy mogli się o tym przekonać, implementacja obu kontrolek `ListBox` jest niemal identyczna. Wyraźną różnicą między nimi będzie technika użyta do identyfikacji zaznaczonych elementów.

Zasadnicze różnice między tym a poprzednim przykładem (*DropDownListDemo*) zostały przedstawione pogrubioną czcionką na listingu 4.28, na którym pokazano zawartość pliku z treścią aplikacji *ListBoxDemo* oraz na listingu 4.29, prezentującym obsługę zdarzeń aplikacji *ListBoxDemo* w pliku ukrytego kodu. Wspomniane różnice obejmują oprócz dwóch kontrolek `ListBox` również modyfikację metody `Page_Load`, wypełniającej te kontrolki oraz dodatkowe obsługi zdarzenia nowych kontrolek.

Listing 4.28. Plik Default.aspx witryny internetowej ListBoxDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka ListBox</title>
</head>
```



Rysunek 4.17. Aplikacja ListBoxDemo

```
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka ListBox</h1>
      <h2>Kontrolka ListBox - Pojedyncze zaznaczenie</h2>
      <asp:ListBox ID="lbSingle" runat="server"
         AutoPostBack="True"
         Rows="6"
         OnSelectedIndexChanged="lbSingle_SelectedIndexChanged">
      </asp:ListBox>
      <br />
      <asp:Label ID="lblSingle" runat="server"></asp:Label>
      <br />
      <h2>Kontrolka ListBox - Wiele zaznaczeń</h2>
      <asp:ListBox ID="lbMulti" runat="server"
         AutoPostBack="True"
         SelectionMode="Multiple"
         OnSelectedIndexChanged="lbMulti_SelectedIndexChanged">
      </asp:ListBox>
      <br />
      <asp:Label ID="lblMulti" runat="server"></asp:Label>
   </div>
   </form>
</body>
</html>
```

Listing 4.29. Obsługa zdarzeń aplikacji ListBoxDemo zawarta w pliku ukrytego kodu

```
protected void Page_Load(object sender, EventArgs e)
{
   if (!IsPostBack)
   {
      //  Tworzymy dwuwymiarową tablicą dla listy.
      //  Pierwszy wymiar zawiera tytuł książki.
      //  Drugi wymiar zawiera numer ISBN.
      string[,] books = {
         {"Programming C#","0596001177"},
         {"Programming Visual Basic .NET","0596004389"},
         {"Programming .NET Windows Applications","0596003218"},
         {"Programming ASP.NET","0596001711"},
         {"WebClasses From Scratch","078972126O"},
         {"Teach Yourself C++ in 21 Days","067232072X"},
         {"Teach Yourself C++ in 10 Minutes","067231603X"},
         {"XML & Java From Scratch","0789724766"},
         {"Complete Idiot's Guide to a Career in Computer Programming","0789719959"},
         {"XML Web Documents From Scratch","0789723166"},
         {"Clouds To Code","1861000952"},
         {"C++: An Introduction to Programming","157576O614"},
         {"C++ Unleashed","0672312395"}
      };

      //  W tym miejscu zapełniamy listę.
      for (int i = 0; i < books.GetLength(0); i++)
      {
         //  Dodajemy zarówno wartości Text, jak i Value.
         lbSingle.Items.Add(new ListItem(books[i, 0], books[i, 1]));
         lbMulti.Items.Add(new ListItem(books[i, 0], books[i, 1]))
      }
   }
}

protected void lbSingle_SelectedIndexChanged(object sender, EventArgs e)
{
   //  Sprawdzamy, czy wybrano jakiś element.
   if (lbSingle.SelectedIndex != -1)
   {
      lblSingle.Text = lbSingle.SelectedItem.Text + " ---> ISBN: " +
         lbSingle.SelectedItem.Value;
   }
}

protected void lbMulti_SelectedIndexChanged(object sender, EventArgs e)
{
   string str = "";
   foreach (ListItem li in lbMulti.Items)
   {
      if (li.Selected == true)
      {
         str += "<br/>" + li.Text + " ---> ISBN: " + li.Value;
      }
   }

   // Alternatywna technika.
   // foreach (int i in lbMulti.GetSelectedIndices())
   // {
   //    ListItem li = lbMulti.Items[i];
   //    str += "<br/>" + li.Text + " ---> ISBN: " + li.Value;
   // }
```

```
    if (str.Length == 0)
       lblMulti.Text = "Nie została wybrana żadna książka.";
    else
       lblMulti.Text = str;
 }
```

Kontrolka `ListBox` posiada dwie dodatkowe właściwości, oprócz tych dziedziczonych z klasy `ListControl`. Dodatkowe właściwości kontrolki zostały przedstawione w tabeli 4.9.

Tabela 4.9. Właściwości kontrolki ListBox, które nie są dziedziczone z klasy ListControl

Nazwa	Typ	Pobierz	Ustaw	Wartości	Opis
`SelectionMode`	ListSelectionMode	x	x	`Single`, `Multiple`	Określa, czy kontrolka `ListBox` znajduje się w trybie zaznaczania pojedynczego (`Single`) czy wielokrotnego (`Multiple`). Domyślnym trybem jest `Single`.
`Rows`	Integer	x	x		Liczba wyświetlanych wierszy. Wartością domyślną jest 4.

Pierwsza kontrolka `ListBox` umieszczona w aplikacji *ListBoxDemo* z identyfikatorem `lbSingle` jest polem listy pozwalającym na pojedyncze zaznaczenie. Właściwość `Row` posiada wartość `6`, co oznacza, że aplikacja wyświetli sześć elementów. Kontrolka zostanie wypełniona więcej niż tylko sześcioma elementami, zatem automatycznie pojawia się pionowy pasek przewijania. Jeżeli zostanie zaznaczony drugi element, wówczas zaznaczenie przy poprzednio wybranym elemencie zostanie usunięte. Podobnie jak w większości przykładów w tym rozdziale, właściwość `AutoPostBack` została ustawiona jako `true`, tak więc efekty zmian będą widoczne natychmiastowo.

Druga kontrolka `ListBox` o identyfikatorze `lbMulti` jest polem wyboru pozwalającym na zaznaczanie wielu opcji. Właściwość `Row` tej kontrolki nie została ustawiona, a więc będą widoczne domyślne cztery wiersze. Ponieważ jest to kontrolka typu `Multiselect`, zostaną zastosowane standardowe techniki Windows do obsługi wielokrotnych zaznaczeń.

Techniki Windows obsługujące wielokrotne zaznaczenia

Większość aplikacji Windows używa tych samych technik do zaznaczania wielu elementów.

Aby dodać *zakres elementów* do zaznaczonej listy, klikamy pierwszy element przeznaczony do zaznaczenia, a następnie przytrzymując klawisz *Shift*, klikamy ostatni element przeznaczony do zaznaczenia. Wszystkie elementy znajdujące się między dwoma klikniętymi zostaną dodane do zaznaczenia.

Możemy również zaznaczyć zakres wielu elementów, klikając lewym przyciskiem myszy pierwszy z nich, a następnie, wciąż trzymając wciśnięty przycisk myszy, przeciągnąć ją na ostatni element zaznaczonego zakresu. Po zwolnieniu przycisku myszy otrzymuje się zaznaczony zakres.

W celu dodania *niesąsiadujących ze sobą elementów* do zaznaczenia, przytrzymujemy klawisz *Ctrl* w trakcie klikania tych elementów.

Usunięcie zaznaczenia z pojedynczego — już zaznaczonego — elementu, wymaga wciśnięcia klawisza *Ctrl* w trakcie kliknięcia danego elementu, co powoduje przełączenie jego stanu zaznaczenia.

Obsługi zdarzeń, które przetwarzają zaznaczenia z tych dwóch pól list są bardzo różne. Obsługa zdarzenia dla pola listy zezwalającego tylko na pojedyncze zaznaczenie jest bardzo podobna do zdarzenia kontrolki `DropDownList` lub każdej innej kontrolki `ListControl` pojedynczego zaznaczenia, na przykład kontrolki `RadioButtonList`.

Przedstawiona na listingu 4.29 obsługa zdarzenia pola listy pozwalającego na dokonanie wielu zaznaczeń zawiera dwie odmienne techniki tworzenia ciągu zaznaczonych elementów. Pierwsza technika jest podobna do użytej w przypadku kontrolki `CheckBoxList`. Polega ona na kolejnym przeglądzie zbioru elementów `ListItem` i sprawdzeniu, czy właściwość `Selected` posiada wartość `true`. Jeżeli wspomniana właściwość posiada wartość `true`, wówczas do ciągu znakowego wyświetlanego w kontrolce `Label` zostają dodane właściwości `Text` i `Value`. Druga technika, która na listingu 4.29 została umieszczona w komentarzu, polega na zastosowaniu metody `ListBox.GetSelectedIndices` (nowość w ASP.NET 2.0) do zwrócenia tablicy liczb całkowitych indeksów wszystkich zaznaczonych elementów. Technika ta polega na przeglądzie tablicy, podczas którego każdy zaznaczony element `ListItem` otrzymuje swoje właściwości `Text` i `Value`.

Kontrolka BulletedList

Kontrolka `BulletedList`, będąca nowością w ASP.NET 2.0, jest kontrolka serwerową ASP.NET, która umożliwia tworzenie elementów witryn analogicznych do uporządkowanych (`<ol>`) i nieuporządkowanych (`<ul>`) list HTML. Wygląd i dostępne funkcje list są określane przez właściwości kontrolki `BulletedList`. Podobnie jak w przypadku innych kontrolek wywodzących się z klasy `LsitControl`, także i `BulletedList` posiada właściwość `Items`, która jest zbiorem obiektów `ListItem`.

Styl znaku wypunktowania jest określony przez właściwość `BulletStyle`. Poprawne wartości są zawarte wewnątrz wyliczenia `BulletStyle` i obejmują wartości takie jak: `Circle`, `Disc`, `Numbered`, `LowerAlpha`, `UpperAlpha`, `LowerRoman` i `UpperRoman`. Jeżeli właściwość `BulletStyle` nie jest ustalona, wówczas domyślną wartością będzie `NotSet`, co powoduje, że przeglądarka sama określa zastosowany styl znaku wypunktowania. Zwykle będzie on taki sam, jaki otrzymujemy ustawiając wartość `Disc`.

W przypadku ustawienia właściwości `BulletStyle` na styl liczbowy bądź alfabetyczny, na przykład `Numbered`, `LowerAlpha`, `UpperAlpha`, `LowerRoman` lub `UpperRoman`, wartość początkowa może zostać ustalona za pomocą właściwości `FirstBulletNumber`. Wartością domyślną jest `1`. Liczbowy styl znaków wypunktowania (`Numbered`, `LowerRoman` lub `UpperRoman`) powoduje wyświetlenie liczb, podczas gdy styl alfabetyczny wyświetla litery w kolejności alfabetycznej.

Właściwość `DisplayMode` określa wygląd oraz zestaw dostępnych funkcji. Może przyjąć jedną z trzech wartości wyliczenia `BulletedListDisplayMode`:

`Text`
: Jest to wartość domyślna i powoduje wyświetlenie zawartości listy w postaci tekstu. W przypadku zastosowania tej wartości, kontrolka nie będzie miała przypisanych żadnych zdarzeń. Oznacza to, że poza przeglądaniem, nie będzie możliwości prowadzenia interakcji z użytkownikiem.

`HyperLink`
: Każdy element `ListItem` zostaje wyświetlony jako podkreślone łącze. Kliknięcie łącza nie powoduje wywołania żadnych zdarzeń po stronie serwera, a formularz *nie zostanie* przekazany z powrotem do serwera. Podobnie jak w przypadku samej kontrolki `HyperLink`, nastąpi wyświetlenie strony dostępnej pod adresem URL podanym w właściwości `Value`, ustawionej dla wybranego elementu `ListItem`.

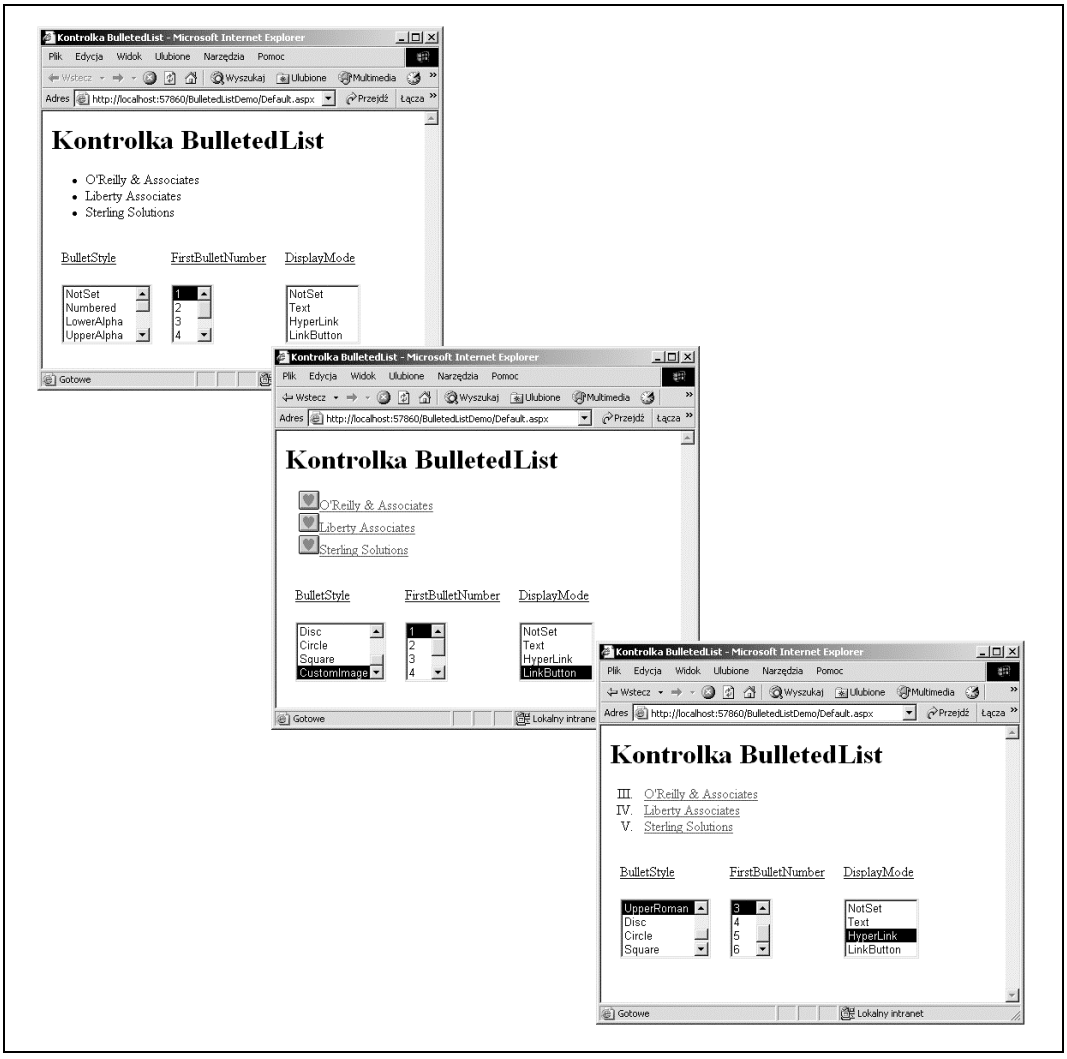
  Właściwość `Target` kontrolki `BulletedList` działa w połączeniu z właściwością `DisplayMode` ustawioną jako `HyperLink`, określając, w którym oknie przeglądarki zostanie wyświetlona docelowa strona. Wartości właściwości `Target` są takie same jak właściwości przedstawione w tabeli 4.3 przy omawianiu kontrolki `HyperLink`.

`LinkButton`
: Każdy element `ListItem` zostanie wyświetlony jako podkreślone łącze, dokładnie tak samo jak w przypadku wartości `HyperLink`, jednak jeśli użytkownik kliknie element, wówczas zostanie wywołane zdarzenie `BulletedList.Click` i nastąpi natychmiastowe przekazanie z powrotem do serwera. Zostanie również uruchomiona obsługa zdarzeń po stronie serwera, określona przez atrybut `OnClick` kontrolki `BulletedList`.

Przykład *BulletedListDemo* pokazany na rysunku 4.18 demonstruje różne style wypunktowania, ich wartości początkowe i tryby wyświetlania, jak również obsługę zdarzeń kontrolki `BulletedList`. Plik z treścią tego przykładu został przedstawiony na listingu 4.30, natomiast metody obsługi zdarzeń z pliku ukrytego kodu zostały przedstawione na listingu 4.31.

Listing 4.30. Plik Default.aspx witryny internetowej BulletedListDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka BulletedList</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka BulletedList</h1>

      <asp:BulletedList ID="bltList" runat="server"
               OnClick="bltList_Click"
               Target="_blank">
         <asp:ListItem Value="http://www.oreilly.com/">
            O'Reilly & Associates</asp:ListItem>
         <asp:ListItem Value="http://www.LibertyAssociates.com">
            Liberty Associates</asp:ListItem>
         <asp:ListItem Value="http://www.stersol.com"
            Text="Sterling Solutions"></asp:ListItem>
      </asp:BulletedList>

      <table cellpadding="10">
         <tr>
            <td colspan="3" id="tdMessage" runat="server">
            </td>
         </tr>
         <tr>
```



Rysunek 4.18. Aplikacja BulletedListDemo wraz ze wszystkimi wartościami domyślnymi

```
            <td>
               <u>BulletStyle</u>
            </td>
            <td>
               <u>FirstBulletNumber</u>
            </td>
            <td>
               <u>DisplayMode</u>
            </td>
         </tr>
         <tr>
            <td>
               <asp:ListBox ID="lbBulletStyle" runat="server"
                     AutoPostBack="true"
                     OnSelectedIndexChanged="lb_SelectedIndexChanged">
                  <asp:ListItem>NotSet</asp:ListItem>
                  <asp:ListItem>Numbered</asp:ListItem>
```

```
                  <asp:ListItem>LowerAlpha</asp:ListItem>
                  <asp:ListItem>UpperAlpha</asp:ListItem>
                  <asp:ListItem>LowerRoman</asp:ListItem>
                  <asp:ListItem>UpperRoman</asp:ListItem>
                  <asp:ListItem>Disc</asp:ListItem>
                  <asp:ListItem>Circle</asp:ListItem>
                  <asp:ListItem>Square</asp:ListItem>
                  <asp:ListItem>DowolnyObrazek</asp:ListItem>
               </asp:ListBox>
            </td>
            <td>
               <asp:ListBox ID="lbFirstBulletNumber" runat="server"
                     AutoPostBack="true"
                     Width="50"
                     OnSelectedIndexChanged="lb_SelectedIndexChanged">
                  <asp:ListItem Selected="True">1</asp:ListItem>
                  <asp:ListItem>2</asp:ListItem>
                  <asp:ListItem>3</asp:ListItem>
                  <asp:ListItem>4</asp:ListItem>
                  <asp:ListItem>5</asp:ListItem>
                  <asp:ListItem>6</asp:ListItem>
               </asp:ListBox>
            </td>
            <td>
               <asp:ListBox ID="lbDisplayMode" runat="server"
                     AutoPostBack="true"
                     OnSelectedIndexChanged="lb_SelectedIndexChanged">
                  <asp:ListItem>NotSet</asp:ListItem>
                  <asp:ListItem>Text</asp:ListItem>
                  <asp:ListItem>HyperLink</asp:ListItem>
                  <asp:ListItem>LinkButton</asp:ListItem>
               </asp:ListBox>
            </td>
         </tr>
      </table>
   </div>
   </form>
</body>
</html>
```

Listing 4.31. Metody obsługi zdarzeń aplikacji ListBoxDemo zawarte w pliku ukrytego kodu default.aspx.cs

```
protected void lb_SelectedIndexChanged(object sender, EventArgs e)
{
   ListBox lb = (ListBox)sender;
   string strID = lb.ID;
   string strValue = lb.SelectedValue;

   switch (strID)
   {
      case "lbBulletStyle":
         BulletStyle style =
            (BulletStyle)Enum.Parse(typeof(BulletStyle), strValue);
         bltList.BulletStyle = style;

         //  Przypadek specjalny dla opcji DowolnyObrazek.
         if (style == BulletStyle.DowolnyObrazek)
         {
            bltList.BulletImageUrl = "heart.bmp";
         }
         break;

      case "lbFirstBulletNumber":
         bltList.FirstBulletNumber = Convert.ToInt32(strValue);
         break;
```

```
        case "lbDisplayMode":
           BulletedListDisplayMode displayMode =
              (BulletedListDisplayMode)Enum.Parse(
                 typeof(BulletedListDisplayMode),
                 strValue);

           bltList.DisplayMode = displayMode;
           break;

        default:
           break;
        }
} //  Zamknięcie lb_SelectedIndexChanged.

protected void bltList_Click(object sender, BulletedListEventArgs e)
{
   BulletedList b = (BulletedList)sender;
   tdMessage.InnerHtml = "Selected index: " + e.Index.ToString() +
      "<br>" +
      "Zaznaczona wartość: " + b.Items[e.Index].Value +
      "<br>";
}
```

Na listingu 4.30 kontrolka `BulletedList` posiada w swoim zbiorze `Items` trzy elementy `ListItems`, które zostały dodane statycznie. Wszystkie elementy listy przedstawiają witryny internetowe. Oczekując użycia trybu `HyperLink DisplayMode`, każdy element `ListItem` posiada ustawioną właściwość `Value`, która zawiera docelowy adres URL. Właściwość `Target` kontrolki `BulletedList` została ustawiona jako `_blank`, co zgodnie z opisem przedstawionym w tabeli 4.3 oznacza, że nowa strona zostanie otworzona w nowym nienazwanym oknie przeglądarki.

Atrybut `OnClick` kontrolki `BulletedList` łączy zdarzenie `Click` z metodą `bltList_Click` umieszczoną w pliku ukrytego kodu. Wiersze kodu tej metody na listingu 4.31 zostały pogrubione.

Obsługa zdarzenia dla wspomnianego zdarzenia `Click` będzie wiązała właściwości `Index` i `Value` kliknięty elementu `ListItem` razem z pewnymi elementami HTML i przypisywała ten ciąg znakowy do właściwości `InnerHtml` kontrolki HTML po stronie serwera. Ta obsługa zdarzeń wymaga argumentu zdarzenia typu `BulletedListEventArgs`, który zawiera pojedynczą właściwość `Index`. Właściwość ta zwraca oparty na zerach indeks klikniętego w zbiorze `Items` elementu `ListItem`.

Jednakże, w celu otrzymania jednej z bieżących właściwości `Text` lub `Value` klikniętego elementu `ListItem`, musimy dysponować odniesieniem do określonej kontrolki `BulletedList`, która wywołała zdarzenie. W omawianym przykładzie występuje tylko jedna kontrolka `BulletedList`, więc jej identyfikator jest nam znany: `bltList`. Jednak tutaj została zastosowana ogólniejsza technika i pojedyncza obsługa zdarzenia będzie funkcjonowała z dowolną liczbą kontrolek. W pierwszej kolejności rzutujemy obiekt wyzwalający zdarzenie (hermetyzowany w `sender`) do obiektu typu `BulletedList`, a następnie indeksujemy w zbiorze `ListItems` wyrażanym przez właściwość `Items` tego obiektu `BulletedList`. Te działania przeprowadzamy w następującym wierszu kodu z listingu 4.31:

```
        "Zaznaczona wartość: " + b.Items[e.Index].Value +
```

Chociaż nie są bezpośrednio powiązane z kontrolką BulletedList, to z wszystkimi trzema kontrolkami ListBox na stronie zostały użyte pewne interesujące techniki.

Wszystkie kontrolki ListBox posiadają właściwość AutoPostBack ustawioną jako true, tak więc wszelkie zmiany wartości będą natychmiast uwzględnione. Dodatkowo, wszystkie trzy kontrolki dla zdarzenia SelectedIndexChanged używają tej samej metody obsługi zdarzeń lb_SelectedIndexChanged. Jest ona zaimplementowana w dwóch następujących atrybutach każdej kontrolki typu ListControl:

```
AutoPostBack = "true";
OnSelectedIndexChanged = "lb_SelectedIndexChanged">
```

Patrząc na metodę lb_SelectedIndexChanged w pliku ukrytego kodu z listingu 4.31, widzimy, że pierwszy wiersz kodu pobiera odniesienie do kontrolki, która wywołała zdarzenie przez rzutowanie sender do obiektu ListBox:

```
ListBox lb = (ListBox)sender;
```

Następnie można otrzymać właściwości ID oraz SelectedValue pola listy.

Blok instrukcji switch jest używany do powzięcia akcji odpowiedniej dla każdego pola listy. Działanie kontrolki ListBox, która ustala właściwość FirstBulletNumber, jest jasne: konwertuje właściwość SelectedValue zawartą w ciągu znakowym zmiennej strValue do postaci liczby całkowitej i przypisuje tę liczbę właściwości FirstBulletNumber:

```
bltList.FirstBulletNumber = Convert.ToInt32(strValue);
```

Bloki case dla dwóch pozostałych kontrolek ListBox są znacznie bardziej interesujące. Ich celem jest określenie zaznaczonego elementu, albo BulletStyle, albo DisplayMode i uwzględnienie tego elementu w kontrolce BulletedList. W obu przypadkach zadanie jest realizowane za pomocą statycznej metody Enum.Parse, która konwertuje nazwę lub wartość wyliczonej stałej na jej wyliczony odpowiednik obiektu. Metodzie trzeba przekazać typ wyliczonej stałej oraz jej wartość.

W przypadku lbBulletStyle (lbDisplayMode jest jego dokładnym odpowiednikiem), Type wyliczenia jest uzyskiwany z operatora typeof, który zwraca obiekt System.Type. Wartość zaznaczonej stałej zostaje zawarta w strValue. Posiadając te dwa argumenty, metoda Enum.Parse zwraca obiekt, który następnie rzutujemy do pożądanego typu i przypisujemy zmiennej:

```
BulletStyle style =
   (BulletStyle)Enum.Parse(typeof(BulletStyle), strValue);
```

Ta zmienna może zostać następnie użyta do ustawienia odpowiedniej właściwości:

```
blList.BulletStyle = style;
```

W przypadku lbBulletStyle musimy możliwości DowolnyObrazek przypisać właściwość BulletImageUrl. Poniżej Czytelnik może bezpośrednio porównać BulletStyle z wyliczoną stałą, aby przekonać się, czy nastąpi dopasowanie:

```
if (style == BulletStyle.DowolnyObrazek)
{
   bltList.BulletImageUrl = "heart.bmp";
}
```

Tabele

Tabele są bardzo ważne w układzie strony internetowej, ponieważ stanowią one jeden z podstawowych sposobów kontrolowania układu strony. W czystym kodzie HTML tabele są tworzone i formatowane przez kilka znaczników, z których wiele posiada odpowiedniki w kontrolkach serwerowych ASP.NET. Jeżeli nie zachodzi potrzeba wykorzystania możliwości oferowanych po stronie serwera, wówczas z powodzeniem można zastosować statyczne znaczniki HTML. Natomiast w sytuacji, gdy zachodzi potrzeba kontroli tabeli w trakcie działania aplikacji, wtedy kontrolki serwerowe stają się odpowiednim rozwiązaniem (można również użyć opisanych w poprzednim rozdziale kontrolek serwerowych HTML, choć nie oferują one spójności implementacji i modelu obiektowego udostępnianego przez kontrolki ASP.NET).

Kontrolki serwerowe ASP.NET używane do tworzenia tabel na stronach internetowych zostały scharakteryzowane w tabeli 4.10.

Tabela 4.10. Kontrolki serwerowe ASP.NET używane do tworzenia tabel na stronach internetowych

Kontrolka serwerowa ASP.NET	Odpowiednik HTML	Opis
`Table`	`<table>`	Kontrolka nadrzędna dla kontrolek `TableRow`. Właściwość `Rows` obiektu `Table` jest zbiorem obiektów `TableRow`.
`TableRow`	`<tr>`	Kontrolka nadrzędna dla kontrolek `TableCell`. Właściwość `Cells` obiektu `TableRow` zawiera zbiór obiektów `TableCell`.
`TableCell`	`<td>`	Zawiera wyświetlaną treść. Właściwość `Text` zawiera tekst HTML. Zbiór `Controls` może zawierać inne kontrolki.
`TableHeaderCell`	`<th>`	Wywodzi się z klasy `TableCell`. Kontrolki wyświetlają komórki nagłówkowe.
`TableHeaderRow`	`<thead>`	Tworzy element wiersza nagłówka.
`TableFooterRow`	`<tfoot>`	Tworzy element wiersza stopki.

Istnieje pewne nałożenie się dostępnych funkcji kontrolek `Table` oraz `Data`. Kontrolki danych, włączając w nie kontrolki `GridView`, `Repeater`, `DataList` i `DataGrid`, zostaną szczegółowo omówione w rozdziale 9. Są one wykorzystywane głównie do wyświetlania danych ze źródeł danych, na przykład bazy danych, pliku XML lub tablicy. Zarówno kontrolka `Table`, jak i kontrolki `Data`, mogą zostać używane do wyświetlania danych w tabeli bądź układzie sformatowanych list. W rzeczywistości, wszystkie te kontrolki są generowane w przeglądarce (lub posiadają opcję takiego generowania) jako tabele HTML (można to sprawdzić wyświetlając po prostu źródło wygenerowanej w przeglądarce strony). Różnice między wymienionymi pięcioma kontrolkami zostały przedstawione w tabeli 4.11.

Aplikacja *TableDemo*, pokazana na rysunku 4.19, demonstruje większość podstawowych funkcji tabel dostępnych w kontrolce `Table`. W tym przykładzie do ustawienia atrybutów czcionki dla próbek wyświetlonych w tabeli zostały zastosowane kontrolki `CheckBoxList` oraz `RadioButtonList`. Następnie utworzono tabelę, która zawiera próbki każdej czcionki zainstalowanej w systemie.

Tabela 4.11. Różnice między kontrolką Table a kontrolkami DataList

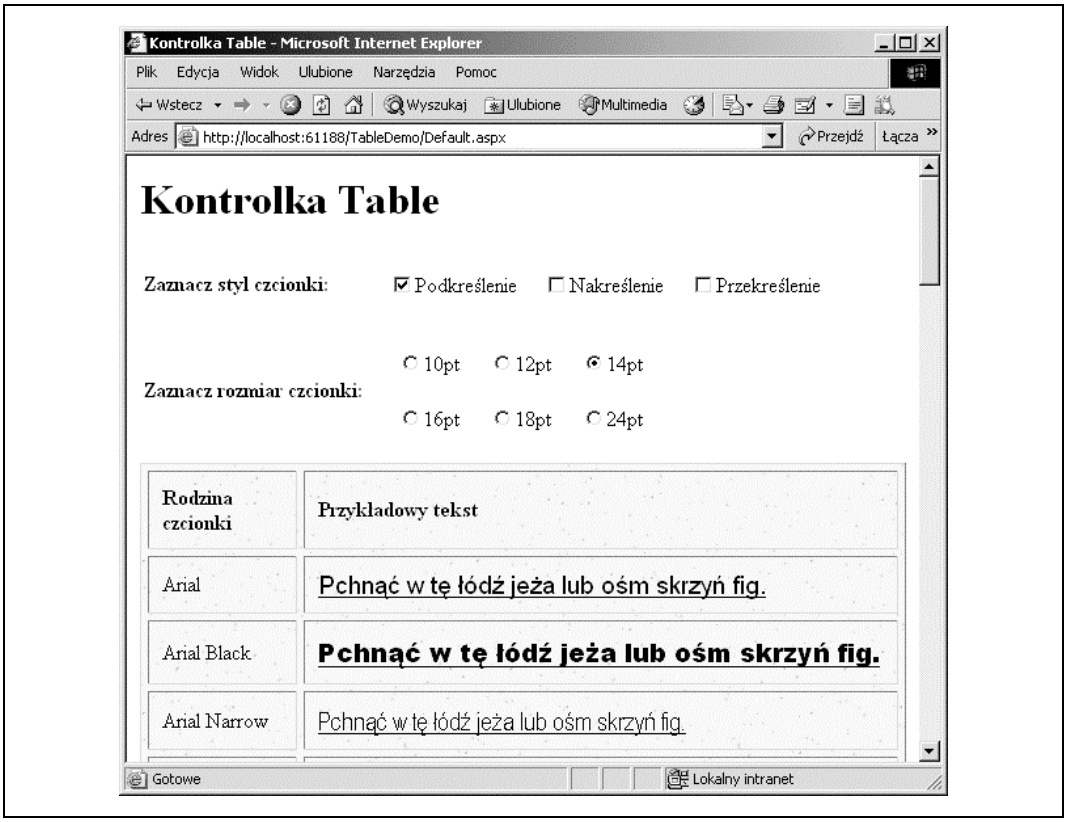
Kontrolka	Użycie	Opis
`Table`	Ogólny układ.	Może zawierać dowolne połączenie tekstu, kodu HTML i innych kontrolek, włączając w to inne tabele. Do kontrolowania wyglądu korzysta z `TableCell`, a nie z szablonów. Nie może łączyć się z danymi, ale może zawierać kontrolki łączące się z danymi.
`Repeater`	Dane tylko do odczytu.	Tylko do odczytu. Do kontrolowania wyglądu korzysta z szablonów. Może się łączyć z danymi. Brak obsługi stronicowania.
`DataList`	Edytowalne dane wyjściowe w postaci listy.	Układem domyślnym jest tabela. Łatwa do dostosowania za pomocą szablonów i stylów. Możliwa do edycji. Może się łączyć z danymi. Brak obsługi stronicowania.
`DataGrid`	Edytowalne dane wyjściowe w postaci listy.	Wyglądem domyślnym jest siatka (to znaczy, jest to tabela możliwa do dostosowania dla własnych potrzeb). Opcjonalnie może korzystać z szablonów. Możliwa do edycji. Może się łączyć z danymi. Obsługuje stronicowanie i sortowanie.
`GridView`	Edytowalne dane wyjściowe w postaci listy.	Podobna do kontrolki `DataGrid`, ale posiada więcej możliwości i wymaga mniejszej ilości kodu do napisania.

Plik z treścią przedstawiony na listingu 4.32 tworzy statyczną tabelę HTML do umieszczenia kontrolek wyboru stylu i wielkości czcionki. Po kilku początkowych nagłówkach znajduje się tam zwykły kod tabeli HTML. Korzystamy więc ze znanych nam znaczników `<table>` zamykających tabelę, wiersze (`<tr>`) oraz komórki tabeli (`<td>`). Nie znajdziemy w tym miejscu żadnych dynamicznych elementów, po prostu powszechnie stosowane techniki wykorzystywania tabel do tworzenia układu strony.

Druga komórka pierwszego wiersza zawiera kontrolkę serwerową `CheckBoxList`, natomiast druga komórka drugiego wiersza zawiera kontrolkę serwerową `RadioButtonList`. Obie kontrolki zostały już omówione we wcześniejszej części tego rozdziału. Obie kontrolki posiadają również kilka wspólnych elementów: atrybut `id`, najważniejszy atrybut `runat` oraz atrybut `AutoPostBack` ustawiony jako `true`, tak więc jakiekolwiek zmiany zostaną natychmiast wprowadzone. Każda z kontrolek posiada różne inne atrybuty, które nadają tabeli pożądany układ.

Listing 4.32. Plik default.aspx aplikacji TableDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka Table</title>
</head>
```



Rysunek 4.19. Aplikacja TableDemo

```
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka Table</h1>
      <table>
         <tr>
            <td>
               <strong>Zaznacz styl czcionki:</strong>
            </td>
            <td>
               <asp:CheckBoxList ID="cblFontStyle" runat="server"
                  AutoPostBack="True"
                  CellPadding="5"
                  CellSpacing="10"
                  RepeatColumns="3"
                  OnInit="cblFontStyle_Init">
               </asp:CheckBoxList>
            </td>
         </tr>
         <tr>
            <td>
               <strong>Zaznacz rozmiar czcionki:</strong>
            </td>
            <td>
               <asp:RadioButtonList ID="rblSize" runat="server"
                     AutoPostBack="True"
```

```
                  CellSpacing="20"
                  RepeatColumns="3"
                  RepeatDirection="Horizontal">
               <asp:ListItem text="10pt" value="10"/>
               <asp:ListItem text="12pt" value="12" selected = "true"/>
               <asp:ListItem text="14pt" value="14"/>
               <asp:ListItem text="16pt" value="16"/>
               <asp:ListItem text="18pt" value="18"/>
               <asp:ListItem text="24pt" value="24"/>
            </asp:RadioButtonList>
         </td>
      </tr>
   </table>

   <asp:Table ID="tbl" runat="server"
         BackImageUrl="Sunflower Bkgrd.jpg"
         Font-Names="Times New Roman"
         Font-Size="12"
         GridLines="Both"
         CellPadding="10"
         CellSpacing="5"
         HorizontalAlign="Left"
         Width="100%">
      <asp:TableHeaderRow HorizontalAlign="Left">
         <asp:TableHeaderCell>Rodzina czcionki</asp:TableHeaderCell>
         <asp:TableHeaderCell Width="80%">Przykładowy tekst</asp:TableHeaderCell>
      </asp:TableHeaderRow>
   </asp:Table>

   </div>
   </form>
</body>
</html>
```

Kontrolka `CheckBoxList` posiada obsługę zdarzenia zdefiniowaną dla inicjalizacji (`OnInit` — pogrubiony wiersz na listingu 4.32) wskazującą na metodę `cblFontStyle_Init`, która jest zawarta w pliku ukrytego kodu. Plik ten przedstawiono na listingu 4.33.

Listing 4.33. Plik default.aspx.cs dla aplikacji TableDemo

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;
using System.Drawing;              // Niezbędne dla FontFamily.
using System.Drawing.Text;         // Niezbędne dla Fonts.

public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      string str = "Pchnąć w tę łódź jeża lub ośm skrzyń fig."; // Pangram[1].
      int i = 0;
```

[1] Pangram — krótkie zdanie zawierające wszystkie litery danego języka (w jęz. ang. będzie to „The quick brown fox jumps over a lazy dog") — *przyp. tłum.*

```
// Pobieramy styl przycisków opcji.
bool boolUnder = false;
bool boolOver = false;
bool boolStrike = false;

foreach(ListItem li in cblFontStyle.Items)
{
   if (li.Selected == true)
   {
      switch (li.Value)
      {
         case "u":
            boolUnder = true;
            break;
         case "o":
            boolOver = true;
            break;
         case "s":
            boolStrike = true;
            break;
      }
   }
}

//   Pobieramy rozmiar czcionki.
int size = Convert.ToInt32(rblSize.SelectedItem.Value);

//   Pobieramy listę wszystkich czcionek zainstalowanych w systemie.
//   Wypełniamy tabelę czcionkami oraz przykładowym tekstem.
InstalledFontCollection ifc = new InstalledFontCollection();
foreach( FontFamily ff in ifc.Families )
{
   TableRow r = new TableRow();

   TableCell cFont = new TableCell();
   cFont.Controls.Add(new LiteralControl(ff.Name));
   r.Cells.Add(cFont);

   TableCell cText = new TableCell();
   Label lbl = new Label();
   lbl.Text = str;

   // ID nie jest tutaj niezbędny.
   // Po prostu pokazujemy, że może zostać ustawiony.
   i++;
   lbl.ID = "lbl" + i.ToString();

   // Ustawiamy nazwę czcionki.
   lbl.Font.Name = ff.Name;

   // Ustawiamy styl czcionki.
   if (boolUnder)
      lbl.Font.Underline = true;
   if (boolOver)
      lbl.Font.Overline = true;
   if (boolStrike)
      lbl.Font.Strikeout = true;

   // Ustawiamy rozmiar czcionki.
   lbl.Font.Size = size;

   cText.Controls.Add(lbl);
   r.Cells.Add(cText);
```

```
            tbl.Rows.Add(r);
        }
    }

    protected void  cblFontStyle_Init(object sender, EventArgs e)
    {
        // Tworzymy tablice elementów do dodania.
        string[] FontStyle = {"Podkreślenie","Nakreślenie", "Przekreślenie"};
        string[] Code = {"u","o","s"};

        for (int i = 0; i < FontStyle.GetLength(0); i++)
        {
            // Dodajemy zarówno właściwości Text, jak i Value.
            this.cblFontStyle.Items.Add(new ListItem(FontStyle[i],Code[i]));
        }
    }
}
```

Powyższy kod jest bardzo podobny do kodu przedstawionego na listingu 4.18, który wypełnia kontrolkę `CheckBoxList` danymi z tablicy. W tym kodzie tworzymy dwie tablice `FontStyle` i `Code`, a następnie wypełniamy nimi właściwości, odpowiednio, `Text` i `Value` elementów `ListItem`.

Z drugiej jednak strony, kontrolka `RadioButtonList` nie posiada obsługi zdarzenia `onInit`, a jej elementy `ListItem` zostały zdefiniowane wewnątrz kontrolki. Witryna wykorzystuje domknięte znaczniki `ListItem`, które zawierają atrybuty określające zarówno właściwość `Text`, jak i również właściwość `Value`. W przypadku dwunastopunktowego przycisku opcji, właściwość `Selected` jest ustalona jako `true`, co oznacza, że jest ona wartością domyślną w trakcie inicjalizacji.

Żadna z tych kontrolek nie posiada innej obsługi zdarzeń. Brak obsługi zdarzeń szczególnie dotyczy `OnSelectedIndexChanged`, ponieważ one występowały w poprzednich przykładach przedstawionych w tym rozdziale. Atrybut `AutoPostBack` ponownie otrzymuje wartość `true`. Jak możemy się przekonać, kontrolka ASP.NET `Table` jest przebudowywana przy każdym wczytaniu strony, co następuje wtedy, gdy zostanie zmieniona kontrolka `CheckBoxList` lub `RadioButtonList`. Aktualna wartość stylu czcionki jest pobierana z kontrolki `CheckBoxList`, natomiast bieżący rozmiar czcionki uzyskujemy z kontrolki `RadioButtonList`.

Warto zwrócić uwagę na dwa dodatkowe polecenia `using` (zostały pogrubione na listingu 4.33) w pliku ukrytego kodu, oprócz tych, które zostały tam umieszczone przez VS2005. Wymienione polecenia są niezbędne do włączenia użycia klas `Font` i `FontFamily` bez konieczności wpisywania pełnych nazw elementów.

Kontrolka `Table` jest najważniejszym elementem analizowanej strony:

```
<asp:Table ID="tbl" runat="server"
      BackImageUrl="Sunflower Bkgrd.jpg"
      Font-Names="Times New Roman"
      Font-Size="12"
      GridLines="Both"
      CellPadding="10"
      CellSpacing="5"
      HorizontalAlign="Left"
      Width="100%">
   <asp:TableHeaderRow HorizontalAlign="Left">
      <asp:TableHeaderCell>Rodzina czcionki</asp:TableHeaderCell>
      <asp:TableHeaderCell Width="80%">Przykładowy tekst</asp:TableHeaderCell>
   </asp:TableHeaderRow>
</asp:Table>
```

Podobnie jak wszystkie kontrolki serwerowe ASP.NET, także kontrolka `Table` dziedziczy z klasy `WebControl` i dlatego też posiada standardowy zestaw właściwości, metod i zdarzeń z tej klasy oraz klas znajdujących się wyżej w hierarchii. Oprócz tego, kontrolka `Table` posiada swoje własne właściwości, które zostały przedstawione w tabeli 4.12. Większość z wymienionych w tabeli właściwości zastosowano w aplikacji *TableDemo*.

Tabela 4.12. Właściwości kontrolki Table, które nie wywodzą się z innych klas

Nazwa	Typ	Pobierz	Ustaw	Wartości	Opis
`BackImageUrl`	String	x	x		Adres URL pliku graficznego wyświetlanego jako tło tabeli. Jeżeli grafika jest mniejsza od tabeli, wówczas zostanie ułożona sąsiadująco.
`Caption`	String	x	x		Tekst generowany do elementu nagłówkowego HTML. Użycie tej właściwości powoduje, że kontrolka staje się dostępniejsza dla użytkowników urządzeń technologii wspomagających, takich jak czytniki ekranu.
`CaptionAlign`	TableCaption-Align	x	x	`NotSet`, `Top`, `Bottom`, `Left`, `Right`	Określa formatowanie elementu nagłówka HTML.
`CellPadding`	Integer	x	x		Odległość podana w pikselach między krawędzią a zawartością komórki tabeli.
`CellSpacing`	Integer	x	x		Odległość podana w pikselach między sąsiadującymi komórkami.
`GridLines`	GridLines	x	x	`Both`, `Horizontal`, `None`, `Vertical`	Określa, czy i jakie linie siatki zostaną wyświetlone w tabeli. Wartością domyślną jest `None`.
`HorizontalAlign`	HorizontalAlign	x	x	`Center`, `Justify`, `Left`, `NotSet`, `Right`	Określa poziome wyrównanie tabeli na stronie. Wartością domyślną jest `NotSet`.

Warto zwrócić uwagę na następujące informacje o kontrolce `Table` w aplikacji *TableDemo*:

- Atrybut `BackImageUrl` kontrolki `Table` wskazuje na plik graficzny umieszczony w tym samym katalogu, w którym znajduje się sam plik *.aspx*. Dlatego też adres URL nie musi być pełny. W przedstawionych kodach korzystamy z pliku *SunflowerBkgrd.jpg*, który został skopiowany z katalogu *C:\Program Files\Common Files\Microsoft Shared\Stationery*. Oczywiście, można zastosować dowolny plik *.jpg* lub atrybut `BackImageUrl` może zostać po prostu pominięty. Dokładniejsza analiza adresowania względnego i absolutnego została zamieszczona w ramce „Położenia plików" w podrozdziale „Kontrolki Button".
- Składnia atrybutów nazwy i rozmiaru czcionki, gdy zostaną zadeklarowane za pomocą składni deklaratywnej jako część kontrolek serwerowych ASP.NET, to `Font-Name` i `Font-Size`. Natomiast kiedy są używane w pliku ukrytego kodu, wówczas ich składnia to `Font.Name` i `Font.Size`.
- Jeżeli atrybut `Width` zostanie ustawiony jako liczba całkowita, ale bez podania jednostek, spowoduje to, że szerokość tabeli zostanie określona liczbą pikseli niezależnie od szerokości okna przeglądarki. Tabela może więc być szersza niż okno przeglądarki internetowej.

- Jeśli atrybut `Width` nie zostanie podany, wówczas szerokość tabeli zostanie dobrana automatycznie, w taki sposób, aby wyświetlić zawartość jej komórek. Jeżeli szerokość okna przeglądarki internetowej nie będzie wystarczająca, to zawartość komórek będzie automatycznie przenoszona do nowego wiersza. W przypadku, kiedy szerokość okna przeglądarki będzie wystarczająca, aby komórki mogły zostać wyświetlone bez automatycznego przenoszenia do nowego wiersza, wtedy szerokość tabeli nie będzie zwiększana.

Wewnątrz kontrolki `Table` jest zagnieżdżona pojedyncza kontrolka `TableHeaderRow`. Ten wiersz zawiera komórki nagłówkowe, wskazane przez kontrolki `TableHeaderCell`.

Wiersze tabeli

Kontrolka `TableRow` jest używana do przedstawienia pojedynczego wiersza w kontrolce `Table`. Podobnie jak w przypadku kontrolki `Table`, także i `TableRow` wywodzi się z klasy `WebControl`. Jak możemy zobaczyć w tabeli 4.13, kontrolka `TableRow` posiada kilka właściwości, które nie są współużytkowane przez wszystkie pozostałe jej siostrzane kontrolki.

Tabela 4.13. Właściwości kontrolki TableRow, które nie są współużytkowane przez inne kontrolki serwerowe ASP.NET

Nazwa	Typ	Pobierz	Ustaw	Wartości	Opis
`HorizontalAlign`	HorizontalAlign	x	x	`Center`, `Justify`, `Left`, `NotSet`, `Right`	Określa poziome wyrównanie zawartości wszystkich komórek w wierszu. Wartością domyślną jest `NotSet`.
`VerticalAlign`	VerticalAlign	x	x	`Bottom`, `Middle`, `NotSet`, `Top`	Określa pionowe wyrównanie zawartości wszystkich komórek w wierszu. Wartością domyślną jest `NotSet`.
`Cells`	TableCellCollection	x			Zestaw obiektów `TableCell` składających się na wiersz.
`TableSection`	TableRowSection	x	x	`TableBody`, `TableFooter`, `TableHeader`	Określa miejsce umieszczenia wiersza w tabeli.

Klasa `TableRow` posiada sześć innych wywodzących się z niej kontrolek lub klas. Zostały one przedstawione w tabeli 4.14.

Tabela 4.14. Kontrolki wywodzące się z TableRow

Kontrolka pochodna	Opis
`DataGridItem`	Wiersz w kontrolce `DataGrid`.
`DetailsViewRow`	Wiersz wewnątrz kontrolki `DetailsView`.
`FormViewRow`	Wiersz wewnątrz kontrolki `FormView`.
`GridViewRow`	Wiersz wewnątrz kontrolki `GridView`.
`TableFooterRow`	Wiersz stopki w kontrolce `Table`.
`TableHeaderRow`	Wiersz nagłówka w kontrolce `Table`.

Wszystkie kontrolki wywodzące się z `TableRow`, za wyjątkiem `DataGridItem` są nowością w ASP.NET 2.0.

Komórki tabeli

Istnieją dwa typy kontrolek komórki tabeli: `TableCell` dla głównej części tabeli oraz `TableHeaderCell` dla komórek nagłówkowych. W aplikacji *TableDemo* zostały zastosowane obydwa typy kontrolek.

Kontrolka `TableHeaderCell` stanowi komórkę nagłówka w kontrolce `Table` i wywodzi się z klasy kontrolki `TableCell`. W rzeczywistości, wszystkie jej właściwości, zdarzenia i metody są dokładnie takie same jak w przypadku kontrolki `TableCell`. Jedyną różnicą między kontrolkami `TableCell` a `TableHeaderCell` jest fakt, że kontrolka `TableHeaderCell` jest generowana w przeglądarce jako element `<th>` zamiast `<td>`. Większość przeglądarek wyświetla komórki `<th>` za pomocą pogrubionej i wyśrodkowanej czcionki. Jak mogliśmy zobaczyć na rysunku 4.19, komórki tabeli zostały pogrubione, ale atrybut `HorizontalAlign` w deklaracji `TableRow` na listingu 4.33 zastąpił domyślne wyrównanie tekstu. Być może istotniejsze jest, że `TableHeaderCell` są na stronie wyraźnie innym typem kontrolek w zbiorze `Collection`.

Żadna z zagnieżdżonych kontrolek `TableHeaderCell` nie posiada ani atrybutu `id`, ani atrybutu `runat`. Wymienione atrybuty są niepotrzebne, ponieważ nigdzie w kodzie nie potrzebujemy programowego dostępu do tych kontrolek.

Tylko jeden wiersz został zdefiniowany statycznie. Pozostałe wiersze tabeli zostały zdefiniowane dynamicznie, przez metodę `Page_Load` zawartą w pliku ukrytego kodu, który przedstawiono na listingu 4.33.

Na listingu 4.32 zawartość komórek nagłówkowych stanowią dosłowne ciągi tekstowe umieszczone pomiędzy otwierającymi a zamykającymi znacznikami kontrolek. Alternatywną możliwością jest użycie domkniętych znaczników oraz określenie zawartości jako właściwości `Text`:

```
<asp:TableHeaderCell text="Rodzina czcionki" />
```

W kontrolce `TableCell` zostaje umieszczona rzeczywista zawartość tabeli. Podobnie jak w przypadku kontrolek `Table` i `TableRow`, wywodzi się ona z klasy `WebControl`. Kontrolki `TableCell` i `TableHeaderCell` posiadają właściwości przedstawione w tabeli 4.15. Właściwości te nie są współużytkowane z innymi siostrzanymi kontrolkami.

W aplikacji *TableDemo* uwzględniono kontrolkę `Table` zawierającą pojedynczy obiekt `TableRow`, który z kolei zawiera parę obiektów `TableHeaderCell`. Metoda `Page_Load` w pliku ukrytego kodu, która jest uruchamiana przy każdym wczytaniu strony, dynamicznie tworzy równoważące wiersze tabeli.

Metoda `Page_Load` dokonuje analizy właściwości `IsPostBack` i sprawdza, czy strona została wczytana po raz pierwszy. Jeżeli wczytanie strony jest wynikiem jej odświeżenia, wówczas wykonanie danego kodu może być niepożądane albo z powodu braku takiej konieczności i kosztu tej operacji, albo z powodu możliwości utracenia lub zmiany informacji o stanie (pełna analiza właściwości `IsPostBack` została przedstawiona w rozdziale 3.).

Jednakże w powyższym przykładzie kod powinien być wykonywany przy każdym wczytaniu strony. W rzeczywistości, kontrolki `CheckBoxList` i `RadioButtonList` posiadają właściwości `AutoPostBack` ustawione jako `true`, co wymusza odświeżenie strony oraz powtórne wygenerowanie tabeli. Za każdym razem, gdy tabela zostaje powtórnie wygenerowana, potrzebny styl czcionki jest odczytywany z kontrolki `CheckBoxList`, natomiast rozmiar czcionki jest pobierany z kontrolki `RadioButtonList`.

Tabela 4.15. Właściwości kontrolek TableCell i TableHeaderCell, które nie są współużytkowane z innymi kontrolkami tabel

Nazwa	Typ	Pobierz	Ustaw	Wartości	Opis
`Associated-HeaderCellID`	String	x	x		Rozdzielona przecinkami lista komórek nagłówka tabeli powiązanych z komórką używaną przez przeglądarki tekstowe do pomocy w nawigacji.
`ColumnSpan`	Integer	x	x		Liczba kolumn w tabeli, na którą dzieli się komórka.
`HorizontalAlign`	HorizontalAlign	x	x	`Center`, `Justify`, `Left`, `NotSet`, `Right`	Określa poziome wyrównanie zawartości komórki. Wartością domyślną jest `NotSet`.
`RowSpan`	Integer	x	x		Liczba wierszy w tabeli, na które dzieli się komórka.
`Text`	String	x	x		Zawartość tekstowa komórki.
`VerticalAlign`	VerticalAlign	x	x	`Bottom`, `Middle`, `NotSet`, `Top`	Określa pionowe wyrównanie zawartości komórki. Wartością domyślną jest `NotSet`.
`Wrap`	Boolean	x	x	`true`, `false`	W przypadku wartości `true` (wartość domyślna), zawartość komórek będzie automatycznie przenoszona do nowego wiersza. Jeżeli wartością będzie `false`, wówczas nie będzie automatycznego przenoszenia do nowego wiersza. Istnieje wzajemne oddziaływanie między właściwością `Wrap` a szerokością komórki.

Metoda `Page_Load` rozpoczyna się od inicjalizacji kilku zmiennych:

```
string str = "Pchnąć w tę łódź jeża lub ośm skrzyń fig.";
int i = 0;
```

Ciąg znakowy `str` jest tekstem wyświetlanym w tabeli, natomiast `i` jest licznikiem wykorzystanym w dalszej części kodu.

Styl lub style czcionki pobiera się z kontrolki `CheckBoxList`. Aby to zrobić, musimy zainicjalizować trzy zmienne typu Boolean używane jako opcje, po jednej dla każdego stylu:

```
boolUnder = true;
boolOver = true;
boolStrike = true;
```

Następnie, za pomocą pętli `foreach` sprawdzamy każdy z obiektów `ListItem` w kontrolce `cblFontStyle CheckBoxList`. Jeżeli pole wyboru zostało zaznaczone przez użytkownika, wówczas zmienna `Boolean` otrzymuje wartość `true` dla każdego stylu czcionki. W tym celu należy sprawdzić, czy właściwość `Selected` obiektu `ListItem` posiada wartość `true`:

```
foreach(ListItem li in cblFontStyle.Items)
{
   if (li.Selected == true)
   {
      switch (li.Value)
      {
         case "u":
            boolUnder = true;
            break;
```

```
            case "o":
               boolOver = true;
               break;
            case "s":
               boolStrike = true;
               break;
         }
      }
   }
```

Uzyskanie rozmiaru czcionki zaznaczonego w kontrolce `RadioButtonList rblSize` jest znacznie łatwiejsze, ponieważ musimy jedynie pobrać właściwość `Value` obiektu `ListItem`, który został zwrócony przez właściwość `SelectedItem`. Otrzymaną liczbę całkowitą umieszczamy następnie w zmiennej `size`:

```
int size = Convert.ToInt32(rblSize.SelectedItem.Value);
```

W tym miejscu dochodzimy do sedna metody. Musimy uzyskać listę wszystkich czcionek zainstalowanych w systemie. Aby wykonać to zadanie, tworzymy nowy egzemplarz obiektu `InstalledFontCollection()`:

```
InstalledFontCollection ifc = new InstalledFontCollection();
```

Następnie, używając pętli `foreach`, kolejno przeglądamy cały zbiór, wyszukując w każdym kroku obiektu `FontFamily`:

```
foreach( FontFamily ff in ifc.Families )
```

Dla każdej rodziny czcionek ze zbioru `FontFamily` tworzymy nowy obiekt `TableRow`:

```
TableRow r = new TableRow();
```

Następnie wewnątrz danego obiektu `TableRow` tworzymy dwa obiekty `TableCell`. Pierwszy obiekt o nazwie `cFont` przechowuje nazwę czcionki, natomiast drugi obiekt o nazwie `cText` zawiera zdefiniowany wcześniej przykładowy ciąg tekstowy. Poniższy kod implementuje komórkę `cFont`:

```
TableCell cFont = new TableCell();
cFont.Controls.Add(new LiteralControl(ff.Name));
r.Cells.Add(cFont);
```

Obiekt `cFont TableCell` wykorzystuje kontrolkę serwerową ASP.NET o nazwie `LiteralControl`. Kontrolka ta jest wykorzystywana do umieszczenia na stronie tekstu oraz elementów HTML. Inaczej niż w przypadku kontrolek dziedziczących z klasy `Control`, jedyną właściwością kontrolki `LiteralControl` jest właściwość `Text`.

Dla komórki zawierającej przykładowy tekst użyjemy odrobinę innej techniki, ponieważ chcemy zachować możliwość operowania właściwościami tekstu i rozmiaru ciągu tekstowego. Po utworzeniu nowego egzemplarza obiektu `TableCell` o nazwie `cText` tworzymy nowy egzemplarz kontrolki `Label`, a jej właściwości `Text` przypisujemy zdefiniowaną wcześniej zmienną `str`:

```
TableCell cText = new TableCell();
Label lbl = new Label( );
lbl.Text = str;
```

Zwiększamy o jednostkę zdefiniowany wcześniej licznik i używamy go do przypisania kontrolce `Label` właściwości `ID`:

```
i++;
lbl.ID = "lbl" + i.ToString();
```

W rzeczywistości ten krok jest niepotrzebny, ponieważ w żadnym miejscu kodu nie będziemy odnosić się z powrotem do określonej komórki. Zdecydowaliśmy się jednak na jego umieszczenie, aby pokazać, w jaki sposób można zrealizować to zadanie.

Następnie przypisujemy nazwę czcionki:

```
lbl.Font.Name = ff.Name;
```

Użyta tutaj składnia różni się od składni ustalającej nazwę czcionki wewnątrz znaczników kontrolki serwerowej ASP.NET (`Font.Name` kontra `Font-Name`).

Używamy ustalonych wcześniej opcji do ustawienia stylów czcionki:

```
if (boolUnder)
   lbl.Font.Underline = true;
if (boolOver)
   lbl.Font.Overline = true;
if (boolStrike)
   lbl.Font.Strikeout = true;
```

Nasza tabela jest tworzona od podstaw przy każdym wczytaniu strony, a wartością domyślną dla wszystkich wymienionych stylów jest brak stylu (na przykład `false`), nie zachodzi więc potrzeba wyraźnego ustawienia tym właściwościom wartości `false`.

Ustalamy rozmiar czcionki, dodajemy obiekt `Label` do obiektu `TableCell`, obiekt `TableCell` dodajemy do obiektu `TableRow`, który z kolei dołączamy do obiektu `Table`:

```
lbl.Font.Size = size;
cText.Controls.Add(lbl);
r.Cells.Add(cText);
tbl.Rows.Add(r);
```

Zadanie zostaje ukończone.

Szerokość komórki

Zagadnienie regulowania szerokości komórek zasługuje na szczególną wzmiankę. Jest to zadanie podobne do regulowania szerokości tabeli, ale równocześnie wystarczająco odmienne, aby wprowadzić pewne zamieszanie. Przyglądając się kodowi HTML na listingu 4.32, możemy zauważyć, że druga komórka w wierszu nagłówka posiada atrybut `Width` o wartości 80%:

```
<asp:TableHeaderCell Width="80%">
   Dowolny tekst
</asp:TableHeaderCell>
```

W przeglądarkach internetowych wszystkie komórki w tej kolumnie będą posiadały tę samą szerokość. Jeżeli żadna z tych komórek nie będzie miała określonej szerokości, kolumna zostanie dopasowana w taki sposób, aby najlepiej przystosować się do wielkości komórek, biorąc pod uwagę wymagania dotyczące szerokości tabeli oraz wielkości okna przeglądarki.

Jeżeli zdefiniowano szerokość dla wielu komórek w kolumnie, wówczas zostanie zastosowana wartość określająca najszerszą komórkę. W celu zwiększenia czytelności, parametr określający szerokość powinien zostać umieszczony w jednym wierszu, najczęściej w pierwszym wierszu tabeli. Z tego powodu atrybut `Width` pojawia się w wierszu nagłówka omawianego przykładu.

Kiedy szerokość komórki jest określona deklaracyjnie jako część znacznika kontrolki serwerowej ASP.NET, może zostać podana albo jako wartość procentowa całej tabeli, jak też uczyniliśmy w powyższym przykładzie, albo jako stała wartość wyrażona w pikselach, na przykład:

```
Width="400";
```

Szerokość komórki może zostać również ustawiona programowo. W takim przypadku składnia jest odrobinę inna. W języku C# kod będzie następujący:

```
TableCell cText = new TableCell();
```

Zmienna `cText` typu `TableCell` zostaje przypisana do nowego egzemplarza. Następnie w utworzonym egzemplarzu `TableCell` można zastosować właściwość `Width` albo wyrażoną w pikselach, albo procentach szerokości tabeli. W celu określenia właściwości `Width` jako 80% szerokości tabeli, używamy następującego wiersza kodu:

```
cText.Width = Unit.Percentage(80);
```

Aby określić szerokość stałą liczbą pikseli, stosuje się *jeden* z poniższych wierszy kodu:

```
cText.Width = Unit.Pixel(400);
cText.Width = 400;
```

Pomiędzy właściwościami `Width` a `Wrap` komórki występuje wzajemne oddziaływanie. Wartość domyślna właściwości `Wrap` to `true`. Jeżeli właściwość `Wrap` otrzyma wartość `false`, wówczas wystąpi jedna z poniższych sytuacji:

- Jeśli nie została określona właściwość `Width`, wówczas zawartość komórek nie będzie przenoszona do nowego wiersza, a kolumna będzie rozszerzać się tak, aby pomieścić najszerszą komórkę.
- Jeśli właściwość `Width` będzie posiadała wartość podaną w pikselach, wtedy właściwość `Wrap` zostanie zastąpiona, a zawartość komórki będzie przenoszona do nowego wiersza, respektując tym samym wartość właściwości `Width`.
- Jeśli właściwość `Width` będzie posiadała wartość wyrażoną w procentach, wówczas zostanie ona zignorowana, a kolumna będzie wystarczająco szeroka, aby wykluczyć jakiekolwiek przenoszenie do nowego wiersza.

Kontrolka Panel

Kontrolka `Panel` jest używana w charakterze pojemnika na inne kontrolki. Posiada ona kilka dostępnych funkcji:

- steruje widzialnością zawartych kontrolek;
- steruje wyglądem zawartych kontrolek;
- ułatwia programowe generowanie kontrolek.

Kontrolka `Panel` wywodzi się z klasy `WebControl` i dodatkowo posiada właściwości przedstawione w tabeli 4.16. Kontrolka `Panel` nie posiada metod lub zdarzeń, które nie są dziedziczone z klas `Control` lub `WebControl`. W szczególności, nie posiada zdarzeń wyzwalanych przez działanie użytkownika.

Tabela 4.16. Właściwości kontrolki Panel, które nie są dziedziczone z klasy Control lub WebControl

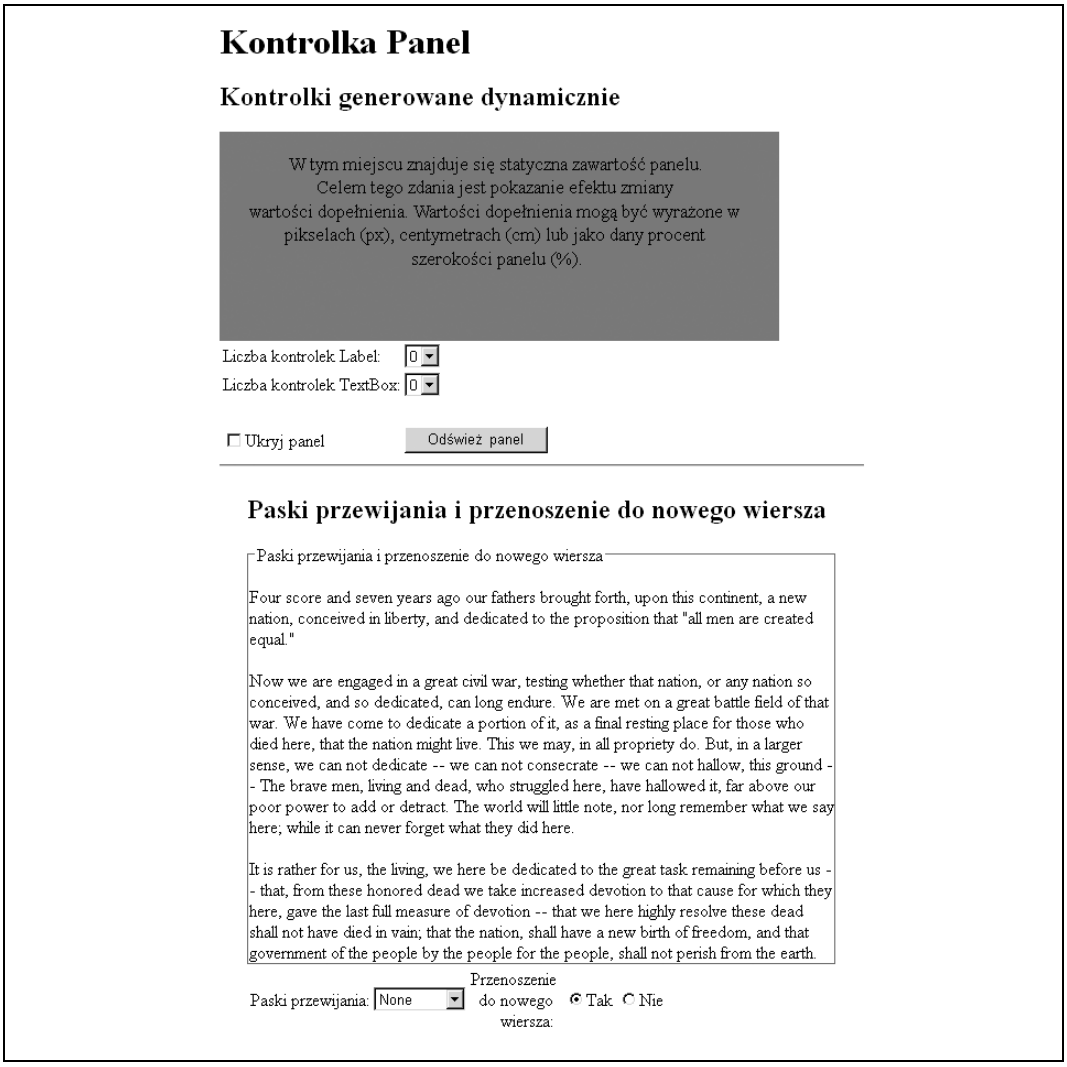
Nazwa	Typ	Pobierz	Ustaw	Wartości	Opis
`BackImageUrl`	String	x	x		Adres URL pliku graficznego wyświetlanego jako tło panelu. Jeżeli obrazek jest mniejszy od panelu, wówczas zostanie ułożony sąsiadująco.
`Direction`	ContentDirection	x	x	`LeftToRight`, `RightToLeft`, `NotSet`	Kierunek wyświetlania tekstu w kontrolce pojemnika. Wartością domyślną jest `NotSet`.
`GroupingText`	String	x	x		Powoduje, że kontrolka `Panel` jest generowana w przeglądarce jako element `<fieldset>` zamiast `<div>`. Wartość tej właściwości jest używana z elementem `<legend>`.
`HorizontalAlign`	HorizontalAlign	x	x	`Center`, `Justify`, `Left`, `NotSet`, `Right`	Określa poziome wyrównanie zawartości. Wartością domyślną jest `NotSet`. Warto zwrócić uwagę, że nie występuje właściwość `VerticalAlign`.
`ScrollBars`	ScrollBars	x	x	`Auto`, `Both`, `Horizontal`, `None`, `Vertical`	Określa widzialność i położenie pasków przewijania. Wartością domyślną jest `None`.
`Wrap`	Boolean	x	x	`true`, `false`	W przypadku wartości `true` (wartość domyślna), zawartość panelu będzie automatycznie przenoszona do nowego wiersza. Jeżeli wartością będzie `false`, wówczas nie będzie automatycznego przenoszenia do nowego wiersza.

Kolejna przykładowa witryna *PanelDemo*, pokazana na rysunku 4.20, zawiera dwie kontrolki `Panel`. Pierwsza z nich prezentuje, w jaki sposób można sterować wyglądem i widocznością kontrolek potomnych oraz jak programowo dodawać kontrolki. Druga kontrolka `Panel` pokazuje użycie właściwości `GroupingText`, `ScrollBars` i `Wrap` do definiowania wyglądu kontrolki.

Dwie deklaracje kontrolek `Panel` z bieżącego przykładu zostały przedstawione na listingu 4.34 pogrubioną czcionką. Pierwsza z nich, o identyfikatorze `pnlDynamic`, posiada między otwierającym a zamykającym znacznikiem kontrolki `Panel` pewną statyczną zawartość. Pozostała zawartość tego panelu jest dodawana dynamicznie, w zależności od zaznaczonych wartości w dwóch rozwijanych listach `ddlLabels` i `ddlBoxes`.

Listing 4.34. Plik default.aspx aplikacji PanelDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka Panel</title>
</head>
```



Rysunek 4.20. Aplikacja PanelDemo

```
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka Panel</h1>
      <h2>Kontrolki generowane dynamicznie</h2>
      <asp:Panel ID="pnlDynamic" runat="server"
         Height="150"
         Width="80%"
         BackColor="Beige"
         Font-Names="Courier New"
         HorizontalAlign="Center"
         Style="padding:20px"
         ScrollBars="Auto">
         W tym miejscu znajduje się statyczna zawartość panelu.
         <br />Celem tego zdania jest pokazanie efektu zmiany
        wartości dopełnienia. Wartości dopełnienia mogą być wyrażone w pikselach (px),
```

```
   centymetrach (cm) lub jako dany procent szerokości panelu (%).
   <p />
</asp:Panel>

<table>
   <tr>
      <td>
         Liczba kontrolek Label:
      </td>
      <td>
         <asp:DropDownList id=ddlLabels runat="server">
            <asp:ListItem text="0" value="0" />
            <asp:ListItem text="1" value="1" />
            <asp:ListItem text="2" value="2" />
            <asp:ListItem text="3" value="3" />
            <asp:ListItem text="4" value="4" />
         </asp:DropDownList>
      </td>
   </tr>
   <tr>
      <td>
         Liczba kontrolek TextBox:
      </td>
      <td>
         <asp:DropDownList id=ddlBoxes runat="server">
            <asp:ListItem text="0" value="0" />
            <asp:ListItem text="1" value="1" />
            <asp:ListItem text="2" value="2" />
            <asp:ListItem text="3" value="3" />
            <asp:ListItem text="4" value="4" />
         </asp:DropDownList>
      </td>
   </tr>
   <tr>
      <td colspan=2>
          
      </td>
   </tr>
   <tr>
      <td>
         <asp:CheckBox id="chkHide" runat="server"
            text="Ukryj panel" />
      </td>
      <td>
         <asp:Button ID="Button1" runat="server"
            text="Odśwież panel" />
      </td>
   </tr>
</table>

<hr/>
<h2>Paski przewijania i przenoszenie do nowego wiersza</h2>

<asp:Panel ID="pnlScroll" runat="server"
      Height="200"px;
      Width="90%"
      GroupingText="Paski przewijania i przenoszenie do nowego wiersza">
   <asp:Label ID="lblPanelContent" runat="server"></asp:Label>
</asp:Panel>
   <br />
<table >
   <tr>
      <td align="right">
         Paski przewijania:
      </td>
```

```
            <td>
               <asp:DropDownList id=ddlScrollBars runat="server"
                     AutoPostback="true"
                     OnSelectedIndexChanged="ddlScrollBars_SelectedIndexChanged">
                  <asp:ListItem text="Brak"  Selected="True" />
                  <asp:ListItem text="Opcja Auto" />
                  <asp:ListItem text="Opcja Both" />
                  <asp:ListItem text="Opcja Horizontal" />
                  <asp:ListItem text="Opcja Vertical" />
               </asp:DropDownList>
            </td>
            <td align="right" width="75">
               Przenoszenie do nowego wiersza:
            </td>
            <td>
               <asp:RadioButtonList ID="rblWrap" runat="server"
                     AutoPostBack="true"
                     RepeatDirection="Horizontal"
                     OnSelectedIndexChanged="rblWrap_SelectedIndexChanged">
                  <asp:ListItem Text="Tak" Value="true"  Selected="True" />
                  <asp:ListItem Text="Nie" Value="false" />
               </asp:RadioButtonList>
            </td>
         </tr>
      </table>
   </div>
   </form>
</body>
</html>
```

W pierwszym panelu zdefiniowano kilka atrybutów, włączając w to `BackColor`, `Height` (w pikselach), `Width` (wyrażona w procentach okna przeglądarki), nazwa czcionki (`Font-Name`) oraz wyrównanie poziome (`HorizontalAlign`). Warto zwrócić uwagę, że ta kontrolka nie posiada właściwości określającej wyrównanie w pionie.

Atrybut `Style` ustawia dopełnienie wzdłuż każdej z czterech stron na 20 pikseli. Alternatywną możliwością jest określenie dopełnienia dla każdej ze stron. W takim przypadku, zgodnie z danymi przedstawionymi w tabeli 4.17, atrybutowi `Style` podajemy wiele wartości. Tak więc, przykładowo, poniższy atrybut ustawi górną, prawą, dolną i lewą wartość dopełnienia jako, odpowiednio, 20, 40, 60 i 20 pikseli:

```
Style="padding: 20px 40px 60px 20px"
```

Skróty dopełnienia, które zostały przedstawione w tabeli 4.17, działają równoznacznie z atrybutami `Style` ustawiającymi również wartości obramowania i marginesów.

Tabela 4.17. Efekt wielu wartości dopełnienia

Liczba podanych wartości	Efekt
1	Wszystkie cztery strony.
2	Pierwsza wartość ustala dla góry i dołu, druga wartość ustala dla strony lewej i prawej.
3	Pierwsza wartość ustala górę, druga ustala stronę lewą i prawą, trzecia wartość ustala dół.
4	Pierwsza wartość ustala górę, druga ustala prawą stronę, trzecia ustala dół, a czwarta wartość ustala lewą stronę.

Atrybut `ScrollBars` posiada ustaloną wartość `Auto`, co powoduje, że poziomy, pionowy lub jeden i drugi pasek przewijania zostaną wyświetlone jedynie w przypadku takiej konieczności. Właściwość `Wrap` domyślnie jest ustawiona jako `true`, zatem wykorzystując dostępne miejsce,

statyczny tekst będzie przenoszony do nowego wiersza. Z tego powodu pierwszy panel nie będzie nigdy wymagał poziomego paska przewijania, jednak jeżeli do panelu dodamy dostatecznie dużą liczbę etykiet i/lub pól tekstowych, wówczas — w razie takiej konieczności — zostanie wyświetlony poziomy pasek przewijania.

Wartość atrybutu `Height` jest liczbą całkowitą określającą liczbę pikseli. Część `px` podana po wartości jest opcjonalna. Na przykład przedstawione poniżej dwa wiersze są równoważne:

```
Height="250px"
Height="250"
```

Inną możliwością jest wyrażenie wartości atrybutu `Height` w procentach, choć wówczas kontrolka `Panel` musi zostać umieszczona wewnątrz pojemnika o stałej wielkości, na przykład w innej kontrolce `Panel`. Jeżeli kontrolka nie zostanie zawarta wewnątrz pojemnika o stałej wielkości i nie będzie posiadała zawartości, wtedy niezależnie od podanej wartości procentowej panel będzie jedynie pojedynczym wierszem.

W przypadku, gdy atrybut `Height` zostanie pominięty, kontrolka `Panel` automatycznie określi swój pionowy rozmiar, tak aby pomieścić wszystkie zawarte w sobie kontrolki.

Atrybut `Width` może być albo liczbą całkowitą określającą piksele, albo procentową wartością okna przeglądarki. W powyższym przykładzie zastosowaliśmy tę drugą opcję. Jeżeli atrybut `Width` zostanie pominięty, wtedy kontrolka `Panel` domyślnie ustawi szerokość na 100%.

W aplikacji *PanelDemo* zostały zdefiniowane dwie statyczne tabele HTML, co miało na celu utworzenie układu kontrolek, które będą sterowały dwoma panelami. Pierwsza tabela, powiązana z pierwszą kontrolką `Panel`, zawiera dwie kontrolki `DropDownList`, kontrolkę `CheckBox` oraz kontrolkę `Button`.

Żadna z kontrolek umieszczonych w tabeli powiązanej z pierwszym panelem nie posiada ustawionej właściwości `AutoPostBack`. Dlatego też, aby zastosować wprowadzone zmiany, użytkownik musi kliknąć przycisk odświeżający formularz. Podczas odświeżania formularza uruchamia się metoda `Page_Load`, której kod przedstawiono na listingu 4.35.

Listing 4.35. Plik default.aspx.cs witryny PanelDemo

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      // W pierwszej kolejności zajmujemy się panelem z dynamicznie
      // generowanymi kontrolkami.
      // Zawartość panelu (Pokaż/Ukryj).
      if (chkHide.Checked)
      {
         pnlDynamic.Visible = false;
      }
      else
```

```
        {
            pnlDynamic.Visible = true;
        }

        // Generowanie kontrolek Label.
        int numlabels = Int32.Parse(ddlLabels.SelectedItem.Value);
        for (int i = 1; i <= numlabels; i++)
        {
            Label lbl = new Label();
            lbl.Text = "Label" + (i).ToString();
            lbl.ID = "Label" + (i).ToString();
            pnlDynamic.Controls.Add(lbl);
            pnlDynamic.Controls.Add(new LiteralControl("<br>"));
        }

        // Generowanie kontrolek TexBox.
        int numBoxes = Int32.Parse(ddlBoxes.SelectedItem.Value);
        for (int i = 1; i <= numBoxes; i++)
        {
            TextBox txt = new TextBox();
            txt.Text = "TextBox" + (i).ToString();
            txt.ID = "TextBox" + (i).ToString();
            pnlDynamic.Controls.Add(txt);
            pnlDynamic.Controls.Add(new LiteralControl("<br>"));
        }

        // Następnie zajmujemy się panelem Scrollbar.
        string strText = "<p>Four score and seven years ago our fathers brought forth,
        upon this continent, a new nation, conceived in liberty, and dedicated to the
        proposition that \"all men are created equal.\"</p>";
        strText += "<p>Now we are engaged in a great civil war, testing whether that
        nation, or any nation so conceived, and so dedicated, can long endure. We are
        met on a great battle field of that war. We have come to dedicate a portion of it,
        as a final resting place for those who died here, that the nation might live.
        This we may, in all propriety do. But, in a larger sense, we can not dedicate --
        we can not consecrate -- we can not hallow, this ground -- The brave men, living
        and dead, who struggled here, have hallowed it, far above our poor power to add
        or detract. The world will little note, nor long remember what we say here;
        while it can never forget what they did here.</p>";
        strText += "<p>It is rather for us, the living, we here be dedicated to the
        great task remaining before us -- that, from these honored dead we take
        increased devotion to that cause for which they here, gave the last full measure
        of devotion -- that we here highly resolve these dead shall not have died in
        vain; that the nation, shall have a new birth of freedom, and that government of
        the people by the people for the people, shall not perish from the earth.</p>";

        lblPanelContent.Text = strText;
    }

    protected void ddlScrollBars_SelectedIndexChanged(object sender, EventArgs e)
    {
        DropDownList ddl= (DropDownList)sender;
        string strValue = ddl.SelectedItem.ToString();

        ScrollBars scrollBar =
            (ScrollBars)Enum.Parse(typeof(ScrollBars), strValue);
        pnlScroll.ScrollBars = scrollBar;
    }

    protected void rblWrap_SelectedIndexChanged(object sender, EventArgs e)
    {
        RadioButtonList rbl = (RadioButtonList)sender;
        pnlScroll.Wrap = Convert.ToBoolean(rbl.SelectedValue);
    }
}
```

Pierwsza część kodu metody `Page_Load` zajmuje się pierwszą kontrolką `Panel`, natomiast druga — dotyczy drugiej kontrolki `Panel`. Wciąż skupiając się na pierwszym panelu, możemy zauważyć, że blok `if-else` włącza lub wyłącza wyświetlanie tego panelu. Kiedy panel nie będzie widoczny, wówczas jego zawartość również nie będzie widoczna. Podobnie, gdy panel stanie się widoczny, to cała jego zawartość też będzie widoczna.

W kodzie występują dwie pętle `for`, po jednej dla etykiet oraz pól tekstowych, które generują zawarte w panelu kontrolki. Po przeprowadzeniu konwersji danych wejściowych odpowiedniej kontrolki `DropDownList` na liczbę całkowitą, pętla `for` wykonuje procedurę określoną liczbę razy.

W każdym z dwóch przypadków procedura jest podobna. Po ustanowieniu nowego egzemplarza kontrolki zostają jej przypisane właściwości `Text` i `ID`. Następnie, kontrolka zostaje dodana do zbioru `Controls` panelu. Na koniec, do zbioru będzie również dodana kontrolka `LiteralControl` zawierająca pewien kod HTML.

Nazwa czcionki, która zostaje podana wewnątrz znaczników kontrolki `Panel`, wpływa na statyczny tekst oraz etykiety w panelu, ale nie na zawartość pól tekstowych.

Drugi panel `pnlScroll` posiada zadeklarowane jedynie trzy atrybuty (poza atrybutami `ID` oraz `runat`): `Height`, `Width` i `GroupingText`. Pierwsze dwa atrybuty zostały już wcześniej opisane, natomiast atrybut `GroupingText` powoduje efekt umieszczenia krawędzi wokół panelu oraz wyświetlenie wartości ciągu znakowego właściwości `GroupingText` jako tytułu wewnątrz krawędzi.

W tym panelu jedyną zawartością jest kontrolka `Label` o identyfikatorze `lblPanelContent`. Właściwość `Text` kontrolki `lblPanelContent` jest ustalona w metodzie `Page_Load` zamiast w postaci długiego ciągu tekstowego umieszczonego w zmiennej `strText`. Wspomniany ciąg tekstowy zawiera pewne elementy akapitu HTML wymuszające znaki nowego wiersza.

Dwie kontrolki powiązane z tym panelem to rozwijana lista, która ustawia wartość właściwości `Scrollbars` oraz lista przycisków opcji służąca do ustawienia właściwości `Wrap`. W obu kontrolkach właściwość `AutoPostBack` została ustalona jako `true`, tak więc nie są wymagane dodatkowe czynności, aby wprowadzone zmiany zostały zastosowane.

W obsłudze zdarzeń dla zdarzenia `SelectedIndexChanged` rozwijanej listy `ddlScrollBars_SelectedIndexChanged`, której kod zapisano pogrubioną czcionką na listingu 4.35, właściwość `Scrollbars` jest ustawiona. Technika ustawienia tej wartości z typu wyliczeniowego `ScrollBars` jest dokładnie taka sama, jaka została wcześniej opisana w przykładzie *BulletedListDemo* na listingu 4.31.

W metodzie obsługi zdarzeń dla zdarzenia `SelectedIndexChanged` listy przycisków opcji, która również została pogrubiona na listingu 4.35, odniesienie do listy przycisków opcji jest uzyskiwane przez rzutowanie obiektu `sender` do zmiennej typu `RadioButtonList`. Następnie, dzięki konwersji `SelectedValue` kontrolki na typ Boolean zostaje odpowiednio ustawiona właściwość `Wrap` panelu `pnlScroll`.

Jeżeli ciąg tekstowy w zmiennej `strText` nie zawiera w sobie żadnych znaczników HTML, wówczas przy właściwości `Wrap` ustawionej jako `false` zostanie wyświetlony jako jeden, długi wiersz. Ponieważ każdy „wiersz" jest umieszczony w znacznikach akapitu, to mimo ustawienia właściwości `Wrap` jako `false`, ciąg znakowy zostanie wyświetlony jako trzy oddzielne wiersze.

Elementy graficzne

Elementy graficzne stanowią ważny aspekt większości witryn internetowych. ASP.NET dostarcza kilku kontrolek serwerowych ASP.NET służących do wyświetlania grafiki. Dwie z nich, `Image` oraz `ImageMap` zostaną omówione w bieżącym podrozdziale. Kontrolka `AdRotator` zostanie przedstawiona w kolejnym rozdziale.

Kontrolka Image

Kontrolka `Image` posiada ograniczony zestaw dostępnych funkcji: jest używana do wyświetlenia pliku graficznego na stronie internetowej lub jeśli ten plik jest niedostępny, to wyświetla podany tekst. Kontrolka nie zapewnia obsługi zdarzeń innych niż te dziedziczone z klasy `Control`, na przykład `Init` i `Load`. Jeżeli dany element graficzny ma funkcjonować jako przycisk (na przykład w celu przechwycenia kliknięć myszą), wówczas lepszym rozwiązaniem jest zastosowanie kontrolki `ImageButton`, która została opisana we wcześniejszej części tego rozdziału.

Oprócz właściwości dziedziczonych z klasy `WebControl`, kontrolka `Image` posiada również właściwości przedstawione w tabeli 4.18.

Tabela 4.18. Właściwości kontrolki Image

Nazwa	Typ	Pobierz	Ustaw	Wartości	Opis
`AlternateText`	String	x	x		Tekst, który zostaje wyświetlony w kontrolce, jeśli plik graficzny jest niedostępny. W przeglądarkach obsługujących funkcję `ToolTip` zostanie on również wyświetlony jako podpowiedź.
`ImageAlign`	ImageAlign	x	x	Zobacz tabela 4.19	Opcje wyrównania względem tekstu na stronie internetowej. Zobacz również tabelę 4.19.
`ImageUrl`	String	x	x		Ares URL wskazujący na położenie pliku graficznego przeznaczonego do wyświetlenia.

Właściwość `ImageUrl` może być albo *względna*, albo *absolutna*. Obie możliwości zostały szczegółowo opisane w ramce „Położenia plików" w podrozdziale „Kontrolki Button".

Istnieje dziesięć możliwych wartości do zastosowania we właściwości `ImageAlign`, wszystkie zostały przedstawione w tabeli 4.19. Jeżeli zachodzi potrzeba zapewnienia precyzyjniejszej kontroli nad elementem graficznym i położeniem tekstu, wówczas prawdopodobnie lepszym rozwiązaniem jest umieszczenie kontrolki `Image` w tabeli.

Na listingu 4.36 oraz na pokazanej na rysunku 4.21 aplikacji *ImageDemo* zobaczymy, w jaki sposób różne wartości `ImageAlign` wpływają na wygląd strony internetowej. Plik ukrytego kodu tej aplikacji został przedstawiony na listingu 4.37.

Tabela 4.19. Opcje właściwości ImageAlign

Wartości	Opis
`NotSet`	Brak ustawień. Jest to wartość domyślna.
`AbsBottom`	Wyrównuje dolną krawędź grafiki z dolną krawędzią największego elementu w tym samym wierszu.
`AbsMiddle`	Wyrównuje środek grafiki ze środkiem największego elementu w tym samym wierszu.
`Top`	Wyrównuje górną krawędź grafiki z górną krawędzią najwyższego elementu w tym samym wierszu.
`Bottom`	Wyrównuje dolną krawędź grafiki z dolną krawędzią pierwszego wiersza tekstu. Opcja taka sama jak `Baseline`.
`Baseline`	Wyrównuje dolną krawędź grafiki z dolną krawędzią pierwszego wiersza tekstu. Opcja taka sama jak `Bottom`.
`Middle`	Wyrównuje środek grafiki z dolną krawędzią pierwszego wiersza tekstu.
`TextTop`	Wyrównuje górną krawędź grafiki z górną krawędzią najwyższego tekstu w tym samym wierszu.
`Left`	Wyrównuje grafikę do lewej krawędzi strony, a otaczający ją tekst znajduje się po prawej stronie.
`Right`	Wyrównuje grafikę do prawej krawędzi strony, a otaczający ją tekst znajduje się po lewej stronie.

Listing 4.36. Plik default.aspx witryny ImageDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>Kontrolka Image</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka Image</h1>

      <font name="Garamond" size ="4">
      Oto przykładowy akapit, który jest używany
      do przedstawienia efektów zastosowania różnych wartości
      właściwości ImageAlign. Jak można zobaczyć, czasami trudno
      jest jasno zdefiniować efekty, a dużo zależy również
      od szerokości okna przeglądarki internetowej
      </font>

      <asp:Image ID="img1" runat="server"
         AlternateText="Dan"
         ImageUrl="Dan at vernal pool.jpg" />
      <hr />
      <asp:Button runat="server" Text="Przykładowy przycisk" />
      <asp:Image ID="img2" runat="server"
         AlternateText="Dan" ImageUrl="Dan at Vernal pool.jpg" />
      <hr />
      <asp:DropDownList ID="ddl" runat="server" AutoPostBack="True">
         <asp:ListItem text="NotSet" />
         <asp:ListItem text="AbsBottom" />
         <asp:ListItem text="AbsMiddle" />
         <asp:ListItem text="Top" />
         <asp:ListItem text="Bottom" />
         <asp:ListItem text="BaseLine" />
         <asp:ListItem text="TextTop" />
         <asp:ListItem text="Left" />
         <asp:ListItem text="Right" />
```

Rysunek 4.21. Aplikacja ImageDemo

```
        </asp:DropDownList>
    </div>
    </form>
</body>
</html>
```

Listing 4.37. Plik default.aspx.cs witryny ImageDemo

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
      switch (ddl.SelectedIndex)
      {
         case 0:
            img1.ImageAlign = ImageAlign.NotSet;
            img2.ImageAlign = ImageAlign.NotSet;
            break;
```

```
            case 1:
               img1.ImageAlign = ImageAlign.AbsBottom;
               img2.ImageAlign = ImageAlign.AbsBottom;
               break;
            case 2:
               img1.ImageAlign = ImageAlign.AbsMiddle;
               img2.ImageAlign = ImageAlign.AbsMiddle;
               break;
            case 3:
               img1.ImageAlign = ImageAlign.Top;
               img2.ImageAlign = ImageAlign.Top;
               break;
            case 4:
               img1.ImageAlign = ImageAlign.Bottom;
               img2.ImageAlign = ImageAlign.Bottom;
               break;
            case 5:
               img1.ImageAlign = ImageAlign.Baseline;
               img2.ImageAlign = ImageAlign.Baseline;
               break;
            case 6:
               img1.ImageAlign = ImageAlign.Middle;
               img2.ImageAlign = ImageAlign.Middle;
               break;
            case 7:
               img1.ImageAlign = ImageAlign.TextTop;
               img2.ImageAlign = ImageAlign.TextTop;
               break;
            case 8:
               img1.ImageAlign = ImageAlign.Left;
               img2.ImageAlign = ImageAlign.Left;
               break;
            case 9:
               img1.ImageAlign = ImageAlign.Right;
               img2.ImageAlign = ImageAlign.Right;
               break;
            default:
               img1.ImageAlign = ImageAlign.NotSet;
               img2.ImageAlign = ImageAlign.NotSet;
               break;
         }
      }
   }
```

Aby zapewnić prawidłowe działanie kodu przedstawionego w witrynie *ImageDemo*, trzeba sobie zapewnić plik z obrazkiem dla właściwości `ImageUrl`. W naszym przykładzie wykorzystaliśmy plik *Dan at vernal pool.jpg*, który umieszczono w katalogu witryny internetowej, aczkolwiek można zastosować dowolny plik graficzny.

Kontrolka ImageMap

Język HTML udostępnia element `<map>` pozwalający na zaimplementowanie elementów graficznych zawierających wiele łączy. Takie konstrukcje nazywamy *mapami obrazkowymi*. Dzięki kontrolce serwerowej `ImageMap` takie funkcje są dostępne również w ASP.NET.

Kontrolka `ImageMap` wywodzi się z klasy `Image`. W celu dostarczenia funkcji mapy obrazkowej, kontrolka `ImageMap` posiada dodatkowo pewną liczbę właściwości i zapewnia obsługę zdarzenia `Click`. Wspomniane właściwości zostały przedstawione w tabeli 4.20.

Tabela 4.20. Właściwości kontrolki ImageMap

Nazwa	Typ	Pobierz	Ustaw	Wartości	Opis
`AlternateText`	String	x	x		Tekst wyświetlony w kontrolce, jeśli plik graficzny jest niedostępny. W przeglądarkach obsługujących funkcję `ToolTip` zostanie on również wyświetlony jako podpowiedź.
`GenerateEmpty-AlternateText`	Boolean	x	x	`true`, `false`	W przypadku wartości `true`, aplikacja wymusza generowanie pustego atrybutu `alt`, nawet jeżeli właściwość `AlternateText` jest pusta (`""`) lub nie została określona. Wartością domyślną jest `false`. Ta właściwość została dostarczona, aby obsługiwać strony internetowe w urządzeniach technologii wspomagających, takich jak czytniki ekranu.
`HotSpotMode`	HotSpotMode	x	x	`Inactive`, `Navigate`, `NotSet`, `PostBack`	Określa domyślny tryb punktu aktywnego lub działania kontrolki, w przypadku kliknięcia takiego punktu. Poszczególnym punktom aktywnym można przypisać różne tryby. I tak tryb `Navigate` powoduje natychmiastowe przejście do adresu URL podanego we właściwości `NavigateUrl`. Natomiast `PostBack` powoduje wysłanie żądania do serwera.
`HotSpots`	HotSpotCollecton	x			Zbiór obiektów `HotSpot` zawartych w kontrolce `ImageMap`.

Każda kontrolka `ImageMap` posiada zbiór obiektów `HotSpot`: są to aktywne obszary obrazka, które zapewniają reakcję na kliknięcie myszą. Obszary te odpowiadają znacznikom HTML `<area>` mapy obrazkowej. Obiekty `HotSpot` albo wywołują na serwerze zdarzenie `Click` (jeśli właściwość `HotSpotMode` jest ustawiona jako `PostBack`), albo w przypadku ustawienia właściwości `HotSpotMode` jako `Navigate` powodują natychmiastowe przejście do adresu URL podanego we właściwości `NavigateUrl`.

Istnieją trzy typy punktów aktywnych:

`RectangleHotSpot`
: Definiuje za pomocą właściwości `Top`, `Bottom`, `Left` i `Right` prostokątny obszar obrazka. Wszystkie piksele są podane względem lewego górnego rogu grafiki.

`CircleHotSpot`
: Definiuje aktywny obszar obrazka w kształcie koła. Właściwości `X` i `Y` określają środek okręgu, a piksele są podane względem lewego górnego rogu grafiki. Właściwość `Radius` określa w pikselach promień okręgu.

`PolygonHotSpot`
: Definiuje wielokątny obszar obrazka. W tym celu stosuje się rozdzieloną przecinkami listę współrzędnych `X` i `Y`, które określają końce linii tworzących obrys zaznaczanego obszaru. Wszystkie piksele są podane względem lewego górnego rogu obrazka.

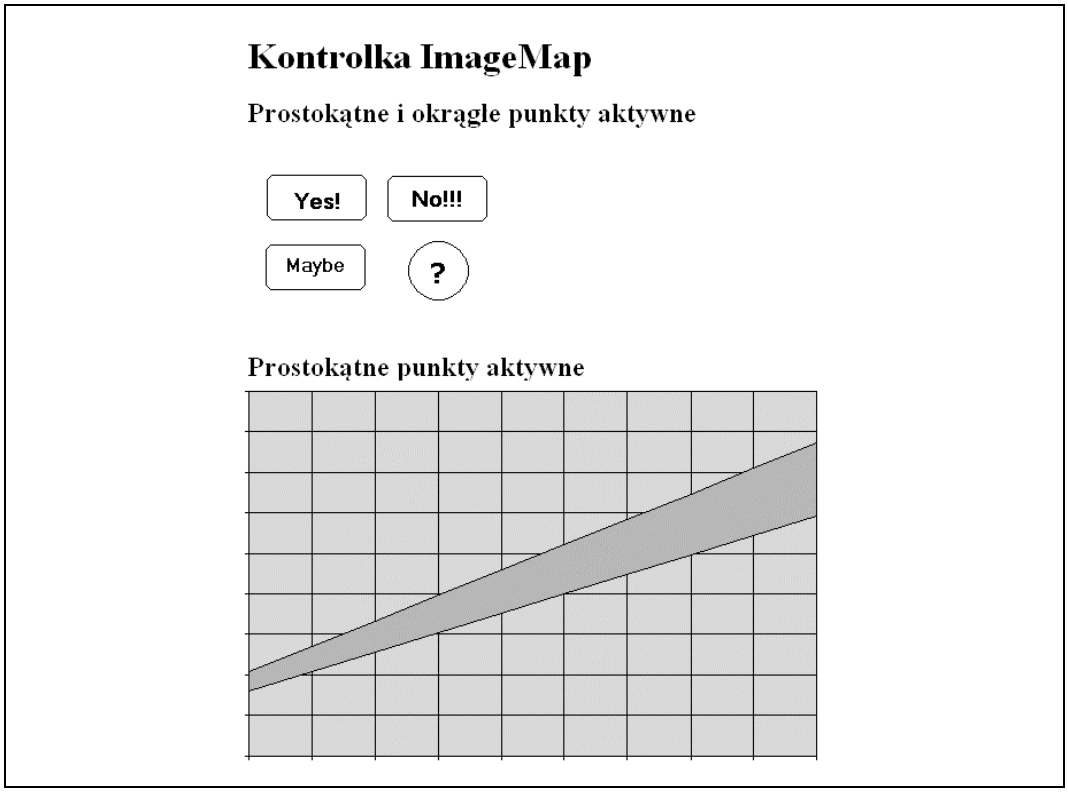
Wszystkie obiekty HotSpot posiadają wspólne właściwości, które zostały przedstawione w tabeli 4.21.

Tabela 4.21. Właściwości obiektu HotSpot

Nazwa	Typ	Pobierz	Ustaw	Wartości	Opis
AlternateText	String	x	x		Tekst wyświetlony w kontrolce, jeśli plik graficzny jest niedostępny. W przeglądarkach obsługujących funkcję ToolTip zostanie on również wyświetlony jako podpowiedź.
HotSpotMode	HotSpotMode	x	x	Inactive, Navigate, NotSet, PostBack	Określa domyślny tryb punktu aktywnego lub działania kontrolki po kliknięciu tego punktu. Poszczególnym punktom aktywnym mogą zostać przypisane różne tryby. I tak tryb Navigate powoduje natychmiastowe przejście do adresu URL podanego we właściwości NavigateUrl. Natomiast PostBack powoduje wysłanie żądania do serwera.
NavigateUrl	String	x	x		Określa adres URL, który zostanie wywołany po kliknięciu punktu aktywnego z właściwością HotSpotMode ustawioną jako Navigate. Dopuszczalne są albo względne, albo absolutne odniesienia, tak jak to opisano w ramce „Położenia plików" w podrozdziale „Kontrolki Button".
PostBackValue	String	x	x		Wartość klikniętego obiektu HotSpot przekazywanego przez argument zdarzeń ImageMapEventArgs. Dotyczy tylko sytuacji, gdy właściwość HotSpotMode jest ustawiona jako PostBack.
Target	String	x	x		Określa okno przeglądarki internetowej, w którym zostanie wyświetlona strona docelowa. Wartości właściwości Target są takie same jak przedstawione w tabeli 4.3 dla kontrolki HyperLink. Dotyczy tylko sytuacji, gdy właściwość HotSpotMode jest ustawiona jako Navigate.

Wszystkie opisane właściwości oraz zdarzenie Click zostały przedstawione w kolejnym przykładzie *ImageMapDemo*. Gotowa strona po kliknięciu punktu aktywnego „Tak" została pokazana na rysunku 4.22. W omawianym przykładzie wykorzystano dwie mapy obrazkowe. Pierwsza, umieszczona w górnej części strony, zawiera cztery punkty aktywne: trzy prostokątne i jeden okrągły. Druga mapa obrazkowa posiada zdefiniowane trzy wielokątne punkty aktywne: jeden nad paskiem, jeden pod paskiem oraz sam pasek jako trzeci punkt.

Plik z treścią dla tej aplikacji został przedstawiony na listingu 4.38, natomiast plik ukrytego kodu został umieszczony na listingu 4.39. Jedyną interesującą częścią w tym drugim pliku jest przedstawiona pogrubioną czcionką metoda obsługi zdarzeń imgmapYesNoMaybe_Click, która jest wykonywana, gdy zostanie kliknięty punkt aktywny z właściwością HotSpotMode ustawioną jako PostBack.



Rysunek 4.22. Aplikacja ImageMapDemo

Listing 4.38. Plik default.aspx aplikacji ImageMapDemo

```
<%@ Page Language="C#" AutoEventWireup="true"  CodeFile="Default.aspx.cs"
Inherits="_Default" %>

<!DOCTYPE html PUBLIC "-//W3C//DTD XHTML 1.1//EN"
"http://www.w3.org/TR/xhtml11/DTD/xhtml11.dtd">

<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server">
   <title>ImageMapDemo</title>
</head>
<body>
   <form id="form1" runat="server">
   <div>
      <h1>Kontrolka ImageMap</h1>
      <h2>Prostokątne i okrągłe punkty aktywne</h2>
      <asp:ImageMap ID="imgmapYesNoMaybe" runat="server"
            ImageUrl="YesNoMaybe.gif"
            HotSpotMode="Postback" OnClick="imgmapYesNoMaybe_Click">
         <asp:RectangleHotSpot
            PostBackValue="Yes"
            Bottom="60" Top="21" Left="17" Right="103"
            AlternateText="Oczywiście, że tak" />
         <asp:RectangleHotSpot
            HotSpotMode=PostBack
            PostBackValue="No"
            Bottom="60" Top="21" Left="122" Right="208"
```

```
            AlternateText="Do diabła, nie"/>
         <asp:RectangleHotSpot
            PostBackValue="Maybe"
            Bottom="122" Top="83" Left="16" Right="101"
            AlternateText="Hmm..., zastanowię się nad tym"/>
         <asp:CircleHotSpot
            HotSpotMode="Navigate"
            X="165" Y="106" Radius="25"
            NavigateUrl="http://localhost/websites/targetpage.aspx"
            Target="_blank"  AlternateText="Będę się musiał nad tym zastanowić."/>
      </asp:ImageMap>
      <asp:Label ID="lblMessage" runat="server" />

      <h2>Prostokątne punkty aktywne</h2>
      <asp:ImageMap ID="imgmapPlot" runat="server"
            ImageUrl="plot.gif"
            HotSpotMode="PostBack"
            OnClick="imgmapYesNoMaybe_Click">
         <asp:PolygonHotSpot Coordinates="4,245,4,3,495,3,495,45,"
            AlternateText="Nad paskiem"
            PostBackValue="Nad paskiem" />
         <asp:PolygonHotSpot Coordinates="4,245,495,45,495,112,3,264"
            AlternateText="W pasku"
            PostBackValue="W pasku" />
         <asp:PolygonHotSpot Coordinates="495,45,495,112,495,320,4,320"
            AlternateText="Pod paskiem"
            PostBackValue="Pod paskiem" />
      </asp:ImageMap>
   </div>
   </form>
</body>
</html>
```

Listing 4.39. Plik default.aspx.cs aplikacji ImageMapDemo

```
using System;
using System.Data;
using System.Configuration;
using System.Web;
using System.Web.Security;
using System.Web.UI;
using System.Web.UI.WebControls;
using System.Web.UI.WebControls.WebParts;
using System.Web.UI.HtmlControls;

public partial class _Default : System.Web.UI.Page
{
   protected void Page_Load(object sender, EventArgs e)
   {
   }
   protected void imgmapYesNoMaybe_Click(object sender, ImageMapEventArgs e)
   {
      lblMessage.Text = "Wartość PostBackValue wynosi " + e.PostBackValue;
   }
}
```

W deklaracji pierwszej mapy obrazkowej `imgMapYesNoMaybe` podano plik graficzny *YesNoMaybe.gif*, który znajduje się w tym samym katalogu razem ze stroną internetową. Ewentualnie można podać względną ścieżkę dostępu, na przykład:

```
ImageUrl="images\YesNoMaybe.gif"
```

lub pełną ścieżkę katalogu, na przykład:

```
ImageUrl="c:\websites\images\YesNoMaybe.gif"
```

albo adres internetowy bądź intranetowy, na przykład:

```
ImageUrl="http://www.dowolna_witryna/images/YesNoMaybe.gif"
```

Domyślna właściwość `HotSpotMode` w przypadku tej mapy obrazkowej jest ustawiona jako `PostBack`. Punkty aktywne `Yes` i `Maybe` opierają się na założeniu, że taka jest wartość tej właściwości, natomiast punkt aktywny `No` wyraźnie określa tę samą wartość. Punkt aktywny w postaci znaku zapytania używa innej wartości właściwości `HotSpotMode`, mianowicie `Navigate`. W tym ostatnim przypadku, właściwości `NavigateUrl` i `Target` zawierają wskazówki dotyczące adresu i sposobu wywołania docelowego obiektu (np. strony internetowej). Dla punktów aktywnych zapewniających odświeżanie strony, atrybut `OnClick` mapy obrazkowej jest połączony ze zdarzeniem `Click` metody `imgmapYesNoMaybe_Click` zawartej w pliku ukrytego kodu. Kod wspomnianej metody zapisano pogrubioną czcionką na listingu 4.39.

Druga mapa obrazkowa `imgmapPlot` definiuje trzy punkty aktywne o nieregularnych kształtach określonych przed współrzędne X, Y. Omawiany przykład zawiera proste punkty aktywne, składają się one jedynie z czterech prostych linii. W bardziej typowych zastosowaniach, na przykład mapie Polski, w której każde województwo będzie zdefiniowane jako punkt aktywny, można określić dziesiątki wierzchołków, jednakże nie należy popadać w przesadę i próbować zbyt dokładnie określić kształt punktu. Większość użytkowników i tak będzie klikać w pobliżu środka punktu aktywnego. Jeżeli użytkownik kliknie blisko krawędzi i wybierze przez pomyłkę sąsiadujący obszar aktywny, to po prostu skorzysta z przycisku *Wstecz* przeglądarki i spróbuje kliknąć odpowiedni punkt raz jeszcze, tym razem zachowując więcej ostrożności.

Argument zdarzenia `Click` jest typu `ImageMapEventArgs` i udostępnia jedną publiczną właściwość: `PostBackValue`. Odpowiada ona właściwości `HotSpot` o takiej samej nazwie, która została zadeklarowana w każdym z obiektów `HotSpot`. Właściwość `PostBackValue` jest otrzymywana w obsłudze zdarzeń `Click` z listingu 4.39, a następnie wykorzystywana do zapełnienia właściwości `Text` znajdującej się na stronie kontrolki `Label`.