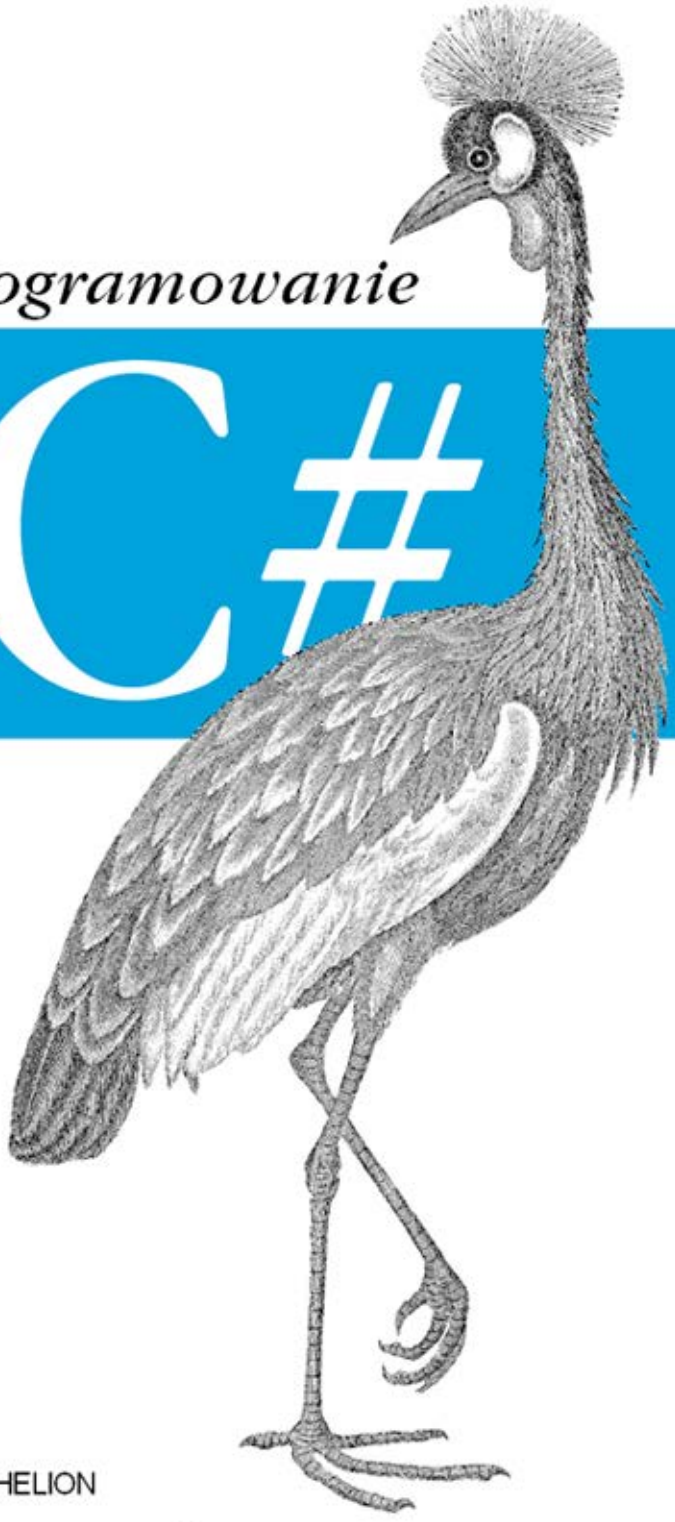
Najlepszy podręcznik poświęcony C#!

WYDANIE VI
Visual Studio 2010, .NET 4.0

Programowanie C#



HELION

O'REILLY®

Ian Griffiths,
Matthew Adams,
Jesse Liberty

Tytuł oryginału: Programming C# 4.0: Building Windows, Web,
and RIA Applications for the .NET 4.0 Framework

Tłumaczenie: Piotr Rajca (wstęp, rozdz. 1 – 11, 16, 19-22),
Łukasz Suma (rozdz. 12 – 15, 17–18)

ISBN: 978-83-246-3701-0

Pliki z przykładami omawianymi w książce można znaleźć pod adresem:
ftp://ftp.helion.pl/przyklady/cshpr6.zip

Wydawnictwo HELION
ul. Kościuszki 1c, 44-100 GLIWICE
tel. 32 231 22 19, 32 230 98 63
e-mail: *helion@helion.pl*
WWW: *http://helion.pl* (księgarnia internetowa, katalog książek)

Drogi Czytelniku!
Jeżeli chcesz ocenić tę książkę, zajrzyj pod adres
http://helion.pl/user/opinie/cshpr6
Możesz tam wpisać swoje uwagi, spostrzeżenia, recenzję.

Printed in Poland.

Spis treści

ROZDZIAŁ 21.

Tworzenie aplikacji w ASP.NET

Programiści coraz więcej swoich aplikacji piszą w taki sposób, by mogły one działać w internecie, a korzystanie z nich odbywało się za pośrednictwem przeglądarek WWW. Jak dowiedzieliśmy się w rozdziale 20., technologia Silverlight pozwala pisać kod C#, który będzie wykonywany w przeglądarce WWW po stronie klienta. Jeśli natomiast chodzi o obsługę aplikacji internetowych po stronie serwera, to .NET Framework udostępnia technologię ASP.NET.

W tym rozdziale skoncentrujemy się na przedstawieniu punktu, w którym spotykają się ASP.NET oraz język C# — technologii Web Forms. ASP.NET jest zagadnieniem bardzo obszernym i jeśli Czytelnik chciałby znaleźć jego obszerną i wyczerpującą prezentację, radzimy sięgnąć po książkę *Programming ASP.NET 3.5, Fourth Edition* napisaną przez Jessego Liberty, Dana Maharry'ego i Dana Hurwitza lub *Learning ASP.NET 3.5, Second Edition* napisaną przez Jessego Liberty, Dana Hurwitza i Dana MacDonalda (obie zostały wydane przez wydawnictwo O'Reilly).

Podstawy technologii Web Forms

Technologia Web Forms przenosi ideę szybkiego programowania aplikacji (RAD — *Rapid Application Development*) do świata programowania aplikacji internetowych. Z poziomu Visual Studio lub programu Visual Web Developer, korzystając z techniki „przeciągnij i upuść", można umieszczać elementy sterujące na formularzach i pisać specjalny „kod ukryty" (ang. *code-behind*) wspomagający ich działanie. Aplikacje tego typu są zazwyczaj wdrażane na serwerze WWW (przeważnie jest to serwer IIS, dostępny niemal we wszystkich wersjach systemu Windows, lub Cassini, wbudowany w Visual Studio w celu testowania pisanych aplikacji), a użytkownicy prowadzą z nimi interakcję, korzystając z przeglądarek WWW.

Technologia ASP.NET obsługuje także inne modele niż Web Forms — można na przykład operować bezpośrednio na poziomie żądań HTTP. Co więcej, platforma .NET 4 udostępnia nowy model MVC (skrót od angielskich słów *model*, *view*, *controller* — model, widok, kontroler). Model ten jest znacznie bardziej skomplikowany, lecz jednocześnie zapewnia znacznie większe możliwości oraz elastyczność, przez co stanowi doskonałe rozwiązanie w przypadku tworzenia złożonych aplikacji internetowych. Jako że niniejsza książka nie jest poświęcona wyłącznie technologii ASP.NET, przedstawimy tu wyłącznie proste przykłady aplikacji tworzonych przy użyciu technologii Web Forms.

Technologia Web Forms udostępnia model, w którym strony WWW są generowane dynamicznie na serwerze i dostarczane do przeglądarki za pośrednictwem internetu. Zapewnia ona możliwość tworzenia stron ASPX zawierających kod HTML oraz kontrolki sieciowe (ang. *web controls*), a także pisania kodu C# implementującego reakcje na czynności wykonywane przez użytkownika na stronie i dodającego do niej dynamiczne treści. Kod C# jest *wykonywany na serwerze*, a dane przez niego wytworzone są integrowane z kontrolkami umieszczonymi na stronie i wspólnie generują kod HTML, który następnie zostaje przesłany do przeglądarki użytkownika.

Koniecznie należy zwrócić uwagę na trzy kluczowe informacje podane w poprzednim akapicie i pamiętać o nich podczas lektury tego rozdziału:

- Strony WWW mogą zawierać zarówno kod HTML, jak i kontrolki sieciowe (opisane w dalszej części rozdziału).
- W technologii ASP.NET kod jest wykonywany na serwerze w środowisku zarządzanym. (Oczywiście technologii tej można używać w połączeniu z technologią AJAX bądź Silverlight, jeśli chcemy także skorzystać z kodu działającego po stronie klienta).
- Kontrolki ASP.NET generują standardowy kod HTML wyświetlany w przeglądarce.

W przypadku formularzy sieciowych tworzonych w technologii Web Forms interfejs użytkownika jest dzielony na dwa elementy: część wizualną (nazywaną także interfejsem użytkownika — UI) oraz logikę jej obsługi. Rozwiązanie to określane jest jako *separacja kodu* (ang. *code separation*) i jest czymś bardzo pożytecznym.

Interfejs użytkownika stron ASP.NET jest umieszczany w plikach z rozszerzeniem *aspx*. W przypadku żądania wykonania formularza serwer generuje kod HTML, a następnie przesyła go do przeglądarki użytkownika. W kodzie stron ASP.NET umieszczane są specjalne kontrolki Web Forms zdefiniowane w przestrzeniach nazw `System.Web` oraz `System.Web.UI` biblioteki klas .NET.

Pisanie stron korzystających z Web Forms przy użyciu Visual Studio nie mogło być prostsze. Wystarczy, że otworzymy formularz, przeciągniemy na niego kilka kontrolek i napiszemy kod, który będzie je obsługiwał. I gotowe! Właśnie napisaliśmy aplikację internetową.

Z drugiej strony, nawet w przypadku korzystania z Visual Studio napisanie solidnej i kompletnej aplikacji internetowej może być onieśmielającym zadaniem. Technologia Web Forms udostępnia niezwykle bogaty interfejs użytkownika — liczba i stopień złożoności oferowanych przez nią kontrolek znacząco wzrosły w ciągu kilku ostatnich lat, podobnie zresztą jak oczekiwania użytkowników odnośnie do ich wyglądu i sposobu działania.

Co więcej, aplikacje internetowe są z założenia aplikacjami rozproszonymi. Zazwyczaj klient takich aplikacji nie znajduje się w tym samym budynku co serwer. W przypadku większości z nich podczas tworzenia interfejsu użytkownika należy uwzględniać czasy opóźnień transmisji sieciowych, przepustowość oraz wydajność serwera, gdyż całkowity czas obsługi żądania może wynieść nawet kilka sekund.

Aby uprościć opisywane tu zagadnienia i umożliwić skoncentrowanie się na aspektach związanych z językiem C#, całkowicie pominiemy tu sprawy dotyczące przetwarzania wykonywanego po stronie klienta i zajmiemy się wyłącznie kontrolkami ASP.NET obsługiwanymi po stronie serwera.

Zdarzenia formularzy sieciowych

Formularze sieciowe są sterowane zdarzeniami. *Zdarzenie* reprezentuje „coś, co się zdarzyło" (więcej informacji na ten temat można znaleźć w rozdziale 5.).

Zdarzenie zostaje zgłoszone, gdy użytkownik kliknie przycisk, wybierze opcję z listy bądź wejdzie w jakąkolwiek inną interakcję z interfejsem użytkownika. Oczywiście w przypadku aplikacji internetowych interakcje te mają miejsce w przeglądarce WWW działającej na komputerze użytkownika, niemniej jednak zdarzenia ASP.NET są obsługiwane na serwerze. Aby takie rozwiązanie mogło działać, konieczne jest wykonanie pełnego cyklu komunikacji z serwerem. Przeglądarka musi przesłać do serwera żądanie, a ten z kolei musi na nie odpowiedzieć — dopiero wtedy zdarzenie zostanie całkowicie obsłużone. To może jednak trochę potrwać, więc w porównaniu z obsługą zdarzeń w klasycznych aplikacjach dla systemu Windows jesteśmy nieco ograniczeni — choć obsługiwanie po stronie serwera niektórych zdarzeń takich jak przesuwanie wskaźnika myszy po ekranie po prostu byłoby praktyczne, to jednak ASP.NET udostępnia jedynie ograniczoną liczbę zdarzeń, na przykład kliknięcie przycisku lub zmianę zawartości pola tekstowego. Są to zdarzenia, które mogą skutkować poważnymi zmianami i których obsługa może być warta wysłania żądania na serwer.

Zdarzenia przesyłane i nieprzesyłane

Zdarzenia przesyłane (ang. *postback events*) to takie, których zgłoszenie powoduje natychmiastowe przesłanie formularza na serwer. Zaliczają się do nich na przykład zdarzenia związane z obsługą kliknięć takie jak zdarzenie `Click` przycisków. Istnieje także liczna grupa zdarzeń *nieprzesyłanych*, czyli takich, których zgłoszenie nie powoduje natychmiastowego przesłania formularza na serwer.

Można zmusić kontrolki generujące zdarzenia nieprzesyłane, by ich zdarzenia powodowały przesłanie formularza. W tym celu wystarczy przypisać właściwości `AutoPost`
`↪Back` wartość `true`.

Zdarzenia nieprzesyłane są zgłaszane w momencie, gdy ASP.NET odkryje, że należy je zgłosić, co może jednak nastąpić ze znacznym opóźnieniem względem momentu, w którym użytkownik wykonał czynność prowadzącą do ich wygenerowania. Na przykład kontrolka `TextBox` udostępnia zdarzenie `TextChanged`. Raczej nie będziemy oczekiwać, że formularz zostanie automatycznie przesłany na serwer w momencie, gdy użytkownik zacznie coś wpisywać w polu tekstowym, dlatego też nie jest to zdarzenie przesyłane. Jeśli użytkownik wypełni kilka pól formularza, serwer nie będzie o tym nic wiedział — ta zmiana stanu następuje po stronie klienta. ASP.NET odkryje ją dopiero wtedy, gdy użytkownik kliknie przycisk, by przesłać formularz na serwer. Dopiero wówczas zostaną zgłoszone zdarzenia `TextChanged` dla wszystkich wypełnionych pól tekstowych. Oznacza to, że w ramach obsługi jednego żądania może zostać obsłużonych wiele różnych zdarzeń.

Stan widoku

Użytkownicy oczekują, że kontrolki tworzące interfejs użytkownika aplikacji będą pamiętały swój stan — zniknięcie wartości wpisanej w polu tekstowym lub opcji zaznaczonej na liście może być bardzo mylące. Niestety WWW jest z natury środowiskiem „bezstanowym"[1].

[1] Choć jest to uzasadnione przez architekturę sieci, nie wpływa to dobrze na łatwość korzystania z niej.

Oznacza to, że każde przesłanie żądania na serwer powoduje utratę stanu z poprzedniego żądania, chyba że programista włoży wiele pracy i starań, by zachować posiadaną wiedzę o sesji. WWW obejmuje całe mnóstwo witryn zawierających formularze, których zawartość jest całkowicie tracona w przypadku, gdy przesłane na serwer dane zawierają jakikolwiek błąd. Programiści muszą wykonać całkiem sporo pracy, by zapobiec takim sytuacjom. Jednak ASP.NET udostępnia mechanizmy pozwalające na automatyczną obsługę pewnych aspektów stanu.

Za każdym razem, gdy formularz zostaje przesłany na serwer, ten przed odesłaniem odpowiedzi do przeglądarki go odtwarza. ASP.NET oferuje mechanizm, który automatycznie zachowuje stan kontrolek obsługiwanych po stronie serwera (`ViewState`). A zatem jeśli formularz zawiera listę, a użytkownik wybrał jedną z dostępnych na niej opcji, to opcja ta pozostanie zaznaczona także po przesłaniu formularza na serwer i ponownym wyświetleniu go w przeglądarce.

Cykl życia stron w technologii Web Forms

Każde żądanie dotyczące strony docierające na serwer powoduje wygenerowanie na nim ciągu zdarzeń. Zdarzenia te składają się na całkowity *cykl życia* strony oraz wszystkich jej komponentów. Cykl ten zaczyna się od żądania zwrócenia strony, które sprawia, że serwer ją wczytuje. Obsługa żądania kończy się natomiast usunięciem strony z pamięci serwera. Ostatecznym celem obsługi żądania jest wygenerowanie i przesłanie do przeglądarki wynikowego kodu HTML.

Jako że ASP.NET jest technologią serwerową, dla niej cykl życia strony wygląda zupełnie inaczej niż z punktu widzenia użytkownika. W chwili gdy użytkownik zobaczy stronę, serwer już dawno skończył ją obsługiwać. Kiedy kod HTML dotrze do przeglądarki, równie dobrze można by wyłączyć serwer i odłączyć go od internetu, a użytkownik w ogóle by tego nie zauważył aż do momentu przesłania kolejnego żądania.

Cykl życia strony jest wyznaczany poniższymi zdarzeniami. Na każdym z etapów jej przetwarzania ASP.NET wykonuje konkretne operacje, jednak z każdym z tych zdarzeń można skojarzyć procedurę obsługi, by wykonać w niej jakieś dodatkowe czynności.

Inicjalizacja
: Inicjalizacja jest pierwszym etapem cyklu życia strony lub kontrolki. To właśnie podczas niej są inicjowane wszelkie ustawienia, które będą potrzebne podczas obsługi żądania.

Wczytanie `ViewState`
: Na tym etapie zostaje określona wartość właściwości `ViewState`. Wartość ta jest przechowywana w ukrytym polu formularza HTML. Kiedy ASP.NET po raz pierwszy generuje kod strony, umieszcza w nim to ukryte pole, a następnie korzysta z niego, by zachować stan strony pomiędzy kolejnymi żądaniami przesyłanymi na serwer. Ciąg znaków zapisany w tym polu jest przetwarzany przez platformę i na jego podstawie określana jest wartość właściwości `ViewState`. Zapewnia to możliwość zarządzania stanem poszczególnych kontrolek pomiędzy kolejnymi wyświetleniami strony, dzięki czemu ich zawartość nie jest za każdym razem przywracana do wartości domyślnej.

Przetworzenie danych zwrotnych
: Podczas tego etapu zostają przetworzone dane przesłane na serwer — tak zwane *dane zwrotne* (ang. *postback data*).

Wczytanie
: Na tym etapie następuje wywołanie metody `CreateChildControls`, które powoduje utworzenie i zainicjowanie kontrolek serwerowych w drzewie sterowania. Stan formularzy zostaje odtworzony, a ich kontrolki zawierają dane przesłane z przeglądarki w żądaniu.

Wysłanie modyfikacji danych zwrotnych
: Jeśli pojawiły się jakiekolwiek różnice pomiędzy poprzednim i bieżącym stanem, to zostaje wywołana metoda `RaisePostDataChangedEvent`, która powoduje zgłoszenie odpowiednich zdarzeń.

Obsługa zdarzeń
: Na tym etapie zostaje obsłużone zdarzenie (zgłoszone po stronie klienta), które spowodowało przesłanie żądania na serwer.

Generowanie wstępne
: To ostatni moment, by zmienić właściwości kontrolek, nim zostaną one wygenerowane. (W przypadku formularzy sieciowych „wygenerowanie" oznacza utworzenie odpowiedniego kodu HTML, który zostanie następnie przesłany do przeglądarki).

Zapisanie stanu
: Na początku cyklu życia strony jej zachowany stan został odczytany i odtworzony z ukrytego pola formularza. Na tym etapie jest on ponownie zapisywany w polu jako łańcuch znaków, co kończy pełny cykl przesyłania stanu pomiędzy klientem i serwerem.

Generowanie
: Podczas tego etapu jest generowany kod wynikowy, który zostanie przesłany do przeglądarki.

Zwolnienie
: To ostatni etap cyklu życia strony. Zapewnia on programiście możliwość wykonania ostatecznych porządków i zwolnienia referencji do wszelkich kosztownych zasobów takich jak połączenia z bazami danych.

Tworzenie aplikacji internetowych

Visual Studio udostępnia dwa sposoby tworzenia aplikacji internetowych. Nie jest to jedynie kwestia wyboru jednego z dwóch przycisków w menu zapewniających dostęp do tej samej możliwości — obie opcje działają całkowicie odmiennie, a Visual Studio nie oferuje wystarczających informacji o różnicach pomiędzy nimi w momencie, gdy trzeba dokonać wyboru. Jednym z tych dwóch sposobów jest skorzystanie z opcji *New Project*, która udostępnia różne szablony projektów ASP.NET umieszczone w sekcji *Visual C#/Web*, pozwalające na generowanie różnych rodzajów *projektów aplikacji internetowych*. Są to pełnoprawne projekty aplikacji Visual Studio generowane i budowane dokładnie tak samo jak wszelkie inne projekty (biblioteki, aplikacje konsolowe czy też aplikacje WPF). Projekty tego typu są nieco „lżejsze" — nie ma w nich pliku *.csproj* reprezentującego projekt. Nie istnieje też konieczność ich budowania, gdyż taki projekt składa się wyłącznie z plików źródłowych i to one później będą kopiowane na serwer WWW. W przykładach zamieszczonych w tym rozdziale wykorzystamy projekt aplikacji internetowej, gdyż jest on najbardziej podobny do innych typów projektów, które mieliśmy okazję poznać we wcześniejszej części książki.

Aby utworzyć prosty formularz internetowy, który wykorzystamy w następnym przykładzie, należy uruchomić Visual Studio .NET i wybrać z menu głównego opcję *File/New/Project*. W oknie dialogowym *New Project* należy następnie zaznaczyć opcję *Visual C#/Web* i wybrać szablon *ASP.NET Empty Web Application*.

Zgodnie z tym, co sugeruje nazwa szablonu, Visual Studio utworzy pustą aplikację internetową. W jej skład będzie początkowo wchodził wyłącznie plik *Web.config* zawierający ustawienia konfiguracyjne witryny. Aby dodać do aplikacji nowy formularz, należy wybrać opcję *Project/Add New Item*, po czym wybrać z listy szablonów wyświetlonej po lewej stronie okna opcję *Visual C#/Web*. Następnie należy wybrać szablon *Web Form* i nadać mu nazwę *HelloWeb.aspx*. W rezultacie Visual Studio utworzy także plik kodu ukrytego o nazwie *HelloWeb.aspx.cs*, który będzie można zobaczyć w panelu *Solution Explorer* po rozwinięciu opcji *HelloWeb.aspx*. (Dodatkowo pojawi się także plik *HelloWeb.aspx.designer.cs*, w którym Visual Studio będzie umieszczać wszelki kod, który musi wygenerować automatycznie. Nie należy w nim umieszczać żadnego własnego kodu, gdyż Visual Studio usuwa ten plik i odtwarza go za każdym razem, gdy musi wprowadzić do wygenerowanego kodu jakieś zmiany).

Pliki kodu ukrytego

Przyjrzyjmy się nieco dokładniej plikom *aspx* oraz plikom kodu ukrytego utworzonym przez Visual Studio. Najpierw zobaczmy kod HTML zapisany w pliku *HelloWeb.aspx*. Podczas edycji plików *aspx* Visual Studio może je wyświetlać w trzech różnych widokach. Domyślnym jest widok źródła (*Source*) prezentujący kod HTML. U dołu okna edytora znajdują się trzy przyciski, które pozwalają przełączać się pomiędzy trzema dostępnymi widokami: widokiem projektu (*Design*) prezentującym zawartość strony w taki sposób, w jaki będzie ona wyświetlana w przeglądarce, widokiem źródła (*Source*) prezentującym nieprzetworzony kod HTML strony oraz widokiem podzielonym (*Split*) prezentującym stronę jednocześnie na dwa opisane wcześniej sposoby. Jak można zauważyć, formularz został utworzony na stronie przy użyciu standardowego znacznika `form` języka HTML:

```
<form id="form1" runat="server">
```

Formularze internetowe ASP.NET wymagają, by na stronie znajdował się przynajmniej jeden formularz HTML umożliwiający zarządzanie interakcją z użytkownikiem, dlatego też Visual Studio utworzyło go podczas dodawania nowej strony *aspx*. Atrybut `runat="server"` jest kluczem do całej magii, która później będzie wykonywana na serwerze. Każdy znacznik zawierający ten atrybut jest traktowany jako kontrolka serwerowa, która powinna zostać wykonana przez ASP.NET Framework na serwerze.

Choć `form` jest standardowym znacznikiem HTML, to atrybut `runat` do tego standardu nie należy. Jednak ASP.NET usuwa ten atrybut z kodu HTML przed jego przesłaniem do przeglądarki. Ma on jakiekolwiek znaczenie wyłącznie na serwerze.

Wewnątrz formularza Visual Studio umieszcza otwierający i zamykający znacznik `div`, pomiędzy którymi można wstawiać własne kontrolki oraz tekst.

Po utworzeniu pustego formularza pierwszą czynnością, jaką Czytelnik zapewne będzie chciał wykonać, będzie umieszczenie na nim jakiegoś tekstu. Po przełączeniu się do widoku źródła można dodawać kod bezpośrednio do pliku HTML. W ten sposób możemy na przykład dodać jakąś zawartość do elementu `div` umieszczonego w sekcji `body` strony *aspx*, tak jak to pokazano na listingu 21.1 (dodany fragment został wyróżniony pogrubioną czcionką).

Listing 21.1. Dodawanie zawartości HTML

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

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title></title>
</head>
<body>
    <form id="form1" runat="server">
    <div>
        Witaj, świecie! Teraz jest: <%= DateTime.Now.ToString() %>
    </div>
    </form>
</body>
</html>
```

W ten sposób na stronie zostanie wyświetlone powitanie oraz aktualny lokalny czas:

```
Witaj, świecie! Teraz jest 2011-12-18 23:40:42
```

Znaczniki `<%` oraz `%>` działają dokładnie tak samo jak we wcześniejszej wersji technologii ASP — oznaczają kod, który został pomiędzy nimi umieszczony (w naszym przypadku jest to kod C#). Znak równości (`=`) znajdujący się bezpośrednio za znacznikiem otwierającym sprawia, że ASP.NET przetworzy umieszczone za nim wyrażenie i wyświetli jego wartość. Wykonajmy zatem tę stronę, naciskając klawisz *F5*.

Dodawanie kontrolek

Kontrolki serwerowe można umieszczać na formularzu na trzy sposoby: samodzielnie pisząc w pliku *aspx* odpowiedni kod, przeciągając je z panelu *Toolbox* (przy czym w tym przypadku można je umieszczać na stronie zarówno w widoku źródła, jak i w widoku projektu) lub ewentualnie pisząc kod, który doda kontrolki w trakcie działania programu. W ramach przykładu załóżmy, że chcemy wyświetlić na stronie trzy przyciski opcji zapewniające użytkownikowi składającemu zamówienie możliwość wyboru jednej z trzech firm kurierskich. W tym celu w kodzie strony, wewnątrz znacznika `<form>`, można by umieścić następujący kod HTML:

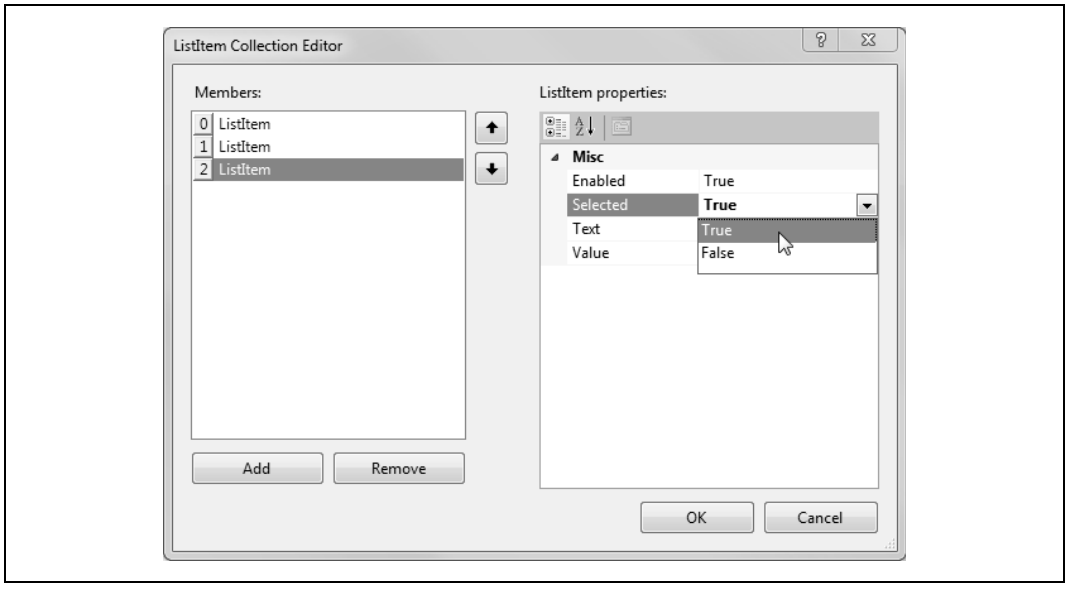
```
<asp:RadioButton GroupName="Shipper" id="Speedy"
    text="Speedy Express" Checked="True" runat="server">
</asp:RadioButton>
<asp:RadioButton GroupName="Shipper" id="United"
    text="United Package" runat="server">
</asp:RadioButton>
<asp:RadioButton GroupName="Shipper" id="Federal"
    text="Federal Shipping" runat="server">
</asp:RadioButton>
```

Znacznik `asp` deklaruje serwerową kontrolkę ASP.NET, która podczas przetwarzania strony zostaje zastąpiona zwyczajnym kodem HTML. Po uruchomieniu aplikacji przeglądarka wyświetli zbiór trzech przycisków opcji — wybór jednego z nich spowoduje usunięcie zaznaczenia pozostałych.

Dokładnie ten sam efekt można uzyskać znacznie łatwiej, przeciągając trzy przyciski opcji z panelu *Toolbox* Visual Studio i upuszczając je na formularzu, a jeszcze łatwiejszym rozwiązaniem będzie przeciągnięcie i upuszczenie na formularzu listy przycisków opcji, która będzie

zarządzać wszystkimi przyciskami jako jedną grupą. Kiedy zaznaczymy kontrolkę przycisku opcji w widoku projektu, pojawi się inteligentny znacznik (ang. *smart tag*), prosząc nas o podanie źródła danych (zapewnia to nam możliwość powiązania przycisku z kolekcją, na przykład pobraną z bazy danych) lub wpisanie ich samemu. Kliknięcie przycisku *Edit items* powoduje wyświetlenie okna *ListItem Collection Editor*, w którym możemy dodać potrzebne nam trzy przyciski opcji.

Każdy z przycisków będzie miał domyślną nazwę `ListItem`, jednak zarówno ich opisy, jak i wartości będzie można samemu podać w odpowiednich właściwościach. Można tam także określić, które z trzech pól będzie początkowo zaznaczone (patrz rysunek 21.1).

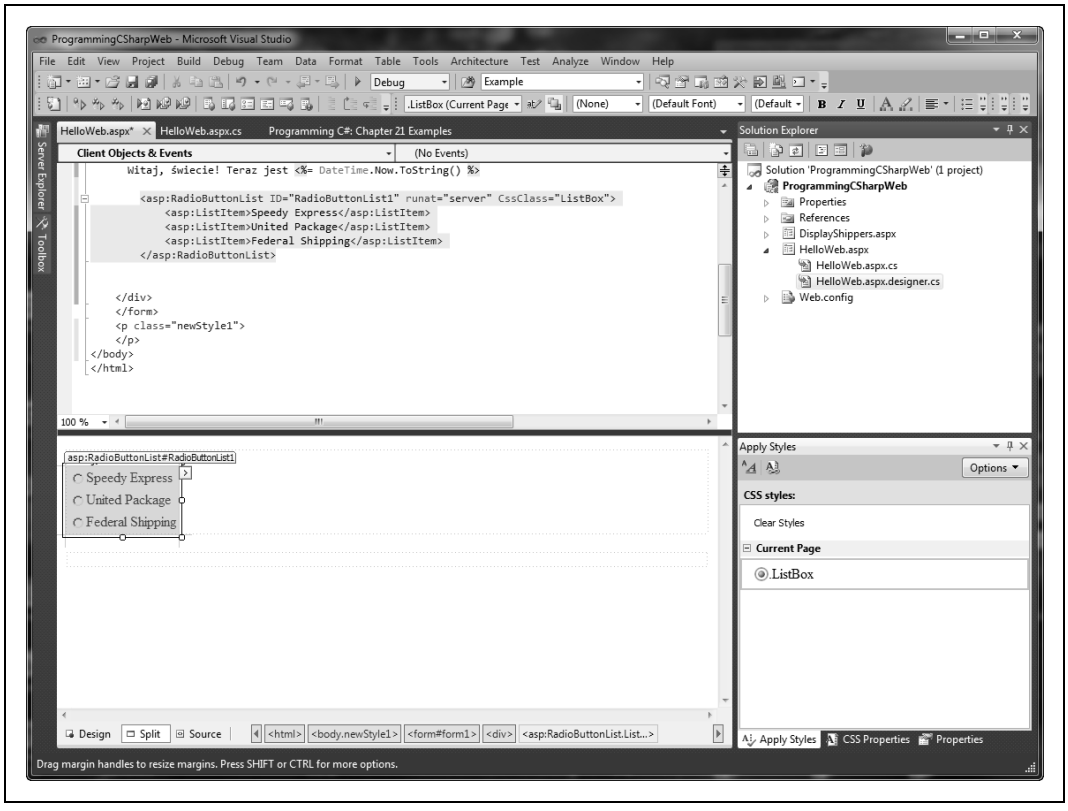


Rysunek 21.1. Kolekcja elementów ListItem

Wygląd listy przycisków opcji można poprawić, modyfikując wartości właściwości wyświetlonych w panelu *Properties*, które określają takie aspekty wyglądu jak: używana czcionka, kolor, liczba kolumn, kierunek powtarzania (domyślnie wybrany jest pionowy) itd. Można to również zrobić, korzystając z rozbudowanego wsparcia, jakie Visual Studio zapewnia w zakresie tworzenia arkuszy stylów CSS (przedstawionego na rysunku 21.2).

Rysunek 21.2 pokazuje, że można przełączać wyświetlane w prawym dolnym rogu Visual Studio panele *Properties* oraz *Apply Styles*. W przedstawionym przykładzie użyliśmy panelu *Properties*, by określić tekst etykiety ekranowej, a następnie panelu *Styles*, by utworzyć styl `ListBox`, który wyświetla wokół listy przycisków obramowanie oraz określa ich czcionkę i kolor. Skorzystaliśmy także z widoku podzielonego, by móc jednocześnie oglądać stronę w widoku źródła oraz projektu.

Sekwencja znaczników wyświetlana automatycznie u dołu okna Visual Studio pokazuje nasze aktualne położenie w strukturze dokumentu. Jak widać, znajdujemy się w elemencie `ListItem` w kontrolce `ListBox`, która jest umieszczona w elemencie `div` wewnątrz formularza `form1`. To naprawdę świetne narzędzie.



Rysunek 21.2. Stosowanie właściwości i stylów

Kontrolki serwerowe

Technologia Web Forms udostępnia dwa rodzaje kontrolek serwerowych. Pierwszym z nich są serwerowe kontrolki HTML. Wyglądają one niemal identycznie jak zwyczajne znaczniki HTML, jednak posiadają dodatkowy atrybut `runat="server"`.

Alternatywą dla serwerowych kontrolek HTML są kontrolki serwerowe ASP.NET (ang. *ASP.NET server controls*; nazywane także czasami kontrolkami internetowymi, ang. *web controls*). Stworzono je po to, by udostępnić nieco wygodniejszy interfejs API do pracy ze standardowymi kontrolkami HTML. Kontrolki internetowe udostępniają nieco bardziej spójny model obiektowy oraz spójne nazewnictwo właściwości. W przypadku kontrolek HTML istnieje na przykład wiele sposobów obsługi wprowadzania danych:

```
<input type="radio">
<input type="checkbox">
<input type="button">
<input type="text">
<textarea>
```

Każda z tych kontrolek działa inaczej i wymaga zastosowania innych atrybutów. To dosyć przykry efekt nieco przypadkowego rozwoju języka HTML we wczesnych latach istnienia WWW. Kontrolki internetowe mają za zadanie znormalizować zbiór dostępnych elementów

sterujących oraz zapewnić spójne wykorzystanie atrybutów w całym ich modelu obiektowym. Poniżej przedstawione zostały kontrolki internetowe odpowiadające kontrolkom HTML widocznym w poprzednim przykładzie.

```
<asp:RadioButton>
<asp:CheckBox>
<asp:Button>
<asp:TextBox rows="1">
<asp:TextBox rows="5">
```

Kod HTML przekazywany do przeglądarki użytkownika nie zawiera znaczników zaczynających się od `asp:` (i bardzo dobrze, gdyż żadna przeglądarka nie wiedziałaby, co z nimi zrobić). ASP.NET konwertuje je na standardowy kod HTML, a zatem z punktu widzenia klienta nie ma żadnej różnicy pomiędzy internetowymi kontrolkami ASP.NET a standardowymi kontrolkami HTML. Wszystko sprowadza się więc do pytania, z jakiego API będziemy chcieli korzystać na serwerze: czy chcemy, by kod na serwerze operował na tych samych elementach i właściwościach, które są używane po stronie klienta, czy też wolimy, by kontrolki były zgodne z konwencjami używanymi we wszystkich innych klasach biblioteki .NET.

W dalszej części rozdziału skoncentrujemy się na kontrolkach internetowych.

Wiązanie danych

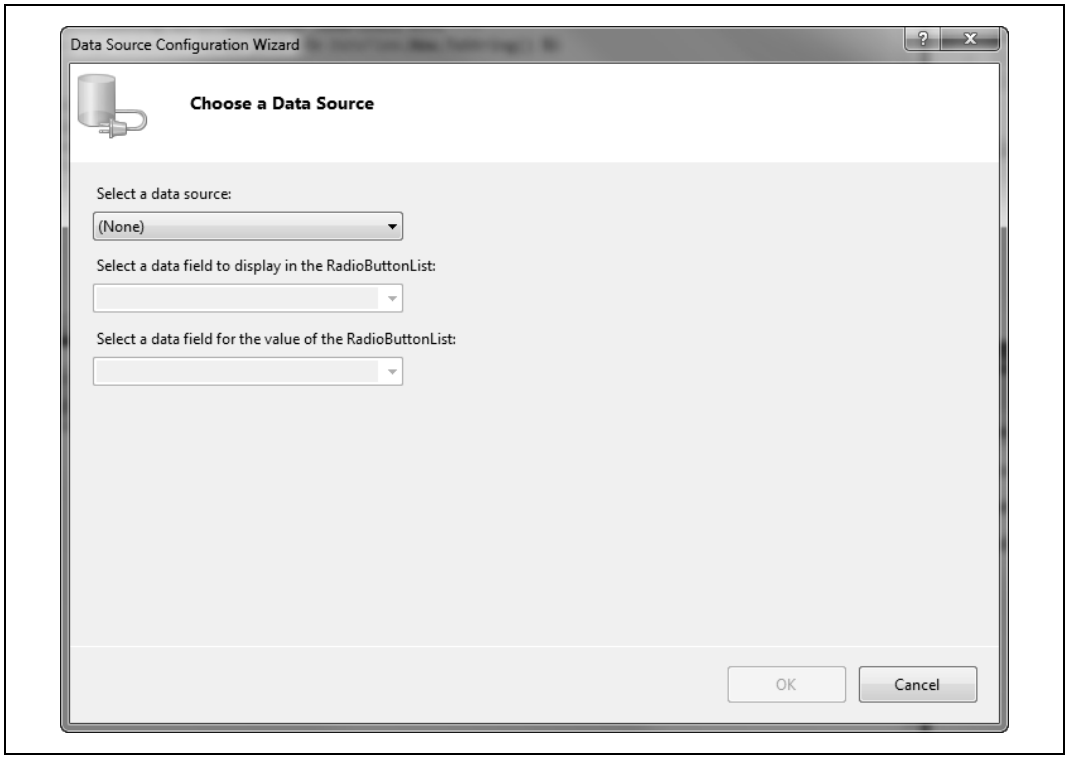
Choć niektóre treści prezentowane przez aplikacje internetowe mogą być stałe, to jednak każda interesująca witryna WWW będzie się zmieniać wraz z upływem czasu. Dlatego też jest wysoce prawdopodobne, że będziemy chcieli, by niektóre kontrolki na stronie wyświetlały dane, które od czasu do czasu mogą podlegać zmianom i które najprawdopodobniej będą przechowywane w bazie danych. Wiele kontrolek ASP.NET można powiązać z danymi, co znacznie ułatwia prezentowanie tych danych oraz ich modyfikację.

W poprzednim podrozdziale na stałe umieściliśmy na formularzu trzy przyciski opcji, po jednym dla każdej z trzech firm kurierskich, które może wybrać użytkownik. Jednak to nie jest najlepsze rozwiązanie — dostawcy się zmieniają, a poza tym istnieje duże prawdopodobieństwo, że w przyszłości będziemy chcieli nawiązać współpracę także z innymi firmami kurierskimi. Nie chcemy, by każda zmiana relacji biznesowych zmuszała nas do ręcznego modyfikowania tych kontrolek. Znacznie bardziej rozsądnym rozwiązaniem będzie zapisanie listy firm kurierskich w bazie danych i powiązanie ich z przyciskami opcji wyświetlanymi na formularzu. W tym podrozdziale dowiemy się, w jaki sposób można utworzyć te kontrolki dynamicznie i powiązać je z informacjami przechowywanymi w bazie danych, korzystając w tym celu z możliwości wiązania z bazą danych zapewnianej przez kontrolkę `RadioButtonList`.

Dodajmy zatem do naszej aplikacji nowy formularz o nazwie *DisplayShippers.aspx* i wyświetlmy go w widoku podzielonym. Teraz musimy przeciągnąć kontrolkę `RadioButtonList` z panelu *Toolbox*, umieszczając ją na naszym nowym formularzu — bądź to w panelu widoku projektu, bądź to w kodzie źródłowym wewnątrz znacznika `<div>`.

Jeśli z lewej strony okna Visual Studio nie jest widoczny panel *Toolbox* z kontrolką przycisków opcji, to można go wyświetlić, wybierając z menu głównego opcję *View/Toolbox*, a następnie rozwijając zakładkę *Standard*. By uporządkować kontrolki, można kliknąć w panelu *Toolbox* prawym przyciskiem myszy i z wyświetlonego menu kontekstowego wybrać opcję *Sort Items Alphabetically*.

Następnie w widoku projektu należy kliknąć na „inteligentnej" ikonie nowej kontrolki — niewielkiej strzałce wyświetlonej w jej prawym górnym rogu. Z wyświetlonego menu należy wybrać opcję *Choose Data Source*, a na ekranie pojawi się okno dialogowe *Data Source Configuration Wizard* przedstawione na rysunku 21.3.



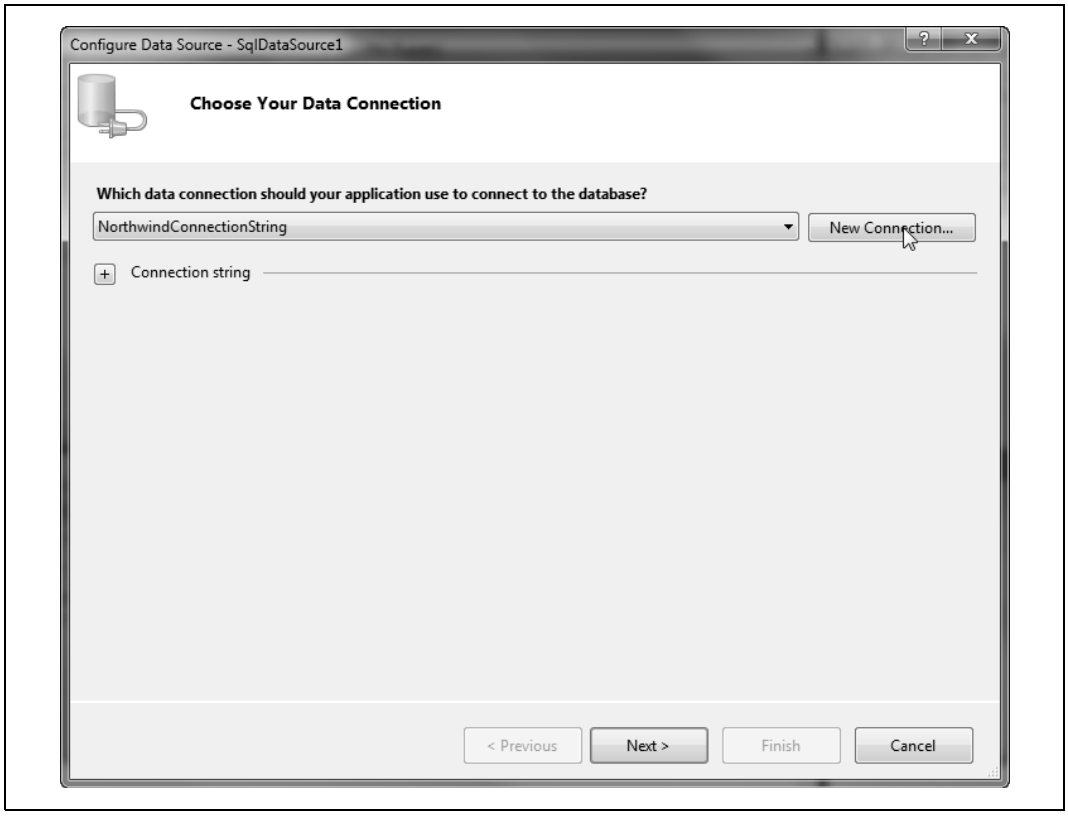
Rysunek 21.3. Okno dialogowe Data Source Configuration Wizard

W oknie tym należy rozwinąć listę *Select a data source* i wybrać z niej opcję <*New Data Source*>. Następnie zostaniemy poproszeni o wybór źródła danych spośród typów danych dostępnych na komputerze. Powinniśmy wybrać opcję *Database*, podać identyfikator i kliknąć przycisk *OK*. Na ekranie zostanie wyświetlone okno dialogowe *Configure Data Source* przedstawione na rysunku 21.4.

Możemy albo wybrać istniejące połączenie z bazą danych, albo utworzyć nowe. W naszym przypadku skorzystamy z tej drugiej możliwości — kliknijmy zatem przycisk *New Connection*, by je skonfigurować. Na ekranie zostanie wyświetlone okno dialogowe *Add Connection*.

Kolejnym krokiem będzie wypełnienie pól formularza: należy wybrać nazwę serwera, podać informacje o sposobie logowania się do niego (w razie wątpliwości należy wybrać opcję *Windows Authentication*) oraz podać nazwę bazy danych (w naszym przykładzie będzie to *Northwind*). Koniecznie powinniśmy także pamiętać o kliknięciu przycisku *Test Connection*, by przetestować połączenie z bazą danych. Kiedy wszystko będzie działać prawidłowo, można kliknąć przycisk *OK*, tak jak to pokazano na rysunku 21.5.

Po kliknięciu przycisku *OK* właściwości połączenia zostaną wpisane do okna dialogowego *Configure Data Source*. Warto jeszcze raz się im przyjrzeć, a jeśli wszystko będzie w porządku,



Rysunek 21.4. Wybieranie połączenia ze źródłem danych

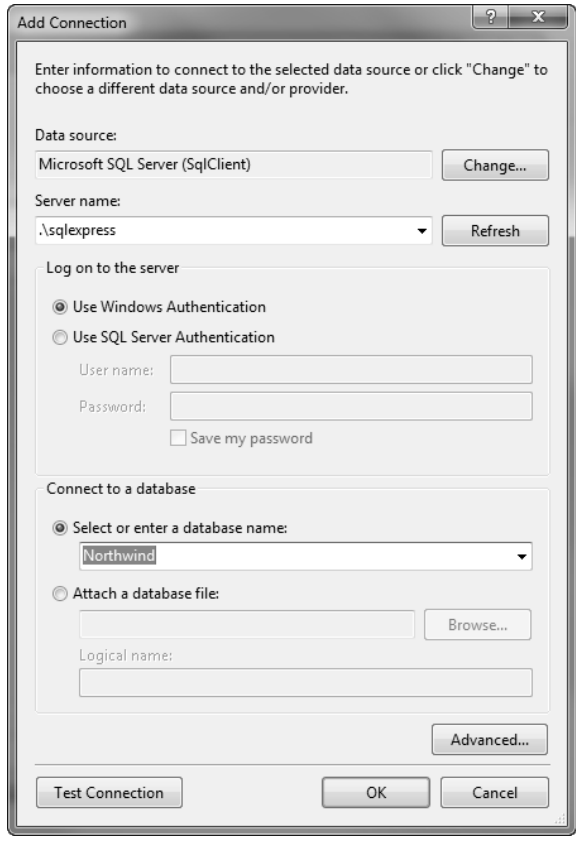
można kliknąć przycisk *Next*. Jeśli chcemy zapisać połączenie w pliku konfiguracyjnym *web.config*, to na kolejnej stronie kreatora powinniśmy podać jego nazwę (np. `NorthWindConnec ↪tionString`).

Po kliknięciu kolejnego przycisku *Next* będziemy mieli możliwość określenia tabel i kolumn, które chcemy pobierać, bądź też podania własnego zapytania SQL lub nazwy procedury składowanej, które zostaną użyte do pobrania danych.

W naszym przykładzie powinniśmy rozwinąć listę *Tables* i przewinąć jej zawartość tak, by była widoczna opcja `Shippers`. Następnie powinniśmy zaznaczyć pola `ShipperID` oraz `CompanyName`, jak pokazano to na rysunku 21.6.

W dalszej kolejności należy ponownie kliknąć przycisk *Next* i przetestować połączenie, by sprawdzić, czy z bazy danych są pobierane oczekiwane wartości (co pokazano na rysunku 21.7).

W końcu nadszedł czas, by połączyć utworzone przed chwilą źródło danych z kontrolką `Radio ↪ButtonList`. Kontrolka ta (podobnie jak większość innych list) rozróżnia wartość wyświetlaną (w naszym przypadku nazwę firmy kurierskiej) oraz wartość wyboru (w naszym przykładzie jest nią identyfikator firmy kurierskiej). Pola te należy wskazać w kreatorze, wybierając je z list rozwijalnych, tak jak to pokazano na rysunku 21.8.



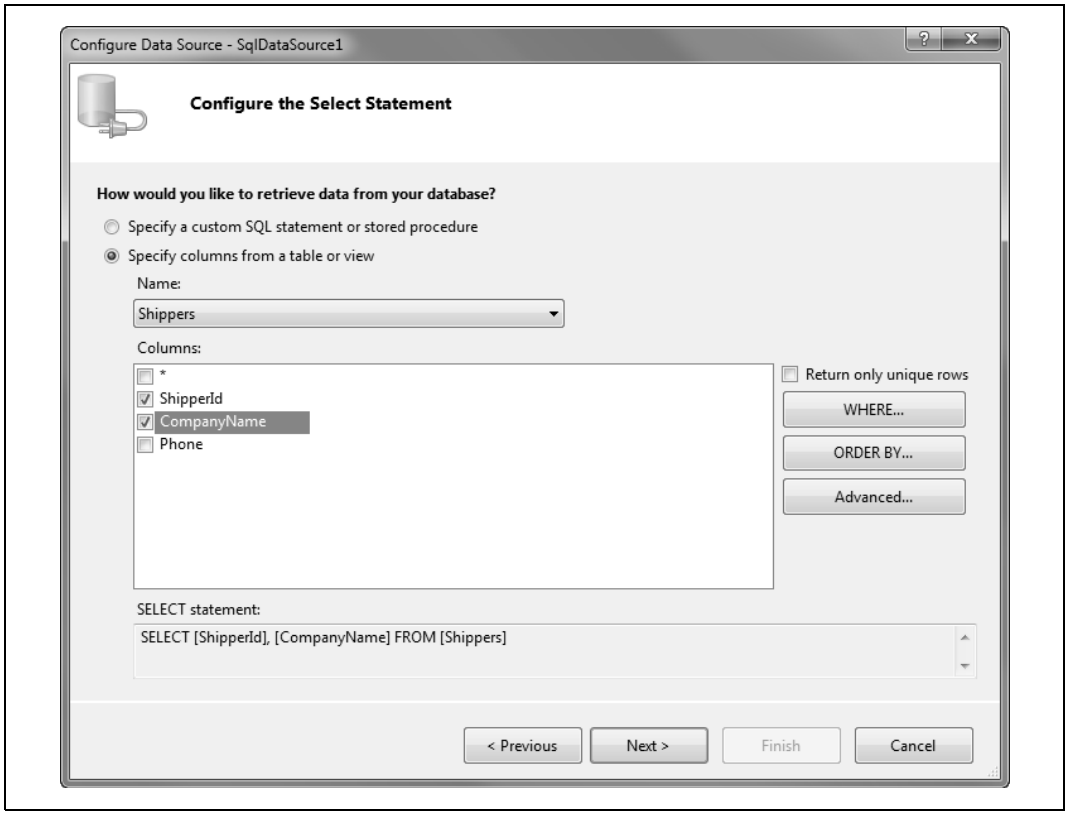
Rysunek 21.5. Okno dialogowe Add Connection

Sprawdzanie kodu

Zanim przejdziemy dalej, należy zwrócić uwagę na kilka zagadnień. Kiedy naciśniemy klawisz *F5*, by uruchomić aplikację, zostanie ona wyświetlona w przeglądarce, a strona, zgodnie z oczekiwaniami, będzie prezentowała grupę przycisków opcji. Możemy teraz wybrać opcję *Choose View/Source*, a przekonamy się, że do przeglądarki został przesłany prosty kod HTML przedstawiony na listingu 21.2.

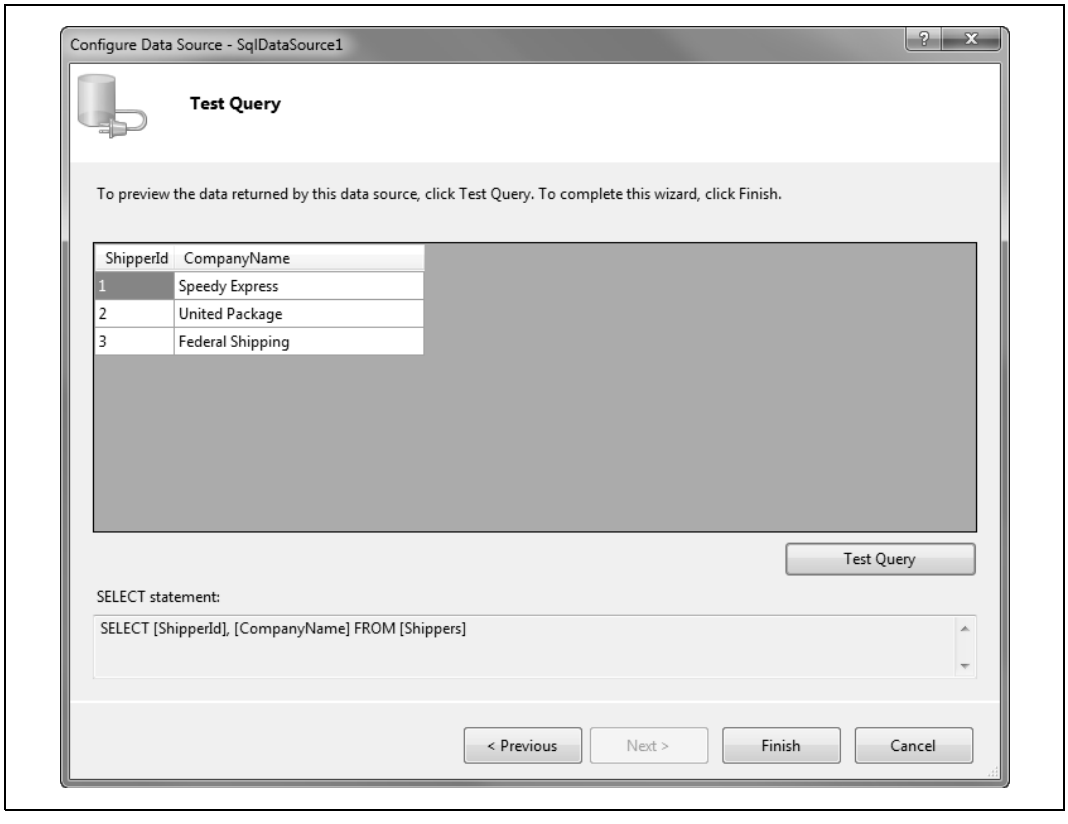
Listing 21.2. Kod źródłowy wygenerowanej strony

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

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

</title></head>
<body>
    <form method="post" action="DisplayShippers.aspx" id="form1">
<div class="aspNetHidden">
```



Rysunek 21.6. Konfiguracja polecenia Select

```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUJMjMzNjY1MzU4D2QWAgIDD2QWAgIBDxAPFgIeC18hRGFOYUJvdW5kZ2QQFQMOU3BlZWR5
IEV4cHJlc3MOVW5pdGVkIFBhY2thZ2UQRmVkZXJhbCBTaGlwcGluZxUDATEBMgEzFCsDA2dnZ2RkZCOksd8
IILjpH4OAdNkxGqjSaORYAA3N2F8zJz4lyxsv" />
</div>

<div class="aspNetHidden">
    <input type="hidden" name="__EVENTVALIDATION" id="__EVENTVALIDATION"
    value="/wEWBQKO2+CfAgL444i9AQL544i9AQL644i9AQL3jKLTDWylFXks1YMe8G5o7AkyHjJysQkO
    Cliwu8U/2yTrYA/Y" />
</div>
    <div>
        <table id="RadioButtonList1">
    <tr>
        <td><input id="RadioButtonList1_0" type="radio"
            name="RadioButtonList1" value="1" />
            <label for="RadioButtonList1_0">Speedy Express</label></td>
    </tr><tr>
        <td><input id="RadioButtonList1_1" type="radio"
            name="RadioButtonList1" value="2" />
            <label for="RadioButtonList1_1">United Package</label></td>
    </tr><tr>
        <td><input id="RadioButtonList1_2" type="radio"
            name="RadioButtonList1" value="3" />
            <label for="RadioButtonList1_2">Federal Shipping</label></td>
    </tr>
```

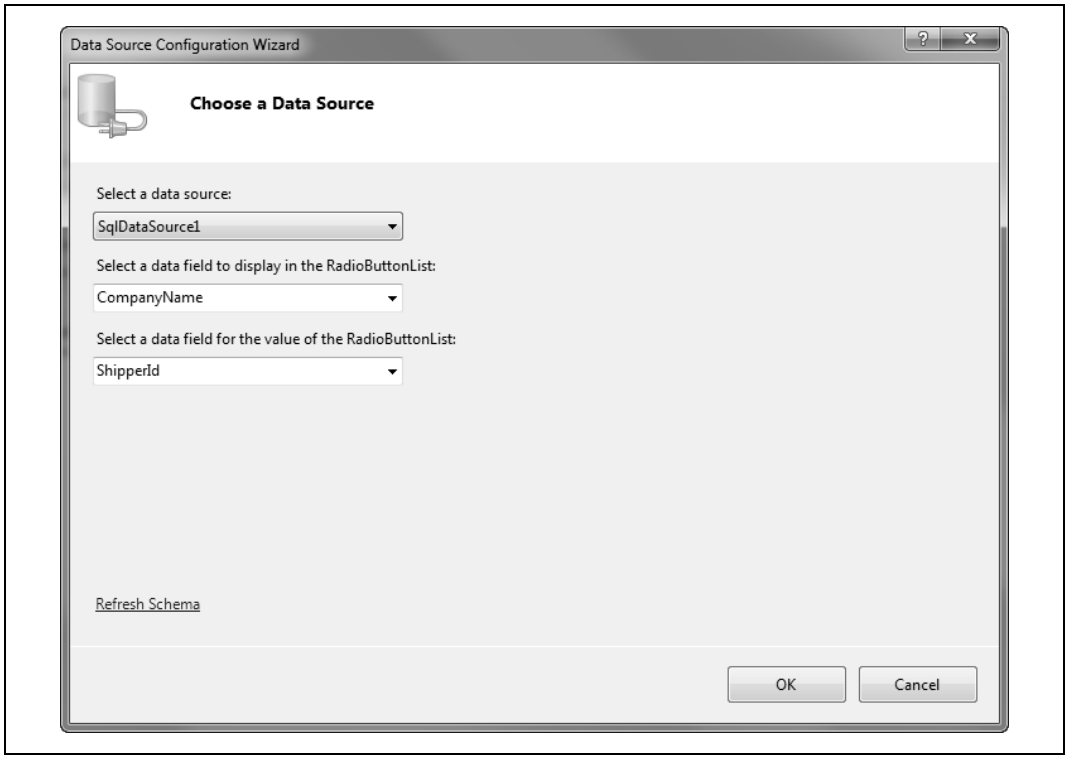


Rysunek 21.7. Testowanie zapytania

```
</table>

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

Zwróćmy uwagę, że w powyższym kodzie HTML nie ma elementu `RadioButtonList`; zawiera on tabelę z komórkami, wewnątrz których są umieszczone standardowe kontrolki HTML oraz etykiety. A zatem ASP.NET przekształcił swoje kontrolki na kod HTML zrozumiały dla każdej przeglądarki WWW.

Wrogo nastawiony użytkownik może utworzyć żądanie wyglądające dokładnie tak jak prawidłowe dane przesłane z naszego formularza, lecz zawierające zupełnie inne, nieoczekiwane przez nas wartości. Może mu to zapewnić możliwość wyboru opcji, która nie powinna być dla niego dostępna (na przykład opcji dostępnej wyłącznie dla uprzywilejowanych użytkowników), bądź nawet umożliwić przeprowadzenie ataku *SQL injection* (wstrzyknięcia kodu SQL). A zatem w przypadkach udostępniania w formularzach HTML istotnych informacji takich jak wartości kluczy głównych należy zachować szczególną ostrożność i pamiętać o tym, że dane pochodzące z formularza wcale nie muszą być tymi, które zostały do niego przesłane. Więcej informacji na temat pisania bezpiecznych aplikacji .NET można znaleźć na stronie *http://msdn.microsoft.com/security/*.



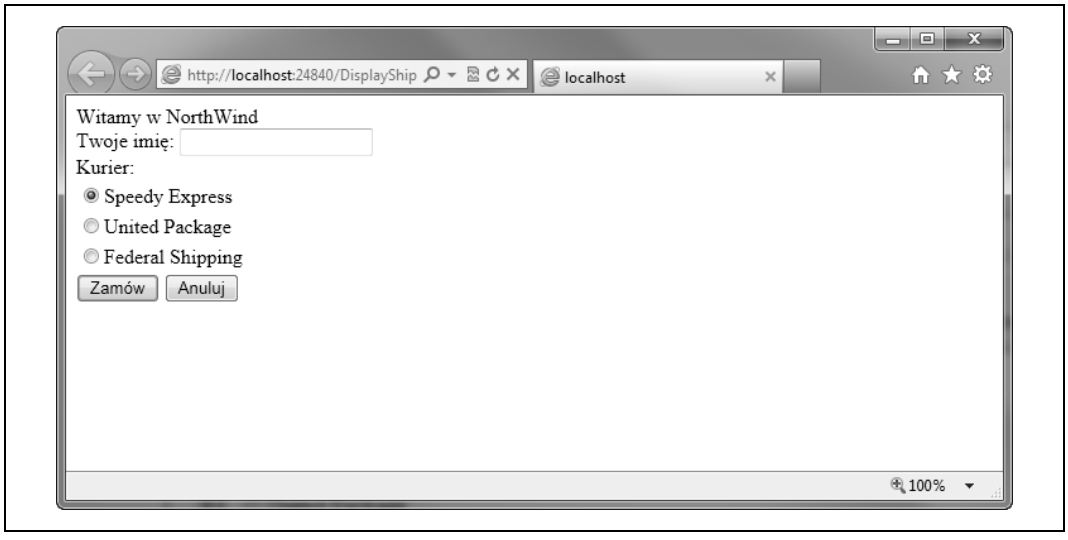
Rysunek 21.8. Wiązanie przycisków opcji ze źródłem danych

Dodawanie kontrolek i formularzy

Wystarczy dodać do naszej przykładowej strony kilka kolejnych kontrolek, by utworzyć kompletny formularz zapewniający możliwość prowadzenia interakcji z użytkownikiem. W tym celu dodamy bardziej odpowiednie powitanie („Witamy w NorthWind"), pole tekstowe pozwalające na podanie imienia, dwa nowe przyciski (*Zamów* oraz *Anuluj*) i tekst, który umożliwi nam wyświetlanie komunikatów dla użytkownika. Wygląd strony po wprowadzeniu tych modyfikacji przedstawiliśmy na rysunku 21.9.

Ten formularz nie wygrałby żadnej nagrody dla najlepszego projektu, niemniej jednak pozwoli nam przedstawić kilka kluczowych zagadnień związanych z formularzami Web Forms.

Nigdy nie spotkaliśmy programisty, który by nie uważał, że potrafi zaprojektować idealny interfejs użytkownika. Jednocześnie nigdy nie udało nam się spotkać takiego, który by to rzeczywiście potrafił. Projektowanie interfejsów użytkownika jest jedną z tych umiejętności (tak jak nauczanie), które każdy w jego własnej opinii posiada, jednak które bardzo niewiele naprawdę utalentowanych osób rozwinęło w wystarczającym stopniu. Jako programiści doskonale znamy swoje ograniczenia: piszemy kod, a ktoś inny rozmieszcza kontrolki na stronie, zapewniając ich odpowiednią użyteczność. Czytelnikom zainteresowanym tymi zagadnieniami gorąco polecamy książkę *Don't Make me Think: A Common Sense Approach to Web Usibility* napisaną przez Steve'a Kruga i wydaną przez wydawnictwo New Riders Press oraz *Why Software Sucks... and What You Can Do About It* napisaną przez Davida Platta i wydaną przez wydawnictwo Addison-Wesley.



Rysunek 21.9. Dokończony formularz z listą firm kurierskich

Listing 21.3 przedstawia kompletny kod pliku *aspx*.

Listing 21.3. Zawartość pliku aspx

```
<%@ Page Language="C#" AutoEventWireup="true"
 CodeBehind="DisplayShippers.aspx.cs"
 Inherits="ProgrammingCSharpWeb.DisplayShippers" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
<title></title>
</head>
<body>
  <form id="form1" runat="server">
  <div>Witamy w NorthWind</div>
  <div>
    Twoje imię:
    <asp:TextBox ID="txtName" runat="server"></asp:TextBox></div>
    <div>Kurier:</div>
  <div>
    <asp:RadioButtonList ID="rblShippers" runat="server"
       DataSourceID="SqlDataSource1" DataTextField="CompanyName"
       DataValueField="ShipperID">
    </asp:RadioButtonList>
    <asp:SqlDataSource ID="SqlDataSource1" runat="server"
       ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
       SelectCommand="SELECT [ShipperID], [CompanyName] FROM [Shippers]">
    </asp:SqlDataSource>
  </div>
  <div>
    <asp:Button ID="btnOrder" runat="server" Text="Zamów" />
    <asp:Button ID="Button2" runat="server" Text="Anuluj" />
  </div>
  <div>
```

```
    <asp:Label id="lblMsg" runat=server></asp:Label>
  </div>
</form>
</body>
</html>
```

Kiedy użytkownik kliknie przycisk *Zamów*, odczytamy wartość wpisaną przez niego w polu tekstowym oraz wyświetlimy komunikat potwierdzający wybór jednej z firm kurierskich. Trzeba pamiętać, że w momencie projektowania formularza nie jest ona znana, gdyż nazwy wszystkich dostępnych firm są pobierane z bazy danych w trakcie działania aplikacji, niemniej jednak możemy pobrać z kontrolki `RadioButtonList` wybraną nazwę lub identyfikator.

Aby to zrobić, należy wyświetlić formularz w widoku projektu i dwukrotnie kliknąć przycisk *Zamów*. W rezultacie Visual Studio wyświetli plik kodu ukrytego oraz utworzy procedurę obsługi zdarzeń `Click` przycisku.

Aby uprościć kod naszej przykładowej aplikacji, nie będziemy sprawdzać, czy użytkownik wpisał swoje imię w polu tekstowym. Informacje na temat kontrolek, które znacznie upraszczają taką weryfikację poprawności danych, można znaleźć w książce *ASP.NET. Programowanie*.

W kodzie obsługującym zdarzenie określamy treść wyświetlanego komunikatu, umieszczając w niej imię odczytane z pola tekstowego oraz tekst i wartość kontrolki `RadioButtonList`:

```
protected void btnOrder_Click(object sender, EventArgs e)
{
    lblMsg.Text = "Witaj, " + txtName.Text.Trim() +
    ". Dziękujemy za zamówienie. " +
    "Wybrałeś firmę kurierską " + rblShippers.SelectedItem.Text +
    " o identyfikatorze " + rblShippers.SelectedValue + ".";
}
```

Po uruchomieniu naszej aplikacji można zauważyć, że początkowo żaden z przycisków opcji nie jest zaznaczony. Podczas wiązania listy ze źródłem danych nie określiliśmy, która z wartości powinna być traktowana jako domyślna. Można to zrobić na kilka różnych sposobów, jednak najprostszym z nich jest dodanie jednego wiersza kodu do metody `Page_Load` automatycznie tworzonej przez Visual Studio:

```
protected void Page_Load(object sender, EventArgs e)
{
    rblShippers.SelectedIndex = 0;
}
```

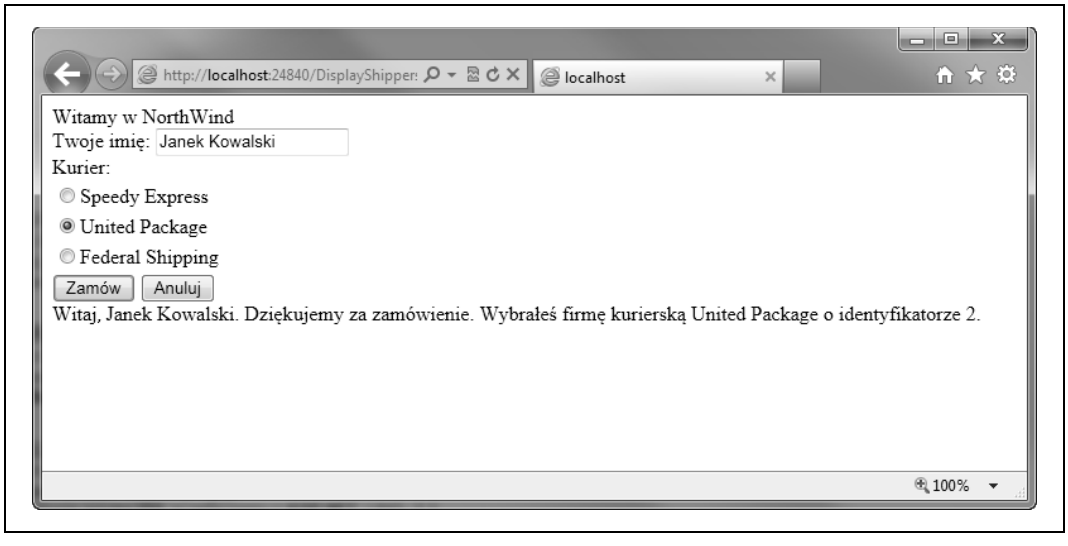
Zastosowanie powyższego kodu sprawi, że zostanie zaznaczony pierwszy przycisk opcji w kontrolce `RadioButtonList`. W powyższym rozwiązaniu występuje jeden dosyć subtelny problem. Otóż kiedy uruchomimy aplikację, zostanie zaznaczony pierwszy przycisk, a jeśli następnie wybierzemy drugi lub trzeci i klikniemy *Wyślij*, to okaże się, że po ponownym wyświetleniu strony znowu będzie zaznaczony przycisk pierwszy. Może się zatem wydawać, że nie da się wybrać żadnej innej opcji poza pierwszą. Dzieje się tak dlatego, że za każdym razem, gdy strona jest wyświetlana, zostaje wykonane zdarzenie `OnLoad`, a procedura jego obsługi zawsze zaznacza pierwszy z przycisków opcji w kontrolce `RadioButtonList`.

Jednak nam chodzi o to, by ten pierwszy przycisk został zaznaczony wyłącznie podczas pierwszego wyświetlenia strony, lecz nie podczas kolejnych wyświetleń następujących po kliknięciu przycisku *Wyślij*.

Aby rozwiązać ten problem, wystarczy umieścić wiersz kodu zaznaczający pierwszy przycisk kontrolki `RadioButtonList` wewnątrz instrukcji warunkowej sprawdzającej, czy formularz został przesłany na serwer:

```
protected void Page_Load(object sender, EventArgs e)
{
    if (!IsPostBack)
    {
        rblShippers.SelectedIndex = 0;
    }
}
```

W momencie wykonywania strony sprawdzana jest wartość właściwości `IsPostBack`. Za pierwszym razem przyjmie ona wartość `false`, zatem pierwsze pole kontrolki zostanie zaznaczone. Kiedy zaznaczymy jakiś przycisk opcji i klikniemy przycisk *Wyślij*, strona zostanie przesłana na serwer w celu jej przetworzenia (co spowoduje wykonanie procedury obsługi `btnOrder_Click`), a następnie zostanie odesłana do klienta. W tym przypadku właściwość `IsPostBack` przyjmie wartość `true`, co sprawi, że kod umieszczony wewnątrz instrukcji `if` nie zostanie wykonany. Dzięki temu, jak pokazano to na rysunku 21.10, po ponownym wyświetleniu strony będzie na niej zaznaczony przycisk wybrany wcześniej przez użytkownika.



Rysunek 21.10. Wybór użytkownika zachowany po ponownym wyświetleniu strony

Listing 21.4 przedstawia kompletny kod ukryty obsługujący nasz przykładowy formularz.

Listing 21.4. Kod ukryty umieszczony w pliku DisplayShippers.aspx.cs

```
using System;

namespace ProgrammingCSharpWeb
{
    public partial class DisplayShippers : System.Web.UI.Page
    {
        protected void Page_Load(object sender, EventArgs e)
        {
            if (!IsPostBack)
```

```
            {
                rblShippers.SelectedIndex = 0;
            }
        }

        protected void btnOrder_Click(object sender, EventArgs e)
        {
            lblMsg.Text = "Witaj, " + txtName.Text.Trim() +
            ". Dziękujemy za zamówienie. " +
            "Wybrałeś firmę kurierską " + rblShippers.SelectedItem.Text +
            " o identyfikatorze " + rblShippers.SelectedValue + ".";
        }
    }
}
```

Podsumowanie

W tym rozdziale pokazaliśmy, jak można utworzyć prostą aplikację ASP.NET, korzystając z technologii Web Forms. Powiązaliśmy listę przycisków opcji z tabelą bazy danych i dodaliśmy kod obsługujący zdarzenia po stronie serwera zapewniający nam możliwość reagowania na czynności wykonywane na stronie przez użytkownika.

Skorowidz

B

C

D

E

F

G

H

I

J

K

L

Ł

M

N

O

P

R

S

Ś

T

U

V

W

X

Z

Ź

Programowanie

O'REILLY®

C#. Programowanie

W dzisiejszych czasach szczególną popularnością cieszą się języki programowania pozwalające na pisanie kodu łatwego do przenoszenia między platformami, ponieważ nikt nie ma czasu na pisanie kilku wersji jednej aplikacji. C# to uniwersalny język, który bez trudu spełnia ten wymóg. Dzięki swojej elastyczności, wydajności oraz mocnemu wsparciu społeczności język C# zdobył uznanie programistów.

Ten bestsellerowy podręcznik pozwoli Ci błyskawicznie poznać wszystkie niuanse języka C# 4.0. Najnowsze wydanie zostało zaktualizowane – zawiera informacje o nowościach w C# oraz opis tego języka i platformy .NET. W trakcie lektury nauczysz się tworzyć skomplikowane programy przy użyciu technik programowania obiektowego i funkcjonalnego. Ponadto sprawdzisz, jaki potencjał kryje język zapytań LINQ oraz jak przesyłać komunikaty za pomocą Windows Communication Foundation (WCF). Dodatkowo poznasz możliwości C# w zakresie tworzenia aplikacji internetowych w technologii Silverlight. Nauka C# jeszcze nigdy nie była tak przyjemna!

- Pisz złożone programy z użyciem technik programowania obiektowego oraz funkcjonalnego.
- Przetwarzaj duże kolekcje danych dzięki wbudowanym w język zapytaniom LINQ.
- Komunikuj się przez sieć za pomocą Windows Communication Foundation (WCF).
- Poznaj zalety technik programowania dynamicznego, dostępnych w C# 4.0.
- Twórz interaktywne aplikacje Windows z Windows Presentation Foundation (WPF).
- Twórz bogate aplikacje internetowe z wykorzystaniem możliwości Silverlight oraz ASP.NET.

Wykorzystaj potencjał języka C#!

Ian Griffiths jest autorem kursu WPF oraz instruktorem w firmie Pluralsight, specjalizującej się w prowadzeniu kursów Microsoft .NET. Pracuje także jako niezależny konsultant. Jest współautorem książek *Windows Forms in a Nutshell*, *Mastering Visual Studio .NET* oraz *Programming WPF*, wydanych przez wydawnictwo O'Reilly.

Matthew Adams jest kierownikiem do spraw tworzenia aplikacji w firmie Digital Healthcare Ltd. oraz autorem wielu artykułów i publikacji dotyczących znaczenia .NET w przemyśle informatycznym.

Jesse Liberty jest starszym kierownikiem programu Microsoft Silverlight. Jest doskonale znany w środowisku jako jeden z czołowych ekspertów oraz autor bestsellerowych książek, takich jak *Programming C# 3.0* (O'Reilly) oraz *ASP.NET 3.5. Programowanie i ASP.NET 2.0 i Ajax* (Helion).

ISBN 978-83-246-3701-0
Cena 129,00 zł
9 788324 637010