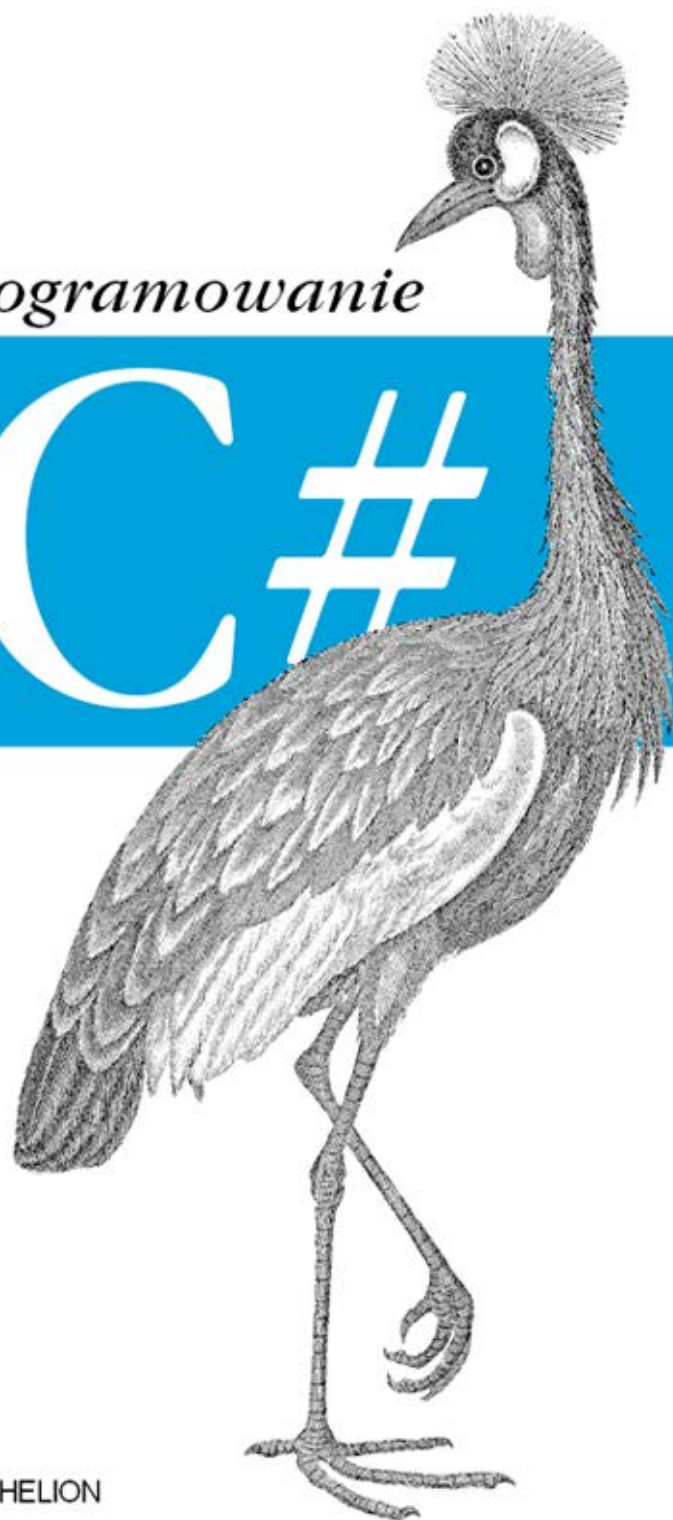


Najlepszy podręcznik poświęcony C#!

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

© Helion 2012
All rights reserved

Authorized Polish translation of the English edition of *Programming C# 4.0, 6th Edition* 9780596159832
© 2010 Ian Griffiths, Matthew Adams

This translation is published and sold by permission of O'Reilly Media, Inc., which owns or controls all rights to publish and sell the same.

All rights reserved. No part of this book may be reproduced or transmitted in any form or by any means, electronic or mechanical, including photocopying, recording or by any information storage retrieval system, without permission from the Publisher.

Wszelkie prawa zastrzeżone. Nieautoryzowane rozpowszechnianie całości lub fragmentu niniejszej publikacji w jakiegokolwiek postaci jest zabronione. Wykonywanie kopii metodą kserograficzną, fotograficzną, a także kopiowanie książki na nośniku filmowym, magnetycznym lub innym powoduje naruszenie praw autorskich niniejszej publikacji.

Wszystkie znaki występujące w tekście są zastrzeżonymi znakami firmowymi bądź towarowymi ich właścicieli.

Autor oraz Wydawnictwo HELION dołożyli wszelkich starań, by zawarte w tej książce informacje były kompletne i rzetelne. Nie biorą jednak żadnej odpowiedzialności ani za ich wykorzystanie, ani za związane z tym ewentualne naruszenie praw patentowych lub autorskich. Autor oraz Wydawnictwo HELION nie ponoszą również żadnej odpowiedzialności za ewentualne szkody wynikłe z wykorzystania informacji zawartych w książce.

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.

• Kup książkę
• Poleć książkę
• Oceń książkę

• Księgarnia internetowa
• Lubię to! » [Nasza społeczność](#)

Spis treści

Wstęp	13
1. Prezentacja C#	19
Dlaczego C#? Dlaczego .NET?	19
Biblioteka klas platformy .NET	20
Styl języka	21
Łatwość konstruowania oprogramowania	22
Kod zarządzany	23
Ciągłość i „ekosystem” Windows	24
C# 4.0, .NET 4.0 oraz Visual Studio 2010	25
Podsumowanie	27
2. Podstawowe techniki programowania	29
Początki	29
Przestrzenie nazw i typy	32
Projekty i solucje	37
Komentarze, regiony oraz czytelność	42
Nieprawidłowe komentarze	43
Komentarze dokumentujące XML	44
Zmienne	45
Typy zmiennych	46
Wyrażenia i instrukcje	52
Instrukcje przypisania	55
Operatory inkrementacji i dekrementacji	55
Instrukcje sterowania przepływem i wyboru	56
Instrukcje if	58
Instrukcje switch oraz case	62
Instrukcje iteracji	64
Instrukcje foreach	65
Instrukcje for	67
Instrukcje while oraz do	69
Przerywanie wykonywania pętli	70

Metody	71
Podsumowanie	74
3. Wyodrębnianie idei przy wykorzystaniu klas i struktur	77
Dziel i rządź	77
Wyodrębnianie idei w formie metod	77
Wyodrębnianie idei przy użyciu obiektów i klas	80
Definiowanie klas	81
Reprezentowanie stanu przy użyciu właściwości	82
Poziomy ochrony	84
Inicjalizacja przy użyciu konstruktora	86
Pola: miejsca do zapisywania danych	90
Pola mogą się zmieniać, lecz stałe nie	92
Pola i właściwości tylko do odczytu	93
Typ enum — powiązane ze sobą stałe	96
Typy wartościowe i referencyjne	100
Zbyt wiele konstruktorów, Panie Mozart	105
Przeciążanie	105
Metody przeciążone oraz domyślne parametry nazwane	106
Inicjalizatory obiektów	108
Definiowanie metod	112
Deklarowanie metod statycznych	115
Pola i właściwości statyczne	116
Konstruktory statyczne	117
Podsumowanie	119
4. Rozszerzalność i polimorfizm	121
Tworzenie asocjacji poprzez kompozycję i agregację	122
Dziedziczenie i polimorfizm	124
Zastępowanie metod w klasach pochodnych	126
Ukrywanie składowych klasy bazowej przy użyciu new	127
Zastępowanie metod przy użyciu modyfikatorów virtual i override	129
Dziedziczenie i ochrona	132
Wywoływanie metod klasy bazowej	134
Dotąd i ani kroku dalej: modyfikator sealed	136
Wymuszanie przesłaniania — metody abstrakcyjne	138
Wszystkie typy dziedziczą po klasie Object	144
Pakowanie i rozpakowywanie typów wartościowych	144
C# nie obsługuje wielokrotnego dziedziczenia implementacji	149
C# obsługuje wielokrotne dziedziczenie interfejsów	149
Tworzenie jednych interfejsów na bazie innych	152
Jawna implementacja interfejsów	153
Ostateczne rozwiązanie: sprawdzanie typów podczas wykonywania programu	157
Podsumowanie	158

5. Delegacje — łatwość komponowania i rozszerzalność	159
Kompozycja funkcyjna wykorzystująca delegacje	166
Typ Action<T> — akcje ogólne	172
Predicate<T> — predykaty ogólne	175
Stosowanie metod anonimowych	177
Tworzenie delegacji przy użyciu wyrażeń lambda	178
Delegacje we właściwościach	180
Ogólne delegacje do funkcji	182
Informowanie klientów za pomocą zdarzeń	186
Udostępnianie dużej liczby zdarzeń	194
Podsumowanie	197
6. Obsługa błędów	199
Kiedy i jak uznać niepowodzenie	204
Zwracanie kodu błędu	207
Debugowanie wartości zwracanych	213
Wyjątki	214
Obsługa wyjątków	219
Kiedy są wykonywane bloki finally?	226
Określanie, jakie wyjątki będą przechwytywane	227
Wyjątki niestandardowe	230
Podsumowanie	232
7. Tablice i listy	233
Tablice	233
Tworzenie i inicjalizacja	234
Własne typy w tablicach	237
Składowe tablic	242
Wielkość tablic	247
List<T>	254
Niestandardowe indeksatory	257
Wyszukiwanie i sortowanie	264
Kolekcje i polimorfizm	264
Tworzenie własnych implementacji IEnumerable<T>	268
Podsumowanie	274
8. LINQ	275
Wyrażenia zapytań	275
Wyrażenia zapytań a wywołania metod	277
Metody rozszerzeń a LINQ	278
Klauzule let	280
Koncepcje i techniki LINQ	281
Delegacje i wyrażenia lambda	281
Styl funkcyjny i kompozycja	283
Wykonywanie opóźnione	284

Operatory LINQ	285
Filtrowanie	285
Porządkowanie	286
Konkatenacja	289
Grupowanie	289
Projekcje	291
Spinanie	298
Robimy się wybredni	299
Testowanie całej kolekcji	300
Agregacja	302
Operacje na zbiorach	304
Łączenie	304
Konwersje	305
Podsumowanie	306
9. Klasy kolekcji	307
Słowniki	307
Popularne zastosowania słowników	309
IDictionary<TKey, TValue>	315
Słowniki i LINQ	317
HashSet oraz SortedSet	318
Kolejki	319
Listy połączone	320
Stosy	321
Podsumowanie	322
10. Łańcuchy znaków	323
Czym są łańcuchy znaków?	324
Typy String i Char	325
Literały łańcuchowe i znakowe	326
Oznaczenie znaków specjalnych	327
Formatowanie wyświetlanych danych	330
Standardowe łańcuchy formatowania liczb	331
Niestandardowe łańcuchy formatujące	337
Daty i godziny	340
W drugą stronę — konwersja łańcuchów na dane innych typów	343
Złożone formatowanie przy użyciu metody String.Format	345
Wrażliwość na ustawienia kulturowe	346
Poznawanie reguł formatowania	348
Uzyskiwanie dostępu do znaków na podstawie indeksów	349
Łańcuchy znaków są niezmiennie	349
Pobieranie ciągu znaków	351
Składanie łańcuchów znaków	352
Ponowne dzielenie łańcuchów	354
Wielkie i małe litery	355

Operacje na tekście	356
StringBuilder — modyfikowalne łańcuchy znaków	357
Odnajdywanie i zastępowanie łańcuchów	361
Wszelkiego typu „puste” łańcuchy znaków	362
Usuwanie białych znaków	365
Sprawdzanie typu znaków	368
Kodowanie znaków	368
Dlaczego kodowanie ma znaczenie	370
Kodowanie i dekodowanie	371
Po co reprezentować łańcuchy w formie sekwencji bajtów?	378
Podsumowanie	378
11. Pliki i strumienie	379
Sprawdzanie katalogów i plików	379
Badanie katalogów	382
Operacje na ścieżkach	383
Ścieżka i aktualny katalog roboczy	384
Zdobywanie informacji o pliku	385
Tworzenie plików tymczasowych	388
Usuwanie plików	389
Powszechnie znane katalogi	390
Bezpieczne łączenie elementów ścieżek	393
Tworzenie i zabezpieczanie hierarchii katalogów	394
Usuwanie katalogu	401
Zapis plików tekstowych	402
Zapis całego pliku tekstowego w jednym wywołaniu	402
Zapis tekstu przy użyciu klasy StreamWriter	403
Gdy pliki schodzą na złą drogę: obsługa wyjątków	406
Określanie i modyfikacja uprawnień	410
Wczytywanie plików do pamięci	414
Strumienie	418
Poruszanie się wewnątrz strumienia	424
Zapis danych przy użyciu strumieni	425
Odczyt, zapis i blokowanie plików	426
Konstruktory klasy FileStream	428
Bufory strumieni	428
Określanie uprawnień podczas tworzenia strumieni	429
Opcje zaawansowane	429
Asynchroniczne operacje na plikach	430
Mechanizm Isolated Storage	433
Magazyny	434
Zapis i odczyt tekstu	435
Definicja izolowania	436

Zarządzanie magazynami użytkownika przy użyciu limitów	440
Zarządzanie magazynami	441
Strumienie, które nie są plikami	444
Strumień adaptujący — CryptoStream	447
Wszystko w pamięci — MemoryStream	448
Reprezentowanie danych binarnych jako tekstu przy użyciu kodowania Base64	449
Podsumowanie	452
12. XML	453
Podstawy XML (krótki przegląd)	453
Elementy	453
XHTML	455
Litera „X” oznacza „rozszerzalny” (eXtensible)	456
Tworzenie dokumentów XML	456
Elementy XML	459
Atrybuty XML	460
Umieszczanie kodu LINQ w LINQ to XML	463
Przeszukiwanie kodu XML za pomocą LINQ	464
Wyszukiwanie pojedynczego węzła	467
Osie wyszukiwania	468
Klauzule where	469
Serializacja XML	469
Dostosowywanie serializacji XML za pomocą atrybutów	472
Podsumowanie	473
13. Sieci	475
Wybór technologii sieciowej	475
Aplikacja WWW z kodem klienta	476
Klient .NET i serwer .NET	480
Klient .NET i usługa WWW pochodząca z zewnątrz	482
Klient zewnętrzny i usługa WWW .NET	483
Platforma WCF	483
Tworzenie projektu WCF	483
Kontrakty WCF	484
Testowy host i klient WCF	486
Udostępnianie usługi WCF	489
Pisanie klienta WCF	496
Dwukierunkowa komunikacja z dwustronnymi kontraktami	504
Protokół HTTP	513
Klient WWW	514
Klasy WebRequest i WebResponse	518
Gniazda	525
Protokoły IP, IPv6 oraz TCP	526
Łączenie się z usługami za pomocą klasy Socket	531
Implementowanie usług za pomocą klasy Socket	535

Inne możliwości związane z siecią	540
Podsumowanie	540
14. Bazy danych	541
Krajobraz możliwości dostępu do danych w ramach platformy .NET	541
Klasyczny mechanizm ADO.NET	542
LINQ i bazy danych	546
Technologie dostępu do danych nieopracowane przez firmę Microsoft	548
WCF Data Services	548
Technologia Silverlight i dostęp do danych	549
Bazy danych	550
Model encji danych	551
Wygenerowany kod	555
Zmiana odwzorowywania	557
Związki	558
Dziedziczenie	565
Zapytania	566
LINQ to Entities	566
Entity SQL	570
Mieszanie języków ESQL oraz LINQ	573
Dostawca ADO.NET EntityClient	574
Kontekst obiektu	574
Obsługa połączenia	574
Tworzenie, aktualizowanie i usuwanie	577
Transakcje	579
Optymistyczna współbieżność	584
Czas życia kontekstu i encji	586
WCF Data Services	587
Podsumowanie	591
15. Podzespoły	593
Komponenty .NET — podzespoły	593
Odwołania	594
Pisanie bibliotek	597
Ochrona	599
Nazwy	602
Podpisywanie i silne nazwy	603
Ładowanie	605
Ładowanie z folderu aplikacji	606
Ładowanie z bufora GAC	606
Ładowanie z pliku Silverlight o rozszerzeniu xap	607
Jawne ładowanie	607
Podsumowanie	609

16. Wątki i kod asynchroniczny	611
Wątki	613
Wątki i systemowy mechanizm szeregujący	615
Stos	617
Pula wątków	624
Powinowactwo oraz kontekst wątków	625
Popularne błędne opinie dotyczące wątków	627
Tworzenie kodu wielowątkowego jest trudne	634
Strategie tworzenia kodu wielowątkowego	637
Podstawowe narzędzia synchronizacji	638
Monitor	639
Inne typy blokad	649
Inne mechanizmy synchronizacji	653
Zdarzenia	653
Odliczanie	654
Programowanie asynchroniczne	655
Model programowania asynchronicznego	656
Programowanie asynchroniczne bazujące na zdarzeniach	659
Doraźne operacje asynchroniczne	660
Task Parallel Library	661
Zadania	661
Obsługa anulowania	668
Obsługa błędów	669
Równoległość danych	671
Metody Parallel.For oraz Parallel.ForEach	671
PLINQ — równoległe LINQ	673
Podsumowanie	674
17. Atrybuty i odzwierciedlanie	675
Atrybuty	675
Typy atrybutów	676
Własne atrybuty	677
Odzwierciedlanie	681
Badanie metadanych	681
Odkrywanie typów	683
Odzwierciedlanie na rzecz określonego typu	684
Późne wiązanie	686
Podsumowanie	689
18. Typ dynamic	691
Styl statyczny kontra styl dynamiczny	691
Styl dynamiczny i automatyzacja COM	693
Typ dynamic	694
Typy obiektów i słowo dynamic	697
Typ dynamic w zastosowaniach niezwiązanych z interoperacyjnością?	707
Podsumowanie	710

19. Współdziałanie z COM i Win32	711
Importowanie kontrolek ActiveX	711
Importowanie kontrolek do projektów .NET	712
Podzespoły współdziałania	714
Bez PIA	716
64 czy 32 bity?	717
Mechanizm P/Invoke	720
Wskaźniki	724
Rozszerzenia składni C# 4.0	729
Właściwości indeksowane	729
Opcjonalny modyfikator ref	730
Podsumowanie	731
20. WPF i Silverlight	733
XAML i kod ukryty	735
XAML i obiekty	739
Elementy i kontrolki	742
Panele układów	743
Elementy graficzne	752
Kontrolki	759
Kontrolki użytkownika	763
Szablony kontrolek	765
Style	767
Menedżer stanu wizualnego	769
Wiązanie danych	771
Szablony danych	773
Podsumowanie	776
21. Tworzenie aplikacji w ASP.NET	777
Podstawy technologii Web Forms	777
Zdarzenia formularzy sieciowych	779
Cykl życia stron w technologii Web Forms	780
Tworzenie aplikacji internetowych	781
Pliki kodu ukrytego	782
Dodawanie kontrolek	783
Kontrolki serwerowe	785
Wiązanie danych	786
Sprawdzanie kodu	789
Dodawanie kontrolek i formularzy	792
Podsumowanie	796
22. Windows Forms	797
Tworzenie aplikacji	798
Dodawanie źródła wiązania	799

Kontrolki	801
Dokowanie i kotwiczenie	806
Wiązanie danych	808
Obsługa zdarzeń	813
Podsumowanie	814
Skorowidz	817

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 pisanie 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.

Koniecznym 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 kontrolki 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 kontrolki 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 uwzględnić 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 obsługiwanych 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 obsługiwane 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 `AutoPostBack` ↪ 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 kontrolki 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 żądaniem 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 kontrolki 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 kontrolki 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 jakiegokolwiek różnice pomiędzy poprzednim i bieżącym stanem, to zostaje wywołana metoda `RaisePostDataChangedEvent`, która powoduje zgłoszenie odpowiednich zdarzeń.

Obsługa zdarzeń

Na tym etapie zostaje obsługane zdarzenie (zgłoszone po stronie klienta), które spowodowało przesłanie żądania na serwer.

Generowanie wstępne

To ostatni moment, by zmienić właściwości kontrolki, 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 „łżejsze” — 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 zobaczymy 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 jakiegokolwiek 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 założymy, 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:

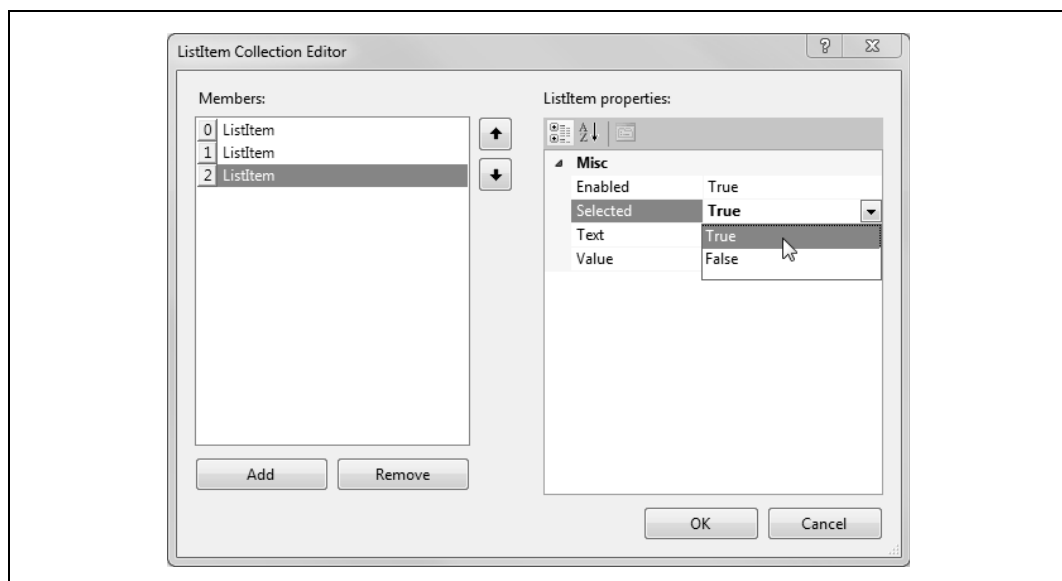
```
<asp:RadioButton GroupName="Shipper" id="Speedy"
  text="Speedy Express" Checked="True" runat="server">
</asp:RadioButton>
<asp:RadioButton GroupName="Shipper" id="United"
  text="United Package" runat="server">
</asp:RadioButton>
<asp:RadioButton GroupName="Shipper" id="Federal"
  text="Federal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

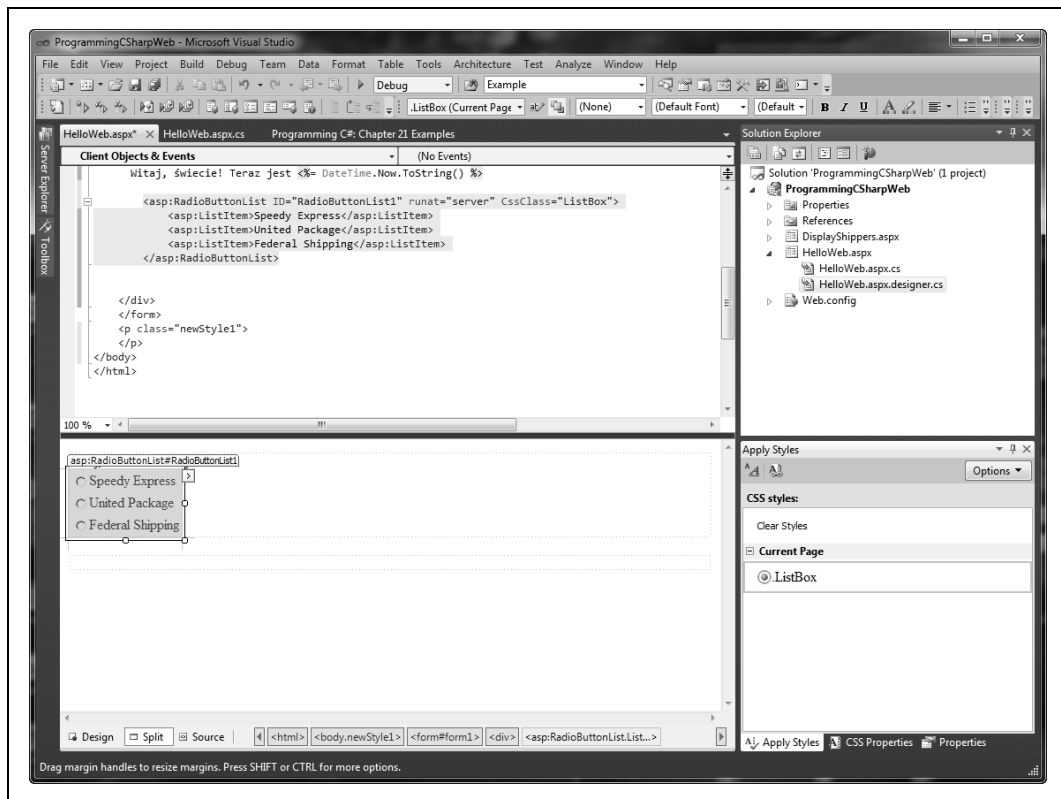


Rysunek 21.1. Kolekcja elementów `ListItem`

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 (przedstawione 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 kontrolki 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 kontrolki 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 kontrolki 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 kontrolki 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 kontrolki ASP.NET można powiązać z danymi, co znacznie ułatwia prezentowanie tych danych oraz ich modyfikację.

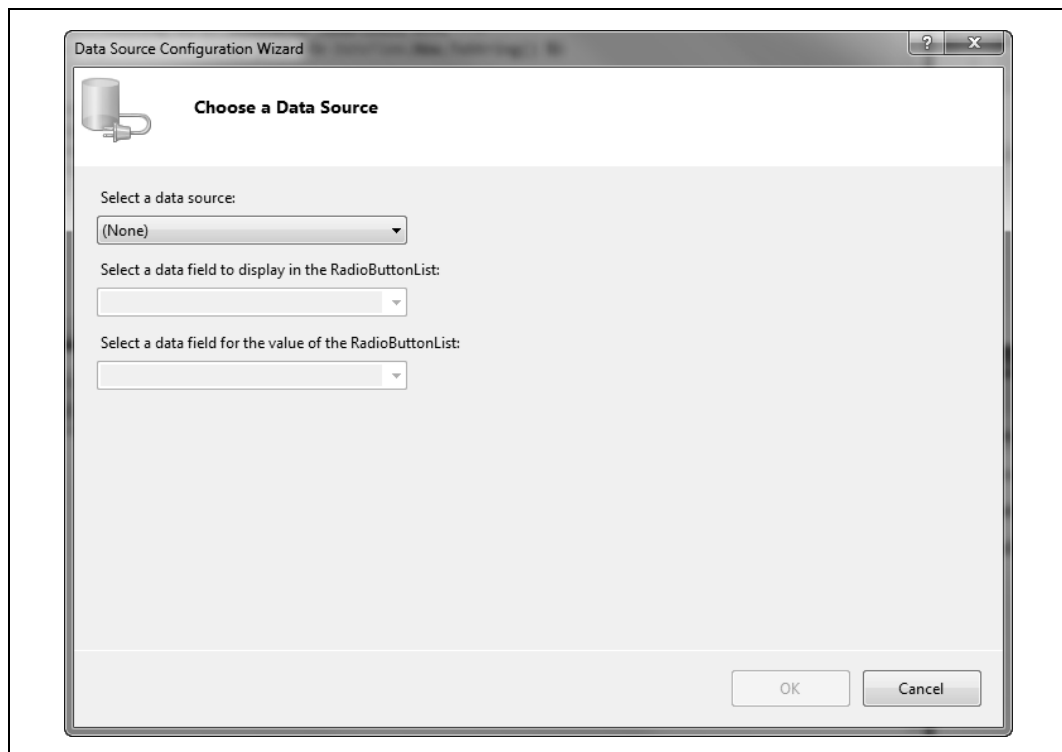
W poprzednim podrozdziale na stałe umieściliśmy na formularzu trzy przyciski opcji, po jednym dla każdej z trzech firm kurierskich, które może wybrać użytkownik. Jednak to nie jest najlepsze rozwiązanie — dostawcy się zmieniają, a poza tym istnieje duże prawdopodobieństwo, że w przyszłości będziemy chcieli nawiązać współpracę także z innymi firmami kurierskimi. Nie chcemy, by każda zmiana relacji biznesowych zmuszała nas do ręcznego modyfikowania tych kontrolki. 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.



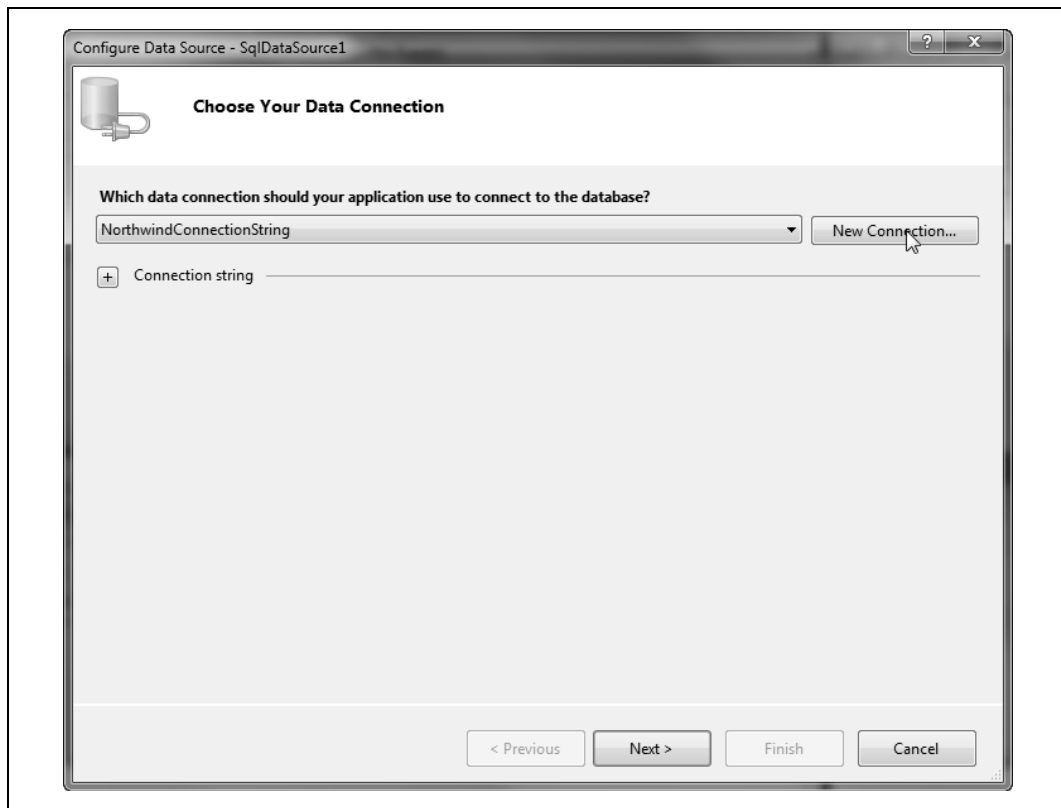
Rysunek 21.3. Okno dialogowe *Data Source Configuration Wizard*

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

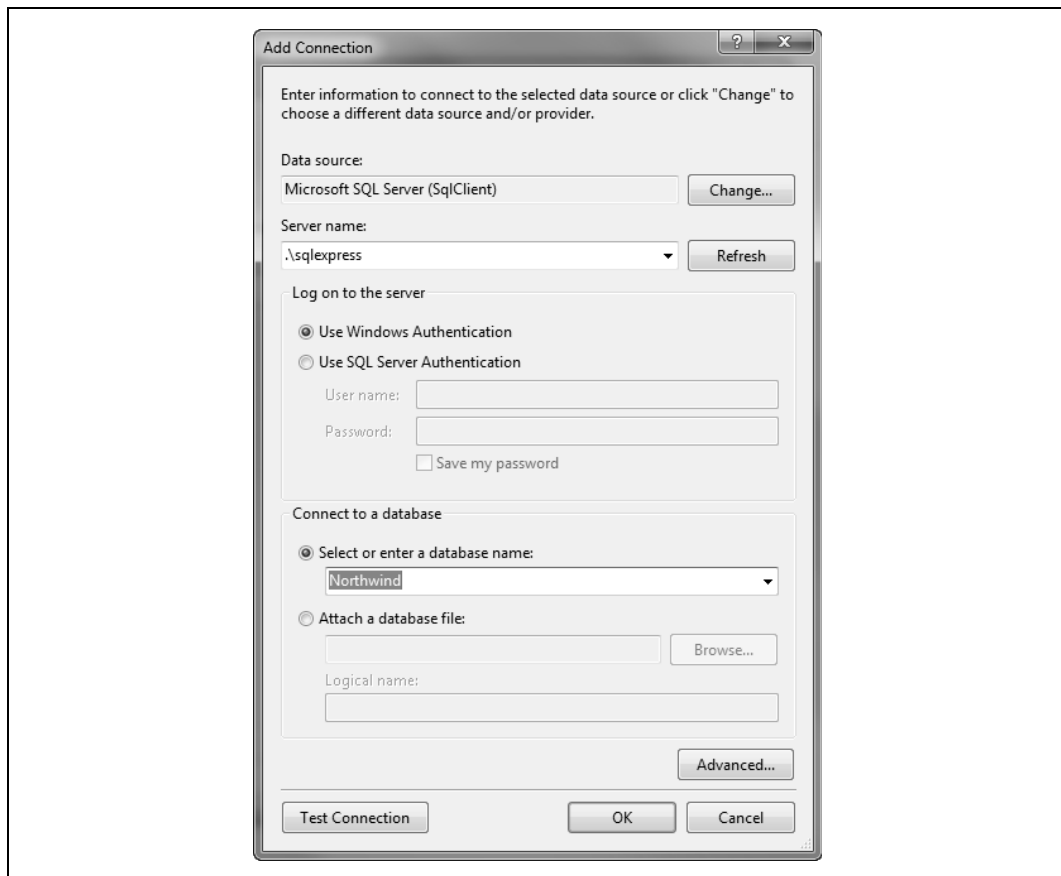
można kliknąć przycisk *Next*. Jeśli chcemy zapisać połączenie w pliku konfiguracyjnym *web.config*, to na kolejnej stronie kreatora powinniśmy podać jego nazwę (np. `NorthWindConnectionString`).

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 `RadioButtonList`. 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

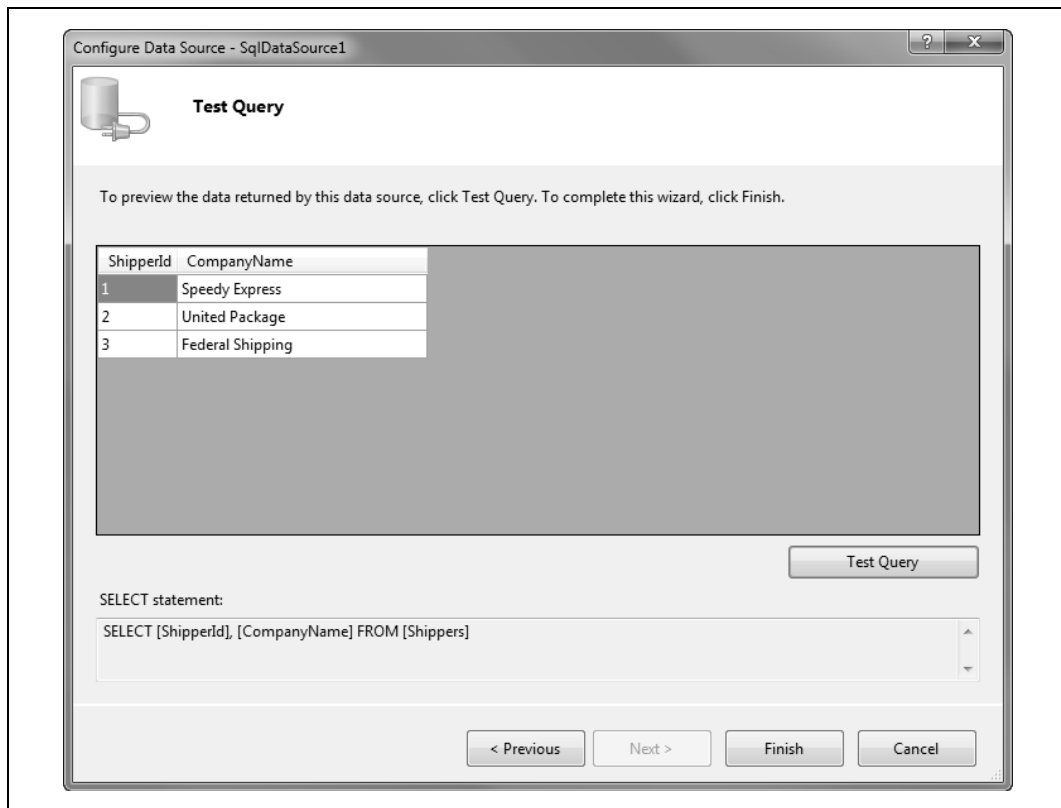
Zanim przejdziemy dalej, należy zwrócić uwagę na kilka zagadnień. Kiedy naciśniemy klawisz *F5*, by uruchomić aplikację, zostanie ona wyświetlona w przeglądarce, a strona, zgodnie z oczekiwaniami, będzie prezentowała grupę przycisków opcji. Możemy teraz wybrać opcję *Choose View/Source*, a przekonamy 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 21.7. Testowanie zapytania

```

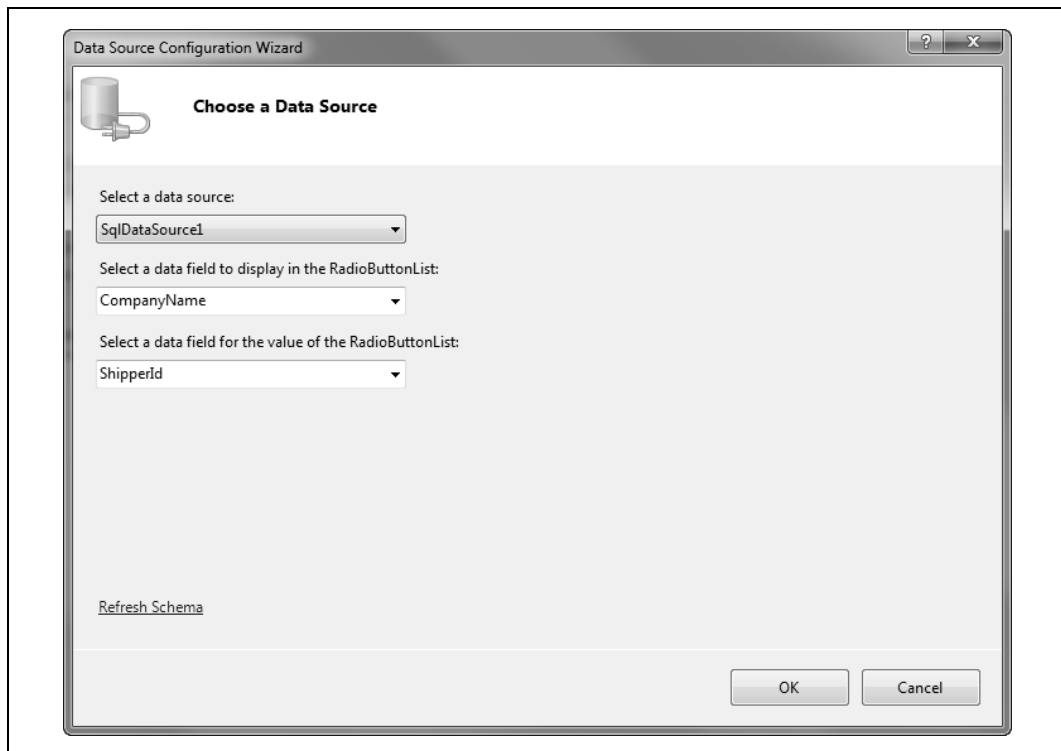
</table>
</div>
</form>
</body>
</html>

```

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

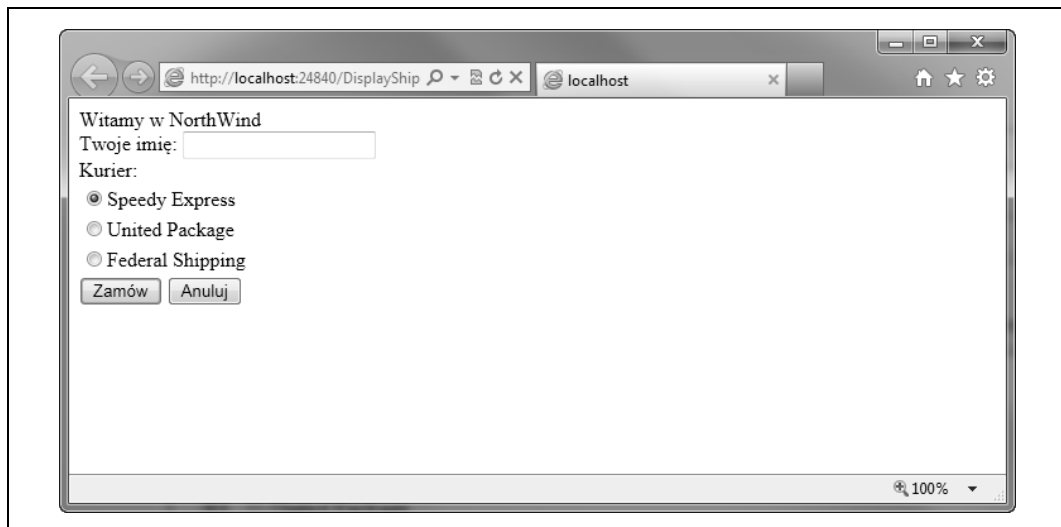
Dodawanie kontrolek i formularzy

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 Usability* 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="rb1Shippers" 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;
}

```

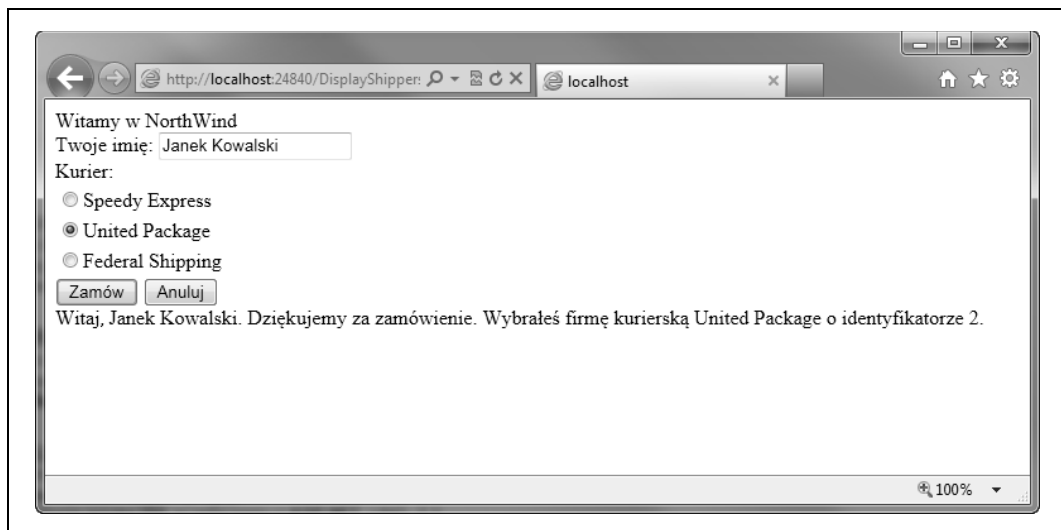
Zastosowanie powyższego kodu sprawi, że zostanie zaznaczony pierwszy przycisk opcji w kontrolce `RadioButtonList`. W powyższym rozwiązaniu występuje jeden dosyć subtelny problem. Otóż kiedy uruchomimy aplikację, zostanie zaznaczony pierwszy przycisk, a jeśli następnie wybierzemy drugi lub trzeci i klikniemy *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.

.NET 4, 16

A

abstrakcyjne klasy bazowe, 138

ACL, access control list, 493

Adams Matthew, 819

ADO.NET, 542

- dostęp do danych, 543

- zbiory danych, 545

ADO.NET EntityClient, 574

adres, 491

adres bazowy, 492

adres IP, 526

adres usługi, 497

adresy IPv4, 526

adresy IPv6, 526

agregacja, 122, 301, 303

AJAX, Asynchronous JavaScript and XML, 476

akcesor

- get, 82, 260

- set, 82, 260

aktualizacja encji, 585

aktualizacje, 480

algorytm

- asymetryczny, 445

- kryptograficzny, 445

- mieszający, 604

- symetryczny, 445

analiza obiektowa, 80

animacje, 757

anulowanie operacji, 193, 669

Any CPU, 719

API, 85

API HTTP, 482

API XML, 456

aplikacja konsolowa, 31, 597

- ChatHost, 490

aplikacja

- Network Monitor, 489

- niepodpisana, 438

- podpisana, 437

- Silverlight, 698, 737

- WWW, 476

APM, Asynchronous Programming Model, 656, 658

architektura x86, 717

args, 36, 57

argumenty opcjonalne, 731

argumenty tablicowe, 237

ASCII, 369

asocjacje, associations, 122, 559

ASP.NET, 21, 489, 777

asynchroniczne metody, 518

asynchroniczne operacje wejścia-wyjścia, 430

asynchroniczne uzyskiwanie odpowiedzi, 520

atak typu SQL injection, 544

atak typu zeroday, 505

Atom, 517

AtomPub, 588

atrybut, attribute, 675

- [Flags], 99

- address, 492

- AttributeUsage, 678

- baseAddress, 492

- DefaultParameterValueAttribute, 109

- DllImport, 721

- Namespace, 498

- OptionalAttribute, 109

- PublicKeyToken, 603

- XmlIgnoreAttribute, 473

atrybuty

- konstruowanie, 678

- możliwe cele, 676

- nazywanie, 678

- podzespołów, 677

- używanie, 679

atrybuty
własne, 677
XML, 460, 740
automatyzacja COM, COM automation, 25, 693, 697
automatyzacja programu Word, 693

B

bardziej znaczący bajt, 376
baza danych, 547, 550
bezpieczeństwo, 493
bezpieczeństwo wątków, 508
białe znaki, whitespace, 83, 277, 364
biblioteka, 37
.NET Framework, 20, 35, 182
ChatServerLibrary, 490
kernel32.dll, 720
mscorlib, 40, 604
Task Parallel Library, 623, 625, 660
TPL, 667
biblioteki
klas, Class Library, 20, 597
kontrolki i XAML, 763
typu, type library, 713
bieżący katalog roboczy, 384
BigInteger, 49
BlockingCollection, 654
blok case, 63
blok catch, 222
blok finally, 226
blokada, 645
blokada hierarchiczna, lock leveling, 645
blokada referencji this, 643
blokada SpinLock, 650
blokady odczytu i zapisu, 651
błąd BadImageFormatException, 718
błędy
programistyczne, 700
sprawdzone jawnie, 216
typograficzne, 362
występujące najczęściej, 200
buforowanie, 428

C

C# 4.0, 16
camelCase, 83
camelCasing, 113
cecha, feature, 664
chronione klasy wirtualne, 187
ciasteczka, cookies, 506, 524
CLI, Common Language Infrastructure, 24, 104

CLR, Common Language Runtime, 21, 23
COM, Component Object Model, 626, 712, 715
CPU, central processing unit, 616
CRUD, Create, Read, Update, and Delete, 577
Ctrl+Alt+C, 213
Ctrl+D, 805
Ctrl+F5, 31, 226
Ctrl+O, 805
Ctrl+Shift+A, 64
Ctrl+Shift+N, 29
Ctrl+T, 805
cykl życia strony, 780
czas bieżący, 538
czas UTC, 534
czas życia kontekstu, 586
czcionka, font, 324
czyszczenie danych, 357

D

DDoS, 538
debugger, 67, 213
debugowanie, 500
debugowanie programów wielowątkowych, 621
debugowanie wartości zwracanych, 213
defekt, bug, 199
definicja
EDM, 575
interfejsu ISet<T>, 318
izolowania, 436
operatora Where, 570
definiowanie
interfejsów, 149
klas, 35, 81
konstruktora, 86
kontraktu, 484
metod, 112
typów, 35
własnych atrybutów, 678
własnych znaczników, 456
deklaracja pola, 91
deklaracja typu wyliczeniowego, 97
deklaracja właściwości, 182
deklarowanie metod statycznych, 115
deklarowanie typów delegacji, 167
deklarowanie zmiennej tablicowej, 236
dekodowanie, 371
dekompiletor, 734
dekompozycja funkcjonalna, 79
delegacja, 166, 180
Action, 174
anonimowa, 177
Check, 175

- DocumentProcess, 169
- EventHandler, 186
- QuickCheck, 175
- delegacje do funkcji, 182
- delegacje i zmienne, 167
- delegacje predykatów, 243
- delegacje we właściwościach, 180
- deserializacja XML, 469
- deszyfracja, 444, 450
- Dijkstra Edsger, 71
- dll, dynamic link library, 37
- DLR, Dynamic Language Runtime, 704
- długość transakcji, 582
- DNS, Domain Name Service, 526
- dodawanie
 - blokowania do klasy, 644
 - elementów, 256
 - encji, 578
 - formularzy, 792
 - komponentu COM, 712
 - kontrolek, 783, 792
 - metody rozszerzenia, 279
 - odwołania, 595
 - odwołania do usługi, 498
 - właściwości, 82, 87
- dokument RFC867, 530
- dokumentacja biblioteki .NET, 41
- dokumentacja MSDN, 101, 227, 659
- DOM, Document Object Model, 456
- domena aplikacji, appdomain, 231, 609
- domknięcie, closure, 184
- dostawca bazy danych LINQ, 547, 570
- dostęp do danych, 542, 550
- dostęp do encji, 574
- dostęp do klasy, 600
- dostęp do łańcuchów połączeń, 576
- dostęp do współdzielonych danych, 583
- Double.NaN, 205
- Double.NegativeInfinity, 205
- Double.PositiveInfinity, 205
- drzewa wyrażeń, expression trees, 179, 571
- dwustronne kontrakty, duplex contracts, 504
- dwustronny klient, 511
- dynamiczne wywoływanie metody, 688
- dyrektywa using, 32
- dyrektywa using System.Linq, 276
- dyski SSD, Solid State Driver, 629
- działanie równoległe, parallel execution, 611
- dzieci, child elements, 454
- dziedziczenie, 124, 132, 565
- dziedzina aplikacji, 609
- dziel i rządź, 77
- dzielenie całkowite, 53

- dzielenie łańcucha znaków, 354
- dzielenie przez zero, 204
- dzielenie zmiennoprzecinkowe, 53

E

- EDM, Entity Data Model, 552
- element, 742
 - behavior, 494
 - ContentPresenter, 767
 - endpoint, 492, 502
 - główny, root element, 454
 - host, 492
 - Image, 756
 - MediaElement, 756
 - Path, 752
 - PostNote, 487
 - przodka, 454
 - serviceDebug, 494
 - właściwości, property element, 743
- elementy, 453
 - formatu, 345
 - graficzne, 752
 - interfejsu, 739
 - potomne, 454
 - XML, 459
- encja SalesOrderHeader, 573
- encje, 553
- Entity Framework, 541, 551, 570
 - modele, 551
 - odwzorowywanie, 551
- enumeratory, 265
- ESQL, Entity SQL, 570
- exe, executable, 37

F

- filtr, 708
- filtr dynamiczny, 708
- filtrowanie, 285
- filtrowanie za pomocą LINQ, 708
- finalizacja, 670
- flaga FileOptions.Asynchronous, 432
- flaga FileOptions.None, 432
- format
 - big-endian, 376
 - daty, 341
 - dziesiętny, 332
 - JPEG, 757
 - JSON, 588
 - liczbowy, 335
 - liczbowy niestandardowy, 337
 - ogólny, 334

- format
 - PNG, 757
 - powrotny, round trip, 336
 - procentowy, 335
 - stałoprzecinkowy, 334
 - szesnastkowy, 332
 - uniwersalny, 342
 - wykładniczy, 333
 - wykładniczy bez określonej precyzji, 334
- formatowanie, 348
- formatowanie danych, 330
- formatowanie wartości walutowych, 331
- formularz, 806
- formularz z powiązаныmi kontrolkami, 810
- funkcja, 78, 178
 - CreateCustomerList, 458
 - CreateFile, 725
 - get_NazwaWlasciwosci, 147
 - MoveFile, 720
 - ReadFile, 726
 - set_NazwaWlasciwosci, 147
- funkcje
 - JavaScript, 699
 - mieszające, 308
 - MIME, 540
- funkcyjne języki programowania, 283

G

- GAC, global assembly cache, 606
- garbage collector, 20
- generator liczb losowych, 446
- generowanie podpisu cyfrowego, 604
- globalna przestrzeń nazw, 34
- globalny bufor podzespołów, 606
- główny podzespół współdziałania, 27, 716
- gniazda, 480, 505, 525
- gniazda działające w podwójnym trybie, 535
- gniazda TCP, 530, 533
- gniazda układu, 747
- gradient, 754
- grafika trójwymiarowa, 758
- Griffiths Ian, 819
- grupowanie, 289
- grupowanie znaków w pary, 370
- gubienie pakietów, 529

H

- hash publicznego klucza, 604
- hermetyzacja metody w obiekcie, 166
- hierarchia elementów, 453
- hierarchia katalogów, 394

- hierarchia klas, 138, 147
- hiperprostokąt, orthotope, 251
- host, 490
- host usługi, 493

I

- IANA, Internet Assigned Numbers Authority, 530
- IL, Intermediate Language, 23, 38
- implementacja
 - interfejsu, 150, 153
 - interfejsu IClockIn, 156
 - interfejsu IEnumerable<T>, 268
 - operatorów LINQ, 280
- importowanie kontrolek, 712
- indeks, 234
- indeksator, 349
- indeksator klasy String, 349
- indeksator niestandardowy, 257
- informacje o błędach, 208, 216
- informacje o pliku, 381, 385
- informacje o typie anonimowym, 295
- informacje o zdarzeniach, 195
- infrastruktura TLS/SSL, 540
- inicjalizator obiektu, 108, 110
- inicjalizator pola, 92
- inicjalizator tablicy, 236
- inicjowanie kolekcji, 236
- inkrementacja, 635
- instrukcja
 - fixed, 728
 - for, 67
 - foreach, 65
 - goto, 63
 - if, 58
 - if ... else, 61
 - switch oraz case, 62
 - throw, 215
 - try-catch, 221
 - while oraz do, 69
 - yield return, 268, 270
- instrukcje
 - iteracji, 64
 - przypisania, 55
 - sterowania przepływem, 56
- integracja systemów, 482
- inteligentny znacznik, smart tag, 784
- IntelliSense, 180
- interfejs
 - IChatService, 498
 - IClockIn, 156
 - ICryptoTransform, 447
 - IDataReader, 542

- IDictionary, 316
- IEnumerable<T>, 265, 268, 317
- IEnumerator<T>, 265
- IFormatProvider, 348
- INamedPerson, 151
- INotifyPropertyChanged, 772
- klienta, 504
- komunikacji zwrotnej, 512
- interoperacyjność, 707
- IP, Internet Protocol, 526
- iterowanie dynamicznych właściwości, 705
- izolowanie, 436, 439, 440

J

- jawna implementacja interfejsu, 153
- jawne odwołanie, 113
- jednowątkowy kod asynchroniczny, 660
- język
 - C#, 13
 - Eiffel, 215
 - ESQL, 571
 - F#, 22
 - IronPython, 705
 - IronRuby, 705
 - JavaScript, 741
 - kompilowany, 38
 - LINQ, 275
 - maszynowy, 38
 - ML, 22
 - obiektowy, 35, 80
 - pośredni, Intermediate Language, 24, 38
 - UML, 123
 - VB.NET, 20
 - WSDL, 485
 - XAML, 735
 - XHTML, 455
 - XML, 453
 - XPath, 468
- JIT, just in time, 717
- JSON, JavaScript Object Notation, 476

K

- kanał alfa, 754
- karta .NET, 595
- karta Browse, 596
- karta COM, 596
- karta Debug, 58
- karta Projects, 599
- kaskadowe operacje usuwania, 579
- catalog mechanizmu Isolated Storage, 443

- catalogi
 - kontrola dostępu, 395
 - prawa dostępu, 396
 - usuwanie rekurencyjne, 405
 - specjalne, 390
 - testowe, 393
- klasa, 80
 - abstrakcyjna, 139
 - Activator, 687
 - Administrator, 142, 146
 - APIFileReader, 725
 - ApplicationException, 231
 - argumentów zdarzenia, 190
 - AutoResetEvent, 653
 - AxHost, 714
 - bazowa, 124, 140
 - bazowa Array, 242
 - BindingList, 813
 - BindingSource, 799
 - CalendarEvent, 243, 292
 - CancelEventArgs, 192
 - CancellationTokenSource, 668
 - ChatServiceClient, 500
 - ChatServiceProxy, 507
 - ConfigurationManager, 576
 - Console, 33, 36, 354
 - ContentControl, 759
 - Control, 742, 765, 801, 802
 - CountdownEvent, 654
 - CredentialCache, 521, 522
 - CryptoStream, 444
 - CultureInfo, 346
 - Customer, 458
 - DataSet, 545
 - DataTable, 545
 - DbException, 230
 - Delegate, 166, 168
 - Directory, 382, 384
 - Dns, 658
 - Document, 159
 - DocumentProcessor, 162, 176, 195
 - Encoding, 372
 - Enumerable, 279
 - Environment, 117, 390
 - Exception, 215
 - ExpandableObject, 705
 - File, 277
 - FileContents, 415, 421
 - FileInfo, 385
 - FileStream, 420, 428
 - FileWebRequest, 518
 - FileWebResponse, 518
 - FireChief, 140

klasa

- FirefighterBase, 142
- FireStation, 145, 155
- Foo, 280
- FrameworkElement, 802
- FtpWebRequest, 518
- FtpWebResponse, 518
- GetFolderPath, 390
- HashSet<T>, 318
- HttpListener, 514
- HttpRequest, 518, 521
- HttpResponse, 518
- Image, 756
- Interlocked, 633
- InvalidOperationException, 215
- IPAddress, 526
- IsolatedStorageFile, 434, 442
- LinkedList<T>, 320
- List<T>, 254, 262, 264
- ManualResetEvent, 653
- MemoryStream, 444, 448, 450
- Monitor, 646
- Mutex, 652
- NamedPerson, 146
- NetworkCredential, 522
- NetworkStream, 533
- Object, 144
- ObjectContext, 555
- Panel, 743
- Parallel, 671
- Path, 383
- pochodna, 124
- Polyline, 752
- ProcessEventArgs, 190
- Queue<T>, 319, 321
- RecordCache, 317
- Shape, 752
- Socket, 531, 537
- SortedSet<T>, 318
- SqlConnection, 542
- Stack<T>, 321
- Stopwatch, 262
- Stream, 516
- StreamReader, 405
- StreamWriter, 403
- String, 349
- StringBuilder, 143, 357, 362
- StringComparer, 288
- SynchronizationContext, 626
- System.Console, 21
- System.Globalization.CultureInfo, 346
- System.Math, 32, 686
- Task, 662
- TaskFactory, 667
- TaskScheduler, 666
- Thread, 618
- ToDoEntry, 809
- TrademarkFilter, 170
- TraineeFirefighter, 139, 140
- TransactionScope, 581
- Turtle, 200, 223
- TurtleException, 230
- Type, 682
- Uri, 515
- VideoBrush, 757
- VisualStateManager, 769
- WaitForIt, 646
- WebClient, 514, 518
- WebProxy, 523
- WebRequest, 518
- WebResponse, 518
- XmlReader, 456
- XmlSerializer, 470
- XmlWriter, 456
- zagnieżdżona, nested class, 174

klasy bazowe ADO.NET, 543

klauzula

- from, 276, 296
- group, 277, 290, 386
- into, 290
- let, 280
- ORDER BY, 568
- orderby, 286
- select, 277
- where, 277, 469, 547
- WHERE, 568

klauzule strażnicze, guard clauses, 215

klawisz F11, 67

klawisz F5, 67

klawisz F7, 738

klawisz skrót, 805

klient .NET, 480

klient Silverlight, 478, 491

klient WCF, 496

klient zewnętrzny, 483

klienty aplikacji WWW, 477, 514

klucz, 308, 445

klucz główny, 589

klucz obcy, 558

klucz publiczny, 604

kod

- ASCII, 354, 369
- kod błędu, 207
- filtrujący, 591
- funkcyjny, 36, 283
- generujący zdarzenia, 191

- HTML, 778, 789
- JavaScript, 698
- modyfikujący informacje, 283
- niebezpieczny, 726
- polimorficzny, 264
- proceduralny, 36
- synchronizujący, 644
- ukryty, code behind, 737, 795
- uzależniony od interfejsu, 772
- wielowątkowy, 637
- wykonywany na serwerze, 778
- XAML, 738, 774
- zarządzany, managed code, 23
- zewnętrzny, 32
- kodowanie, 371
- kodowanie Base64, 449
- kodowanie Latin-1, 368
- kodowanie Mac-Roman, 369
- kodowanie tekstu, 371
- kodowanie znaków, 368
- kolejka, 319
 - FIFO, 321, 664
 - LIFO, 321, 664
- kolejkowanie elementów, 625
- kolejność
 - bloków catch, 221
 - elementów tablicy, 244
 - inicjalizatorów pól statycznych, 118
 - parametrów, 105
 - wywoływania metod, 135
- kolekcja, 264
- kolekcja BlockingCollection, 654
- kolekcja ogólna List<T>, 145
- kolekcja unikalnych wartości, 318
- kolekcje enumerowalne, 266
- kolumna dyskryminatora, 565
- kolumna rowguid, 558
- kombinacja wartości, 99
- komentarz, 43
- kompilacja, 38
- kompilator C#, 35
- kompozycja, composition, 123
- komunikacja dwukierunkowa, 503
- komunikacja pomiędzy programami, 496
- komunikat o błędzie, 84, 86, 103, 105, 125, 139, 141, 154, 221, 260, 598
- konfiguracja
 - polecenia Select, 790
 - procesora dokumentów, 165
 - proxy, 522
 - serwera, 511
- WCF, 490
 - źródła wiązania, 800
- konflikt nazw, 152
- konkatenacja, 289, 296
- konkatenacja łańcuchów znaków, 353
- konkatenacja tablicy, 353
- konstruktor, 86
 - bezargumentowy, 109
 - domyślny, 104, 109
 - klasy StringBuilder, 358
 - statyczny, 117
- kontekst obiektu, object context, 555, 574
- kontekst wątków, 625
- kontrakt, contract, 484, 492
- kontrakt dwustronny, 504
- kontrakt usługi, service contract, 484
- kontrakty WCF, 484
- kontrolka
 - ListBox, 761, 773, 802, 814
 - ListView, 802, 804, 813
 - SplitContainer, 802
 - TreeView, 762
- kontrolki, 742, 759, 801
 - ActiveX, 711, 715
 - ASP.NET, 778
 - serwerowe, 785
 - serwerowe ASP.NET, 785
 - sieciowe, web controls, 778
 - użytkownika, user controls, 762
 - Windows Forms, 797
 - z wieloma elementami, 760
 - z zawartością, 759
 - zadokowane, 806
 - zakotwiczone, 807
- kontynuacje, 665
- konwersja, 305
- konwersja danych na Base64, 450
- konwersja liczb, 330
- konwersja łańcuchów, 343
- koordynacja wątków, 646
- kopiowanie wartości, 101
- kowariantne argumenty typów, 267
- kradzież zadań, 665
- krotność, multiplicity, 563
- krój pisma, typeface, 324
- kryptografia, 445
- kryterium sortowania, 287
- krzywa Beziera, 753
- kształty, 752
- kwantyfikator egzystencjalny, 301
- kwantyfikator ogólny, 301

L

Learning WCF, 488
leniwa enumeracja, 273
leniwe kolekcje, 270
leniwe przetwarzanie, 271
Liberty Jesse, 819
liczba logicznych procesorów, 629
liczba wątków, 616, 623
liczby całkowite, 46
liczby zmiennoprzecinkowe, 49
limit, 452
LINQ, Language Integrated Query, 20, 233, 275
 generowanie elementów XML, 463
 przeszukiwanie kodu XML, 464
 tworzenie kodu XML, 463
LINQ to Entities, 544, 566
LINQ to Objects, 275, 383, 547
LINQ to SQL, 546, 548, 570
LINQ to XML, 456
Liskov Barbara, 125
lista, 264
 ACL, 493
 bibliotek w projekcie, 40
 inicjalizatora słownika, 309
 inicjalizatorów, 234, 235
 kontroli dostępu, 493
 metod, 700
 metod na stosie, 74
 parametrów, 36
 plików, 381
 przycisków, 784
 szablonów, 483
 usług, 487
listy list, 258
listy połączone, 320
literał, 52
literał łańcuchowy, 326
literał znakowy, 327
literały XML, 23
localhost, 528
losowa nazwa pliku, 390
losowość, 446
LSB, least-significant byte, 376
LSP, Liskov Substitution Principle, 125

Ł

ładowanie encji dla asocjacji, 561
ładowanie jawne, 607
ładowanie podzespołów, 605
ładowanie z bufora GAC, 606
ładowanie z folderu aplikacji, 606

ładowanie z pliku Silverlight, 607
łańcuch formatujący, 345
łańcuch jako klucz w słowniku, 350
łańcuch połączenia z SQL Serverem, 576
łańcuch UserAgent, 518
łańcuch znaków, 72, 324
łańcuchy URL, 514
łączenie, 304
łączenie elementów ścieżek, 393
łączenie klienta z serwerem, 496
łączenie leniwych enumeracji, 272
łączenie łańcuchów znakowych, 143, 571
łączenie się z usługami, 531

M

magazyn, store, 434
magazyn ciasteczek, 524
magazyn użytkownika, 440
magazyny izolowane, 436
mały odstęp, thin space, 366
MapReduce, 303
mechanizm
 Hibernate, 548
 IntelliSense, 701
 Isolated Storage, 433, 440
 komunikacji, 491
 odwzorowywania, 557
 odzyskiwania pamięci, 724
 ORM, 548
 P/Invoke, 720, 722
 szeregujący, scheduler, 615, 666
 wymiany metadanych, 512
 zmiany nazw, 139
media, 756
MEF, Managed Extensibility Framework, 608
metadane, metadata, 512, 593, 675
metoda, 35, 71, 78
 AddAnything, 702
 AddFirst, 320
 AddLast, 320
 AddNumbers, 235, 270
 AddProcess, 175
 AddRange, 256
 Aggregate, 303
 APM, 667
 Array.Copy, 246
 Array.Resize, 247, 254
 Array.Sort, 245
 Backwards, 279
 BeginGetRequestStream, 520
 BeginTransaction, 583
 BeginWrite, 431

ClockIn, 145, 157
CompareBytes, 422
CompareFiles, 415
CompareTo, 245
Compile, 570
Configure, 169
Connect, 513
Console.ReadKey, 374
Console.ReadLine, 692
Contains, 318
ContinueWith, 665
Convert.FromBase64String, 450
Convert.ToBase64String, 450
CopyTo, 246, 425
Cos, 687
CreateComInstanceFrom, 687
CreateEncryptor, 447
CreateInstance, 687
CreateInstanceFrom, 687
CreateTestFiles, 401
DateTime.ParseExact, 345
Delete, 389
Directory.CreateDirectory, 394
Directory.EnumerateFiles, 276
Directory.GetFile, 412
Directory.SetAccessControl, 410
Disconnect, 507
DisplayMatches, 380, 385, 414, 422
Dispose, 587
Dns.GetHostEntry, 535
double.Parse, 57
DownloadData, 514
DownloadFile, 514
DownloadString, 514, 655
Drive, 202
DynamicWhere, 710
Element, 468
Elements, 468
Encoding, 324
EncryptString, 446
End, 658
EndGetRequestStream, 520
EnlistTransaction, 583
Enqueue, 319
Equals, 308
ExecuteReader, 543
ExtinguishFire, 140
File, 402
File.Create, 431
File.CreateText, 435
File.EndWrite, 432
File.Exists, 406
File.OpenRead, 420, 426
File.OpenWrite, 426
File.ReadAllBytes, 419
File.ReadAllLines, 72
File.ReadAllText, 628
Find, 244
FindAll, 242
FindIndex, 244
FindLast, 244
FindLastIndex, 244
First, 299
Flush, 429
FlushFinalBlock, 448
GenerateIV, 445
GenerateKey, 445
GetAccessControl, 410
GetAllFilesInDirectory, 271, 275, 276
GetBytes, 372
GetCurrentDirectory, 384
GetEnumerator, 265
GetEventsByDay, 289
GetFiles, 383
GetFolderPath, 392
GetFullPath, 384
GetHashCode, 308
GetLength, 253
GetRandomFileName, 389
GetStream, 520
GetTempFileName, 388
GetType, 470
GetValue, 234
GetWebRequest, 518
GetWebResponse, 518
HandleError, 217
Include, 561
IncreaseQuotaTo, 441
IndexOf, 244, 361
InspectDirectories, 380, 381, 386
Invoke, 168
IsNullOrEmpty, 143
Join, 620
LastIndexOf, 244
Listen, 536
Load, 561
LoadFile, 608
Main, 36, 115, 189
MakeTestDirectories, 388, 409
Monitor.Enter, 642
Monitor.Exit, 642
myString.Replace, 692
odczytująca liczby z pliku, 72
OpenReadAsync, 517
OpenWrite, 517
OpenWriteAsync, 517

metoda

- Parallel.For, 671
- Parallel.ForEach, 673
- Path.Combine, 394
- POST, 517
- PostNote, 503, 509
- Process, 164, 169
- QueueUserWorkItem, 624
- ReadAllBytes, 414
- ReadAllLines, 414
- ReadAllText, 414
- ReadKey, 658
- ReadNumbersFromFile, 73
- Refresh, 585, 586
- RemoveAccessRuleSpecific, 410
- RemoveFirst, 320
- RollCall, 145
- Rotate, 202
- RunFor, 227
- SaveChanges, 583
- SecretKeyAndIV, 445
- Seek, 424
- Select, 278
- SendMessage, 107
- SetCurrentDirectory, 384
- SetValue, 234
- Sort, 244
- Split, 236
- String.Concat, 353
- String.Format, 345
- String.Join, 354
- String.Split, 237, 355
- Substring, 351, 367
- Task.WaitAll, 663
- Thread.Sleep, 649
- ToArray(), 73
- ToBuffer, 450
- ToLower, 355
- ToString, 330, 460
- ToUpper, 355
- Trim, 365, 366
- TrimEnd, 365
- TrimStart, 365, 367
- TryGetValue, 313, 315
- TryParse, 344
- UploadData, 516
- UploadFile, 516
- UploadString, 516
- WaitUntilReady, 648
- Where, 278
- WriteAllLines, 402
- WriteLine, 36

metody

- abstrakcyjne, 138, 139
- anonimowe, anonymous methods, 177
- asynchroniczne, 516
- klasy List<T>, 264
- klasy Path, 383
- przeciążone, 106
- rozszerzeń, extension methods, 278, 703
- rozszerzeń dla klasy String, 352
- statyczne, 115
- metody wirtualne, 130
- metody wytwórcze, factory methods, 108, 110
- metody zagnieżdżone, 622
- mniej znaczący bajt, 376
- mobilny profil, 392
- model
 - DOM, 698
 - encji danych, 552
 - jednowątkowy, 660
 - konceptyjny, conceptual model, 552
 - MVC, 777
 - programowania asynchronicznego, 520, 656
 - widoku, view model, 775
- modelowanie oprogramowania, 123
- M-odstęp, em space, 366
- modyfikacja kontraktu, 485
- modyfikacja nagłówka, 519
- modyfikacja procesora dokumentów, 172
- modyfikacja zmiennych zewnętrznych, 184
- modyfikator
 - abstract, 130, 150
 - const, 92
 - dostępu, 85, 132
 - new, 127
 - out, 214, 267
 - override, 129
 - protected, 132
 - protected internal, 132
 - readonly, 638
 - ref, 214, 730
 - sealed, 136
 - unsafe, 724
 - virtual, 130
- modyfikowanie encji, 577
- monitor, 639
- MSB, most-significant byte, 376
- multipleksing, 612
- muteks, 652

N

nadekspresja, overexpression, 664
narzędzie ildasm, 147
narzędzie SQL Profiler, 567
nasłuchiwanie przychodzących połączeń TCP, 535
naśladowanie, impersonation, 437
nawias klamrowy, 35, 65
nawias kwadratowy, 36
nazwa Control, 33
nazwa podzespołu, 602
nazwa zastępcza, alias, 48
NHibernate, 548
niejawna konwersja, 54
niezmiennosc, 638
niezmiennosc łańcucha znaków, 349
NIST, National Institute of Standards and Technology, 531
N-odstęp, en space, 366
no-PIA, 27
notatka, 486
numer portu, 530
numer wersji, 603
numerowanie elementów, 297

O

obciążenie procesora, 630
obiekt, 80
 APIFileReader, 727
 ASCIIEncoding, 727
 CalendarEvent, 244
 CancellationToken, 668
 COM, 696, 697
 DirectorySecurity, 410
 Encoding, 405
 FileDetails, 381
 FileInfo, 281
 FileNameGroup, 381
 FileSecurity, 429
 ImageSource, 776
 INamedPerson, 152
 IntPtr, 428
 IsolatedStorageFile, 443
 List<double>, 73
 List<T>, 641
 pośredniczący, proxy, 500
 SafeFileHandle, 428
 ServiceHost, 495
 Socket, 534
 StreamReader, 451
 StreamWriter, 435
 StringBuilder, 378

TrademarkFilter, 171
TransactionScope, 581
 typu Program, 72
obiekty .NET, 699
obiekty ExpandableObject, 705
obliczanie odległości, 115
obrazy, 755
obsługa
 anulowania, 668
 błędów, 211, 413, 669
 powinowactwa, 626
 stronicowania, 590
 wariancji, 267
 wątków, 617
 wyjątków, 219, 406, 412
 zdarzeń, 811, 813
ochrona, 599
ochrona wewnętrzna, internal, 600
odliczanie, 654
odnajdywanie wartości, 310
odpytywanie, polling, 656
odwołanie, reference, 594
odwołanie do pliku DLL, 596
odwołanie do tablicy, 253
odwołanie do zasobów EDM, 576
odwoływanie zdarzenia, 192
odwzorowywanie, mapping, 551
odzwierciedlanie, reflection, 675, 682
 badanie metadanych, 681
 na rzecz typu, 684, 686
 odkrywanie typów, 681, 683
 późne wiązanie, 681, 686
 tworzenie typów, 681
odzyskiwanie pamięci, 20
ognisko wprowadzania, input focus, 770
okno
 Add Connection, 789
 Add New Item, 552
 Add Reference, 595
 Add Service Reference, 496, 501
 Configure Data Source, 787
 Data Source Configuration Wizard, 787
 debuggera, 218
 konsoli, 495
 Mapping Details, 554
 Mapping Details, 555
 Model Browser, 554
 New Project, 30
 Solution Explorer, 557
 wiersza poleceń, 31
opakowanie obiektu komunikacji zwrotnej, 512
opcja Remove and Sort, 34
operacja wykonywana bez połączenia, 545

operacje
 asynchroniczne, 660
 na katalogach, 382
 na plikach, 382
 na ścieżkach, 383
 wejścia-wyjścia, 407

operator
 &&, 60
 /, 52
 ||, 61
 +=", 93, 634
 =, 92
 Aggregate, 302
 All, 301
 AND, 99
 Any, 301
 as, 157
 Average, 302
 Cast, 305
 Concat, 289, 304
 Count, 302
 dekrementacji --, 56
 Distinct, 304
 Except, 304
 First, 299
 group, 384
 inkrementacji ++, 56
 Intersect, 304
 Last, 299
 LastOrDefault, 299
 Max, 302
 Min, 302
 new, 108, 118
 OR, 99
 porównania, 60
 rzutowania, 54
 SelectMany, 297
 Single, 300
 Sum, 302
 Take, 299
 ToArray, 305
 ToDictionary LINQ, 317
 ToList, 305
 trójargumentowy, 61
 Union, 304
 Zip, 298

operatory LINQ, 279, 283, 285
opóźnione wykonanie, deferred execution, 566
opróżnianie buforów, 429
optymistyczna współbieżność, optimistic concurrency, 584
ORM, Object Relational Mapper, 548
osadzanie usługi WCF, 495
osie wyszukiwania, 468

P

pakiet, 526
pakowanie, boxing, 144
pamięć izolowana, isolated storage, 379
pamięć podręczna, 523

panel
 Canvas, 746
 DockPanel, 747
 Grid, 743, 744
 Properties, 784
 Solution Explorer, 39, 594, 713
 StackPanel, 745, 749
 Toolbox, 712, 783

parametr domyślny, 107
parametry nazwane, 109
parametry typów, type parameters, 255
PascalCase, 83
PascalCasing, 113
peer-to-peer, 476
pełna wersja .NET Framework, 733

pętla
 do while, 70
 for, 68
 foreach, 65
 while, 69

PIA, primary interop assembly, 27, 716
pisanie bibliotek, 597
platforma .NET, 13
platforma ASP.NET, 33
platforma WCF, 483, 505
platforma Windows Form, 21

plik
 App.config, 490, 497, 502, 574
 AssemblyInfo.cs, 39, 602
 ChatService.cs, 484
 Class1.cs, 597
 DisplayShippers.aspx.cs, 795
 IChatService.cs, 484
 MyData.svc, 589
 Program.cs, 30
 Reference.cs, 498
 standardsettings.txt, 443
 web.config, 495
 Web.config, 782

pliki
 pobieranie informacji, 385
 porównywanie zawartości, 414
 powtarzające się, 379
 tworzenie, 388
 usuwanie, 389
 wczytywanie do pamięci, 414
 wykrywanie duplikatów, 433
 zapisywanie, 402

- aspx, 778, 793
- exe, 38
- dll, 38
- edmx, 553, 555
- kodu ukrytego, 782
- msi, 39
- testowe, 387
- tymczasowe, 388
- XAML, 738, 764
- xap, 607
- PLINQ, równoległe LINQ, 673
- pobieranie
 - adresów IP, 528
 - ciasteczek, 524
 - danych, 514, 517
 - danych ASCII, 533
 - plików, 382
 - znaków, 351
- POCO, Plain Old CLR Object, 556
- podpis cyfrowy, 603
- podzespoły o silnych nazwach, 603
- podzespoły współdziałania, interop assemblies, 714
- podzespół, 437, 593, 683
- pojedyncza instancja usługi, 507
- pojemność obiektu StringBuilder, 360
- pole bitowe, bit fields, 99
- pole, field, 90
- pole logiczne canGo, 646
- pole statyczne, 116, 508
- pole tylko do odczytu, 95, 118
- pole wyboru Create directory for solution, 30
- polecenie Select, 790
- polecenie Set as Startup Project, 495
- polimorfizm, polimorphism, 125, 264
- połączenie z bazą danych, 567, 574
- połączenie ze źródłem danych, 788
- pomiar wydajności listy, 262
- porównywanie, 308
- porównywanie elementów, 245
- porównywanie nazw, 707
- port, 530, 536
- porządkowanie, 286
- powiadomienia, 645
- powiązania, 160
- powinowactwo do wątku, thread affinity, 625
- powrót karetki, carriage return, 354
- poziom ochrony, 84, 599
 - internal, 84
 - privat, 84
 - protected, 85
 - protected internal, 85
- późne ładowanie danych, lazy loading, 561
- późne wiązanie, late binding, 687
- prawa dostępu, 395
- prawo Moore'a, 629
- predykat, predicate, 175
- prezentacja oddzielona, separated presentation, 775
- priorityety operatorów, 54
- proces wdrażania oprogramowania, 482
- procesor dokumentów, 164, 171, 173
- procesor logiczny, 616
- procesory wielordzeniowe, 616
- profil podstawowy, basic profile, 477
- program
 - Adobe Illustrator, 755
 - Adobe Photoshop, 755
 - charmap.exe, 373
 - Expression Blend, 740
 - storeadm.exe, 441
 - WCF Test Client, 487
 - Wireshark, 489
- programowanie
 - asynchroniczne, 612, 655
 - dynamiczne, 26
 - funkcyjne, 20
 - wielowątkowe, 612
- projekcja, projection, 291
- projekt, 37
 - bazy danych, 551
 - biblioteki klas, 597
 - ChatClient, 499
 - ChatHost, 495
 - ChatServerLibrary, 490
 - Mono, 25
 - Moonlight, 25, 734
- projektowanie według kontraktu, 215
- protokoły TCP/IP, 526
- protokoły usług sieciowych WS-*, 506
- protokół
 - Daytime Protocol, 530
 - HTTP, 311, 505, 513
 - HTTPS, 540
 - IP, 526
 - kontroli transmisji, 528
 - TCP, 505, 529
- proxy, 500, 522
- przechowywanie danych, 311
- przechwytywanie błędów, 213
- przechwytywanie wyjątków, 221
- przeciążanie, 105
- przekazanie delegacji, 243
- przekazywanie danych, 695
- przekształcenia afiniczne, affine transformations, 750

- przekształcenie RotateTransform, 752
- przeładowanie, 704
- przełączenie kontekstu, 623
- przerwanie działania pętli, 71
- przerwanie operacji, 192
- przesłanie udostępnionych metod, 133
- przesłanie właściwości, 769
- przestrzeń kodowa, code space, 370
- przestrzeń nazw, namespace, 32, 41
 - ChatService, 498
 - Reflection, 681
 - System, 33
 - System.Collections.Concurrent, 654
 - System.Collections.Generic, 33, 307
 - System.Collections.Generics, 254
 - System.Data.EntityClient, 574
 - System.Data.SqlClient, 32
 - System.Diagnostics, 262
 - System.Dynamic, 706
 - System.IO, 32, 277
 - System.Linq, 33, 279
 - System.Net, 526
 - System.Net.Mail, 540
 - System.Net.Mime, 540
 - System.Net.NetworkInformation, 540
 - System.Net.PeerToPeer, 540
 - System.Net.Security, 540
 - System.Reflection, 608, 682
 - System.Text, 33
 - System.Threading.Tasks, 661
 - System.Transactions, 581
 - System.Xml.Serialization, 469
- przesyłanie danych, 516
- przetwarzanie dokumentów, 160, 163
- przetwarzanie łańcuchów znaków, 356
- przycisk Show All Files, 498
- przyciski z marginesami, 750
- przypinanie deklaratywne, declarative pinning, 726
- pula wątków, thread pool, 624, 625
- punkt końcowy
 - adres, address, 491
 - kontrakt, contract, 491
 - wiązanie, binding, 491
- punkt przerwania, breakpoint, 67
- punkt wejścia, entry point, 37, 594
- punkt wstrzymania, 213
- punkty kodowe, code points, 370
- pusta referencja, 364
- pusty łańcuchów znaków, 363
- pusty wiersz, 364

R

- rachunek lambda, 178
- RAD, Rapid Application Development, 777
- redukowanie, reduce, 303
- refaktoryzacja, 74, 128
- referencja, 100, 125, 128
- referencje do obiektów, 238, 304
- referencje do referencji do obiektu, 730
- reguła dostępu, 410
- rejestrwanie komunikatów, 183
- relacja, 559
- relacja dziedziczenia, 124
- relacja rodzic-dziecko, 663, 667
- relacje pomiędzy tabelami, 304
- relacyjna baza danych, 551
- relacyjny magazyn danych, 549
- renderowanie obrazów, 479
- REST, Representational State Transfer, 479
- RIA, Rich Internet Application, 19, 476, 733
- rodzaj izolacji, 439
- rodzeństwo, sibling elements, 454
- rodzic, parent element, 454
- router, 526
- rozpakowywanie, 145
- rozproszona odmowa usługi, 538
- rozszerzenie języka C#, 215
- równoległe LINQ, 673
- równoległość danych, 671
- równoważność typów, type equivalence, 716
- różnica pomiędzy tablicami i indeksatorami, 261
- rzadkie tablice, sparse array, 314
- rzutowanie, cast, 144

S

- scenariusz działania wielowątkowego, 643
- schemat przechowywania danych, store schema, 552
- Schemat XML, XML Schema, 471
- sekcja References, 40
- separacja kodu, code separation, 243, 778
- separator, 237
- serializacja, serialization, 231
- serializacja XML, 469, 472
- Service Pack, 199
- serwer .NET, 480
- sesja, 506
- siatka, 743
- siatka danych, data grid, 814
- silna nazwa, strong name, 437, 604
- Silverlight, 25, 477, 549, 733
 - dostęp do danych, 549
 - elementy graficzne, 752

- gniazda układu, 747
- typy paneli, 743
- składnia C# 4.0, 729
- składnia indeksatorów, 729
- składnia zapytań LINQ, 281
- składowa klasy, 113
- składowe zdarzeń, 190
- słownik, dictionary, 195, 307, 416, 508
 - przeglądanie zawartości, 316
 - rzadka tablica, 314
 - właściwości dynamiczne, 313
- słownik Properties, 313
- słowo
 - Callback, 512
 - pop, 74
 - push, 74
 - System, 32
- słowo kluczowe
 - abstract, 139
 - base, 134
 - break, 63, 70
 - catch, 219
 - class, 136, 138, 238
 - do, 70
 - dynamic, 27, 295, 313, 689, 699
 - enum, 96
 - event, 186
 - finally, 219
 - in, 278
 - interface, 150
 - lock, 642, 674
 - namespace, 34
 - new, 127, 293
 - null, 100
 - params, 237
 - partial, 557
 - ref, 694
 - static, 35, 72, 114, 691
 - struct, 103
 - this, 104, 279
 - try, 219
 - using, 32
 - value, 88
 - VALUE, 573
 - var, 691
 - void, 35, 86
- SMT, simultaneous multithreading, 616
- solucja, solution, 37
- sortowanie danych, 245, 287
- sortowanie przy użyciu LINQ, 283
- specyfikacja CDMA, 24
- specyfikacja Daytime Protocol, 532
- specyfikacja oprogramowania, 81
- spinanie, 298
- SpinLock, 650
- spłaszczanie list, 296
- sprawdzanie błędów, 208
- sprawdzanie pojemności, 359
- sprawdzanie typów, 157
- sprawdzanie zakresów, 205
- SQL Server, 543
- SQL Server 2008 Express, 550
- stała, 92
 - Environment.NewLine, 354
 - String.Empty, 362
- stan nasłuchu, 537
- stan statyczny, 116
- stan widoku, 779
- standard OCX, 711
- sterta, 101
- stopień tablicy, ang. rank, 253
- stos, 101, 321
- stos wywołań, call stack, 74, 214
- stos wywołań wątku, 617
- strażnik, guard, 215
- strefa czasowa GMT, 342
- strona kodowa, code page, 369
- strona kodowa Windows-1252, 369
- strumienie
 - aktualne położenie, 425
 - buforowanie, 428
 - odczyt, 419
 - porównywanie, 423
 - szyfrowanie, 444
 - zamykanie, 422
 - zapis danych, 425
- strumień, stream, 379, 418
 - adaptujący, 447
 - CryptoStream, 447
 - IsolatedStorageFileStream, 436
 - MemoryStream, 448
 - obiektów, 292
 - szyfrowania i deszyfracji, 451
 - szyfrujący, 444
 - wyjściowy, 448
- styl dynamiczny, 693, 696
- styl funkcyjny, 283
- styl przycisku, 768
- styl statyczny, 691
- style, 767
- subskrybent, 195
- subskrypcja zdarzeń, 188
- symbol
 - #, 337
 - , (przecinek), 337
 - . (kropka), 337
 - 0, 337

- symulator robota, 201
- synchronizacja, 639, 653
- synchronizacja wątków, 508
- system 64-bitowy, 717
- systemowy mechanizm obsługi błędów, 226
- szablon, template, 765
 - ADO.NET Entity Data Model, 552
 - Console Application, 30
 - Empty ASP.NET Web Application, 588
 - WCF Service Library, 484
- szablony
 - danych, 773
 - kontrolek, 765, 769
 - projektów ASP.NET, 781
- szybkość działania procesorów, 629
- szyfrowanie, 444
- szyfrowanie łańcucha znaków, 446

Ś

- ścieżka, 384
- śląd stosu, 494
- śmiertelny uścisk, deadly embrace, 636
- środowisko uruchomieniowe, 23

T

- tabela SalesOrderDetail, 578
- tabela SalesOrderHeader, 557
- tablica, array, 233
- tablica bajtów, 448, 776
- tablica łańcuchów znaków, 233
- tablice
 - elementy typu referencyjnego, 239
 - elementy typu wartościowego, 240
 - kopiowanie elementów, 246
 - odwołania do elementów, 253
 - odwołania do jednego obiektu, 239
 - określanie typu elementów, 236
 - porządkowanie elementów, 244
 - tworzone dynamicznie, 235
 - typ i liczba elementów, 235
 - wyszukiwanie elementów, 242, 264
- tablice nieregularne, jagged arrays, 247, 250
- tablice o zmiennej wielkości, 254
- tablice prostokątne, 251, 252
- tablice tablic, 248
- TCP, Transmission Control Protocol, 528, 529
- technologia
 - .NET, 21
 - ActiveX, 626
 - AJAX, 476, 778
 - ASP, 783

- ASP.NET, 13
- Authenticode, 603
- COM, 626, 711
- Entity Framework, 546
- Flash, 733
- IntelliSense, 715
- LINQ, 275
- ODBC, 542
- RIA, 476
- Silverlight, 19, 25, 436, 468
- Silverlight 2, 13
- Silverlight 3, 698
- Silverlight 4, 699
- SMT, 616
- WCF, 313
- WCF Data Services, 549
- Web Forms, 777
- Windows Forms, 310, 797
- WPF, 310, 626, 743, 797
- tekst jawny, plain text, 445
- tekst zaszyfrowany, 445
- test procesu przetwarzania, 174
- testowanie
 - aplikacji, 513
 - kodu, 64
 - kolekcji, 300
 - programu, 58
 - robota, 202
 - zapytania, 791
- testowy host, 490
- testowy host usługi WCF, 486
- testowy klient WCF, 486
- testy usługi, 484
- TPL, Task Parallel Library, 661
- transakcja niejawną, implicit transaction, 582
- transakcje, 579
- transakcje automatyczne, 583
- treść, content, 454
- tryb ConcurrencyMode.Single, 508
- tryb debugowania, 31
- tworzenia
 - interfejsu użytkownika, 733
 - aplikacji internetowych, 781
 - aplikacji Silverlight, 735
 - aplikacji w ASP.NET, 777
 - asocjacji, 122
 - delegacji, 172
 - dokumentu XML, 457, 461
 - hierarchii klas, 135
 - interfejsów, 152
 - interfejsu użytkownika, 736, 740
 - klucza i wektora początkowego, 445
 - kodu wielowątkowego, 634

- kodu XML, 463
- łańcuchów znaków, 358
- niebezpiecznego kontekstu, 726
- nowego typu, 294
- nowego wyjątku, 225
- obiektu argumentu zdarzenia, 191
- obiektu Thread, 623
- programu, 29
- projektu WCF, 483
- solucji, 597
- tablicy, 234, 238
- tablicy dynamiczne, 235
- tablicy tablic, 248
- typów, 681
- wątków, 613, 624
- własnych klas wyjątków, 231
- własnych typów enumerowalnych, 268
- właściwości, 90

typ, 32

- Action<>, 172
- anonimowy, anonymous type, 293
- bazowy, governing type, 97
- BigInteger, 49, 273
- Boolean, 59
- Char, 325, 385
- Control, 33
- DateTime, 340
- DateTimeOffset, 340, 341
- Dictionary, 315
- dynamic, 694, 701, 707, 731
- enum, 96
- FileAccess, 427
- FileMode, 426
- FileOptions, 430
- FileShare, 427
- Object, 295
- ogólny, generic type, 172, 255
- Predicate<T>, 175
- referencyjny, 619
- String, 325
- System.Double, 205
- TimeSpan, 107
- var, 696
- wartościowy, 102
- wygenerowany przez kompilator, 281
- wyliczeniowy, 96

typy

- atrybutów, 676
- liczbowe, 47
- liczbowe wbudowane, 144
- obiektów, 697
- referencyjne, 100, 238

- tablic, 236
- wyjątków, 228
- zmiennoprzecinkowe, 49
- zmiennych, 46

U

- udostępnianie
 - danych, 588
 - encji, 589
 - metod, 133
 - usług, 489
- ujednolicone identyfikatory zasobów, 515
- ujednolicone lokalizatory zasobów, 515
- ukrywanie metod, 128
- ukrywanie składowych klasy bazowej, 127
- UML, Unified Modeling Language, 123
- Unicode, 369
- unikatowość użytkowników, 509
- uprawnienia, 396, 400
 - administracyjne, 493
 - czynne, 396
 - do nasłuchiwania komunikatów, 493
 - odbieranie, 398, 409
 - określanie, 411
 - pełne do katalogu, 410
 - przydzielanie, 410
 - przywracanie, 410
 - specjalne, 396
- uprawnienie ListDirectory, 409
- URI, Uniform Resource Identifier, 479, 515
- URL, Uniform Resource Locator, 515
- uruchamianie debuggera, 217
- uruchamianie mechanizmu obsługi błędów, 226
- uruchamianie wielu projektów, 501
- usługa
 - ChatService, 487
 - RESTful, 479
 - Service1, 484
 - WCF, 477, 489
 - WWW, 482
 - WWW .NET, 483
- ustanowienie sesji, 507
- ustawienia kulturowe, 347
- usuwanie
 - białych znaków, 365
 - dyrektyw using, 34
 - encji, 579
 - katalogu, 401
 - plików, 389
 - podzespołu, 608
 - znaków, 366

uszkodzenie danych, 641
UTF-16, 370, 375
UTF-8, 370, 405
uwierzytelnianie, 521
uwierzytelnianie podstawowe, basic
 authentication, 522
uwierzytelnianie skrótowe, digest authentication,
 522
uwierzytelnianie zintegrowane, 521
uwięzienie, livelock, 636

V

VBA, Visual Basic for Applications, 693
Visual Studio, 29
Visual Studio 2010, 25

W

W3C, World Wide Web Consortium, 453
warstwa, 160
wartości domyślne parametrów, 109
wartości typu FileMode, 427
wartość FileSystemRights, 429
wartość inicjująca, seed, 446
wartość null, 158
wartość SeekOrigin.Current, 425
wartość typu bool, 59
wątek główny, 614, 649
wątki, threads, 613
 blokada SpinLock, 650
 blokowanie, 639
 izolacja, 637
 koordynacja, 630, 646
 liczba, 637
 monitor, 649
 niezmiennosc, 638
 synchronizacja, 638
 uwięzienie, 636
 współdzielenie informacji, 637
 współużytkowanie informacji, 633
 wyścig, 634
 zablokowanie, 628
 zakleszczenie, 636
wątki programowe, 616
WCF, Windows Communication Foundation, 13,
 21, 475, 483
WCF Data Services, 548, 587
WCF Service Host, 486, 490
WCF Service Library, 490
WCF Test Client, 486, 488
Web Forms, 777
 cykl życia strony, 780

wektor początkowy, initialization vector, 445
wiązanie, 491
 WS-ADDRESSING, 491
 wsDualHttpBinding, 511
 wsHttpBinding, 491, 511
 WS-SECURITY, 491
wiązanie danych, 771, 786, 808
widok zapytania, query view, 558
wielkość pliku, 385
wielokrotne dziedziczenie implementacji, 149
wielokrotne dziedziczenie interfejsów, 149
wielowątkowość, 612
wielowątkowość współbieżna, simultaneous
 multithreading, 616
więzy, 577, 578
Win32, 711
Win32 API, 720
Windows Forms, 797
 tworzenie aplikacji, 798
witryna MSDN, 83
własne interfejsy, custom interfaces, 698
własne szablony, 766
własny obiekt dynamiczny, 706
właściwości dołączone, attached properties, 744
właściwości dynamiczne, 313
właściwości indeksowane, indexed properties, 729
właściwości kontrolki, 714
właściwości nawigacji, navigation properties, 559
właściwości układów, 747
właściwość, 82
 AddressFamily, 528
 AutoReverse, 758
 CachePolicy, 523
 Capacity, 358
 CompareTo, 245
 ConcurrencyMode, 508
 Content, 767
 ContentTemplate, 776
 Count, 47
 Credentials, 521
 Direction, 96
 Exception.Data, 230
 HeaderTemplate, 776
 ID, 300
 includeExceptionDetailInFaults, 494
 InnerException, 225
 ItemTemplate, 776
 LayoutTransform, 751
 Length, 247, 358
 LongLength, 247
 Message, 215
 Method, 167
 MyProperty, 729

- nawigacji, 578
- Position, 112, 814
- provider, 575
- Quota, 441
- SalesOrderHeaders, 561
- SessionMode, 505
- StartTime, 245, 288
- statyczna, 116
- stream.Length, 420
- SystemDirectory, 117
- Target, 167
- Template, 765
- tylko do odczytu, 94
- typu bool, 95
- typu wyliczeniowego, 97
- VisualStateGroups, 770
- zawierająca delegację, 180
- włókna, fibers, 617
- WPF, Windows Presentation Foundation, 13, 733, 757
- WSDL, Web Service Definition Language, 485
- wskaźnik do bufora, 724
- wskaźniki, 724, 727
- współbieżność, 612, 671
- współbieżność precyzyjna, fine-grained concurrency, 664
- współdziałanie, interoperability, 711
- współdzielenie kontraktów, 499
- współdzielenie stanu przez wątki, 618
- współużytkowany stan programu, 622
- wstrzyknięcie kodu SQL, 791
- wyjątek, exception, 214
 - AddressAlreadyInUseException, 501
 - ArgumentException, 228
 - ArgumentNullException, 229
 - DirectoryNotFoundException, 407
 - DriveNotFoundException, 407
 - FileLoadException, 407
 - FileNotFoundException, 407
 - FormatException, 57
 - IndexOutOfRangeException, 234
 - InvalidCastException, 145
 - InvalidOperationException, 245, 300, 306
 - IOException, 407
 - NotSupportedException, 448, 569
 - NullReferenceException, 245
 - OptimisticConcurrencyException, 585
 - OverflowException, 420
 - PathTooLongException, 407
 - RuntimeBindeException, 703
 - SqlException, 230
 - Unauthorized AccessException, 409
 - UpdateException, 578
- wyjątki
 - blok catch, 221
 - blok finally, 227
 - generowane przez FileInfo, 413
 - istniejące, 230
 - operacji wejścia-wyjścia, 406
 - ponowne zgłaszanie, 223
 - przechwytywanie, 221
 - przekazywanie na wyższy poziom, 226
 - sygnalizowanie błędów, 214
 - systemu zabezpieczeń, 406
 - testowanie, 229
 - typowe, 406
 - typy, 227
 - własne, 230
 - zgłaszanie, 224, 228
- wykonywanie odroczone, deferred execution, 273, 547
- wyładowywanie, 608
- wyłączenie programu WCF Service Host, 496
- wyrażenia o postaci instrukcji, 179
- wyrażenia zapytań, query expressions, 275, 566
- wyrażenie, 52
- wyrażenie lambda, 178, 244, 281, 662
- wyrównanie elementu, 748
- wysunięcie wiersza, line feed, 354
- wyszukiwanie
 - nazwy ustawienia, 494
 - plików, 379
 - pojedynczego węzła, 467
 - tekstu, 361
 - w dokumencie XML, 465
- wyścig, race, 634
- wyświetlenie kategorii na liście, 773
- wywłaszczenie wątku, 617
- wywoływanie
 - funkcji Win32 API, 722
 - konstruktora, 104
 - metody Include, 562
 - metod klasy bazowej, 134
 - metody wirtualnej, 131
 - usługi sieciowej, 500
- wzorzec, 110

X

- XBAP WPF, 477
- XBAP, XAML Browser Application, 477
- XHTML, 455
- XML, eXtensible Markup Language, 453

Z

- zachowania, behaviors, 758
- zadania zwracające wyniki, 664
- zadanie, 661
- zadanie code-based, 670
- zaginanie, fold, 303
- zakleszczenie, deadlock, 508, 636
- zakres adresów, 493
- zaległości, backlog, 536
- zapis pliku tekstowego, 402
- zapis tekstu, 403, 435
- zapora sieciowa, 505
- zapytania
 - ESQL, 572
 - LINQ, 276, 317, 591
 - LINQ to Entities, 569
 - LINQ to Objects, 569
 - SQL, 544, 567
- zapytanie
 - definiujące, defining query, 558
 - grupujące, 290
 - jako wywołanie metody, 277
 - łańcuchowe, 568
 - oparte na łańcuchach znakowych, 572
 - orderedOrders, 568
 - zdegenerowane, 292
- zarządzanie
 - kluczami, 605
 - magazynami, 441
 - sesją, 512
- zastępowanie tekstu, 361
- zastrzyk SQL, SQL injection, 544
- zdalne wywołanie metody, 478
- zdarzenia, 653
 - deklaracja, 186
 - kod, 191
 - nazwa, 186
 - odwoływanie, 192
 - przechowywanie informacji, 195
 - rejestracja, 188
 - składowe, 190
 - subskrypcja, 188
 - typ delegacji, 186
 - zgłaszanie, 187
- zdarzenie, event, 186
 - domyślne, 813
 - DownloadProgressChanged, 516
 - OpenReadCompleted, 517
 - przesyłane, postback event, 779
 - Processing, 192
- zgłaszanie wyjątków, 219
- zgłaszanie zdarzeń, 187
- ziarno, 446
- zintegrowany język zapytań, 275
- złączenia, 304
- złożenie, assembly, 593
- zmiana mechanizmu komunikacji, 511
- zmienna, 45
 - iteracyjna, 65
 - lokalna, 113
 - lokalna projected, 295
 - lokalna w zapytaniu, 282
 - myDictionary, 307
 - myKey, 350
 - overlaps, 301
 - result, 315
 - this, 114
 - zakresu, range variable, 276
- znacznik, tag, 453
- <param>, 44
- <returns>, 44
- <summary>, 44
- asp, 783
- elementu pustego, 455, 459
- znak
 - &, 805
 - @, 329
 - dwukropka, 343
 - kropki, 37
 - odstępu, 42
 - odwrotnego ukośnika (\), 327
 - pusty, null character, 327
 - średnika, 42
 - ukośnika, 343
 - wykrzyknika, 69
- znaki
 - /*, 43
 - //, 43
 - specjalne, 327
 - sterujące, 328
- związek, 558
 - jeden-do-jednego, 564
 - jeden-do-wielu, 562
 - potrójny, ternary, 563
 - wiele-do-wielu, 564
- zwracanie wartości, 219

Ż

- źródło danych, 771
- źródło wiązania, binding source, 799

PROGRAM PARTNERSKI

GRUPY WYDAWNICZEJ HELION



1. ZAREJESTRUJ SIĘ
2. PREZENTUJ KSIĄŻKI
3. ZBIERAJ PROWIZJĘ

Zmień swoją stronę WWW
w działający bankomat!

Dowiedz się więcej i dołącz już dzisiaj!

<http://program-partnerski.helion.pl>

GRUPA WYDAWNICZA

 **Helion SA**

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).

helion.pl
księgarnia
internetowa

Nr katalogowy: 8608



Księgarnia internetowa:
<http://helion.pl>



Zamówienia telefoniczne:
0 801 339900
0 601 339900



Helion

Sprawdź najnowsze promocje:

🔗 <http://helion.pl/promocje>

🔗 <http://helion.pl/bestsellery>

Zamów informacje o nowościach:

🔗 <http://helion.pl/novosci>

Helion SA

ul. Kościuszki 1c, 44-100 Gliwice

tel.: 32 230 98 63

e-mail: helion@helion.pl

<http://helion.pl>

sięgnij po WIĘCEJ



KOD KORZYŚCI

ISBN 978-83-246-3701-0



Cena 129,00 zł