

» Idź do

- Spis treści
- Przykładowy rozdział

» Katalog książek

- Katalog online
- Zamów drukowany katalog

» Twój koszyk

- Dodaj do koszyka

» Cennik i informacje

- Zamów informacje o nowościach
- Zamów cennik

» Czytelnia

- Fragmenty książek online

» Kontakt

Helion SA
ul. Kościuszki 1c
44-100 Gliwice
tel. 032 230 98 63
e-mail: helion@helion.pl
© Helion 1991-2008

ASP.NET 3.5 z wykorzystaniem C# i VB. Zaawansowane programowanie

Autor: Bill Evjen, Scott Hanselman, Devin Rader

Tłumaczenie: Paweł Dyl, Marek Pałczyński

ISBN: 978-83-246-1852-1

Tytuł oryginału: [Professional ASP.NET 3.5: in C# and VB](#)

Format: 172x245 , stron: 1792

Oprawa: twarda



Kompletne źródło informacji na temat ASP.NET!

- Jak rozpocząć pracę z ASP.NET?
- Jakie kontrolki dostarcza ASP.NET?
- Jak wykorzystać technologię AJAX?

ASP.NET powstał jako odpowiedź firmy Microsoft na rosnącą popularność aplikacji internetowych. Jest on częścią technologii .NET oraz bezpośrednim następcą rozwiązania ASP, dostępnym na rynku od stycznia 2002 roku. Od tego czasu ASP.NET zdobywa sobie coraz większą popularność. Teraz dzięki wykorzystaniu technologii .NET możesz projektować dynamiczne strony, korzystając z dowolnego języka obsługującego to rozwiązanie.

Dzięki tej książce dowiesz się, jak to robić przy użyciu języków C# i Visual Basic. Jednak zanim napiszesz pierwszą linię kodu, warto poznać nowe środowisko pracy, tak aby tworzenie stron przebiegało szybko, wygodnie i bezproblemowo. W kolejnych rozdziałach zdobędziesz wiedzę na temat dostępnych kontrolki, metod pracy ze stronami wzorcowymi oraz sposobów stosowania kompozycji i skórek. Ponadto nauczysz się wykorzystywać źródła danych oraz zarządzać informacjami przy użyciu ADO.NET. Książka ta sprawdzi się znakomicie zarówno w rękach początkującego, jak i zaawansowanego programisty.

- Środowisko pracy
- Dyrektywy strony ASP.NET 3.5
- Obsługa zdarzeń
- Wykorzystanie kontrolki serwerowych
- Wykonywanie skryptów po stronie klienta
- Walidacja danych
- Zastosowanie kompozycji i skórek
- Źródła danych
- Zarządzanie danymi przy użyciu ADO.NET
- Tworzenie zapytań z wykorzystaniem LINQ
- Konfiguracja i uruchomienie usługi IIS7
- Model dostawców
- Platformy portalowe
- Wsparcie dla HTML i CSS
- Wykorzystanie technologii AJAX
- Bezpieczeństwo aplikacji internetowych w ASP.NET
- Obsługa sytuacji wyjątkowych
- Debugowanie kodu
- Wykorzystanie plików i strumieni
- Obsługa żądań HTTP
- Możliwości Silverlight

Twórz dynamiczne rozwiązania, korzystając z nowoczesnych technologii!

Spis treści

Wstęp	25
Rozdział 1. Środowiska do tworzenia aplikacji i stron	49
Opcje lokalizacji aplikacji	49
Wbudowany serwer WWW	50
IIS	51
FTP	52
Strony internetowe wymagające FrontPage Extensions	53
Opcje struktury strony ASP.NET	54
Model inline	56
Model code-behind	58
Dyrektywy strony ASP.NET 3.5	61
@Page	62
@Master	64
@Control	66
@Import	67
@Implements	69
@Register	69
@Assembly	70
@PreviousPageType	70
@MasterType	70
@OutputCache	71
@Reference	71
Zdarzenia strony ASP.NET	72
Praca z mechanizmem postback	74
Mechanizm cross-page posting	74
Katalogi aplikacji ASP.NET	81
Katalog \AppCode	81
Katalog \App_Data	86
Katalog \App_Themes	86
Katalog \App_GlobalResources	87
\App_LocalResources	87
\App_WebReferences	87
\App_Browsers	87

Kompilacja	88
Build Providers	92
Korzystanie z wbudowanych klas BuildProvider	93
Korzystanie z własnych klas BuildProvider	94
Global.asax	99
Praca z klasami w VS 2008	103
Podsumowanie	108

Rozdział 2. Kontrolki serwerowe ASP.NET i skrypty po stronie klienta 109

Kontrolki serwerowe ASP.NET	109
Typy kontrolek serwerowych	110
Tworzenie aplikacji z wykorzystaniem kontrolek serwerowych	112
Praca ze zdarzeniami kontrolek serwerowych	114
Dodawanie stylu do kontrolek serwerowych	117
Przegląd wspólnych właściwości kontrolek	117
Zmiana stylu za pomocą kaskadowych arkuszy stylów	118
Kontrolki serwerowe HTML	122
Omówienie klasy bazowej HtmlControl	125
Omówienie klasy HtmlContainerControl	126
Omówienie wszystkich klas HTML	127
Praca z klasą HtmlGenericControl	128
Zarządzanie stronami i kontrolkami serwerowymi za pomocą JavaScript	129
Korzystanie z Page.ClientScript.RegisterClientScriptBlock	131
Korzystanie z Page.ClientScript.RegisterStartupScript	133
Korzystanie z Page.ClientScript.RegisterClientScriptInclude	135
Funkcja zwrotna po stronie klienta	135
Porównanie postback z funkcją zwrotną	135
Korzystanie z możliwości funkcji zwrotnej — proste podejście	138
Korzystanie z funkcji zwrotnych z jednym parametrem	142
Użycie mechanizmu funkcji zwrotnej — przykład bardziej zaawansowany	145
Podsumowanie	150

Rozdział 3. Kontrolki serwerowe Web ASP.NET 151

Wprowadzenie do kontrolek serwerowych Web	151
Kontrolka serwerowa Label	153
Kontrolka serwerowa Literal	155
Kontrolka serwerowa TextBox	155
Użycie metody Focus()	157
Użycie AutoPostBack	157
Użycie AutoCompleteType	159
Kontrolka serwerowa Button	160
Właściwość CausesValidation	160
Właściwość CommandName	160
Przyciski, które współpracują z JavaScript po stronie klienta	162
Kontrolka serwerowa LinkButton	164
Kontrolka serwerowa ImageButton	164
Kontrolka serwerowa HyperLink	166
Kontrolka serwerowa DropDownList	166
Wizualne usuwanie elementów z kolekcji	169

Kontrolka serwerowa ListBox	171
Umżliwienie wyboru kilku pozycji	171
Przykład użycia kontrolki ListBox	171
Dodawanie elementów do kolekcji	174
Kontrolka serwerowa CheckBox	174
W jaki sposób sprawdzić, czy pole wyboru jest zaznaczone	175
Przypisanie wartości do pola wyboru	176
Wyrównywanie tekstu kontrolki CheckBox	176
Kontrolka serwerowa CheckBoxList	177
Kontrolka serwerowa RadioButton	180
Kontrolka serwerowa RadioButtonList	182
Kontrolka serwerowa Image	183
Kontrolka serwerowa Table	184
Kontrolka serwerowa Calendar	187
Wybieranie daty za pomocą kontrolki Calendar	187
Wybieranie formatu daty pobieranej z kalendarza	189
Wybór dni, tygodni lub miesięcy	189
Praca z zakresami dat	190
Zmiana stylu i zachowania kalendarza	192
Kontrolka serwerowa AdRotator	196
Kontrolka serwerowa Xml	198
Kontrolka serwerowa Panel	199
Kontrolka serwerowa Placeholder	201
Kontrolka serwerowa BulletedList	202
Kontrolka serwerowa HiddenField	207
Kontrolka serwerowa FileUpload	209
Pobieranie plików za pomocą kontrolki FileUpload	209
Nadawanie ASP.NET właściwych praw do pobierania plików	212
Zrozumienie limitów rozmiaru plików	214
Wczytywanie wielu plików na tej samej stronie	215
Przekazywanie pobranego pliku do obiektu Stream	218
Przenoszenie zawartości pliku z obiektu Stream do tablicy bajtów	219
Kontrolki serwerowe MultiView oraz View	220
Kontrolka serwerowa Wizard	224
Dostosowanie nawigacji po stronach	226
Użycie atrybutu AllowReturn	226
Praca z atrybutem StepType	227
Wstawianie nagłówka w kontrolce Wizard	228
Praca z systemem nawigacji kontrolki Wizard	228
Obsługa zdarzeń kontrolki Wizard	229
Użycie kontrolki Wizard do pokazania elementów formularza	231
Kontrolka serwerowa ImageMap	236
Podsumowanie	238
Rozdział 4. Walidacyjne kontrolki serwerowe	239
Zrozumienie procesu walidacji	239
Walidacja po stronie klienta a walidacja po stronie serwera	240
Kontrolki walidacyjne ASP.NET	242
Przyczyny walidacji	243
Kontrolka serwerowa RequiredFieldValidator	244
Kontrolka serwerowa CompareValidator	249

Kontrolka serwerowa RangeValidator	252
Kontrolka serwerowa RegularExpressionValidator	257
Kontrolka serwerowa CustomValidator	258
Kontrolka serwerowa ValidationSummary	263
Wyłączanie walidacji po stronie klienta	266
Korzystanie z obrazków i dźwięków w powiadomieniach o błędach	268
Praca z grupami walidacyjnymi	269
Podsumowanie	273
Rozdział 5. Praca ze stronami wzorcowymi	275
Do czego są nam potrzebne strony wzorcowe?	275
Podstawy stron wzorcowych	278
Pisanie kodu stron wzorcowych	279
Pisanie kodu strony z zawartością	282
Łączenie różnych typów stron i języków	286
Określanie, której strony wzorcowej użyć	288
Praca z tytułem strony	289
Praca z kontrolkami i właściwościami strony wzorcowej	290
Określanie domyślnej zawartości na stronie wzorcowej	297
Programowe przypisywanie strony wzorcowej	299
Osadzanie stron wzorcowych	300
Strony wzorcowe dostosowane do przeglądarek	304
Porządek wywoływania zdarzeń	306
Buforowanie stron wzorcowych	306
ASP.NET AJAX i strony wzorcowe	307
Podsumowanie	310
Rozdział 6. Kompozycje i skórki	311
Korzystanie z kompozycji ASP.NET	311
Przypisywanie kompozycji pojedynczej stronie ASP.NET	311
Stosowanie stylów do całej aplikacji	313
Usuwanie kompozycji z kontrolki serwerowych	314
Usuwanie kompozycji ze stron	315
Stosowanie kompozycji podczas korzystania ze stron wzorcowych	316
Działanie atrybutu StyleSheetTheme	316
Tworzenie własnych kompozycji	317
Tworzenie właściwej struktury katalogów	317
Tworzenie skórki	318
Umieszczanie w kompozycjach plików CSS	320
Wstawianie do kompozycji obrazków	323
Definiowanie wielu opcji skórek	326
Programowa praca z kompozycjami	328
Programowe przypisywanie kompozycji strony	328
Programowe przypisanie właściwości SkinID kontrolki	329
Kompozycje, skórki i własne kontrolki	330
Podsumowanie	334
Rozdział 7. Wiązanie danych w ASP.NET 3.5	335
Kontrolki źródeł danych	335
Kontrolka SqlDataSource	337
Kontrolka LinqDataSource	350
Kontrolka AccessDataSource	355

Kontrolka XmlDataSource	356
Kontrolka ObjctDataSource	357
Kontrolka SiteMapDataSource	362
Konfiguracja buforowania kontrolki źródła danych	362
Przechowywanie informacji o połączeniu	363
Użycie kontroltek list umożliwiających wiązanie z kontrolkami źródeł danych	366
GridView	366
Edycja danych rekordu GridView	383
Usuwanie danych GridView	390
DetailsView	393
Wstawianie, modyfikacja i usuwanie danych za pomocą DetailsView	398
ListView	400
FormView	410
Inne kontrolki umożliwiające wiązanie danych	414
DropDownList, ListBox, RadioButtonList oraz CheckBoxList	414
TreeView	415
AdRotator	415
Menu	416
Składnia rozwijanego wiązania danych	416
Zmiany w składni wiązania danych	417
Wiązanie danych XML	418
Wyrażenia i klasy do budowania wyrażeń	419
Podsumowanie	424
Rozdział 8. Zarządzanie danymi za pomocą ADO.NET	425
Podstawowe możliwości ADO.NET	426
Podstawowe zadania ADO.NET	426
Pobieranie danych	427
Wstawianie danych	428
Aktualizacja danych	429
Usuwanie danych	430
Podstawowe przestrzenie nazw i klasy ADO.NET	431
Korzystanie z obiektu Connection	432
Korzystanie z obiektu Command	434
Korzystanie z obiektu DataReader	435
Korzystanie z DataAdapter	438
Korzystanie z parametrów	440
Opis obiektów DataSet oraz DataTable	443
Typowany DataSet	448
Korzystanie z bazy danych Oracle i ASP.NET	449
Kontrolka serwerowa DataList	451
Przegląd dostępnych wzorców	452
Praca z ItemTemplate	452
Praca z innymi wzorcami układów graficznych	455
Praca z wieloma kolumnami	457
Kontrolka serwerowa ListView	458
Przegląd dostępnych wzorców	458
Korzystanie ze wzorców	459
Tworzenie wzorca układu graficznego	461
Tworzenie ItemTemplate	462
Tworzenie EditItemTemplate	463

Tworzenie EmptyItemTemplate	464
Tworzenie InsertItemTemplate	464
Wyniki	465
Wykorzystanie Visual Studio do zadań ADO.NET	466
Tworzenie połączenia ze źródłem danych	467
Praca z projektantem DataSet	470
Korzystanie z obiektu DataSet CustomersOrders	475
Asynchroniczne wywoływanie poleceń	479
Asynchroniczne metody klasy SqlCommand	480
Interfejs IAsyncResult	481
AsyncCallback	482
Klasa WaitHandle	482
Sposoby przetwarzania asynchronicznego w ADO.NET	483
Asynchroniczne połączenia	500
Podsumowanie	501
Rozdział 9. Zapytania z wykorzystaniem LINQ	503
LINQ to Objects	503
Tradycyjne metody zapytań	504
Zamiana tradycyjnych zapytań na LINQ	511
Grupowanie danych	519
Inne operatory LINQ	520
Złączenia LINQ	520
Paginacja za pomocą LINQ	522
LINQ to XML	523
Łączenie danych XML	526
LINQ to SQL	528
Zapytania Insert, Update oraz Delete z wykorzystaniem LINQ	537
Rozszerzanie LINQ	542
Podsumowanie	542
Rozdział 10. Praca z XML oraz LINQ to XML	543
Podstawy XML	544
XML InfoSet	546
Definicja schematu XSD-XML	547
Edycja plików XML oraz schematów XML w Visual Studio 2008	549
XmlReader oraz XmlWriter	552
Korzystanie z XmlDocument zamiast XmlReader	555
Korzystanie ze schematu oraz XmlTextReader	556
Walidacja względem schematu przy użyciu XmlDocument	558
Korzystanie z optymalizacji NameTable	560
Pobieranie typów .NET CLR z dokumentów XML	562
ReadSubtree oraz XmlSerialization	564
Tworzenie obiektów CLR z dokumentów XML za pomocą LINQ to XML	566
Tworzenie XML za pomocą XmlWriter	567
Tworzenie XML za pomocą LINQ to XML	569
Udoskonalenia obiektów XmlReader oraz XmlWriter w wersji 2.0	572
XmlDocument oraz XPathDocument	573
Problemy z DOM	573
XPath, XPathDocument oraz XmlDocument	574

Obiekty DataSet	578
Zapisywanie obiektów DataSet w postaci XML	578
XmlDataDocument	580
Kontrolka XmlDataSource	582
XSLT	585
XslCompiledTransform	588
Debugowanie XSLT	592
XML i bazy danych	593
FOR XML AUTO	594
SQL Server 2005 oraz typ danych XML	598
Podsumowanie	605
Rozdział 11. IIS 7	607
Modularna architektura usługi IIS 7	607
Serwer sieci Web	609
Narzędzia do zarządzania	612
Usługa publikowania za pomocą protokołu FTP	612
Rozszerzalna struktura usługi IIS 7	613
Zintegrowany potok przetwarzania żądań serwera IIS 7 i środowiska ASP.NET	613
Przygotowanie serwera WWW zgodnie z własnymi potrzebami	615
Zależności pakietów instalacyjnych	616
Instalacja usługi IIS 7 w systemie Windows Vista	617
Instalacja usługi IIS 7 w systemie Windows Server 2008	617
Instalacja za pomocą programu wiersza polecenia	618
Instalacja nienadzorowana	619
Uaktualnianie systemu	620
Menedżer internetowych usług informacyjnych (IIS)	620
Pule aplikacji	622
Witryny	626
Hierarchiczna konfiguracja	628
Delegowanie funkcji	633
Przenoszenie aplikacji z serwera IIS 6 do serwera IIS 7	636
Podsumowanie	637
Rozdział 12. Wprowadzenie do modelu dostawców	639
Zrozumienie modelu dostawców	640
Model dostawców w ASP.NET 3.5	641
Ustawianie dostawcy, aby współpracował z Microsoft SQL Server 7.0, 2000, 2005 lub 2008	643
Dostawcy członkostwa	649
Dostawcy ról	653
Dostawca personalizacji	658
Dostawca SiteMap	659
Dostawcy SessionState	661
Dostawcy WebEvent	664
Dostawcy konfiguracji	672
Dostawca WebParts	675
Konfigurowanie dostawców	677
Podsumowanie	678

Rozdział 13. Rozszerzanie modelu dostawców	679
Dostawcy są jedną warstwą w rozbudowanej architekturze	679
Modyfikacja programowa z wykorzystaniem atrybutów	680
Ułatwienie wprowadzania hasła za pomocą SqlMembershipProvider	681
Nakładanie silnych restrykcji na hasło za pomocą SqlMembershipProvider	684
Analiza ProviderBase	685
Tworzenie własnych klas dostawców	687
Tworzenie aplikacji CustomProvider	687
Tworzenie wymaganego szkieletu klasy	689
Tworzenie magazynu danych XML	692
Definiowanie instancji dostawcy w pliku web.config	693
Nieimplementowane metody i właściwości klasy MembershipProvider	694
Implementacja metod i właściwości klasy MembershipProvider	695
Korzystanie z XmlMembershipProvider podczas logowania użytkownika	703
Rozszerzanie istniejących dostawców	704
Ograniczenie możliwości zarządzania rolami za pomocą nowego dostawcy LimitedSqlRoleProvider	705
Korzystanie z nowej klasy dostawcy LimitedSqlRoleProvider	709
Podsumowanie	713
Rozdział 14. Nawigacja portalu	715
Mapy portalu w postaci plików XML	716
Kontrolka serwerowa SiteMapPath	718
Właściwość PathSeparator	720
Właściwość PathDirection	722
Właściwość ParentLevelsDisplayed	722
Właściwość ShowToolTips	723
Elementy potomne kontrolki SiteMapPath	723
Kontrolka serwerowa TreeView	724
Wbudowane style kontrolki TreeView	728
Badanie składników kontrolki TreeView	729
Wiązanie kontrolki TreeView z plikiem XML	729
Wybór wielu opcji w kontrolce TreeView	732
Przypisywanie do kontrolki TreeView własnych ikon	736
Używanie linii w celu połączenia węzłów	737
Programistyczna praca z kontrolką TreeView	739
Kontrolka serwerowa Menu	745
Przypisywanie do kontrolki Menu różnych stylów	747
Zdarzenia Menu	752
Wiązanie kontrolki Menu z plikiem XML	753
Dostawca danych SiteMap	755
ShowStartingNode	755
StartFromCurrentNode	756
StartingNodeOffset	757
StartingNodeUrl	758
SiteMap API	758
Mapowanie URL	761
Lokalizacja mapy portalu	762
Tworzenie pliku Web.sitemap korzystającego z lokalizacji	762
Wprowadzanie modyfikacji w pliku Web.config	763
Tworzenie plików podzespołów z zasobami (.resx)	764
Testowanie wyników	765

Security trimming	767
Ustawienie zarządzania rolami dla administratorów	767
Ustawianie sekcji administratorów	769
Włączanie security trimming	770
Zagnieżdżanie plików SiteMap	772
Podsumowanie	775

Rozdział 15. Personalizacja 777

Model personalizacji	778
Tworzenie właściwości personalizacji	779
Dodawanie prostej właściwości personalizacji	779
Korzystanie z właściwości personalizacji	780
Dodawanie grup właściwości personalizacji	784
Korzystanie z grupowanych właściwości personalizacji	785
Definiowanie typów właściwości personalizacji	785
Korzystanie z własnych typów	786
Ustawianie wartości domyślnych	789
Tworzenie właściwości personalizacji tylko do odczytu	789
Personalizacja anonimowa	790
Umożliwienie anonimowej identyfikacji użytkowników	790
Praca z anonimową identyfikacją	793
Anonimowe opcje właściwości personalizacji	794
Uwagi na temat przechowywania profili anonimowych użytkowników	795
Programowy dostęp do personalizacji	796
Migracja użytkowników anonimowych	796
Personalizacja profili	798
Określanie, czy korzystać z automatycznego zapisu	799
Dostawcy personalizacji	800
Praca z SQL Server Express Edition	800
Praca z Microsoft SQL Server 7.0, 2000, 2005, 2008	801
Korzystanie z wielu dostawców	804
Zarządzanie profilami aplikacji	804
Właściwości klasy ProfileManager	805
Metody klasy ProfileManager	805
Tworzenie strony ProfileManager.aspx	806
Omówienie kodu strony ProfileManager.aspx	809
Uruchomienie strony ProfileManager.aspx	811
Podsumowanie	812

Rozdział 16. Członkostwo i zarządzanie rolami 813

Uwierzytelnianie	814
Autoryzacja	814
Uwierzytelnianie ASP.NET 3.5	814
Ustalanie członkostwa na portalach	814
Wstawianie użytkowników	818
Pobieranie danych uwierzytelniających	833
Praca z zarejestrowanymi użytkownikami	841
Pokazywanie liczby użytkowników online	843
Obsługa haseł	845
Autoryzacja ASP.NET 3.5	850
Korzystanie z kontrolki serwerowej LoginView	850
Konfiguracja systemu zarządzania rolami na stronie	853

Dodawanie i pobieranie ról w aplikacji	856
Usuwanie ról	858
Dodawanie użytkowników do ról	859
Pobieranie wszystkich użytkowników określonej roli	860
Pobieranie wszystkich ról określonego użytkownika	862
Usuwanie użytkowników z ról	863
Sprawdzanie, czy użytkownicy przypisani są do ról	863
Wyjaśnienie sposobu buforowania ról	865
Korzystanie z Web Site Administration Tool	866
Publiczne metody API członkostwa	867
Publiczne metody API ról	867
Podsumowanie	868
Rozdział 17. Platformy portalowe oraz Web Parts	869
Wprowadzenie do Web Parts	870
Tworzenie dynamicznych i modularnych portali	871
Wprowadzenie do kontrolki WebPartManager	872
Praca z układami stref	873
Omówienie kontrolki WebPartZone	877
Zezwolenie użytkownikowi na zmianę trybu strony	879
Modyfikacja stref	891
Praca z klasami platformy portalowej	898
Tworzenie własnych kontrolki Web Part	902
Łączenie Web Parts	908
Tworzenie dostawcy Web Part	909
Tworzenie kontrolki Web Part konsumenta	912
Łączenie obiektów Web Parts na stronie ASP.NET	914
Trudności podczas pracy z kontrolkami Web Part oraz stronami wzorcowymi	917
Podsumowanie	918
Rozdział 18. Projekt HTML oraz CSS w ASP.NET	919
Uwagi	920
Ogólne informacje na temat HTML oraz CSS	920
Wprowadzenie do CSS	921
Tworzenie arkuszy stylów	922
Reguły CSS	925
Dziedziczenie CSS	933
Układ i położenie elementów	935
Praca z HTML oraz CSS w Visual Studio	943
ASP.NET 2.0 CSS — Friendly Control Adapters	951
Podsumowanie	951
Rozdział 19. ASP.NET AJAX	953
Zrozumienie potrzeby stosowania AJAX	953
Przed AJAX	954
AJAX zmienia ten stan rzeczy	955
ASP.NET AJAX oraz Visual Studio 2008	959
Technologie po stronie klienta	959
Technologie działające po stronie serwera	960
Tworzenie aplikacji za pomocą ASP.NET AJAX	961

Aplikacje ASP.NET AJAX	962
Tworzenie prostej strony ASP.NET niekorzystającej z AJAX	964
Tworzenie prostej strony ASP.NET z użyciem AJAX	965
Kontrolki ASP.NET AJAX po stronie serwera	971
Kontrolka ScriptManager	972
Kontrolka ScriptManagerProxy	974
Kontrolka Timer	975
Kontrolka UpdatePanel	977
Kontrolka UpdateProgress	982
Korzystanie z wielu kontrolek UpdatePanel	985
Podsumowanie	988
Rozdział 20. ASP.NET AJAX Control Toolkit	989
Pobieranie i instalacja	989
Nowe wzorce Visual Studio	991
Dodawanie nowych kontrolek do okna narzędziowego Visual Studio 2008	991
Kontrolki ASP.NET AJAX	993
Kontrolki rozszerzające ASP.NET AJAX Control Toolkit	996
AlwaysVisibleControlExtender	996
AnimationExtender	999
AutoCompleteExtender	1000
CalendarExtender	1004
CollapsiblePanelExtender	1005
ConfirmButtonExtender oraz ModalPopupExtender	1007
DragPanelExtender	1009
DropDownExtender	1011
DropShadowExtender	1013
DynamicPopulateExtender	1015
FilteredTextBoxExtender	1018
HoverMenuExtender	1020
ListSearchExtender	1021
MaskedEditExtender oraz MaskedEditValidator	1023
MutuallyExclusiveCheckBoxExtender	1026
NumericUpDownExtender	1027
PagingBulletedListExtender	1028
PopupControlExtender	1029
ResizableControlExtender	1031
RoundedCornersExtender	1033
SliderExtender	1035
SlideShowExtender	1036
TextBoxWatermarkExtender	1039
ToggleButtonExtender	1041
UpdatePanelAnimationExtender	1042
ValidationCalloutExtender	1043
Kontrolki serwerowe ASP.NET AJAX Control Toolkit	1045
Kontrolka Accordion	1045
Kontrolka NoBot	1047
Kontrolka PasswordStrength	1049
Kontrolka Rating	1050
Kontrolka TabContainer	1052
Podsumowanie	1053

Rozdział 21. Bezpieczeństwo	1055
Uwierzytelnianie i autoryzacja	1056
Techniki uwierzytelniania	1056
Węzeł <authentication>	1057
Uwierzytelnianie Windows	1058
Uwierzytelnianie na podstawie formularzy	1068
Uwierzytelnianie z wykorzystaniem mechanizmu Microsoft Passport	1079
Uwierzytelnianie w dostępie do określonych plików i katalogów	1079
Autoryzacja programowa	1080
Właściwość User.Identity	1081
Metoda User.IsInRole()	1082
Uzyskiwanie dodatkowych informacji z obiektu WindowsIdentity	1083
Element <identity> i tryb personifikacji	1086
Zabezpieczenia serwera IIS	1089
Ograniczenie zakresu adresów IP i nazw domenowych	1089
Rozszerzenia plików	1091
Korzystanie z konsoli ASP.NET MMC	1094
Konsola menedżera usługi IIS 7.0	1095
Podsumowanie	1096
Rozdział 22. Zarządzanie informacjami o stanie aplikacji	1097
Jakie opcje są do wyboru?	1098
Obiekt Session platformy ASP.NET	1101
Sesje a model zdarzeń	1101
Konfiguracja mechanizmu zarządzania sesją	1103
Sesje wewnętrzne	1104
Sesje zewnętrzne	1113
Sesje zapisywane w serwerach SQL	1119
Rozszerzenie sesji o inne mechanizmy dostawców danych	1123
Sesje bez plików cookie	1124
Wybór odpowiedniego sposobu podtrzymywania sesji	1126
Obiekt Application	1126
Łańcuchy zapytania	1127
Dane cookie	1128
Odsyłanie danych i przekazywanie danych między stronami	1129
Ukryte pola formularza, mechanizmy ViewState oraz ControlState	1131
Wykorzystanie kolekcji HttpContext.Current.Items do przechowywania krótkookresowych wartości	1136
Podsumowanie	1138
Rozdział 23. Buforowanie	1139
Buforowanie	1139
Buforowanie danych wyjściowych	1140
Buforowanie części strony (kontrolki użytkownika)	1143
Podmiana wartości w buforowanej treści	1145
Buforowanie po stronie klienta i obiekt HttpCachePolicy	1147
Buforowanie programowe	1149
Buforowanie danych za pomocą obiektu Cache	1150
Nadzorowanie pracy pamięci podręcznej środowiska ASP.NET	1151
Zależności wpisów pamięci podręcznej	1151

Zależności bufora SQL	1157
Dodanie bazy danych do listy bazy danych uwzględnianych w zależnościach bufora SQL ...	1158
Dodanie tabeli do list tabel uwzględnianych w zależnościach bufora SQL	1159
Sprawdzenie zmian w konfiguracji usługi SQL Server 2000	1159
Analiza włączonych tabel	1161
Usunięcie tabeli z listy tabel uwzględnianych w zależnościach bufora SQL	1161
Usunięcie bazy danych z listy baz uwzględnianych w zależnościach bufora SQL	1161
Zależności bufora SQL w bazie danych SQL Server 2005	1162
Konfiguracja aplikacji ASP.NET	1163
Testowanie mechanizmu unieważniania danych bufora SQL	1165
Odwołanie do więcej niż jednej tabeli w kodzie strony	1168
Powiązanie zależności bufora SQL z obiektem Request	1168
Powiązanie zależności bufora SQL z obiektem Cache	1169
Podsumowanie	1172

Rozdział 24. Debugowanie i obsługa błędów 1173

Wsparcie w czasie projektowania	1173
Powiadomienia o błędach składni	1174
Okna Immediate i Command	1176
Lista zadań	1177
Śledzenie kodu	1178
Klasy System.Diagnostics.Trace oraz Page.Trace	1178
Śledzenie kodu na poziomie strony	1179
Śledzenie pracy aplikacji	1179
Przeglądanie danych wynikowych	1179
Śledzenie pracy komponentów	1184
Przekazywanie danych ze śledzenia kodu	1186
Obiekty TraceListener	1186
Przełączniki diagnostyczne	1192
Zdarzenia sieciowe	1194
Debugowanie	1196
Potrzebne elementy	1196
Usługi IIS i ASP.NET Development Server	1198
Uruchomienie sesji debugowania	1199
Nowe narzędzia ułatwiające debugowanie	1203
Debugowanie klienckiego kodu JavaScript	1207
Debugowanie procedur składowanych SQL	1209
Wyjątki i obsługa błędów	1210
Przechwytywanie wyjątku na stronie	1211
Obsługa wyjątków aplikacji	1212
Kody statusowe HTTP	1213
Podsumowanie	1215

Rozdział 25. Pliki i strumienie 1217

Dyski, katalogi i pliki	1218
Klasa DriveInfo	1218
Klasy Directory i DirectoryInfo	1221
Klasy File i FileInfo	1228
Przetwarzanie ścieżek dostępu	1233
Właściwości plików i katalogów, ich atrybuty oraz listy kontroli dostępu	1237

Odczyt i zapis plików	1244
Strumienie	1245
Obiekty odczytu i zapisu	1249
Kompresowanie danych strumieni	1254
Wykorzystanie portów szeregowych	1260
Komunikacja sieciowa	1261
Klasy WebRequest i WebResponse	1262
Przesyłanie poczty elektronicznej	1269
Podsumowanie	1270
Rozdział 26. Kontrolki użytkownika i kontrolki serwerowe	1271
Kontrolki użytkownika	1272
Utworzenie kontrolki użytkownika	1272
Interakcje z kontrolkami użytkownika	1275
Dynamiczne ładowanie kontrolki użytkownika	1277
Kontrolki serwerowe	1283
Przygotowanie projektu kontrolki WebControl	1283
Atrybuty sterujące	1289
Wyświetlanie kontrolki	1291
Dodawanie atrybutów znaczników	1295
Definicje stylu HTML	1297
Motywy tematyczne i skórki	1300
Dodanie elementów kodu klienckiego	1302
Wykrywanie parametrów przeglądarki	1312
Mechanizm ViewState	1315
Wywoływanie zdarzeń powodujących odesłanie strony	1319
Obsługa odsyłanych danych	1323
Kontrolki złożone	1325
Kontrolki szablonowe	1328
Zachowanie kontrolki w środowisku projektowym	1336
Podsumowanie	1356
Rozdział 27. Moduły HTTP i obsługa żądań	1357
Przetwarzanie żądań HTTP	1357
IIS 5 (IIS 6) i ASP.NET	1358
IIS 7 i ASP.NET	1358
Przetwarzanie żądań ASP.NET	1359
Moduły HTTP	1360
Zmiana wynikowych danych HTTP	1362
Przepisywanie adresów URL	1365
Symbole wieloznaczne	1369
Procedury obsługi żądań HTTP	1371
Odwzorowanie rozszerzenia pliku w serwerze IIS	1376
Podsumowanie	1379
Rozdział 28. Obiekty biznesowe	1381
Korzystanie z obiektów biznesowych w środowisku ASP.NET 3.5	1381
Tworzenie wstępnie skompilowanych obiektów biznesowych platformy .NET	1382
Wykorzystanie wstępnie skompilowanych obiektów biznesowych w aplikacji ASP.NET ..	1385
Wykorzystanie komponentów COM w środowisku .NET	1387
Komponent Runtime Callable Wrapper	1387
Wykorzystanie obiektów COM w kodzie ASP.NET	1389

Obsługa błędów	1394
Wdrażanie komponentów COM w aplikacjach .NET	1397
Odwołania do kodu .NET z poziomu kodu niezarządzonego	1399
Moduł COM-Callable Wrapper	1399
Współdziałanie komponentów .NET z obiektami COM	1402
Wczesne czy późne wiązanie	1405
Obsługa błędów	1406
Wdrażanie komponentów .NET z aplikacjami COM	1408
Podsumowanie	1410
Rozdział 29. Budowanie i wykorzystywanie usług	1411
Komunikacja między rozproszonymi systemami	1411
Budowa prostej XML-owej usługi sieciowej	1414
Dyrektywa WebService	1415
Plik klasy bazowej usługi sieciowej	1416
Udostępnianie zbiorów danych w formie dokumentów SOAP	1417
Interfejs usługi sieciowej	1420
Korzystanie z nieskomplikowanych XML-owych usług sieciowych	1423
Dodawanie odwołań	1424
Wywoływanie usługi sieciowej w kodzie aplikacji klienckiej	1426
Protokoły transportowe usług sieciowych	1429
Żądania HTTP GET	1430
Żądania HTTP POST	1433
Żądania SOAP	1434
Przeciążanie metod sieciowych	1434
Buforowanie odpowiedzi usług sieciowych	1438
Nagłówki SOAP	1439
Tworzenie usług sieciowych uwzględniających nagłówki SOAP	1439
Wykorzystanie nagłówków SOAP w odwołaniach do usługi sieciowej	1441
Wykorzystanie żądań SOAP 1.2	1444
Asynchroniczne odwołania do usług sieciowych	1446
Windows Communication Foundation	1449
Krok w stronę architektury bazującej na usługach	1450
Przegląd technologii WCF	1451
Tworzenie usług WCF	1451
Aplikacja korzystająca z usługi WCF	1460
Dodanie odwołania do usługi	1460
Kontrakty danych	1464
Przestrzenie nazw	1469
Podsumowanie	1469
Rozdział 30. Lokalizacja oprogramowania	1471
Ustawienia kulturowe i regionalne	1471
Typy kulturowe	1472
Wątki ASP.NET	1473
Ustawienia kulturowe serwera	1477
Ustawienia kulturowe klienta	1478
Tłumaczenie wartości i zmiana sposobu zachowania aplikacji	1480
Pliki zasobów ASP.NET 3.5	1488
Wykorzystanie zasobów lokalnych	1488
Wykorzystanie zasobów globalnych	1495
Edytor zasobów	1498
Podsumowanie	1498

Rozdział 31. Konfiguracja	1499
Ogólne informacje na temat konfiguracji	1500
Pliki konfiguracyjne serwera	1501
Plik konfiguracyjny aplikacji	1504
W jaki sposób są odczytywane ustawienia konfiguracyjne?	1504
Wykrywanie zmian w plikach konfiguracyjnych	1505
Format pliku konfiguracyjnego	1505
Wspólne ustawienia konfiguracyjne	1506
Ciągi połączeń	1506
Konfiguracja stanu sesji	1508
Konfiguracja kompilacji	1512
Parametry przeglądarek	1515
Niestandardowe komunikaty o błędach	1517
Uwierzytelnianie	1518
Identyfikacja użytkowników anonimowych	1522
Autoryzacja	1523
Blokowanie ustawień konfiguracyjnych	1525
Konfiguracja strony ASP.NET	1526
Włączane pliki	1528
Parametry pracy środowiska ASP.NET	1529
Konfiguracja procesu roboczego ASP.NET	1532
Przechowywanie ustawień aplikacji	1534
Programowe przetwarzanie plików konfiguracyjnych	1535
Ochrona ustawień konfiguracyjnych	1542
Edycja pliku konfiguracyjnego	1546
Tworzenie własnych sekcji konfiguracyjnych	1549
Wykorzystanie obiektu NameValueFileSectionHandler	1549
Wykorzystanie obiektu DictionarySectionHandler	1551
Wykorzystanie obiektu SingleTagSectionHandler	1552
Wykorzystanie własnej procedury obsługi ustawień konfiguracyjnych	1553
Podsumowanie	1555
Rozdział 32. Narzędzia monitorujące pracę serwisu	1557
Dzienniki zdarzeń	1557
Odczytywanie informacji z dziennika zdarzeń	1558
Zapis informacji w dzienniku zdarzeń	1561
Wskaźniki wydajności	1563
Przeglądanie liczników wydajności za pomocą narzędzi administracyjnych	1564
Narzędzie administracyjne uruchamiane w przeglądarce	1567
Śledzenie kodu aplikacji	1572
Monitorowanie kondycji aplikacji	1572
Model dostawcy danych systemu monitorowania kondycji aplikacji	1573
Konfiguracja systemu monitorowania kondycji aplikacji	1575
Zapis zdarzeń na podstawie parametrów konfiguracyjnych	
— uruchomienie przykładowej aplikacji	1583
Przekazywanie zdarzeń do serwera SQL	1584
Buforowanie zdarzeń sieciowych	1587
Wysyłanie informacji o zdarzeniach za pomocą poczty elektronicznej	1590
Podsumowanie	1595

Rozdział 33. Administracja i zarządzanie1597

Aplikacja ASP.NET Web Site Administration Tool	1597
Zakładka Home	1599
Zakładka Security	1599
Zakładka Application	1609
Zakładka Provider	1613
Konfiguracja środowiska ASP.NET w systemie Vista	1615
Kompilacja platformy .NET	1616
Globalizacja platformy .NET	1617
Profil platformy .NET	1618
Role platformy .NET	1619
Poziomy zaufania platformy .NET	1620
Użytkownicy platformy .NET	1620
Ustawienia aplikacji	1622
Ciągi połączenia	1622
Strony i formanty	1624
Dostawcy	1624
Stan sesji	1624
Poczta e-mail SMTP	1626
Podsumowanie	1626

Rozdział 34. Pakowanie i instalacja aplikacji1627

Instalowane elementy	1628
Czynności poprzedzające instalację	1628
Metody instalowania aplikacji WWW	1629
Program XCopy	1629
Opcja Copy Web Site środowiska Visual Studio	1632
Instalowanie wstępnie skompilowanej aplikacji WWW	1636
Utworzenie programu instalatora	1638
Szczegółowa analiza opcji instalatora	1648
Właściwości projektu instalacyjnego	1649
Edytor systemu plików	1653
Edytor rejestru	1657
Edytor typów plików	1658
Edytor interfejsu użytkownika	1659
Edytor niestandardowych operacji	1661
Edytor warunków uruchomienia	1663
Podsumowanie	1664

Dodatek A Wykorzystanie projektów wcześniejszych wersji ASP.NET1665

Przeniesienie nie jest trudne	1665
Równoległa praca wielu wersji platformy	1666
Aktualizacja aplikacji ASP.NET	1666
Łączenie wersji — uwierzytelnianie na bazie formularzy	1668
Aktualizacja — zarezerwowane foldery ASP.NET	1669
Format XHTML stron ASP.NET 3.5	1670
Brak plików .js w ASP.NET 3.5	1672
Konwertowanie aplikacji ASP.NET 1.x w środowisku Visual Studio 2008	1673
Przeniesienie aplikacji ze środowiska ASP.NET 2.0 do 3.5	1678

Dodatek B Narzędzia wspomagające pracę w środowisku ASP.NET	1681
Łatwiejsze debugowanie	1682
Firebug	1682
YSlow	1683
IE Developer Toolbar oraz Firefox WebDeveloper	1685
Aptana Studio — środowisko programistyczne języka JavaScript	1686
Narzędzia optymalizacji kodu — dotTrace i ANTS	1687
Źródła informacji	1689
PositionIsEverything.net, QuirksMode.org oraz HTMLDog.com	1689
Visibone	1689
www.asp.net	1689
Czyszczenie kodu	1690
Refactor! for ASP.NET	1690
Code Style Enforcer	1691
Packer for .NET — narzędzie zmniejszające rozmiar skryptu JavaScript	1692
Dodatki do środowiska Visual Studio	1694
Dodatek do Visual Studio ASPX Edit Helper	1694
Power Toys Pack Installer	1695
Rozszerzanie środowiska ASP.NET	1696
ASP.NET AJAX Control Toolkit	1696
ELMAH — rejestracja i obsługa błędów	1697
ISAPI_Rewrite	1698
Narzędzia programistyczne ogólnego przeznaczenia	1700
Internetowy konwerter kodu	1700
WinMerge i narzędzia wyszukujące różnice w kodzie	1701
Reflector	1701
CR_Documentor	1702
Process Explorer	1703
Podsumowanie	1704
Dodatek C Silverlight	1705
Rozszerzanie aplikacji ASP.NET za pomocą SilverLight	1705
Krok 1. Prosta aplikacja ASP.NET	1707
Wyszukiwanie grafiki wektorowej	1708
Konwertowanie grafiki wektorowej na XAML	1711
Narzędzia do podglądu i edycji XAML	1712
Integracja z istniejącą stroną ASP.NET	1719
Obsługa zdarzeń Silverlight w JavaScript	1720
Dostęp do elementów Silverlight w zdarzeniach JavaScript	1722
Podsumowanie	1724
Dodatek D Serwisy internetowe o ASP.NET	1725
Blogi autorów książki	1725
Inne blogi na temat ASP.NET	1725
Witryny internetowe	1726
Skorowidz	1727

4

Walidacyjne kontrolki serwerowe

Patrząc na okno *Toolbox* w Visual Studio 2008 — zwłaszcza wtedy, gdy czyta się rozdziały 2. i 3., w których omówiono różne kontrolki serwerowe pozostające do dyspozycji — można być porażonym ilością kontrolki serwerowych udostępnianych przez ASP.NET 3.5. W niniejszym rozdziale omówiono specyficzny typ kontrolki serwerowych, które można znaleźć w oknie *Toolbox*: **walidacyjne kontrolki serwerowe**.

Walidacyjne kontrolki serwerowe są serią kontrolki, które pozwalają pracować z danymi wprowadzonymi przez użytkowników końcowych w elementach formularza tworzonej aplikacji. Kontrolki pozwalają zadbać o poprawność danych wpisywanych na formularzu.

Zanim przejdziemy do omówienia sposobów ich użycia, przyjrzyjmy się dokładnie procesowi walidacji.

Zrozumienie procesu walidacji

Ludzie tworzyli aplikacje internetowe przez wiele lat. Zwykle było to spowodowane potrzebą udostępnienia lub pobrania informacji. W tym rozdziale skupimy się na aspekcie pobierania informacji przez aplikacje internetowe. Podczas pobierania danych w aplikacji ważne jest to, aby były to dane **poprawne**. Jeżeli dane nie są poprawne, wtedy nie ma większego sensu w gromadzeniu ich.

Walidacja jest procesem wielostopniowym i stanowi zbiór reguł, które nakłada się na zbierane dane. Tych reguł może być dużo lub mało i mogą być ścisłe lub dość luźne. Zależy to jedynie od potrzeb twórcy aplikacji. Nie istnieje żaden perfekcyjny sposób walidacji, ponieważ niektórzy użytkownicy mogą znaleźć jakiś sposób oszukania tych procedur, bez względu na zastosowane reguły. Cały problem tkwi w znalezieniu złotego środka pomiędzy niewielką ilością zasad oraz ścisłą kontrolą, która nie będzie miała wpływu na użyteczność aplikacji.

Dane zbierane do procesu walidacji pochodzą z formularzy aplikacji. Formularze zbudowane są z różnych typów elementów HTML, które są tworzone za pomocą tradycyjnych elementów HTML, kontrolek serwerowych HTML ASP.NET oraz kontrolek serwerowych Web ASP.NET. Wszystko to na końcu i tak staje się zbiorem elementów HTML wchodzącym w skład formularzy. Są to na przykład pola tekstowe, przyciski opcji, przyciski wyboru, listy rozwijane i wiele innych.

Pracując z przykładami zaprezentowanymi w tym rozdziale, będzie można zauważyć różne typy reguł walidacyjnych, które można dodać do elementów formularza. Należy pamiętać, że nie ma możliwości sprawdzenia, czy dane są prawdziwe. Można jedynie wprowadzić zasady, które pomagają odpowiedzieć na pytania typu:

- Czy coś zostało wpisane w polu tekstowym?
- Czy dane wpisane w polu tekstowym posiadają format adresu e-mail?

Warto także zwrócić uwagę na to, że możliwe jest zastosowanie więcej niż jednej reguły walidacji do elementu formularza HTML (przykłady zostaną pokazane w dalszej części tego rozdziału). W rzeczywistości do każdego elementu można zastosować tyle reguł walidacyjnych, ile tylko potrzeba. Dodanie kolejnych reguł do elementów zwiększa poziom sprawdzania poprawności danych.

Należy pamiętać, że pobieranie danych na stronach internetowych jest jedną z najważniejszych funkcji internetu. Należy więc zadbać o to, aby zebrane dane posiadały pewną wartość i miały jakieś znaczenie. Można o to zadbać, eliminując przypadki, w których zbierane informacje nie spełniają nakreślonych reguł.

Walidacja po stronie klienta a walidacja po stronie serwera

Początkujący twórcy aplikacji internetowych mogą nie być świadomi różnicy pomiędzy walidacją po stronie klienta i walidacją po stronie serwera. Przypuśćmy, że użytkownik końcowy po uzupełnieniu kontrolek formularza naciska przycisk *Zatwierdź*. ASP.NET pakuje formularz do postaci **żądania** i wysyła je do serwera, na którym ta aplikacja jest umieszczona. W tym punkcie cyklu żądanie – odpowiedź można przeprowadzić proces sprawdzania poprawności wprowadzonych informacji. Takie podejście nazywamy **walidacją po stronie serwera**, ponieważ wszystko dzieje się na serwerze.

Z drugiej strony można umieścić skrypt (zwykle pod postacią kodu JavaScript). Wysyłany jest on razem ze stroną do użytkownika końcowego i umożliwia sprawdzenie poprawności danych wprowadzonych do formularza, **zanim** zostanie on przesłany do serwera aplikacji. W tym przypadku mamy do czynienia z **walidacją po stronie klienta**.

Oba typy walidacji mają swoje wady i zalety. Programiści Active Server Pages 2.0/3.0 (w czasach klasycznego ASP) mieli świadomość wad i zalet tych rozwiązań, ponieważ cały proces sprawdzania poprawności danych wykonywali własnoręcznie. Wielu programistów spędziło wiele dni z klasycznym ASP. W tym czasie wprowadzili różne techniki walidacji, które spełniają pewne wymagania związane z wydajnością i bezpieczeństwem.

Walidacja po stronie klienta jest szybka. Użytkownik natychmiast otrzymuje odpowiedź. To coś, czego spodziewają się użytkownicy końcowi na każdym formularzu. Jeżeli z formularzem coś jest nie tak, wtedy walidacja po stronie klienta powoduje, że użytkownik natychmiast jest o tym informowany. Walidacja po stronie klienta przerzuca obowiązek przetwarzania danych i sprawdzania ich poprawności na klienta. Oznacza to, że nie trzeba używać mocy obliczeniowej na serwerze do przetwarzania tych samych informacji, ponieważ klient wykonał już całą pracę.

Jak można się domyślić na podstawie powyższego, walidacja po stronie klienta jest formą sprawdzania poprawności bardziej narażoną na różne niebezpieczeństwa. Gdy strona generowana jest w przeglądarce użytkownika, wtedy można dość łatwo podejrzec jej źródła (poprzez kliknięcie prawym przyciskiem myszy i wybranie opcji *Pokaż źródła*). Po wykonaniu takiej czynności można zobaczyć cały kod HTML strony. Oprócz tego można obejrzeć cały kod JavaScript, który na tej stronie został umieszczony. Jeżeli poprawność danych sprawdzana jest po stronie klienta, to dla sprawnego hakera nie stanowi żadnego problemu odesłanie spreparowanego formularza (zawierającego wartości, które są przez niego pożądane). Serwer może je wtedy odebrać jako prawidłowe. Istnieją także takie przypadki, gdy użytkownik zwyczajnie zablokuje w swojej przeglądarce obsługę skryptów — w ten sposób może uczynić walidację całkowicie bezużyteczną.

W związku z powyższym walidacja po stronie klienta powinna być rozważana w kategoriach wygody i ułatwienia życia użytkownikowi końcowemu. Nigdy nie powinien to być mechanizm zapewniający aplikacji bezpieczeństwo.

Bezpieczniejszą formą sprawdzania poprawności danych jest walidacja po stronie serwera. Walidacja po stronie serwera oznacza, że wszystkie procedury kontrolujące poprawność wykonywane są na serwerze, a nie na kliencie. Jest to bezpieczniejsze, ponieważ tego etapu nie da się w łatwy sposób ominąć. Dane na formularzu sprawdzane są przez kod serwera (C# lub VB) na serwerze. Jeżeli formularz nie jest prawidłowy, wtedy odsyłany jest do klienta jako nieprawidłowy. Jest to bezpieczniejsze, ale taka walidacja po stronie serwera może być wolna. Dzieje się tak, ponieważ strona musi być przesłana do zdalnego komputera i tam sprawdzona. Użytkownik końcowy na pewno nie będzie zadowolony, gdy po odczekaniu 20 sekund na odpowiedź dowie się, że wpisał swój adres e-mail w nieprawidłowym formacie.

Jaka jest zatem prawidłowa ścieżka? Generalnie obie są dobre! Najlepszym podejściem jest wykonanie walidacji po stronie klienta, a potem, po przejściu przez ten pierwszy etap i przesłaniu formularza na serwer, wykonywana jest walidacja po stronie serwera. Takie podejście jest najlepsze spośród wszystkich. Jest bezpieczne, ponieważ hakerzy nie mogą tak zwyczajnie ominąć procesu walidacji. Można oszukać walidację po stronie klienta, ale dane i tak zostaną jeszcze raz sprawdzone po przesłaniu ich na serwer. Taki sposób walidacji jest także dość efektywny — pozwala uzyskać szybkość i elegancję walidacji po stronie klienta.

Kontrolki walidacyjne ASP.NET

W czasach klasycznego ASP.NET programiści poświęcali dużo czasu na obsługę różnych schematów walidacji. Z tego powodu wraz z pierwszą wersją ASP.NET wprowadzono serię walidacyjnych kontroltek serwerowych, które pozwalają w łatwy sposób przeprowadzić sprawdzenie poprawności danych formularza.

Samo wprowadzenie walidacyjnych kontroltek serwerowych przez ASP.NET to nie wszystko. Kontrolki są wyjątkowo sprytnie. Jak już zostało wcześniej napisane, jednym z zadań programistów klasycznego ASP było zdecydowanie, gdzie przeprowadzać walidację — czy na kliencie, czy na serwerze. Walidacyjne kontrolki serwerowe eliminują ten problem, ponieważ ASP.NET wykrywa przeglądarkę i na tej podstawie podejmuje właściwą decyzję.

Oznacza to, że jeżeli aplikacja obsługuje JavaScript, wtedy ASP.NET przeprowadza walidację po stronie klienta. Jeżeli przeglądarka klienta nie obsługuje JavaScript i walidacji po stronie klienta, wtedy kod JavaScript jest pomijany, a cała walidacja przeprowadzana jest po stronie serwera.

Najlepsze z tego wszystkiego jest to, że pomimo pomyślnego umieszczenia na stronie walidacji po stronie klienta ASP.NET w dalszym ciągu przeprowadza walidację po stronie serwera w momencie otrzymania danej strony. W ten sposób nie ma żadnych kompromisów związanych z bezpieczeństwem. Decyzyjna natura walidacyjnych kontroltek serwerowych oznacza, że można tworzyć strony ASP.NET tak dobre, jak tylko one mogą być. Nie trzeba szukać żadnego wspólnego mianownika bezpieczeństwa i szybkości.

Obecnie w ASP.NET 3.5 dostępnych jest 6 kontroltek walidacyjnych. Od czasu wprowadzenia pierwszej wersji technologii ASP.NET nie pojawiły się żadne nowe kontrolki. W ASP.NET 2.0 wprowadzono jednak kilka nowych możliwości, takich jak grupy walidacji oraz nowe możliwości zastosowania JavaScript. Obie te techniki omówione są w tym rozdziale. Do dyspozycji mamy następujące walidacyjne kontrolki serwerowe:

- `RequiredFieldValidator`,
- `CompareValidator`,
- `RangeValidator`,
- `RegularExpressionValidator`,
- `CustomValidator`,
- `ValidationSummary`.

Praca z walidacyjnymi kontrolkami serwerowymi ASP.NET nie różni się niczym od pracy z innymi kontrolkami serwerowymi ASP.NET. Każda z tych kontroltek może być przeciągnięta i upuszczona na powierzchnię projektową, ale może być także wprowadzona bezpośrednio do kodu strony ASP.NET. Kontrolki mogą być modyfikowane w taki sposób, aby odpowiadały one potrzebom aplikacji. W ten sposób aplikacja może uzyskać unikalny wygląd. Wiele przykładów pracy z tymi kontrolkami pojawi się w dalszej części rozdziału.

Jeżeli walidacyjne kontrolki serwerowe nie spełniają wszystkich oczekiwań, wtedy zajdzie potrzeba napisania własnych kontrollek walidacyjnych. Istnieją jednak kontrolki napisane przez osoby trzecie, na przykład takie jak Validation and More Petera Bluma (VAM) z www.peterblum.com/VAM. Na wspomnianej stronie znajduje się ponad 40 kontrollek walidacyjnych ASP.NET.

W poniższej tabeli opisano funkcjonalność każdej z dostępnych walidacyjnych kontrollek serwerowych.

Walidacyjna kontrolka serwerowa	Opis
RequiredFieldValidator	Dbą o to, aby użytkownik nie opuścił danego pola formularza.
CompareValidator	Pozwala porównać dane wprowadzone przez użytkownika z innym elementem za pomocą operatora porównania (równe, większe niż, mniejsze niż i tak dalej).
RangeValidator	Sprawdza, czy wartość wprowadzona przez użytkownika mieści się w podanym zakresie liczb lub znaków.
RegularExpressionValidator	Sprawdza, czy wpis użytkownika jest zgodny ze wzorcem zdefiniowanym przez wyrażenie regularne. To dobra kontrolka do sprawdzenia adresu e-mail oraz numeru telefonu.
CustomValidator	Sprawdza wpis użytkownika za pomocą własnej logiki walidacyjnej.
ValidationSummary	Wyświetla wszystkie komunikaty o błędach wszystkich kontrollek walidacyjnych w jednym miejscu na stronie.

Przyczyny walidacji

Walidacja nie jest przeprowadzana nagle. Pojawia się w wyniku odpowiedzi na zdarzenie. W większości przypadków jest to zdarzenie naciśnięcia przycisku. Kontrolki serwerowe `Button`, `LinkButton` oraz `ImageButton` posiadają możliwość uruchomienia procesu walidacji na formularzu. Jest to zachowanie domyślne. Po przeciągnięciu i upuszczeniu kontrolki `Button` na formularz otrzymujemy następujący rezultat:

```
<asp:Button ID="Button1" runat="server" Text="Button" />
```

Przeglądając właściwości kontrolki `Button`, można zauważyć, że właściwość `CausesValidation` ustawiona jest na `True`. Jak już wspomniano, jest to ustawienie domyślne — wszystkie przyciski na stronie, bez względu na to, gdzie są, powodują uruchomienie procesu walidacji.

Jeżeli na stronie ASP.NET znajduje się wiele przycisków i nie ma potrzeby, aby każdy z nich wywoływał proces walidacji, wtedy można ustawić właściwość `CausesValidation` na `False` dla tych przycisków, które powinny zignorować proces walidacji (na przykład przycisk *Anuluj*):

```
<asp:Button ID="Button1" runat="server" Text="Anuluj" CausesValidation="false" />
```


Kontrolka serwerowa RequiredFieldValidator

Kontrolka RequiredFieldValidator zwyczajnie sprawdza, czy do elementu HTML formularza zostało **coś** wprowadzone. To prosta kontrolka walidacyjna, ale jest ona używana najczęściej. Kontrolka RequiredFieldValidator musi być wstawiona dla wszystkich elementów, które muszą spełniać regułę postaci **wartość wymagana**.

Na listingu 4.1 pokazano przykładowy sposób użycia kontrolki serwerowej RequiredFieldValidator.

Listing 4.1. Przykładowy sposób użycia kontrolki serwerowej RequiredFieldValidator

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        If Page.IsValid Then
            Label1.Text = "Strona jest prawidłowa!"
        End If
    End Sub
</script>

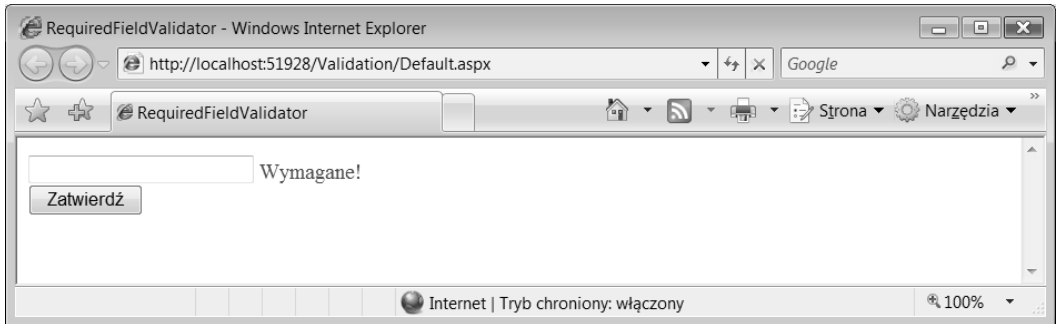
<html xmlns="http://www.w3.org/1999/xhtml" >
<head runat="server" id="Head1">
    <title>RequiredFieldValidator</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
            <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
                Runat="server" Text="Wymagane!" ControlToValidate="TextBox1">
</asp:RequiredFieldValidator>
            <br />
            <asp:Button ID="Button1" Runat="server" Text="Zatwierdź"
                OnClick="Button1_Click" />
            <br />
            <br />
            <asp:Label ID="Label1" Runat="server"></asp:Label>
        </div>
    </form>
</body>
</html>
```

C#

```
<%@ Page Language="C#" %>

<script runat="server">
    protected void Button1_Click(Object sender, EventArgs e) {
        if (Page.IsValid) {
            Label1.Text = " Strona jest prawidłowa!";
        }
    }
</script>
```

Zbudujmy stronę i spróbujmy ją uruchomić. Na stronie pokaże się proste pole tekstowe oraz przycisk. Spróbujmy teraz pozostawić pole tekstowe puste i nacisnąć przycisk *Zatwierdź*. Wyniki pokazane są na rysunku 4.1.



Rysunek 4.1

Popatrzmy teraz na kod z przykładu. Na początku można zauważyć, że przy kontrolkach `TextBox`, `Button` i `Label` nic się nie zmieniło. Wstawione są one tak, jakby nie była stosowana żadna technika walidacji. Na stronie umieszczono jednak prostą kontrolkę `RequiredFieldValidator`. Na kilka właściwości tej kontrolki należy zwrócić szczególną uwagę, ponieważ będą stosowane w każdej tworzonej walidacyjnej kontrolce serwerowej.

Pierwszą z tych ważnych właściwości jest `Text`. Wartość tej właściwości pokazywana jest użytkownikowi końcowemu na stronie wtedy, gdy walidacja nie powiedzie się. W pokazanym przykładzie jest to zwykły łańcuch znaków `Wymagane!`. Drugą właściwością, o której warto wspomnieć, jest `ControlToValidate`. Właściwość wykorzystywana jest do połączenia walidacyjnej kontrolki serwerowej z tym elementem formularza ASP.NET, który ma być sprawdzony. W tym przypadku właściwość wskazuje na jedyny element formularza — pole edycji.

W przykładzie można zauważyć, że komunikat o błędzie tworzony jest na podstawie atrybutu kontrolki `<asp:RequiredFieldValidator>`. Zadanie realizowane jest za pomocą atrybutu `Text`. Pokazano to na listingu 4.2.

Listing 4.2. Wykorzystanie atrybutu `Text`

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
  Runat="server" Text="Wymagane!" ControlToValidate="TextBox1">
</asp:RequiredFieldValidator>
```

Ten sam komunikat o błędzie można umieścić pomiędzy otwierającym i zamykającym węzłem `<asp:RequiredFieldValidator>`. Pokazano to na listingu 4.3.

Listing 4.3. Umieszczenie wartości pomiędzy węzłami

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
  Runat="server" ControlToValidate="TextBox1">
  Wymagane!
</asp:RequiredFieldValidator>
```

Sprawdzanie wygenerowanych wyników

Kontrolka `RequiredFieldValidator` używa walidacji po stronie klienta, jeżeli tylko przeglądarka na to pozwala. Można samemu przekonać się o obecności walidacji po stronie klienta (jeżeli przeglądarka pozwala), klikając na stronie prawym przyciskiem myszy i wybierając z menu podręcznego podgląd źródła. W kodzie strony można dostrzec funkcje JavaScript pokazane na listingu 4.4.

Listing 4.4. Wygenerowany kod JavaScript

```
... znaczniki strony usunięte dla przejrzystości

<script type="text/javascript">
<!--
function WebForm_OnSubmit() {
if (ValidatorOnSubmit() == false) return false;
return true;
}
// -->

... znaczniki strony usunięte dla przejrzystości

<script type="text/javascript">
<!--
var Page_Validators = new Array(document.getElementById("RequiredFieldValidator1"));
// -->
</script>

<script type="text/javascript">
<!--
var RequiredFieldValidator1 = document.all ? document.all["RequiredFieldValidator1"] :
document.getElementById("RequiredFieldValidator1");
RequiredFieldValidator1.controltovalidate = "TextBox1";
RequiredFieldValidator1.text = "Wymagane!";
RequiredFieldValidator1.evaluationfunction = "RequiredFieldValidatorEvaluateIsValid";
RequiredFieldValidator1.initialvalue = "";
// -->
</script>

... znaczniki strony usunięte dla przejrzystości

<script type="text/javascript">
<!--
var Page_ValidationActive = false;
if (typeof(ValidatorOnLoad) == "function") {
ValidatorOnLoad();
}

function ValidatorOnSubmit() {
if (Page_ValidationActive) {
return ValidatorCommonOnSubmit();
}
else {
return true;
}
}
// -->
</script>
```

W kodzie strony można zauważyć kilka zmian w elementach formularza (zwykłych kontrolkach formularza). Obsługują one teraz zatwierdzanie formularza oraz skojarzoną z tym zdarzeniem walidację.

Korzystanie z właściwości InitialValue

Kolejną ważną właściwością podczas pracy z kontrolką `RequiredFieldValidator` jest właściwość `InitialValue`. Czasami istnieją elementy, które zawierają pewną domyślną wartość początkową (na przykład pobraną z jakiegoś źródła danych). Takie elementy formularza mogą przedstawiać pewną wartość, która musi być zmieniona, zanim formularz zostanie przesłany na serwer.

Korzystając z właściwości `InitialValue`, określa się w kontrolce `RequiredFieldValidator` początkową wartość elementu. Użytkownik końcowy musi zmienić wartość tekstową tej kontrolki przed zatwierdzeniem formularza. Na listingu 4.5 pokazano przykład użycia tej właściwości.

Listing 4.5. Praca z właściwością InitialValue

```
<asp:TextBox ID="TextBox1" Runat="server">Moja wartość początkowa</asp:TextBox>
&nbsp;
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
  Runat="server" Text="Wartość pola tekstowego musi być zmieniona!"
  ControlToValidate="TextBox1" InitialValue="Moja wartość początkowa">
</asp:RequiredFieldValidator>
```

W tym przypadku właściwość `InitialValue` zawiera wartość `Moja wartość początkowa`. Po zbudowaniu i uruchomieniu strony pole tekstowe również zawiera tę wartość. Kontrolka `RequiredFieldValidator` wymaga zmiany w tym polu. Tylko wtedy strona zostanie zakwalifikowana jako poprawna.

Uniemożliwienie pustych wpisów i równoczesne wymaganie zmian

W poprzednim przykładzie użycia właściwości `InitialValue` pojawił się interesujący problem. Po pierwsze, jeżeli aplikacja z takim kodem zostanie uruchomiona, wtedy użytkownik może przejść przez proces walidacji, przesyłając stronę bez żadnej wartości wpisanej w polu edycji. Puste pole edycji nie spowoduje wygenerowania błędu walidacji, ponieważ kontrolka `RequiredFieldValidator` jest teraz przebudowana tak, aby zmusić użytkownika wyłącznie do **zmiany** domyślnej wartości pola edycji (co użytkownik zrobił, usuwając starą wartość). Zmieniając kontrolkę `RequiredFieldValidator` w ten sposób, nie nakłada się żadnego ograniczenia, które wymaga wpisania **czegoś** w polu edycji — zmieniona musi być tylko wartość początkowa. Proces walidacji może być ominięty poprzez usunięcie wszystkiego, co wpisano w polu tekstowym.

Istnieje jednak sposób obejścia tego problemu. Dochodzimy tu do techniki, o której już wcześniej wspomniano — na formularzu można umieścić wiele reguł walidacyjnych. Niektóre z nich mogą być przypisane do tego samego elementu. Aby nałożyć wymóg zmiany

Tak samo jak podczas pracy z kontrolką edycji, tak i w tym przypadku kontrolka `RequiredFieldValidator` skojarzona jest z kontrolką `DropDownList` za pomocą właściwości `ControlToValidate`. Lista rozwijana, do której podpięta jest kontrolka walidacyjna, posiada wartość początkową — Wybierz zawód. Oczywiście taka wartość nie jest właściwa w momencie przesyłania formularza na serwer. Znowu należy użyć właściwości `InitialValue` kontrolki `RequiredFieldValidator`. Wartością tej kontrolki jest wartość początkowa listy rozwijanej. Zmusza to oczywiście użytkownika do wyboru jednego z zawodów z listy rozwijanej. W przeciwnym razie nie będzie on w stanie przesłać strony na serwer.

Kontrolka serwerowa `CompareValidator`

Kontrolka `CompareValidator` pozwala wykonywać porównania pomiędzy dwoma elementami formularza, a także służy do porównania dwóch wartości umieszczonych we wskazanych elementach formularza. Można na przykład określić, że wartością elementu formularza musi być liczba całkowita większa niż inna zadana wartość. Można także wskazać, że wartościami muszą być łańcuchy znaków, daty lub inne typy, które są do dyspozycji.

Walidacja względem innych kontrolek

Jednym z dość często stosowanych sposobów wykorzystania kontrolki `CompareValidator` jest użycie jej do porównania dwóch elementów formularza. Przypuśćmy, że dana jest aplikacja, która w celu uzyskania dostępu do stron przez użytkowników wymaga podania hasła. Tworzymy zatem jedno pole edycji, które będzie służyło do pobrania hasła, i drugie pole tekstowe, które będzie służyło do potwierdzenia tego hasła. W związku z tym, że pole tekstowe działa w trybie wprowadzania hasła, użytkownik końcowy nie widzi tego, co wpisuje — zna tylko liczbę wprowadzonych znaków. Aby zmniejszyć prawdopodobieństwo pomyłki przy wpisywaniu hasła i zapobiec wprowadzeniu do systemu złego hasła, użytkownik proszony jest o potwierdzenie tego hasła. Po wprowadzeniu hasła do systemu należy porównać te dwa pola edycji i sprawdzić, czy wartości w nich wpisane są takie same. Jeżeli są takie same, wtedy jest duże prawdopodobieństwo, że użytkownik wpisał hasło poprawnie i może ono być wprowadzone do systemu. Jeżeli dwa pola tekstowe nie są zgodne, wtedy formularz należy traktować jako nieprawidłowy. W przykładzie z listingu 4.8 pokazano tę sytuację.

Listing 4.8. Wykorzystanie `CompareValidator` do porównania wartości z wartościami innych kontrolek

VB

```
<%@ Page Language="VB" %>
<script runat="server">

    Protected Sub Button1_Click(sender As Object, e As EventArgs)
        If Page.IsValid Then
            Label1.Text = "Hasła są zgodne"
        End If
    End Sub

</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>CompareFieldValidator</title>
```

```

</head>
<body>
  <form id="Form1" runat="server">
    <p>
      Hasło<br>
      <asp:TextBox ID="TextBox1" Runat="server"
        TextMode="Password"></asp:TextBox>
      &nbsp;
      <asp:CompareValidator ID="CompareValidator1"
        Runat="server" ErrorMessage="Hasła nie są zgodne!"
        ControlToValidate="TextBox2"
        ControlToCompare="TextBox1"></asp:CompareValidator>
    </p>
    <p>
      Potwierdź hasło<br>
      <asp:TextBox ID="TextBox2" Runat="server"
        TextMode="Password"></asp:TextBox>
    </p>
    <p>
      <asp:Button ID="Button1" OnClick="Button1_Click"
        Runat="server" Text="Zaloguj"></asp:Button>
    </p>
    <p>
      <asp:Label ID="Label1" Runat="server"></asp:Label>
    </p>
  </form>
</body>
</html>
C#
<%@ Page Language="C#" %>

<script runat="server">

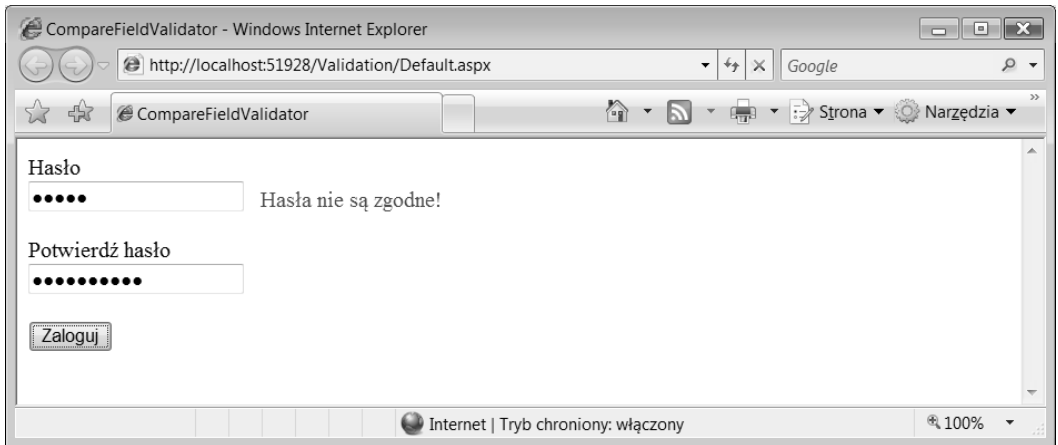
  protected void Button1_Click(Object sender, EventArgs e) {
    if (Page.IsValid)
      Label1.Text = "Hasła są zgodne";
  }

</script>

```

Patrząc na kontrolkę `CompareValidator` na formularzu, można zauważyć, że jest ona podobna do kontrolki `FieldValidator`. Kontrolka `CompareValidator` posiada właściwość `ControlToValidate`, która łączy się z jednym z elementów formularza na stronie. W tym przypadku potrzebna jest tylko jedna kontrolka `CompareValidator`, ponieważ wymagane jest tylko jedno porównanie. W pokazanym przykładzie sprawdzana jest zgodność wartości pól edycji `TextBox2` oraz `TextBox1`. W związku z tym użyto właściwości `ControlToCompare`. Określa się w ten sposób wartość porównywaną z `TextBox2`. W tym przykładzie wartością jest `TextBox1`.

Jest to wyjątkowo proste. Jeżeli dwa pola tekstowe w momencie zatwierdzania strony posiadają różne wartości, wtedy w przeglądarce wyświetlana jest wartość właściwości `Text` kontrolki `CompareValidator`. Przykład pokazany jest na rysunku 4.2.



Rysunek 4.2

Walidacja względem stałej

Kontrolki CompareValidator można użyć nie tylko do walidacji wartości względem wartości w innych kontrolkach, ale też do porównania wartości względem stałych określonych typów danych. Przypuśćmy, że na formularzu rejestracyjnym umieszczono pole edycji, w którym użytkownik wpisuje swój wiek. W większości tego typu przypadków celem jest pobranie liczby całkowitej, a nie czegoś w postaci aa lub bb. Na listingu 4.9 pokazano, w jaki sposób można wymusić podanie liczby całkowitej.

Listing 4.9. Wykorzystanie CompareValidator do walidacji względem stałej

```
Wiek:
<asp:TextBox ID="TextBox1" Runat="server" MaxLength="3">
</asp:TextBox>
 
<asp:CompareValidator ID="CompareValidator1" Runat="server"
  Text="Musisz wpisać liczbę"
  ControlToValidate="TextBox1" Type="Integer"
  Operator="DataTypeCheck"></asp:CompareValidator>
```

W pokazanym przykładzie użytkownik końcowy musi wpisać w polu tekstowym liczbę. Jeżeli spróbuje obejść proces walidacji, wpisując wartość nieprawidłową, która zawiera coś innego niż cyfry, wtedy strona będzie traktowana jako nieprawidłowa. Kontrolka CompareValidator wyświetla wartość właściwości Text.

Aby określić typ danych wykorzystywany w porównaniach, korzysta się z właściwości Type. Właściwość Type może przyjmować następujące wartości:

- Currency,
- Date,
- Double,
- Integer,
- String.

Wprowadzenie ograniczenia ze względu na typ to nie wszystko, co można zrobić przy wykorzystaniu tej kontrolki. Można także zadbać o to, aby wartość była porównywana z określoną stałą. Istnieje sposób na sprawdzenie, czy wartość wpisana w elemencie formularza jest większa niż, mniejsza niż, równa, większa niż lub równa i mniejsza niż lub równa pewnej wartości. Przykład pokazano na listingu 4.10.

Listing 4.10. Wykonywanie porównań za pomocą kontrolki CompareValidator

```
Wiek:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
    &nbsp;
<asp:CompareValidator ID="CompareValidator1" Runat="server"
    Operator="GreaterThan" ValueToCompare="18"
    ControlToValidate="TextBox1"
    Text="Aby się zapisać musisz mieć więcej niż 18 lat" Type="Integer">
</asp:CompareValidator>
```

W tym przypadku kontrolka CompareValidator oprócz połączenia się z kontrolką oraz nałożenia wymogu wprowadzenia liczby całkowitej robi coś jeszcze. Wykorzystywane są również atrybuty Operator oraz ValueToCompare. Dzięki nim można się upewnić, że liczba jest większa niż 18. Jeżeli zatem użytkownik wprowadzi liczbę równą 18 lub mniejszą, wtedy walidacja nie powiedzie się i strona będzie traktowana jako nieprawidłowa.

Właściwość Operator może przyjmować następujące wartości:

- Equal,
- NotEqual,
- GreaterThan,
- GreaterThanEqual,
- LessThan,
- LessThanEqual,
- DataTypeCheck.

Właściwość ValueToCompare to miejsce, gdzie umieszcza się wartość wykorzystywaną w porównaniu. W pokazanym przykładzie jest to liczba 18.

Kontrolka serwerowa RangeValidator

Kontrolka RangeValidator jest podobna do kontrolki CompareValidator. Pozwala ona sprawdzić, czy wartość wprowadzona przez użytkownika lub jego wybór znajduje się w podanym zakresie. Za pomocą kontrolki CompareValidator sprawdza się, czy wartość jest większa albo mniejsza niż wskazana stała. Jako przykład zastosowania kontrolki niech posłuży nieco zmodyfikowany przykład formularza pobierającego wiek. Nieco inny jest rodzaj walidacji wprowadzonej wartości. Pokazano go na listingu 4.11.

Listing 4.11. Wykorzystanie kontrolki RangeValidator do sprawdzenia wartości całkowitej

```
Wiek:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
    &nbsp;
<asp:RangeValidator ID="RangeValidator1" Runat="server"
```

```
ControlToValidate="TextBox1" Type="Integer"  
Text="Wiek musi być pomiędzy 30 i 40"  
MaximumValue="40" MinimumValue="30"></asp:RangeValidator>
```

W pokazanym przykładzie umieszczono pole edycji, za pomocą którego pobierany jest wiek użytkownika. Kontrolka `RangeValidator` analizuje wprowadzoną wartość i sprawdza, czy mieści się ona w zakresie od 30 do 40. Wykonywane jest to za pomocą właściwości `MaximumValue` oraz `MinimumValue`. Kontrolka `RangeValidator` pozwala także zadbać o to, aby wprowadzona wartość była liczbą całkowitą. W tym celu wykorzystywana jest właściwość `Type`, która ustawiona jest na `Integer`. Kolekcja zrzutów ekranów na rysunku 4.3 pokazuje działanie omawianej kontrolki.



Rysunek 4.3

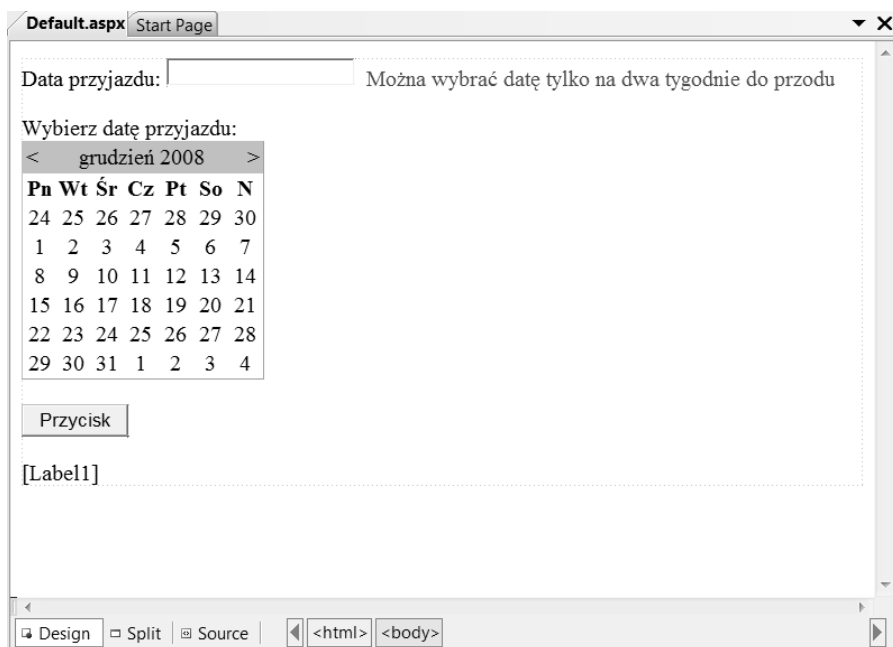
Jak można zauważyć na zrzutach ekranów z rysunku 4.3, wartość mniejsza niż 30 zostaje przez kontrolkę `RangeValidator` zasygnalizowana. To samo dzieje się w przypadku wartości większych niż 40. Wartość pomiędzy 30 i 40 (w tym przypadku 36) spełnia warunki nałożone przez kontrolkę walidacyjną.

Kontrolka `RangeValidator` nie jest stosowana wyłącznie do porównywania liczb (w takich sytuacjach jest jednak najczęściej wykorzystywana). Może ona posłużyć także do walidacji zakresu łańcuchów znaków oraz innych elementów, włączając w to daty pobrane z kalendarza. Domyślnie atrybut `Type` każdej z kontrolki walidacyjnych ustawiony jest na `String`. Kontrolka `RangeValidator` może posłużyć do sprawdzenia, czy wartość w innej kontrolce serwerowej (takiej jak na przykład `Calendar`) mieści się w podanym zakresie dat.

Przypuśćmy, że tworzony jest formularz internetowy, który pozwala na pobranie od użytkownika daty przyjazdu. Data przyjazdu musi mieścić się w zakresie dwóch tygodni od daty bieżącej. Kontrolka `RangeValidator` może być z łatwością dostosowana do takich potrzeb.

W związku z tym, że zakres dat musi być generowany dynamicznie, atrybuty `MaximumValue` oraz `MinimumValue` muszą być przypisane programowo w zdarzeniu `Page_Load`. W oknie projektanta przykładowa strona powinna wyglądać podobnie do tej z rysunku 4.4.

Rysunek 4.4



Cała idea polega na tym, że użytkownik będzie wybierał datę w kontrolce `Calendar`, a jego wybór będzie wyświetlany w kontrolce `TextBox`. Potem, gdy użytkownik kliknie na formularzu przycisk, wtedy w przypadku wybrania nieprawidłowej daty zostanie o tym powiadomiony. Jeżeli wybrana data jest prawidłowa, wtedy pokazywana jest ona przez kontrolkę `Label`. Kod przykładowy pokazany jest na listingu 4.12.

Listing 4.12. Wykorzystanie kontrolki RangeValidator do sprawdzenia wartości daty w postaci łańcucha znaków**VB**

```

<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
        RangeValidator1.MinimumValue = DateTime.Now.ToShortDateString()
        RangeValidator1.MaximumValue = DateTime.Now.AddDays(14).ToShortDateString()
    End Sub

    Protected Sub Calendar1_SelectionChanged(ByVal sender As Object, _
        ByVal e As System.EventArgs)
        TextBox1.Text = Calendar1.SelectedDate.ToShortDateString()
    End Sub

    Protected Sub Button1_Click(ByVal sender As Object, _
        ByVal e As System.EventArgs)
        If Page.IsValid Then
            Label1.Text = "Data przybycia ustawiona jest na: " & TextBox1.Text
        End If
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>Sprawdzenie poprawności daty</title>
</head>
<body>
    <form id="form1" runat="server">
        Data przyjazdu:
        <asp:TextBox ID="TextBox1" runat="server"></asp:TextBox>&nbsp;
        <asp:RangeValidator ID="RangeValidator1" runat="server"
            Text="Można wybrać datę tylko na dwa tygodnie do przodu"
            ControlToValidate="TextBox1" Type="Date"></asp:RangeValidator><br />
        <br />
        Wybierz datę przyjazdu:<br />
        <asp:Calendar ID="Calendar1" runat="server"
            OnSelectionChanged="Calendar1_SelectionChanged"></asp:Calendar>
        &nbsp;
        <br />
        <asp:Button ID="Button1" runat="server" Text="Przycisk"
            OnClick="Button1_Click" />
        <br />
        <br />
        <asp:Label ID="Label1" runat="server"></asp:Label>
    </form>
</body>
</html>

```

C#

```

<%@ Page Language="C#" %>

<script runat="server">
    protected void Page_Load(object sender, EventArgs e)
    {
        RangeValidator1.MinimumValue = DateTime.Now.ToShortDateString();
    }

```

```

        RangeValidator1.MaximumValue = DateTime.Now.AddDays(14).ToShortDateString();
    }

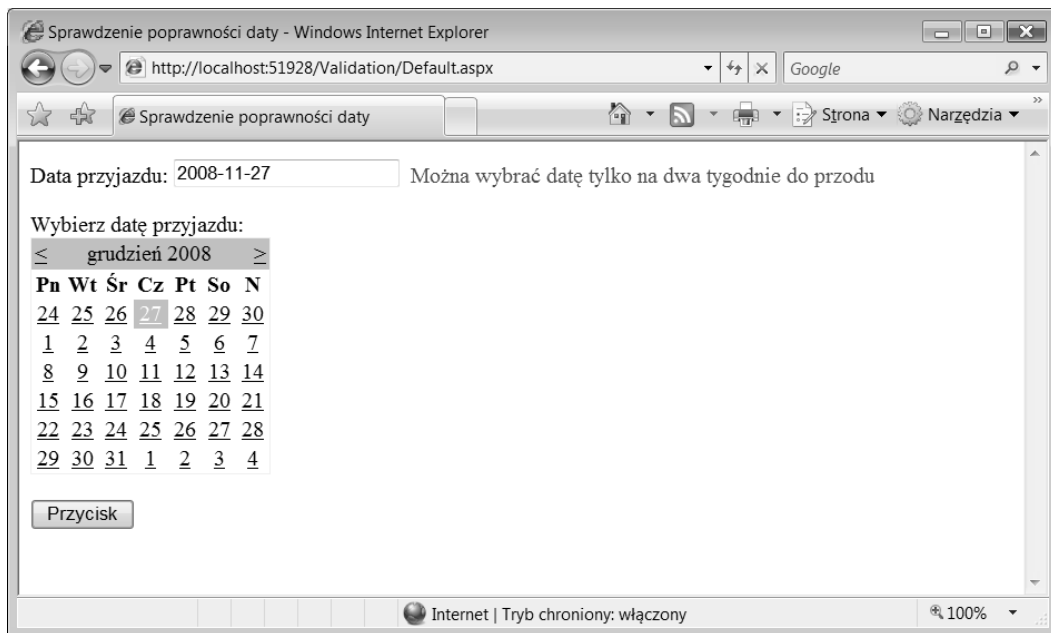
    protected void Calendar1_SelectionChanged(object sender, EventArgs e)
    {
        TextBox1.Text = Calendar1.SelectedDate.ToShortDateString();
    }

    protected void Button1_Click(object sender, EventArgs e)
    {
        if (Page.IsValid)
        {
            Label1.Text = "Data przybycia ustawiona jest na: " +
                TextBox1.Text.ToString();
        }
    }
</script>

```

W pokazanym kodzie widać, że podczas wczytywania strony atrybuty `MinimumValue` oraz `MaximumValue` ustawiane są dynamicznie. W tym przypadku wartością `MinimumValue` staje się wynik wywołania `DateTime.Now.ToShortDateString()`, a wartością `MaximumValue` data późniejsza o 14 dni.

Po wybraniu daty przez użytkownika jej wartość wyświetlana jest w kontrolce `TextBox1`. Wykonywane jest to w procedurze obsługi zdarzenia `Calendar1_SelectionChanged`. Po wybraniu daty i kliknięciu przycisku na stronie wykonywana jest procedura obsługi zdarzenia `Button1_Click`, a poprawność walidacji całej strony sprawdzana jest za pomocą właściwości `Page.IsValid`. Nieprawidłowa strona pokaże wyniki zaprezentowane na rysunku 4.5.



Rysunek 4.5

Kontrolka serwerowa `RegularExpressionValidator`

Jedną ze wspaniałych kontroltek, którą programiści najczęściej wykorzystują, jest `RegularExpressionValidator`. Kontrolka oferuje wysoki poziom elastyczności podczas wprowadzania do formularza różnych reguł. Wykorzystując kontrolkę `RegularExpressionValidator`, można sprawdzić zgodność wprowadzonych danych z pewnym wzorcem, wyrażonym za pomocą wyrażenia regularnego.

Oznacza to, że można zdefiniować pewną strukturę, która będzie potem wykorzystywana do sprawdzania zgodności z danymi wprowadzonymi przez użytkownika. Można na przykład zdefiniować strukturę, która wymusza wprowadzenie danych w postaci adresu e-mail lub adresu internetowego URL. Jeżeli wartość nie jest zgodna z definicją, wtedy strona traktowana jest jako nieprawidłowa. Na listingu 4.13 pokazano, w jaki sposób sprawdzić poprawność danych wprowadzonych do pola edycji pod kątem ich zgodności z formatem adresu e-mail.

Listing 4.13. Sprawdzenie, czy pole edycji zawiera właściwy adres e-mail

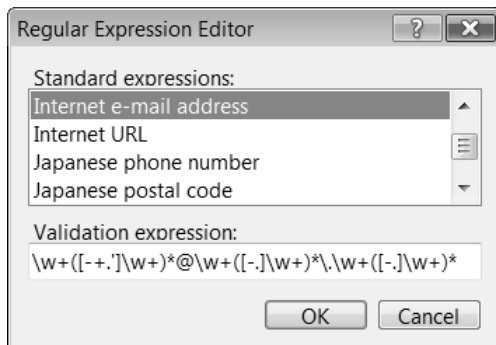
```
E-mail:
<asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
  &nbsp;
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
  Runat="server" ControlToValidate="TextBox1"
  Text="Musisz wpisać właściwy adres e-mail"
  ValidationExpression="\w+([-.\w+)*@\w+([-.\w+)*\.\w+([-.\w+)*]">
</asp:RegularExpressionValidator>
```

Tak jak to jest w przypadku innych walidacyjnych kontroltek serwerowych, kontrolka `RegularExpressionValidator` używa właściwości `ControlToValidate` do połączenia się z inną kontrolką. Zawiera także właściwość `Text`, która zawiera tekst wyświetlany na ekranie po niepomyślnym zakończeniu się procesu walidacji. Unikatową właściwością tej kontrolki jest `ValidationExpression`. Właściwość przechowuje wartość tekstową, która jest wyrażeniem regularnym stosowanym przez kontrolkę do sprawdzania poprawności danych wejściowych.

Dzięki Visual Studio 2008 praca z wyrażeniami regularnymi jest nieco łatwiejsza. Wszystko to dzięki modułowi *Regular Expression Editor*. W edytorze umieszczono kilka najczęściej używanych wyrażeń regularnych, które można zastosować w kontrolce `RegularExpressionValidator`. Aby przejść do okna edytora, należy przełączyć się na widok *Design*. Należy się jednocześnie upewnić, że kontrolka serwerowa `RegularExpressionValidator1` w widoku *Design* jest podświetlona i pokazane są jej właściwości. Teraz, aby uruchomić *Regular Expression Editor*, należy w oknie *Properties Visual Studio* kliknąć obok właściwości `ValidationExpression`. Edytor pokazany jest na rysunku 4.6.

Korzystając z tego edytora, można odnaleźć wyrażenia regularne dla takich wartości, jak na przykład adresy e-mail, URL, kody pocztowe, numery telefonów, numery ubezpieczeniowe. Narzędzie *Regular Expression Editor* pomaga w korzystaniu z tych dość często skomplikowanych wyrażeń regularnych. Pokazną kolekcję takich wyrażeń można znaleźć w internecie. Na przykład na witrynie `RegExLib` pod adresem www.regexlib.com.

Rysunek 4.6



Kontrolka serwerowa CustomValidator

Do tej pory pokazaliśmy zestaw kontrolki walidacyjnych, które można natychmiast wykończyć. W większości przypadków pokazane kontrolki walidacyjne doskonale sprawdzają się podczas wprowadzania reguł walidacyjnych na formularzach. Czasami jednak żadna z tych kontrolki nie będzie odpowiednia i trzeba będzie polegać wyłącznie na tym, co się zrobi samemu. To właśnie do takich zastosowań stworzono kontrolkę CustomValidator.

Kontrolka CustomValidator pozwala utworzyć własne algorytmy walidacji, które będą stosowane zarówno po stronie klienta, jak i po stronie serwera. Skorzystanie z takiej możliwości pozwala sprawdzić poprawność wartości lub obliczeń wykonanych w warstwie danych (na przykład w bazie danych) lub upewnić się, czy na wejściu pojawia się liczba spełniająca pewne matematyczne założenia (na przykład czy jest parzysta, czy nieparzysta). Wszystko to można zrobić dzięki kontrolce CustomValidator.

Korzystanie z walidacji po stronie klienta

Jedną z zalet kontrolki CustomValidator wartych uwagi jest możliwość łatwego wprowadzenia mechanizmu walidacji po stronie klienta. Większość programistów ma swoje własne kolekcje funkcji JavaScript, które znajdują zastosowanie w ich aplikacjach. Dzięki kontrolce CustomValidator te funkcje mogą być szybko i łatwo zaimplementowane na stronie.

Przyjrzyjmy się prostemu formularzowi, który pobiera od użytkownika liczbę. Wykorzystano w nim kontrolkę CustomValidator do wprowadzenia walidacji po stronie klienta. Jej zadaniem jest sprawdzenie, czy liczba jest podzielna przez 5. Pokazano to na listingu 4.14.

Listing 4.14. Użycie kontrolki CustomValidator do przeprowadzenia walidacji po stronie klienta

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Label1.Text = "PRAWIDŁOWA LICZBA!"
    End Sub
</script>
```

```

<html xmlns="http://www.w3.org/1999/xhtml">
<head id="Head1" runat="server">
  <title>CustomValidator</title>

  <script type="text/javascript">
    function validateNumber(oSrc, args) {
      args.IsValid = (args.Value % 5 == 0);
    }
  </script>

</head>
<body>
  <form id="form1" runat="server">
    <div>
      <p>
        Liczba:
        <asp:TextBox ID="TextBox1"
          Runat="server"></asp:TextBox>
        &nbsp;
        <asp:CustomValidator ID="CustomValidator1"
          Runat="server" ControlToValidate="TextBox1"
          Text="Liczba musi być podzielna przez 5"
          ClientValidationFunction="validateNumber">
        </asp:CustomValidator>
      </p>
      <p>
        <asp:Button ID="Button1" OnClick="Button1_Click"
          Runat="server" Text="Button"></asp:Button>
      </p>
      <p>
        <asp:Label ID="Label1" Runat="server"></asp:Label>
      </p>
    </div>
  </form>
</body>
</html>

```

C#

```

<%@ Page Language="C#" %>

<script runat="server">

  protected void Button1_Click(Object sender, EventArgs e) {
    Label1.Text = "PRAWIŁOWA LICZBA!";
  }

</script>

```

W pokazanym formularzu można zauważyć kilka rzeczy. Po pierwsze, jest to prosty formularz; zawiera tylko jedno pole tekstowe, w którym użytkownik może wprowadzić dane. Użytkownik naciska przycisk i wywoływana jest procedura obsługi zdarzenia `Button1_Click`, która uzupełnia na stronie zawartość kontrolki `Label1`. Cała ta operacja wykonywana jest tylko wtedy, gdy na stronie wykonany zostanie proces walidacji, a dane wejściowe przejdą wszystkie testy.

Jedyną rzeczą, która wyróżnia tę stronę spośród innych tego typu, jest obecność drugiego bloku `<script>` w sekcji `<head>`. To własny kod JavaScript. Należy zauważyć, że Visual Studio 2008 umożliwia stosowanie tego typu konstrukcji. Działa to nawet wtedy, gdy zachodzi potrzeba przełączania się pomiędzy widokami *Design* oraz *Source* — to coś, z czym kiepsko radziły sobie poprzednie wersje Visual Studio. Funkcja `validateNumber` napisana w JavaScript pokazana jest poniżej:

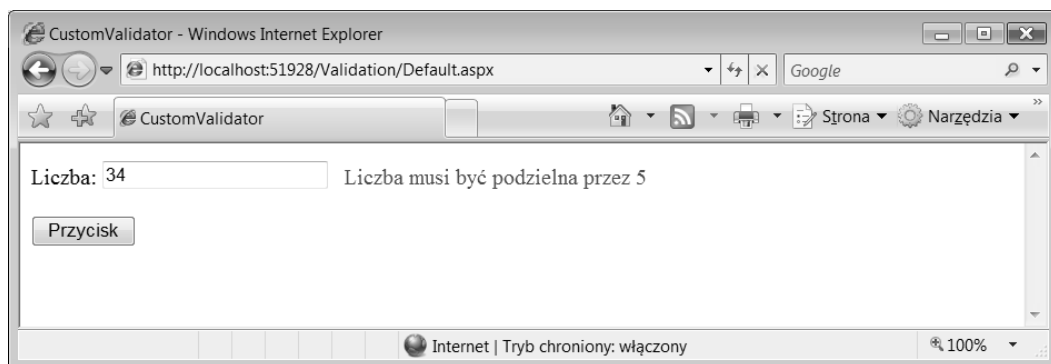
```
<script type="text/javascript">
    function validateNumber(oSrc, args) {
        args.IsValid = (args.Value % 5 == 0);
    }
</script>
```

Druga sekcja `<script>` jest kodem JavaScript używanym w kontrolce `CustomValidator` i jest wywoływana podczas przeprowadzania procesu sprawdzania poprawności danych wpisanych w polu edycji. Wstawiane funkcje JavaScript używają właściwości `args.IsValid` i ustawiają ją na `True` lub `False`, w zależności od wyniku przeprowadzonych testów sprawdzających. W tym przypadku wartość wprowadzona przez użytkownika (`args.Value`) jest sprawdzana pod kątem podzielności przez 5. Zwrócona wartość logiczna przypisywana jest do właściwości `args.IsValid`, która na ostatnim etapie tego procesu używana jest przez kontrolkę `CustomValidator`.

Kontrolka `CustomValidator`, tak jak inne kontrolki przedstawione wcześniej, korzysta z właściwości `ControlToValidate`. Za pomocą tej właściwości kontrolka może być powiązana z określonym elementem strony. Właściwością, którą należy się w tym przypadku zainteresować, jest `ClientValidationFunction`. Wartość w postaci łańcucha znaków przypisywana do tej funkcji określa funkcję po stronie klienta, która ma być wywołana w momencie sprawdzania formularza przez kontrolkę `CustomValidator`. W tym przypadku jest to `validateNumber`:

```
ClientValidationFunction="validateNumber"
```

Po uruchomieniu strony i wprowadzeniu nieprawidłowego wpisu można otrzymać wynik pokazany na rysunku 4.7.



Rysunek 4.7

Tworząc własne procedury walidacji po stronie serwera, można stworzyć tak zaawansowane algorytmy, jakie są tylko aplikacji potrzebne. Użycie walidacji po stronie serwera może się przydać wtedy, gdy zachodzi potrzeba sprawdzenia danych wprowadzonych przez użytkownika

i porównania ich z dynamicznymi wartościami pochodzącymi z plików XML, baz danych lub innego źródła.

Jako przykład własnej walidacji po stronie serwera za pomocą kontrolki `CustomValidator` może posłużyć jej odpowiednik po stronie klienta. Zróbmy to samo. Stwórzmy procedurę walidującą po stronie serwera, która sprawdza podzielność liczby przez 5. Pokazano to na listingu 4.15.

Listing 4.15. Wykorzystanie kontrolki `CustomValidator` do przeprowadzenia walidacji po stronie serwera

VB

```
<%@ Page Language="VB" %>

<script runat="server">
    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        If Page.IsValid Then
            Label1.Text = "PRAWIDŁOWY WPIS!"
        End If
    End Sub

    Sub ValidateNumber(sender As Object, args As ServerValidateEventArgs)
        Try
            Dim num As Integer = Integer.Parse(args.Value)
            args.IsValid = ((num mod 5) = 0)
        Catch ex As Exception
            args.IsValid = False
        End Try
    End Sub
</script>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
    <title>CustomValidator</title>
</head>
<body>
    <form id="form1" runat="server">
        <div>
            <p>
                Liczba:
                <asp:TextBox ID="TextBox1"
                    Runat="server"></asp:TextBox>
                &nbsp;
                <asp:CustomValidator ID="CustomValidator1"
                    Runat="server" ControlToValidate="TextBox1"
                    Text="Liczba musi być podzielna przez 5"
                    OnServerValidate="ValidateNumber"></asp:CustomValidator>
            </p>
            <p>
                <asp:Button ID="Button1" OnClick="Button1_Click"
                    Runat="server" Text="Przycisk"></asp:Button>
            </p>
            <p>
                <asp:Label ID="Label1" Runat="server"></asp:Label>
            </p>
        </div>
    </form>
```

```

</body>
</html>

C#

<%@ Page Language="C#" %>

<script runat="server">

    protected void Button1_Click(Object sender, EventArgs e) {
        if (Page.IsValid) {
            Label1.Text = "PRAWIDŁOWY WPIS!";
        }
    }

    void ValidateNumber(object source, ServerValidateEventArgs args)
    {
        try
        {
            int num = int.Parse(args.Value);
            args.IsValid = ((num%5) == 0);
        }
        catch(Exception ex)
        {
            args.IsValid = false;
        }
    }
}

</script>

```

Zamiast umieszczać w kodzie funkcję JavaScript, w tym przykładzie wykorzystano funkcję po stronie serwera — `ValidateNumber`. Funkcja `ValidateNumber`, tak jak wszystkie funkcje przystosowane do pracy z kontrolką `CustomValidator`, musi jako parametr przyjmować obiekt `ServerValidateEventArgs`. Dzięki temu do funkcji sprawdzającej zostaną przekazane właściwe dane. Funkcja `ValidateNumber` nie robi niczego ciekawego. Po prostu sprawdza, czy liczba jest podzielna przez 5.

Wewnątrz własnej funkcji, która została zaprojektowana do pracy z kontrolką `CustomValidator`, otrzymujemy wartość pochodzącą z elementu formularza poprzez obiekt `args.Value`. Ustawiamy potem właściwość `args.IsValid` na `True` lub `False`, w zależności od wyników procedury sprawdzającej. W przykładzie można zauważyć, że `args.IsValid` ustawiane jest na `False` wtedy, gdy liczba nie jest podzielna przez 5, i wtedy, gdy wyrzucony zostanie wyjątek (może on zostać wygenerowany wtedy, gdy zamiast liczby z elementu formularza otrzymamy jakiś tekst). Po utworzeniu własnej funkcji walidującej należy przejść do kolejnego kroku. Wstawmy zatem kontrolkę `CustomValidator` na stronę, jak pokazano na poniższym listingu:

```

<asp:CustomValidator ID="CustomValidator1"
    Runat="server" ControlToValidate="TextBox1"
    Text="Liczba musi być podzielna przez 5"
    OnServerValidate="ValidateNumber"></asp:CustomValidator>

```

Aby skojarzyć kontrolkę `CustomValidator` z funkcją umieszczoną w kodzie po stronie serwera, korzysta się z atrybutu `OnServerValidate`. Wartością przypisywaną do tej właściwości jest nazwa funkcji — w tym przypadku jest to `ValidateNumber`.

Wykonanie kodu z tego przykładu spowoduje wysłanie strony z powrotem na serwer i tam zostanie wykonana walidacja (dzięki funkcji `ValidateNumber`). Potem strona jest przeładowywana i wywoływane jest zdarzenie `Page_Load`. Na listingu 4.15 można zauważyć, że sprawdzana jest wartość określająca, czy strona jest prawidłowa. Wykonywane jest to za pomocą właściwości `Page.IsValid`:

```
If Page.IsValid Then
    Label1.Text = "PRAWIDŁOWY WPIS!"
End If
```

Jednoczesne użycie walidacji po stronie serwera i po stronie klienta

Jak już zostało wcześniej wspomniane, należy pomyśleć o bezpieczeństwie formularzy i zadbać o to, aby zbierane przez formularz dane były prawidłowe. Z tego powodu, decydując się na walidację po stronie klienta (pokazano to na listingu 4.14), należy jednocześnie przerobić funkcje działające po stronie klienta na ich odpowiedniki po stronie serwera. Po wykonaniu tego kroku powinno się skojarzyć funkcje po stronie klienta i po stronie serwera z kontrolką `CustomValidator`. W przypadku stosowania algorytmu walidującego z listingu 4.14 i listingu 4.15 można użyć na stronie dwóch funkcji walidacyjnych i tak zmodyfikować kontrolkę `CustomValidator`, aby wskazywała na obie te funkcje. Pokazano to na listingu 4.16.

Listing 4.16. Kontrolka `CustomValidator` z walidacją po stronie klienta i po stronie serwera

```
<asp:CustomValidator ID="CustomValidator1"
    Runat="server" ControlToValidate="TextBox1"
    Text="Liczba musi być podzielna przez 5"
    ClientValidationFunction="validateNumber"
    OnServerValidate="ValidateNumber"></asp:CustomValidator>
```

Jak widać na zaprezentowanym przykładzie, nie ma żadnego problemu związanego z jednoczesnym użyciem atrybutów `ClientValidationFunction` oraz `OnServerValidate`.

Kontrolka serwerowa `ValidationSummary`

Kontrolka `ValidationSummary` nie jest kontrolką, która przeprowadza jakiegoś rodzaju testy sprawdzające poprawność danych wpisanych na formularzu. Jest to kontrolka typowo raportująca, która jest wykorzystywana przez inne kontrolki walidacyjne na stronie. Kontrolka walidacyjna `ValidationSummary` może być użyta do zebrania wszystkich zgłoszonych błędów walidacyjnych na stronie i może stanowić alternatywę wyświetlania błędów przez każdą kontrolkę walidacyjną indywidualnie.

Taka możliwość może być wykorzystana na większych formularzach, które wykorzystują pełny proces walidacji. W takim przypadku rozwiązaniem bardziej przyjaznym dla użytkownika może być umieszczenie wszystkich zgłoszonych błędów w sposób jednolity i łatwy do ogarnięcia. Takie komunikaty o błędach mogą być wyświetlone za pomocą listy, listy wypunktowanej lub paragrafu.

Domyślnie kontrolka `ValidationSummary` pokazuje listę zgłoszonych błędów walidacji w postaci listy wypunktowanej. Pokazano to na listingu 4.17.

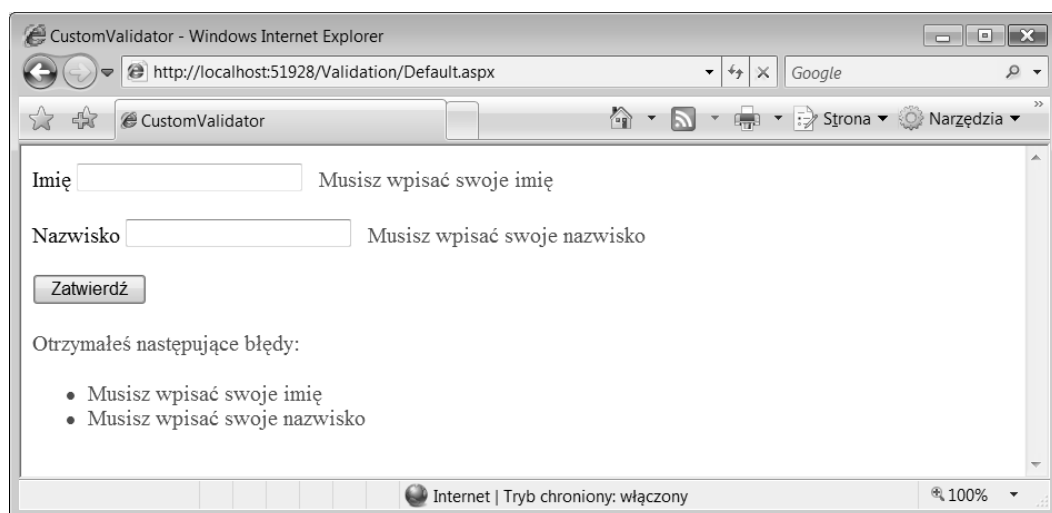
Listing 4.17. Przykład fragmentu strony korzystającej z kontrolki ValidationSummary

```

<p>Imię
  <asp:TextBox ID="TextBox1" Runat="server"></asp:TextBox>
  &nbsp;
  <asp:RequiredFieldValidator ID="RequiredFieldValidator1"
    Runat="server" ErrorMessage="Musisz wpisać swoje imię"
    ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
</p>
<p>Nazwisko
  <asp:TextBox ID="TextBox2" Runat="server"></asp:TextBox>
  &nbsp;
  <asp:RequiredFieldValidator ID="RequiredFieldValidator2"
    Runat="server" ErrorMessage="Musisz wpisać swoje nazwisko"
    ControlToValidate="TextBox2"></asp:RequiredFieldValidator>
</p>
<p>
  <asp:Button ID="Button1" OnClick="Button1_Click" Runat="server"
    Text="Zatwierdź"></asp:Button>
</p>
<p>
  <asp:ValidationSummary ID="ValidationSummary1" Runat="server"
    HeaderText="Otrzymałeś następujące błędy:">
  </asp:ValidationSummary>
</p>
<p>
  <asp:Label ID="Label1" Runat="server"></asp:Label>
</p>

```

Zadaniem przykładowego kodu jest pobranie imienia i nazwiska użytkownika. Każde pole tekstowe na formularzu ma powiązaną ze sobą kontrolkę `RequiredFieldValidator`. Po zbudowaniu i uruchomieniu strony oraz naciśnięciu przez użytkownika przycisku *Zatwierdź* bez wpisywania w pola edycji żadnych wartości powstaną dwa błędy. Wynik działania aplikacji pokazany jest na rysunku 4.8.



Rysunek 4.8

We wcześniejszych przykładach walidacji kontrolki na formularzu błędy pojawiają się obok każdego z pól tekstowych. Można jednak zauważyć, że kontrolka `ValidationSummary` pokazuje błędy walidacyjne w postaci czerwonej listy wypunktowanej w miejscu, w którym ta kontrolka się znajduje. W większości przypadków nie ma potrzeby wyświetlania tych samych błędów na stronie dwukrotnie. Można zmienić to zachowanie, korzystając nie tylko z właściwości `ErrorMessage`, ale także z właściwości `Text` kontrolki walidacyjnych, tak jak było to wykonywane w tym rozdziale do tej pory. Zaprezentowane podejście pokazano na listingu 4.18.

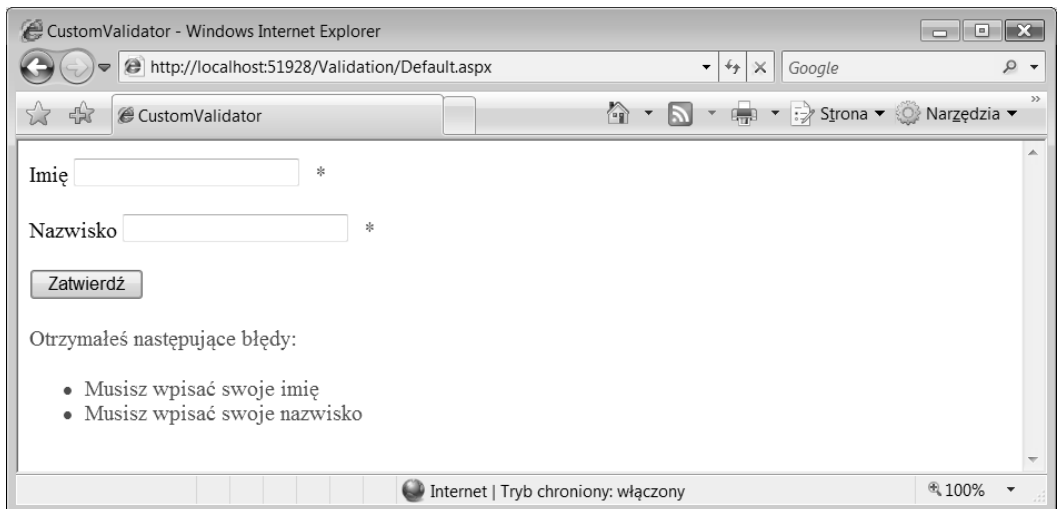
Listing 4.18. Wykorzystanie właściwości `Text` kontrolki walidacyjnej

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
  Runat="server" ErrorMessage="Musisz wpisać swoje imię" Text="*"
  ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
```

lub

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
  Runat="server" ErrorMessage="Musisz wpisać swoje imię"
  ControlToValidate="TextBox1">*</asp:RequiredFieldValidator>
```

Na listingu 4.18 pokazano dwa sposoby wykonania tego samego zadania. Pierwszy z nich polega na użyciu właściwości `Text`. Drugi polega na umieszczeniu wartości pomiędzy znacznikami elementu `<asp:RequiredFieldValidator>`. Wprowadzenie do kontrolki walidacyjnych takich zmian spowoduje powstanie wyników pokazanych na rysunku 4.9.



Rysunek 4.9

Aby otrzymać taki wynik, należy pamiętać, że kontrolka `ValidationSummary` w celu wyświetlenia błędów walidacji korzysta z właściwości `ErrorMessage`. Właściwość `Text` jest używana przez kontrolki walidacyjne i nie jest w żaden sposób przetwarzana przez kontrolkę `ValidationSummary`.

Oprócz list wypunktowanych można stosować inne formaty wyświetlania wyników. Zmian można dokonać za pomocą właściwości `DisplayMode`. Kontrolka może przyjmować następujące wartości:

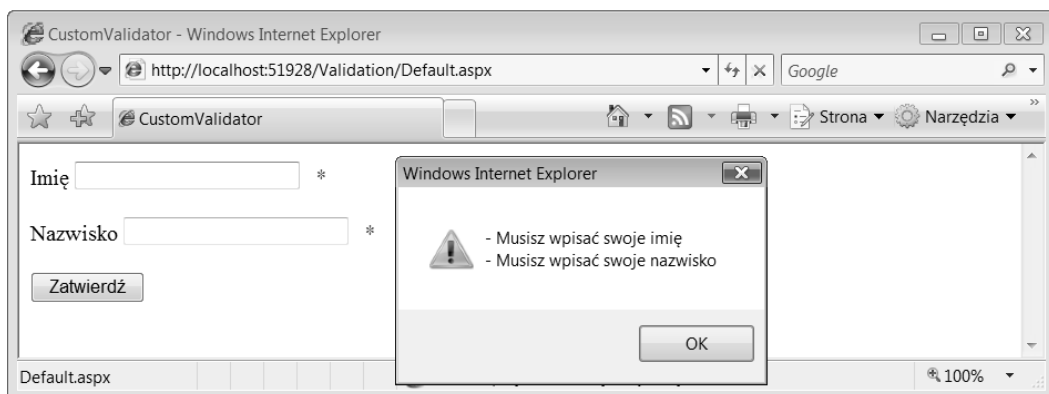
- `BulletList`,
- `List`,
- `SingleParagraph`.

Zamiast wyświetlać wyniki na stronie, można także wykorzystać okno dialogowe. Na listingu 4.19 pokazano przykład takiego podejścia.

Listing 4.19. Wykorzystanie okna dialogowego do raportowania błędów walidacji

```
<asp:ValidationSummary ID="ValidationSummary1" Runat="server"
  ShowMessageBox="True" ShowSummary="False"></asp:ValidationSummary>
```

W pokazanym przykładzie właściwość `ShowSummary` ustawiona jest na `False`, co oznacza, że błędy walidacyjne nie będą wyświetlane na stronie. Jednak w związku z tym, że właściwość `ShowMessageBox` ustawiona jest na `True`, wszystkie błędy walidacji będą wyświetlone w oknie dialogowym. Pokazano to na rysunku 4.10.



Rysunek 4.10

Wyłączanie walidacji po stronie klienta

Walidacyjne kontrolki serwerowe automatycznie przeprowadzają walidację po stronie klienta (jeżeli aplikacja wysyłająca żądanie potrafi poprawnie przetworzyć utworzony kod JavaScript). Czasami jednak zachodzi konieczność przejęcia kontroli nad tego typu zachowaniem.

Można wyłączyć walidację tych kontrolki po stronie klienta w taki sposób, aby nie wysyłały one do aplikacji żądających żadnych funkcji walidujących. Takie podejście umożliwia przeprowadzanie wyłącznie walidacji po stronie serwera, bez względu na rzeczywiste możliwości kontrolki walidacyjnych. Istnieje kilka możliwości wyłączenia tego typu funkcjonalności.

Pierwszy sposób może być zrealizowany na poziomie kontrolki. Każda z walidacyjnych kontrolki serwerowych posiada właściwość o nazwie `EnableClientScript`. Właściwość jest domyślnie ustawiona na `True`. Ustawienie właściwości na `False` blokuje wysyłanie funkcji JavaScript na stronę klienta. Zamiast tego walidacja przeprowadzana jest na serwerze. Przykład użycia tej właściwości pokazany jest na listingu 4.20.

Listing 4.20. Zablokowanie w kontrolce walidacyjnej walidacji po stronie klienta

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
Text="*" ControlToValidate="TextBox1" EnableClientScript="false">
```

Można także zablokować walidację po stronie klienta programowo. Pokazano to na listingu 4.21.

Listing 4.21. Programowe blokowanie walidacji po stronie klienta

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    RequiredFieldValidator1.EnableClientScript = False
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e)
    RequiredFieldValidator1.EnableClientScript = false;
End Sub
```

Kolejną opcją jest wyłączenie walidacji na kliencie dla wszystkich kontrolki na stronie w zdarzeniu `Page_Load`. Może to być przydatne wtedy, gdy trzeba dynamicznie zdecydować o zezwoleniu na walidację po stronie klienta lub o jej zablokowaniu. Pokazano to na listingu 4.22.

Listing 4.22. Blokowanie walidacji po stronie klienta dla wszystkich kontrolki w zdarzeniu `Page_Load`

VB

```
Protected Sub Page_Load(ByVal sender As Object, ByVal e As System.EventArgs)
    For Each bv As BaseValidator In Page.Validators
        bv.EnableClientScript = False
    Next
End Sub
```

C#

```
protected void Page_Load(object sender, EventArgs e) {
    foreach(BaseValidator bv in Page.Validators)
    {
        bv.EnableClientScript = false;
    }
}
```

Instrukcja `For Each` przeglądająca każdą instancję klasy `BaseValidator` w składowej `Validators` strony ASP.NET może wyłączyć walidację po stronie klienta wszystkich kontrolki, które zostały umieszczone na tej stronie.

Korzystanie z obrazków i dźwięków w powiadomieniach o błędach

Do tej pory wyświetlaliśmy proste wiadomości tekstowe, które informowały o pojawieniu się błędów w walidacyjnych kontrolkach serwerowych. W większości przypadków takie rozwiązanie jest wystarczające — prosty komunikat tekstowy informuje użytkownika końcowego o tym, że wprowadził do formularza jakąś nieprawidłową wartość, która nie spełnia nałożonych wymagań.

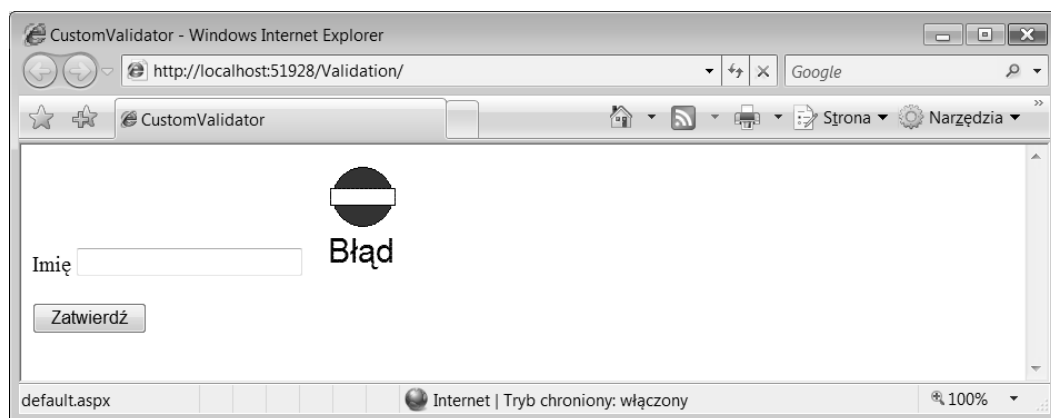
Interesującą wskazówką jest to, że nie trzeba wcale używać tekstu — w celu powiadomienia użytkownika można także używać dźwięków i obrazków.

W tym celu korzysta się z właściwości `Text`, która jest dostępna w każdej kontrolce walidacyjnej. Aby do zasygnalizowania błędu użyć obrazka, można po prostu przypisać do właściwości właściwy kod HTML. Pokazano to na listingu 4.23.

Listing 4.23. Używanie obrazków do powiadomienia o błędach

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
  Runat="server" Text=''
  ControlToValidate="TextBox1"></asp:RequiredFieldValidator>
```

Jak można zauważyć na przykładzie, zamiast wyświetlać tekst na stronie, do właściwości przypisywany jest tekst HTML. Ten fragment tekstu HTML używany jest do wyświetlenia obrazka. Należy zwrócić szczególną uwagę na obecność cudzysłowów pojedynczych i podwójnych. W ten sposób po wygenerowaniu strony w przeglądarce nie pojawią się żadne błędy. Kod pokazany na listingu 4.23 spowoduje wygenerowanie strony podobnej do tej z rysunku 4.11.



Rysunek 4.11

Kolejnym interesującym rozwiązaniem jest możliwość dodania do strony powiadomienia dźwiękowego, które będzie używane przy błędach. Można to zrobić w taki sam sposób, jaki został wykorzystany przy powiadomieniu obrazkowym. Na listingu 4.24 pokazano przykład.

Listing 4.24. Wykorzystanie dźwięku do powiadomienia o błędach

```

<asp:RequiredFieldValidator ID="RequiredFieldValidator1"
  Runat="server" Text='<bgsound src="C:\Windows\Media\tada.wav">'
  ControlToValidate="TextBox1" EnableClientScript="False">
</asp:RequiredFieldValidator>

```

Dużo różnych dźwięków systemu Windows można znaleźć w katalogu C:\Windows\Media. W tym przykładzie właściwość `Text` przyjmuje wartość elementu `<bgsound>`. Zadaniem tego elementu jest umieszczenie na formularzu dźwięku (działa tylko z Internet Explorer). Dźwięk odgrywany jest tylko w momencie wywołania przez użytkownika kontrolki walidacyjnej.

Pracując z powiadomieniami dźwiękowymi, trzeba zablokować skrypty po stronie klienta. Jeżeli się tego nie zrobi, wtedy dźwięk zostanie odegrany już w momencie wczytania strony w przeglądarce, bez względu na to, czy pojawi się jakiś błąd.

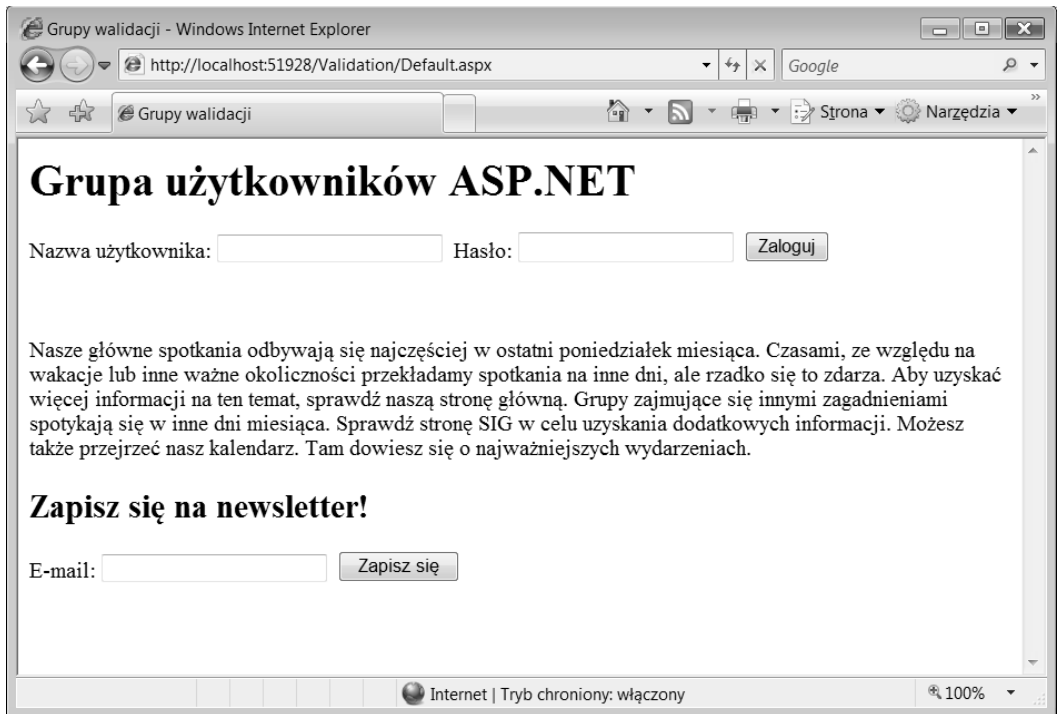
Praca z grupami walidacyjnymi

W większości przypadków programiści umieszczali na jednej stronie większą ilość formularzy. W ASP.NET 1.0/1.1 było to możliwe, ponieważ kliknięcia różnych przycisków mogły wywoływać różne zdarzenia. Korzystając z takiego rozwiązania, można się było narazić na pewne problemy.

Jeden z tych problemów polegał na tym, że pojawiały się pewne trudności z posiadaniem walidacji dla wszystkich kontrolki na każdym formularzu strony. Różne kontrolki walidacyjne były często przypisywane do dwóch różnych formularzy na stronie. Gdy użytkownik zatwierdzał jeden z formularzy, wtedy uruchamiane były procedury walidacyjne na drugim formularzu (ponieważ użytkownik nie pracował z tym formularzem). Błędy na drugiej stronie mogą więc zablokować zatwierdzenie pierwszego formularza.

Na rysunku 4.12 pokazano przykładową stronę grupy użytkowników ASP.NET, która zawiera dwa formularze.

Jeden formularz jest przeznaczony dla członków grupy. Za jego pomocą podaje się nazwę użytkownika i hasło, które są potrzebne do wejścia do strefy tylko dla członków. Drugi formularz na stronie jest przeznaczony dla wszystkich użytkowników i pozwala zapisać się do newslettera. Każdy z formularzy posiada swój przycisk i związane z nim kontrolki walidacyjne. Problem pojawia się wtedy, gdy ktoś przesyła informacje za pomocą jednego formularza. Przypuśćmy, że mamy do czynienia z członkiem grupy, który wpisuje nazwę użytkownika, hasło i naciska przycisk *Zaloguj*. Kontrolki walidacyjne dla newslettera mogą zasygnalizować błąd, ponieważ w tym formularzu nie uzupełniono adresu e-mail. Jeżeli ktoś jest zainteresowany korzystaniem z newslettera, wtedy umieszcza tylko adres e-mail w ostatnim polu i naciska przycisk *Zapisz się*. Kontrolka walidacyjna z pierwszego formularza sygnalizuje błąd, ponieważ nie wprowadzono nazwy użytkownika i hasła do pierwszego formularza.



Rysunek 4.12

W ASP.NET 3.5 pojawiła się właściwość `ValidationGroup`, która pozwala podzielić kontrolki walidacyjne na kilka niezależnych grup. W momencie naciśnięcia przycisku na stronie pozwala ona aktywować tylko określone kontrolki walidacyjne. Na listingu 4.25 pokazano przykład podzielenia kontrolki walidacyjnych na stronie grupy użytkowników na dwie różne grupy.

Listing 4.25. Korzystanie z właściwości `ValidationGroup`

```
<%@ Page Language="VB" %>

<html xmlns="http://www.w3.org/1999/xhtml" >
<head id="Head1" runat="server">
  <title>Grupy walidacji</title>
</head>
<body>
  <form id="form1" runat="server">
    <div>
      <h1>Grupa użytkowników ASP.NET</h1>
      <p>Nazwa użytkownika:
      <asp:TextBox ID="TextBox1" Runat="server">&nbsp;</asp:TextBox> &nbsp; Hasło:
      <asp:TextBox ID="TextBox2" Runat="server"
        TextMode="Password">&nbsp;</asp:TextBox> &nbsp;
      <asp:Button ID="Button1" Runat="server" Text="Zaloguj"
        ValidationGroup="Login" />
      <br />
      <asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
        Text="** Musisz wpisać nazwę użytkownika!"
```

```

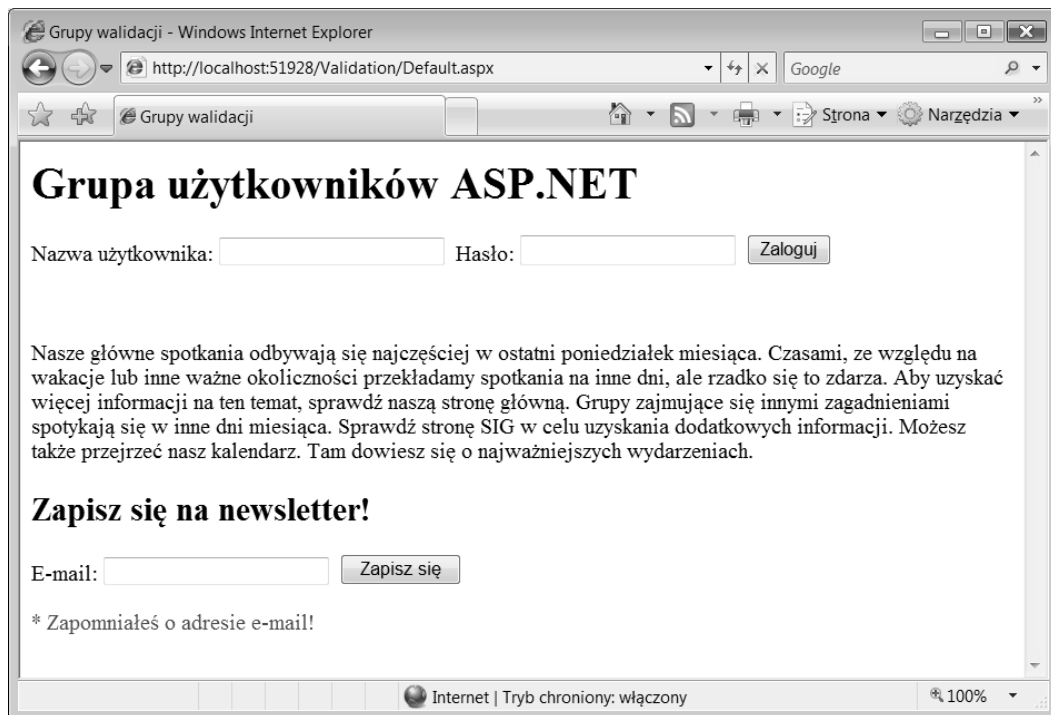
        ControlToValidate="TextBox1" ValidationGroup="Login">
</asp:RequiredFieldValidator>
<br />
<asp:RequiredFieldValidator ID="RequiredFieldValidator2" Runat="server"
    Text="* Musisz wpisać hasło!"
    ControlToValidate="TextBox2" ValidationGroup="Login">
</asp:RequiredFieldValidator>
<p>
Nasze główne spotkania odbywają się najczęściej w ostatni poniedziałek
miesiąca. Czasami, ze względu na wakacje lub inne ważne okoliczności
przekładamy spotkania na inne dni, ale rzadko się to zdarza. Aby uzyskać
więcej informacji na ten temat, sprawdź naszą stronę główną. Grupy zajmujące
się innymi zagadnieniami spotykają się w inne dni miesiąca. Sprawdź stronę
SIG w celu uzyskania dodatkowych informacji. Możesz także przejrzeć nasz
kalendarz. Tam dowiesz się o najważniejszych wydarzeniach.<br />
</p>
<h2>Zapisz się na newsletter!</h2>
<p>E-mail:
<asp:TextBox ID="TextBox3" Runat="server"></asp:TextBox>&nbsp;
<asp:Button ID="Button2" Runat="server" Text="Zapisz się"
    ValidationGroup="Newsletter" />&nbsp;
<br />
<asp:RegularExpressionValidator ID="RegularExpressionValidator1"
    Runat="server"
    Text="* Musisz wpisać właściwy adres e-mail!"
    ControlToValidate="TextBox3" ValidationGroup="Newsletter"
    ValidationExpression="\w+([-+.]\w+)*@\w+([-+.]\w+)*\.\\w+([-.]\w+)*">
</asp:RegularExpressionValidator>
<br />
<asp:RequiredFieldValidator ID="RequiredFieldValidator3" Runat="server"
    Text="* Zapomniałeś o adresie e-mail!"
    ControlToValidate="TextBox3" ValidationGroup="Newsletter">
</asp:RequiredFieldValidator>
</p>
</div>
</form>
</body>
</html>

```

Właściwość `ValidationGroup` na przykładzie została pogrubiona. Można zauważyć, że wartością właściwości jest łańcuch znaków. Warto także zwrócić uwagę na to, że nie tylko kontrolki walidacyjne posiadają tę właściwość. Kluczowe kontrolki serwerowe także mają właściwość `ValidationGroup`, ponieważ zdarzenia takie jak kliknięcia myszą także muszą być powiązane z określoną grupą walidacyjną.

W pokazanym przykładzie każdy z przycisków przypisany jest do innej grupy walidacyjnej. Pierwszy przycisk jako wartości używa `Login`, drugi z nich używa wartości `Newsletter`. Każda z kontrolki walidacyjnych także powiązana jest z określoną grupą walidacyjną. Dzięki takiemu rozróżnieniu, jeżeli użytkownik kliknie na stronie przycisk *Zaloguj*, ASP.NET jest w stanie rozpoznać grupę. Konsekwencją jest to, że przycisk pracuje wyłącznie z walidacyjnymi kontrolkami serwerowymi należącymi do tej samej grupy. ASP.NET ignoruje walidacyjne kontrolki przypisane do innych grup walidacyjnych.

Korzystając z tego udogodnienia, można zastosować wiele grup z regułami walidacyjnymi, które będą wywoływane tylko wtedy, gdy chcemy. Pokazano to na rysunku 4.13.



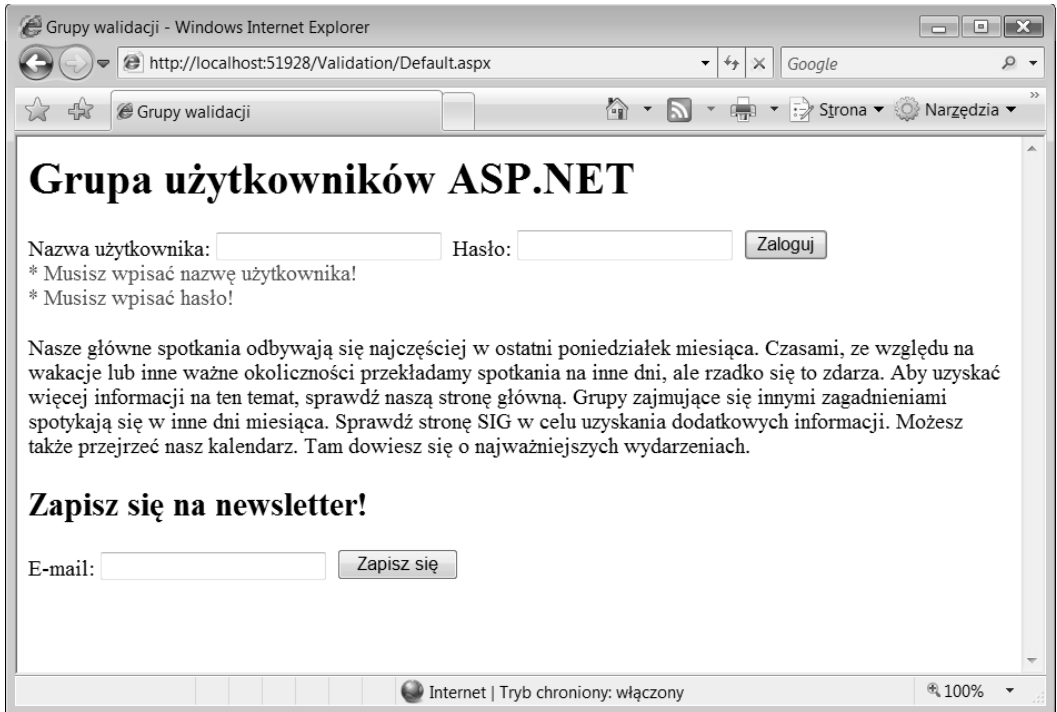
Rysunek 4.13

Kolejnym udogodnieniem dodanym do kontrolki walidacyjnej jest właściwość `SetFocusOnError`. Właściwość pobiera wartość typu logicznego. Jeżeli podczas zatwierdzania formularza sygnalizowany jest błąd, wtedy właściwość umieszcza fokus na elemencie, który ten błąd zawiera. Właściwość `SetFocusOnError` pokazana jest na poniższym listingu:

```
<asp:RequiredFieldValidator ID="RequiredFieldValidator1" Runat="server"
  Text="* Musisz wpisać nazwę użytkownika!"
  ControlToValidate="TextBox1" ValidationGroup="Login" SetFocusOnError="true">
</asp:RequiredFieldValidator>
```

Jeżeli `RequiredFieldValidator1` zasygnalizuje błąd, ponieważ użytkownik nie wstawił żadnej wartości w polu `TextBox1`, wtedy strona jest odrysowywana z fokusem ustawionym na kontrolce `TextBox1`. Pokazano to na rysunku 4.14.

Należy także zwrócić uwagę na to, że wiele kontrolki walidacyjnych może zawierać ustawioną na `True` właściwość `SetFocusOnError`. Jeżeli w takim przypadku wystąpi więcej błędów walidacji, wtedy fokus otrzyma pierwszy element formularza. Jako przykład weźmy pokazany formularz. Jeżeli zarówno pole tekstowe z nazwą użytkownika (`TextBox1`), jak i pole tekstowe z hasłem (`TextBox2`) zawierają błędy walidacji, wtedy fokus zostanie ustawiony na polu tekstowym z nazwą użytkownika. Jest to bowiem pierwsza kontrolka tego formularza z błędem.



Rysunek 4.14

Podsumowanie

Kontrolki walidacyjne są miłym dodatkiem dla tych programistów, którzy przenoszą się z Active Server Pages na ASP.NET. Skrywają one wielkie możliwości pod postacią łatwej w użyciu paczki i tak jak wiele rzeczy w świecie .NET mogą być w prosty sposób dostosowywane. Dzięki temu będą działały dokładnie tak, jak sobie tego zażyczymy.

Należy pamiętać, że celem formularzy jest zbieranie danych. Takie zbieranie danych nie ma jednak większego sensu, gdy dane są nieprawidłowe. Oznacza to, że trzeba zastosować pewne reguły sprawdzania poprawności danych, które będą implementowane na formularzach za pomocą zestawu różnych kontroltek — walidacyjnych kontroltek serwerowych.

W niniejszym rozdziale omówiono różne walidacyjne kontrolki, włączając w to:

- `RequiredFieldValidator`,
- `CompareValidator`,
- `RangeValidator`,
- `RegularExpressionValidator`,

- RegularExpressionValidator,
- CustomValidator,
- ValidationSummary.

Oprócz zaprezentowania podstawowych kontrolki walidacyjnych w rozdziale poruszono także zagadnienia związane ze stosowaniem walidacji po stronie klienta i walidacji po stronie serwera.