

IDŹ DO

PRZYKŁADOWY ROZDZIAŁ



SPIS TREŚCI

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

ASP.NET 2.0. Księga eksperta

Autor: Stephen Walther

Tłumaczenie: Robert Górczyński

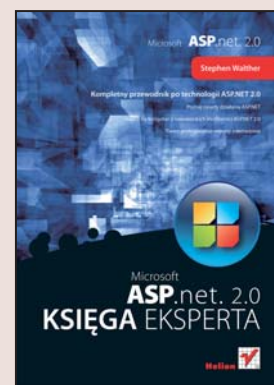
ISBN: 978-83-246-0689-4

Tytuł oryginału: [ASP.NET 2.0 Unleashed](#)

Format: B5, stron: 1024

oprawa twarda

Zawiera CD-ROM



Kompletny przewodnik po technologii ASP.NET 2.0

- Poznaj zasady działania ASP.NET
- Naucz się korzystać z nowatorskich możliwości ASP.NET 2.0
- Twórz profesjonalne witryny internetowe

ASP.NET to popularna technologia firmy Microsoft służąca do tworzenia wysoce interaktywnych i skalowalnych witryn internetowych. Jej najnowsza wersja, ASP.NET 2.0, nie tylko zawiera ponad 50 nowych kontroltek, ale również szereg innowacyjnych właściwości. Funkcje te pozwalają między innymi na dostęp do danych bez konieczności pisania kodu, łatwe nadawanie wspólnego stylu witrynom, dzięki stronom wzorcowym i tematów, budowanie portali za pomocą kontroltek Web Parts oraz wykorzystanie zalet technologii AJAX.

„ASP.NET 2.0. Księga eksperta” to profesjonalne i kompletne źródło wiedzy o tej technologii. Znajdziesz tu informacje o wszystkich wbudowanych kontrolkach ASP.NET 2.0 oraz o sposobach opracowania własnych. Nauczysz się tworzyć efektywne witryny internetowe, a także dbać o ich bezpieczeństwo oraz wysoką skalowalność. Dowiesz się, jak łatwo można dodawać nowe, jednolite strony do aplikacji sieciowych oraz zarządzać ich stanem. Przeczytasz o efektywnej obsłudze bufora w celu poprawy wydajności witryny. Poznasz także wszystkie nowe możliwości ASP.NET 2.0.

Do książki dołączona jest płyta z przykładowym kodem w językach VB.NET i C#, który możesz szybko wykorzystać na własnych witrynach. „ASP.NET 2.0. Księga eksperta” to prawdziwa skarbnica informacji, którą powinien mieć każdy programista ASP.NET.

- Działanie platformy ASP.NET
- Stosowanie wbudowanych kontroltek
- Tworzenie własnych kontroltek
- Sprawdzanie poprawności danych
- Nadawanie stylu witrynie przy użyciu stron wzorcowych i tematów
- Dostęp do danych i wyświetlanie ich
- Tworzenie własnych komponentów
- Obsługa nawigacji po witrynie
- Zapewnianie bezpieczeństwa witryny
- Obsługa stanu aplikacji
- Tworzenie portali przy użyciu kontroltek Web Parts
- Wykorzystanie technologii AJAX

Wykorzystaj potencjał ASP.NET 2.0 i twórz lepsze witryny internetowe

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



Spis treści

O autorze	21
Wprowadzenie	23
Część I Tworzenie stron ASP.NET	29
Rozdział 1. Ogólny opis platformy ASP.NET	31
ASP.NET i platforma .NET	34
Zrozumienie biblioteki Framework Class Library	34
Zrozumienie środowiska Common Language Runtime	40
Zrozumienie kontrolek ASP.NET	42
Ogólny opis kontrolek ASP.NET	42
Zrozumienie kontrolek HTML	44
Zrozumienie i obsługa zdarzeń kontrolek	45
Zrozumienie stanu widoku	49
Zrozumienie stron ASP.NET	51
Zrozumienie zagadnień kompilacji dynamicznej	51
Zrozumienie drzewa kontrolek	53
Używanie stron ukrytego kodu	54
Obsługa zdarzeń strony	57
Używanie właściwości Page.IsPostBack	59
Debugowanie i śledzenie stron ASP.NET	60
Śledzenie wykonywania strony	63
Instalowanie platformy ASP.NET	66
Podsumowanie	67
Rozdział 2. Używanie kontrolek standardowych	69
Wyświetlanie informacji	69
Używanie kontrolki Label	69
Używanie kontrolki Literal	72
Przyjmowanie danych wejściowych od użytkownika	74
Używanie kontrolki TextBox	74
Używanie kontrolki CheckBox	78
Używanie kontrolki RadioButton	80
Wysyłanie danych formularzy WWW	82
Używanie kontrolki Button	82
Używanie kontrolki LinkButton	83
Używanie kontrolki ImageButton	85

Używanie skryptów po stronie klienta z kontrolkami Button	88
Przekazywanie danych między stronami	89
Określanie przycisku domyślnego	90
Obsługiwanie zdarzenia Command	90
Wyświetlanie obrazków	91
Używanie kontrolki Image	91
Używanie kontrolki ImageMap	92
Używanie kontrolki Panel	95
Używanie kontrolki HyperLink	98
Podsumowanie	98

Rozdział 3. Używanie kontrolki sprawdzania poprawności danych ... 101

Ogólny opis kontrolki sprawdzania poprawności danych	101
Kontrolki sprawdzania poprawności danych i JavaScript	103
Używanie właściwości Page.IsValid	104
Ustawianie właściwości Display	105
Podświetlanie błędów dotyczących sprawdzania poprawności danych	106
Używanie grup sprawdzania poprawności	108
Wyłączenie sprawdzania poprawności danych	109
Używanie kontrolki RequiredFieldValidator	110
Używanie kontrolki RangeValidator	112
Używanie kontrolki CompareValidator	113
Używanie kontrolki RegularExpressionValidator	116
Używanie kontrolki CustomValidator	118
Używanie kontrolki ValidationSummary	120
Tworzenie własnych kontrolki sprawdzania poprawności danych	123
Tworzenie kontrolki LengthValidator	124
Tworzenie kontrolki AjaxValidator	125
Podsumowanie	127

Rozdział 4. Używanie kontrolki zaawansowanych 129

Przyjmowanie plików przekazywanych do serwera	130
Zapisywanie plików w systemie plików	131
Zapisywanie plików w bazie danych	132
Przekazywanie dużych plików	134
Wyświetlanie kalendarza	136
Tworzenie mechanizmu do wybierania daty	139
Generowanie kalendarza na podstawie danych z tabeli bazy danych	140
Wyświetlanie reklam	141
Przechowywanie listy reklam w pliku XML	142
Przechowywanie listy reklam w tabeli bazy danych	143
Śledzenie właściwości Impressions oraz przekierowań użytkowników na witrynę reklamodawcy	144
Wyświetlanie różnych widoków strony	146
Wyświetlanie strony z zastosowaniem zakładek	146
Wyświetlanie formularza składającego się z wielu części	147
Wyświetlanie kreatora	148
Podsumowanie	151

Część II Projektowanie witryn internetowych ASP.NET153**Rozdział 5. Projektowanie witryn internetowych z wykorzystaniem stron wzorcowych 155**

Tworzenie stron wzorcowych	156
Tworzenie zawartości domyślnej	158
Zagnieżdżanie stron wzorcowych	159
Używanie obrazków i łączy na stronach wzorcowych	160
Rejestrowanie stron wzorcowych w pliku konfiguracyjnym aplikacji	162
Modyfikowanie zawartości strony wzorcowej	163
Używanie atrybutu Title	164
Używanie właściwości Page Header	164
Udostępnianie właściwości strony wzorcowej	165
Używanie metody FindControl na stronach wzorcowych	166
Dynamiczne ładowanie stron wzorcowych	166
Dynamiczne wczytywanie stron wzorcowych na wielu stronach z treścią	169
Podsumowanie	170

Rozdział 6. Projektowanie witryn internetowych z zastosowaniem tematów 171

Tworzenie tematów	172
Dodawanie skórek do tematów	172
Tworzenie skórek o danych nazwach	174
Atrybut Theme kontra atrybut StylesheetTheme	175
Blokowanie tematów	176
Rejestrowanie tematów w pliku konfiguracyjnym aplikacji	178
Dodawanie kaskadowych arkuszy stylów do tematów	179
Dodawanie wielu arkuszy stylów do tematu	180
Zmiana projektu strony za pomocą kaskadowych arkuszy stylów	181
Tworzenie tematów globalnych	183
Dynamiczne stosowanie tematów	184
Dynamiczne stosowanie skórek	185
Podsumowanie	187

Rozdział 7. Tworzenie własnych kontroltek za pomocą kontroltek użytkownika 189

Tworzenie kontroltek użytkownika	190
Rejestrowanie kontrolki użytkownika w pliku konfiguracyjnym aplikacji	191
Udostępnianie właściwości kontrolki użytkownika	192
Udostępnianie zdarzeń kontrolki użytkownika	193
Tworzenie kontrolki AddressForm	194
Technologia AJAX i kontrolki użytkownika	195
Dynamiczne wczytywanie kontroltek użytkownika	197
Używanie dyrektywy Reference	198
Tworzenie kreatora składającego się z wielu stron	199
Podsumowanie	202

Część III Dostęp do danych203**Rozdział 8. Ogólny opis zagadnień dostępu do danych 205**

Używanie kontrolki DataBound	205
Praca z kontrolkami List	206
Praca z tabelarycznymi kontrolkami DataBound	207
Praca z hierarchicznymi kontrolkami DataBound	208
Praca z innymi kontrolkami	209
Używanie kontrolki DataSource	210
Używanie parametrów ASP.NET z kontrolkami DataSource	212
Używanie programowego dołączania danych	213
Zrozumienie szablonów i wyrażeń dotyczących dołączania danych	215
Używanie szablonów	216
Używanie wyrażeń dołączania danych	217
Używanie dwukierunkowych wyrażeń dołączania danych	220
Ogólny opis bazy danych SQL Server 2005 Express	221
Funkcje bazy danych SQL Server Express	221
Narzędzia do zarządzania bazą danych SQL Server 2005 Express	222
Serwerowe bazy danych kontra lokalne bazy danych	224
Przykładowa aplikacja WWW wykorzystująca bazę danych	227
Podsumowanie	230

Rozdział 9. Używanie kontrolki SqlDataSource 231

Nawiązywanie połączeń z bazą danych	232
Nawiązywanie połączenia z bazą Microsoft SQL Server	232
Nawiązywanie połączenia z innymi bazami danych	234
Przechowywanie ciągów tekstowych połączenia w pliku konfiguracyjnym aplikacji	235
Szyfrowanie ciągów tekstowych połączenia	236
Wykonywanie poleceń bazy danych	237
Wykonywanie poleceń SQL	238
Wykonywanie procedur składowanych	239
Filtrowanie rekordów bazy danych	240
Zmiana trybu źródła danych	241
Obsługa błędów wynikających z wykonywania poleceń SQL	242
Anulowanie wykonywania polecenia	244
Używanie parametrów ASP.NET z kontrolką SqlDataSource	244
Używanie obiektu ASP.NET Parameter	246
Używanie obiektu ASP.NET ControlParameter	247
Używanie obiektu ASP.NET CookieParameter	249
Używanie obiektu ASP.NET FormParameter	249
Używanie obiektu ASP.NET ProfileParameter	250
Używanie obiektu QueryStringParameter	252
Używanie obiektu SessionParameter	252
Programowe wykonywanie poleceń kontrolki SqlDataSource	253
Dodawanie parametrów ADO.NET	253
Wykonywanie poleceń Insert, Update i Delete	254
Wykonywanie poleceń Select	254
Buforowanie danych za pomocą kontrolki SqlDataSource	256
Podsumowanie	257

Rozdział 10. Używanie kontroltek List	259
Ogólny opis kontroltek List	259
Deklarowanie elementów ListItem	259
Dołączanie do źródła danych	261
Określanie wybranego elementu listy	263
Dodawanie elementów danych	264
Włączenie automatycznego odświeżania	265
Używanie zbioru Items	266
Praca z kontrolką DropDownList	267
Praca z kontrolką RadioButtonList	268
Praca z kontrolką ListBox	269
Praca z kontrolką CheckBoxList	271
Praca z kontrolką BulletedList	272
Utworzenie własnej kontrolki List	275
Podsumowanie	277
Rozdział 11. Używanie kontrolki GridView	279
Podstawy kontrolki GridView	279
Wyświetlanie danych	280
Zaznaczanie danych	281
Używanie kluczy danych	282
Sortowanie danych	283
Stronicowanie danych	286
Edytowanie danych	289
Wyświetlanie pustych danych	292
Formatowanie kontrolki GridView	293
Używanie stanu widoku z kontrolką GridView	295
Używanie pól w kontrolce GridView	296
Używanie pól BoundField	297
Używanie pól CheckBoxField	299
Używanie pól CommandField	299
Używanie pól ButtonField	301
Używanie pól HyperLinkField	303
Używanie pól ImageField	304
Używanie pól TemplateField	306
Praca ze zdarzeniami kontrolki GridView	307
Podświetlanie rekordów danych kontrolki GridView	309
Wyświetlanie podsumowań kolumn	310
Wyświetlanie zagnieżdżonych formularzy typu strona główna/strona szczegółowa	310
Rozszerzanie możliwości kontrolki GridView	312
Tworzenie pola dla długiego tekstu	312
Tworzenie pola DeleteButtonField	314
Tworzenie pola ValidatedField	315
Podsumowanie	316
Rozdział 12. Używanie kontroltek DetailsView i FormView	317
Używanie kontrolki DetailsView	317
Wyświetlanie danych za pomocą kontrolki DetailsView	317
Używanie pól w kontrolce DetailsView	318
Wyświetlanie pustych danych za pomocą kontrolki DetailsView	320

Stronicowanie danych w kontrolce DetailsView	320
Uaktualnianie danych za pomocą kontrolki DetailsView	324
Wstawianie danych za pomocą kontrolki DetailsView	328
Usuwanie danych za pomocą kontrolki DetailsView	328
Praca ze zdarzeniami kontrolki DetailsView	329
Formatowanie kontrolki DetailsView	331
Używanie kontrolki FormView	333
Wyświetlanie danych za pomocą kontrolki FormView	333
Stronicowanie danych w kontrolce FormView	334
Edytowanie danych za pomocą kontrolki FormView	336
Wstawianie danych za pomocą kontrolki FormView	338
Usuwanie danych za pomocą kontrolki FormView	339
Podsumowanie	340

Rozdział 13. Używanie kontrolki Repeater i DataList 341

Używanie kontrolki Repeater	341
Wyświetlanie danych za pomocą kontrolki Repeater	342
Używanie szablonów w kontrolce Repeater	343
Obsługa zdarzeń kontrolki Repeater	345
Używanie kontrolki DataList	346
Wyświetlanie danych za pomocą kontrolki DataList	346
Wyświetlanie danych w wielu kolumnach	348
Używanie szablonów w kontrolce DataList	348
Zaznaczanie danych w kontrolce DataList	350
Edytowanie danych za pomocą kontrolki DataList	351
Formatowanie kontrolki DataList	353
Podsumowanie	354

Część IV Tworzenie komponentów 355

Rozdział 14. Tworzenie komponentów 357

Tworzenie komponentów podstawowych	358
Komponenty a kompilacja dynamiczna	359
Łączenie w katalogu App_Code komponentów napisanych w różnych językach	360
Deklarowanie metod	361
Deklarowanie pól i właściwości	362
Deklarowanie konstruktorów	364
Przeciążanie metod i konstruktorów	365
Deklarowanie przestrzeni nazw	366
Tworzenie klas częściowych	367
Dziedziczenie i klasy MustInherit	368
Deklarowanie interfejsów	369
Używanie modyfikatorów dostępu	370
Lista IntelliSense a komponenty	370
Używanie ASP.NET wewnątrz komponentu	371
Tworzenie bibliotek komponentów	372
Kompilowanie bibliotek komponentów	373
Dodawanie odniesienia do biblioteki klas	377
Zagadnienia dotyczące architektury komponentów	381
Tworzenie aplikacji składającej się z wielu warstw	382
Tworzenie warstwy interfejsu użytkownika	383

Tworzenie warstwy logiki biznesowej	383
Tworzenie warstwy dostępu do danych	385
Podsumowanie	385
Rozdział 15. Używanie kontrolki ObjectDataSource	387
Reprezentacja obiektów za pomocą kontrolki ObjectDataSource	387
Dołączanie danych do komponentu	388
Dołączanie danych obiektu DataReader	389
Dołączanie danych do obiektu DataSet	389
Dołączanie danych do usługi WWW	390
Używanie parametrów w kontrolce ObjectDataSource	392
Używanie różnych typów parametrów	393
Przekazywanie obiektów jako parametrów	393
Stronicowanie, sortowanie i filtrowanie danych za pomocą kontrolki ObjectDataSource	396
Używanie interfejsu stronicowania	396
Stronicowanie w źródle danych	397
Interfejs użytkownika sortowania	399
Sortowanie w źródle danych	400
Filtrowanie danych	401
Obsługa zdarzeń kontrolki ObjectDataSource	403
Dodawanie i modyfikowanie parametrów	404
Obsługiwanie błędów metod	405
Obsługiwanie zdarzenia ObjectCreating	406
Kwestie współbieżności a kontrolki ObjectDataSource	406
Rozbudowa kontrolki ObjectDataSource	407
Tworzenie własnej kontrolki ObjectDataSource	408
Tworzenie własnych obiektów parametrów	409
Podsumowanie	410
Rozdział 16. Tworzenie komponentów dostępu do danych	413
Model Connected Data Access	414
Używanie obiektu Connection	416
Używanie obiektu Command	422
Używanie obiektu DataReader	429
Model Disconnected Data Access	433
Używanie obiektu DataAdapter	434
Używanie obiektu DataTable	436
Używanie obiektu DataView	440
Używanie obiektu DataSet	441
Wykonywanie asynchronicznych poleceń bazy danych	442
Używanie asynchronicznych metod ASP.NET	443
Używanie asynchronicznych stron ASP.NET	444
Tworzenie obiektów bazy danych za pomocą platformy .NET	446
Włączenie integracji CLR	446
Tworzenie za pomocą platformy .NET typów zdefiniowanych przez użytkownika	447
Tworzenie warstwy dostępu do danych za pomocą typu zdefiniowanego przez użytkownika	450
Tworzenie procedur składowanych za pomocą platformy .NET	452
Utworzenie podzespołu procedury składowanej	452
Podsumowanie	454

Część V Nawigacja po witrynie internetowej 457**Rozdział 17. Używanie kontrolki nawigacyjnych 459**

Zrozumienie zagadnienia mapy witryny internetowej	459
Używanie kontrolki SiteMapPath	460
Formatowanie kontrolki SiteMapPath	463
Używanie kontrolki Menu	464
Deklaracyjne dodawanie elementów menu	465
Używanie kontrolki Menu z kontrolką MultiView	467
Dołączanie mapy witryny	468
Dołączanie do pliku XML	469
Dołączanie danych pochodzących z bazy danych	471
Formatowanie kontrolki Menu	472
Używanie szablonów w kontrolce Menu	476
Używanie kontrolki TreeView	477
Deklaracyjne dodawanie węzłów drzewka	478
Wyświetlanie pól wyboru w kontrolce TreeView	479
Dołączanie do mapy witryny	480
Dołączanie do pliku XML	481
Dołączanie do danych pochodzących z bazy danych	482
Używanie funkcji Populate On Demand oraz technologii AJAX	483
Formatowanie kontrolki TreeView	486
Budowanie kontrolki hierarchicznego źródła danych SQL	490
Podsumowanie	492

Rozdział 18. Używanie mapy witryny 493

Używanie kontrolki SiteMapDataSource	493
Ustawianie właściwości kontrolki SiteMapDataSource	494
Używanie klasy SiteMap	496
Używanie klasy SiteMapNode	497
Zaawansowana konfiguracja mapy witryny	500
Używanie funkcji Security Trimming	500
Łączenie wielu map witryn	503
Tworzenie własnych atrybutów mapy witryny	504
Tworzenie własnych dostawców mapy witryny	506
Tworzenie dostawcy AutoSiteMapProvider	507
Tworzenie dostawcy SqlSiteMapProvider	508
Generowanie pliku Google SiteMap	511
Podsumowanie	513

Rozdział 19. Nawigacja zaawansowana 515

Ponowne odwzorowanie adresów URL	515
Tworzenie własnego modułu UrlRemapper	517
Używanie klasy VirtualPathProvider	520
Ograniczenia klasy VirtualPathProvider	521
Zrozumienie klasy VirtualPathProvider	521
Rejestrowanie klasy VirtualPathProvider	523
Przechowywanie witryny internetowej w bazie danych Microsoft SQL Server	524
Podsumowanie	526

Część VI Bezpieczeństwo529**Rozdział 20. Używanie kontrolki Login531**

Ogólny opis kontrolki Login	532
Używanie kontrolki Login	534
Automatyczne przekierowanie użytkownika na stronę logowania	535
Automatyczne ukrywanie kontrolki Login przed uwierzytelnionymi użytkownikami	536
Używanie szablonu w kontrolce Login	536
Przeprowadzanie własnego uwierzytelniania w kontrolce Login	538
Używanie kontrolki CreateUserWizard	539
Konfigurowanie pól formularza kontrolki CreateUserWizard	540
Wysyłanie wiadomości e-mail po utworzeniu użytkownika	542
Automatyczne przekierowanie użytkownika ze strony logowania	544
Automatyczne generowanie hasła	544
Używanie szablonów w kontrolce CreateUserWizard	546
Dodawanie kroków do kontrolki CreateUserWizard	548
Używanie kontrolki LoginStatus	548
Używanie kontrolki LoginName	549
Używanie kontrolki ChangePassword	550
Wysyłanie w wiadomości e-mail zmienionego hasła	551
Używanie szablonów w kontrolce ChangePassword	552
Używanie kontrolki PasswordRecovery	553
Wysyłanie oryginalnego hasła użytkownika	553
Wymaganie pytania bezpieczeństwa i odpowiedzi na nie	555
Używanie szablonów w kontrolce PasswordRecovery	556
Używanie kontrolki LoginView	556
Używanie ról w kontrolce LoginView	557
Podsumowanie	558

Rozdział 21. Używanie mechanizmu ASP.NET Membership559

Konfigurowanie uwierzytelniania	560
Konfigurowanie uwierzytelniania na podstawie formularzy	561
Używanie uwierzytelniania na podstawie formularzy bez plików cookie	563
Używanie opcji SlidingExpiration w uwierzytelnianiu na podstawie formularzy	564
Używanie uwierzytelniania na podstawie formularzy pomiędzy aplikacjami	565
Używanie uwierzytelniania na podstawie formularzy pomiędzy domenami	567
Używanie klasy FormsAuthentication	569
Używanie klasy User	571
Konfigurowanie autoryzacji	572
Autoryzacja na podstawie roli	574
Autoryzowanie plików na podstawie ich położenia	574
Używanie autoryzacji w przypadku obrazków i innego rodzaju plików	575
Używanie autoryzacji w klasycznych stronach ASP	577
Używanie mechanizmu ASP.NET Membership	578
Używanie interfejsu programowania aplikacji mechanizmu Membership	579
Szyfrowanie oraz kodowanie haseł użytkowników	582
Modyfikowanie wymagań dotyczących haseł użytkowników	584
Blokowanie złych użytkowników	585

Konfigurowanie dostawcy SqlMembershipProvider	586
Konfigurowanie dostawcy ActiveDirectoryMembershipProvider	588
Tworzenie własnego dostawcy mechanizmu Membership	593
Używanie menedżera ról	595
Konfigurowanie dostawcy SqlRoleProvider	596
Konfigurowanie dostawcy WindowsTokenRoleProvider	601
Konfigurowanie dostawcy AuthorizationStoreRoleProvider	602
Buforowanie ról w pliku cookie przeglądarki	604
Używanie interfejsu programowania aplikacji klasy Roles	606
Podsumowanie	608

Część VII Budowanie aplikacji ASP.NET 609

Rozdział 22. Obsługa stanu aplikacji 611

Używanie plików cookie przeglądarki internetowej	612
Ograniczenia bezpieczeństwa plików cookie	613
Tworzenie plików cookie	614
Odczytywanie plików cookie	615
Ustawianie właściwości cookie	617
Usuwanie plików cookie	618
Praca z plikami cookie zawierającymi wiele wartości	618
Używanie stanu sesji	619
Przechowywanie w stanie sesji danych pochodzących z bazy danych	620
Używanie obiektu Session	621
Obsługa zdarzeń sesji	622
Kontrolowanie momentu wygaśnięcia sesji	623
Używanie stanu sesji bez wykorzystywania mechanizmu cookies	624
Konfigurowanie magazynu stanu sesji	627
Konfigurowanie trybu SQL Server stanu sesji	630
Używanie profili	633
Tworzenie grup Profile	635
Obsługa anonimowych użytkowników	636
Migracja profilu anonimowego użytkownika	638
Dziedziczenie obiektu Profile z klasy własnej	639
Tworzenie złożonych właściwości Profile	640
Automatyczne zapisywanie obiektów Profile	642
Uzyskanie dostępu do obiektu Profile z poziomu komponentów	644
Używanie menedżera profili	645
Konfigurowanie dostawcy obiektu Profile	646
Tworzenie własnego dostawcy obiektu Profile	647
Podsumowanie	650

Rozdział 23. Buforowanie stron aplikacji oraz danych 651

Ogólny opis buforowania	651
Używanie buforowania danych wyjściowych strony	652
Zróżnicowanie bufora danych wyjściowych w zależności od parametru	653
Zróżnicowanie bufora danych wyjściowych w zależności od kontrolki	655
Zróżnicowanie bufora danych wyjściowych w zależności od nagłówka	656
Zróżnicowanie bufora danych wyjściowych w zależności od przeglądarki internetowej	657
Zróżnicowanie bufora danych wyjściowych w zależności od własnej funkcji	657

Określenie położenia bufora	658
Tworzenie zależności plików bufora danych wyjściowych strony	660
Programowe ustalenie okresu ważności bufora danych wyjściowych strony	661
Programowe operacje na buforze danych wyjściowych strony	662
Tworzenie profili buforowania danych wyjściowych strony	663
Używanie częściowego buforowania stron	664
Używanie funkcji post-cache substitution	664
Buforowanie za pomocą kontrolki użytkownika	666
Współdzielenie buforowanych danych wyjściowych kontrolki użytkownika	668
Programowa obsługa bufora kontrolki użytkownika	669
Tworzenie zależności plików bufora kontrolki użytkownika	669
Buforowanie dynamicznie wczytywanych kontrolki użytkownika	670
Używanie buforowania źródeł danych	671
Używanie polityki bezwzględnego czasu wygaśnięcia bufora	672
Używanie polityki opóźniania czasu wygaśnięcia bufora	673
Buforowanie za pomocą kontrolki ObjectDataSource	674
Buforowanie za pomocą kontrolki XmlDataSource	675
Tworzenie klucza zależności bufora kontrolki źródła danych	675
Używanie buforowania danych	676
Używanie interfejsu programowania aplikacji obiektu Cache	677
Dodawanie elementów do bufora	678
Dodawanie elementów z polityką bezwzględnego czasu wygaśnięcia ważności	678
Dodawanie elementów z polityką opóźniania czasu wygaśnięcia ważności	680
Dodawanie elementów z zależnościami	680
Określanie priorytetu elementu bufora	681
Konfigurowanie bufora	681
Używanie bufora zależności SQL	682
Używanie bufora zależności SQL w trybie monitorowania	684
Konfigurowanie bufora zależności SQL w trybie monitorowania	684
Używanie bufora zależności SQL w trybie monitorowania wraz z buforowaniem danych wyjściowych strony	687
Używanie bufora zależności SQL w trybie monitorowania wraz z buforowaniem kontrolki źródeł danych	688
Używanie bufora zależności SQL w trybie monitorowania wraz z buforowaniem danych	688
Używanie bufora zależności SQL w trybie przekazywania	689
Konfigurowanie bufora zależności SQL w trybie przekazywania	690
Używanie bufora zależności SQL w trybie przekazywania wraz z buforowaniem danych wyjściowych strony	692
Używanie bufora zależności SQL w trybie przekazywania wraz z buforowaniem kontrolki źródeł danych	693
Używanie bufora zależności SQL w trybie przekazywania wraz z buforowaniem danych	694
Podsumowanie	694
Rozdział 24. Lokalizowanie aplikacji na wiele języków	697
Ustalanie bieżącej kultury	698
Ręczne ustalanie kultury	699
Automatyczne określanie kultury	702
Ustawianie kultury w pliku konfiguracyjnym aplikacji	703
Kultura a kontrolki ASP.NET	704

Używanie klasy CultureInfo	704
Używanie klasy CultureInfo w celu sformatowania wartości ciągu tekstowego	705
Porównywanie i sortowanie wartości ciągów tekstowych	706
Tworzenie zasobów lokalnych	707
Jawne wyrażenia lokalizacyjne	707
Niejawne wyrażenia lokalizacyjne	709
Używanie zasobów lokalnych wraz z właściwościami strony	710
Programowe pobieranie zasobów lokalnych	710
Tworzenie zasobów globalnych	711
Programowe pobieranie zasobów globalnych	713
Używanie ściśle określonych wyrażen lokalizacyjnych	713
Używanie kontrolki Localize	713
Podsumowanie	714

Rozdział 25. Praca ze środowiskiem wykonawczym HTTP 715

Tworzenie własnej klasy BuildProvider	715
Tworzenie prostej klasy BuildProvider	716
Tworzenie komponentu dostępu do danych klasy BuildProvider	718
Tworzenie własnej klasy ExpressionBuilder	722
Tworzenie klasy LookupExpressionBuilder	723
Tworzenie procedur obsługi HTTP	726
Tworzenie ogólnej procedury obsługi	726
Implementowanie interfejsu IHttpHandler	727
Rejestrowanie rozszerzeń za pomocą serwera Internet Information Server	729
Tworzenie asynchronicznej procedury obsługi HTTP	732
Praca z aplikacjami oraz modułami HTTP	734
Tworzenie pliku Global.asax	735
Tworzenie własnych modułów HTTP	737
Podsumowanie	739

Rozdział 26. Konfigurowanie aplikacji 741

Ogólny opis konfiguracji witryny internetowej	741
Używanie narzędzia Web Site Administration Tool	743
Używanie narzędzia ASP.NET Microsoft Management Console Snap-In	744
Sekcje konfiguracyjne ASP.NET	746
Zastosowanie ustawień konfiguracyjnych na podanej ścieżce dostępu	748
Blokowanie ustawień konfiguracyjnych	749
Dodawanie własnych ustawień aplikacji	750
Umieszczanie ustawień konfiguracyjnych w pliku zewnętrznym	752
Używanie API konfiguracji	753
Odczytywanie sekcji konfiguracyjnych z bieżącej aplikacji	754
Otwieranie pliku konfiguracyjnego	756
Otwieranie pliku konfiguracyjnego z serwera zdalnego	756
Używanie klas konfiguracyjnych	758
Modyfikowanie sekcji konfiguracyjnych	759
Ustanawianie nowej witryny internetowej	761
Tworzenie własnych sekcji konfiguracyjnych	762
Tworzenie zbioru elementów konfiguracyjnych	764

Tworzenie zaszyfrowanych sekcji konfiguracyjnych	767
Szyfrowanie sekcji za pomocą narzędzia aspnet_regiis	768
Programowe szyfrowanie sekcji	769
Wdrażanie zaszyfrowanych plików konfiguracyjnych aplikacji	770
Podsumowanie	773

Część VIII Budowanie aplikacji za pomocą Web Parts775

Rozdział 27. Wprowadzenie do kontrolki Web Parts777

Ogólny opis platformy Web Parts	778
Obszary Web Part Zone	779
Tryby wyświetlania Web Part	780
Personalizacja Web Part	781
Tworzenie prostej aplikacji Web Part	782
Używanie obszaru Catalog Zone	789
Używanie kontrolki DeclarativeCatalogPart	789
Używanie kontrolki PageCatalogPart	791
Używanie kontrolki ImportCatalogPart	792
Używanie obszaru Editor Zone	795
Używanie kontrolki AppearanceEditorPart	796
Używanie kontrolki BehaviorEditorPart	796
Używanie kontrolki LayoutEditorPart	800
Używanie kontrolki PropertyGridEditorPart	801
Używanie obszaru Connection Zone	803
Łączenie kontrolki Web Parts	803
Łączenie prostych kontrolki Web Parts	804
Łączenie kontrolki Web Parts dołączonych do źródeł danych	806
Dynamiczne łączenie kontrolki Web Parts	808
Używanie elementu Transformer podczas łączenia kontrolki Web Parts	808
Podsumowanie	811

Rozdział 28. Tworzenie kontrolki Web Parts813

Tworzenie prostych kontrolki Web Parts	813
Kontrolka Web Parts Witaj świecie	814
Standardowe właściwości kontrolki Web Parts	816
Tworzenie klasy podstawowej dla kontrolki Web Parts na bazie kontrolki użytkownika	818
Używanie rozszerzonego zestawu właściwości kontrolki Web Parts	819
Filtrowanie kontrolki Web Parts	824
Filtrowanie za pomocą filtru autoryzacji	824
Filtrowanie za pomocą ścieżki dostępu kontrolki użytkownika	827
Filtrowanie względem rodzaju kontrolki	828
Tworzenie własnych elementów verb kontrolki Web Parts	829
Tworzenie elementów verb wykonujących kod po stronie serwera	829
Tworzenie elementów verbs wykonujących kod po stronie klienta	830
Tworzenie elementów verb na poziomie obszaru	831
Wyświetlanie stron pomocy kontrolki Web Parts	832
Zarządzanie kontrolkami Web Parts za pomocą kontrolki WebPartManager	832
Podsumowanie	838

Rozdział 29. Personalizacja kontrolki Web Parts 839

Ogólny opis personalizacji	839
Używanie klasy WebPartPersonalization	840
Tworzenie menedżera personalizacji	841
Konfigurowanie personalizacji	843
Konfigurowanie personalizacji o zasięgu użytkownika oraz współdzielonym	843
Konfigurowanie bazy danych na potrzeby personalizacji	845
Tworzenie spersonalizowanych kontrolki Web Parts	847
Praca z właściwościami złożonymi, które można personalizować	848
Używanie interfejsu IPersonalizable	851
Administrowanie personalizacją	852
Tworzenie własnych dostawców personalizacji	853
Tworzenie dostawcy QueryStringPersonalizationProvider	854
Tworzenie dostawcy AnonymousPersonalizationProvider	857
Podsumowanie	860

Rozdział 30. Rozbudowa możliwości platformy Web Parts 861

Tworzenie własnych obszarów dla kontrolki Web Parts	861
W jaki sposób funkcjonują obszary Web Part Zone?	862
Tworzenie obszaru Photo Web Part Zone	862
Tworzenie obszaru MultiColumn Web Part Zone	864
Tworzenie obszaru Menu Web Part Zone	866
Tworzenie własnych obszarów Catalog Zone	870
W jaki sposób funkcjonuje obszar Catalog Zone?	870
Tworzenie kontrolki ReflectionCatalogPart	872
Tworzenie kontrolki DragDropCatalogZone	872
Tworzenie kontrolki TemplatedCatalogZone	874
Tworzenie własnych obszarów Editor Zone	877
W jaki sposób działają obszary Editor Zone?	878
Tworzenie prostej własnej kontrolki typu Editor Part	878
Tworzenie kontrolki TemplatedEditorPart	881
Tworzenie własnych trybów wyświetlania kontrolki Web Parts	882
Podsumowanie	885

Część IX Tworzenie własnych kontrolki 887**Rozdział 31. Tworzenie własnych kontrolki 889**

Ogólny opis tworzenia własnych kontrolki	889
Tworzenie w pełni generowanych kontrolki	890
Budowanie kontrolki złożonych	895
Budowanie kontrolki hybrydowych	896
Stan widoku oraz stan kontrolki	897
Obsługiwanie stanu widoku	898
Obsługa stanu kontrolki	899
Przetwarzanie danych przekazywanych do serwera oraz zdarzeń	900
Obsługa przekazywania danych do serwera	901
Obsługa zdarzeń przekazywanych do serwera	902
Praca ze zbiorem właściwości kontrolki	906
Używanie atrybutu ParseChildren	906
Używanie metody AddParsedSubObject()	909
Używanie klasy ControlBuilder	910

Usprawnienie pracy w trybie Design	910
Zastosowanie w kontrolce atrybutów na czas projektowania	911
Tworzenie klasy ControlDesigner	913
Tworzenie klasy ContainerControlDesigner	914
Dodawanie elementów Smart Tasks	915
Podsumowanie	917

Rozdział 32. Integrowanie kodu JavaScript we własnych kontrolkach 919

Używanie klasy ClientScriptManager	920
Określanie możliwości przeglądarki internetowej	921
Tworzenie kontrolki JavaScript	923
Tworzenie kontrolki NewWindowLink	923
Tworzenie kontrolki WebWindow	925
Tworzenie kontrolki ClientTabs	926
Tworzenie kontrolki AJAX	928
Implementowanie technologii AJAX	929
Tworzenie kontrolki ServerTimeButton	930
Tworzenie kontrolki AJAX ComboBox	931
Podsumowanie	933

Rozdział 33. Tworzenie kontrolki źródeł danych, które używają szablonów 935

Tworzenie kontrolki używających szablonów	935
Implementacja interfejsu ITemplate	936
Tworzenie szablonu domyślnego	938
Obsługa uproszczonego dołączania danych	939
Obsługiwanie dwukierunkowego dołączania danych	940
Tworzenie kontrolki źródeł danych, które używają szablonów	942
Tworzenie kontrolki DivView	943
Tworzenie kontrolki AjaxDivView	944
Tworzenie kontrolki AjaxFormView	947
Podsumowanie	950

Część X Przykładowa aplikacja 951

Rozdział 34. Budowanie aplikacji typu e-commerce 953

Ogólny opis aplikacji typu e-commerce	953
Używanie stron wzorcowych, tematów oraz kontrolki użytkownika	958
Budowanie biblioteki komponentów	960
Tworzenie własnego dostawcy mapy witryny	962
Tworzenie koszyka na zakupy	963
Ochrona numerów kart kredytowych	964
Obsługa obrazków	966
Pobieranie danych za pomocą technologii AJAX	966
Poprawianie wydajności aplikacji za pomocą buforowania	967
Zgodność ze standardami WWW	968
Podsumowanie	969

Skorowidz 971

Rozdział 1.

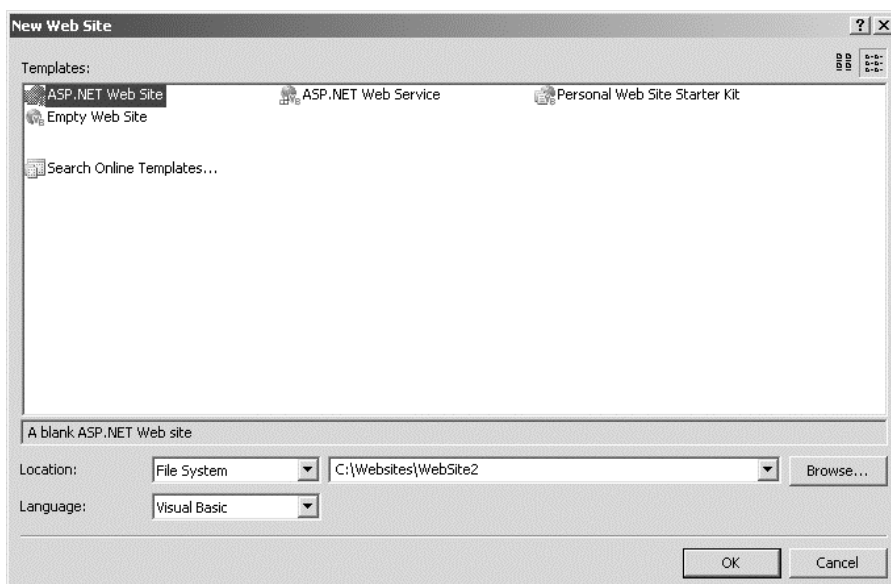
Ogólny opis platformy ASP.NET

Pracę rozpoczniemy od zbudowania prostej strony ASP.NET.



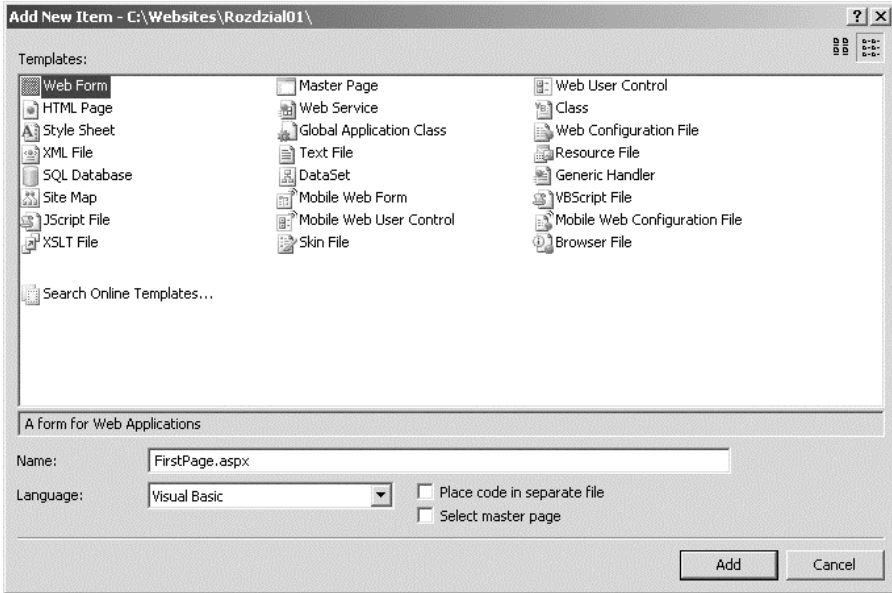
Informacje dotyczące instalowania ASP.NET znajdują się w ostatnim podrozdziale bieżącego rozdziału.

Jeżeli Czytelnik korzysta z pakietu Visual Web Developer lub Visual Studio .NET, to w pierwszej kolejności musi utworzyć nową witrynę internetową. Uruchamiamy więc pakiet Visual Web Developer i wybieramy opcję *New Web Site* z menu *File*, co spowoduje wyświetlenie okna dialogowego *New Web Site* (zobacz rysunek 1.1). W polu *Location* wyświetlonego okna dialogowego należy podać nazwę katalogu, w którym ma zostać utworzona witryna internetowa, a następnie kliknąć przycisk *OK*.



Rysunek 1.1. Tworzenie nowej witryny internetowej

Po utworzeniu nowej witryny internetowej można do niej dołączyć stronę ASP.NET. W tym celu z menu *Website* wybieramy opcję *Add New Item*. Ten krok spowoduje wyświetlenie kolejnego okna dialogowego, w którym zaznaczamy element *Web Form* oraz podajemy nazwę *FirstPage.aspx* w polu *Name*. Należy się upewnić, że w wyświetlonym oknie dialogowym nie zostało zaznaczone żadne z pól wyboru *Place code in Separate File* oraz *Select Master Page*. Kliknięcie przycisku *Add* spowoduje utworzenie nowej strony ASP.NET (zobacz rysunek 1.2).



Rysunek 1.2. Dodawanie nowej strony ASP.NET

Kod pierwszej strony ASP.NET został zawarty w pliku *Rozdzial01\FirstPage.aspx*.



Plata CD dołączona do książki zawiera również wersje napisane w języku C# wszystkich przykładów przedstawionych na stronach książki w języku Visual Basic .NET.

Zawarta w pliku *Rozdzial01\FirstPage.aspx* strona ASP.NET powoduje wyświetlenie krótkiego komunikatu powitalnego oraz aktualnej daty i godziny. Stronę tę możemy wyświetlić w przeglądarce internetowej po kliknięciu strony prawym przyciskiem myszy i wybraniu opcji *View in Browser* (zobacz rysunek 1.3).

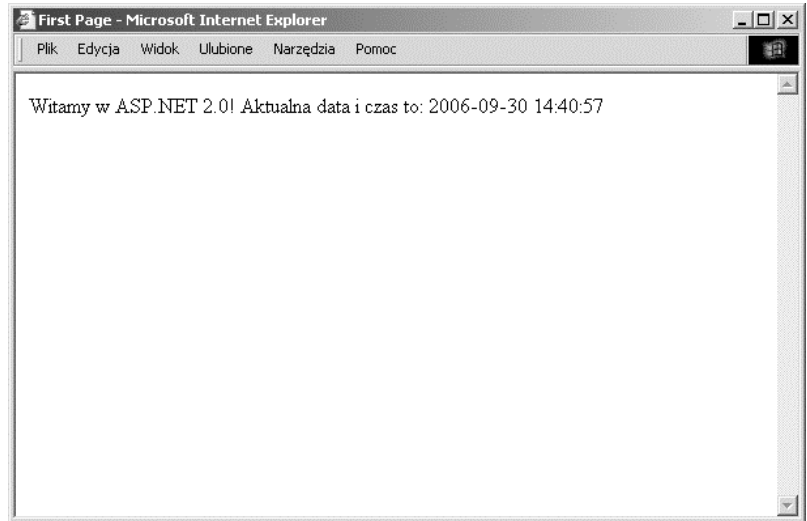
Strona *Rozdzial01\FirstPage.aspx* jest wyjątkowo prosta, jednakże prezentuje większość wspólnych elementów strony ASP.NET. Omawiana strona zawiera więc dyrektywę, blok kodu deklaracyjnego oraz blok generowania strony.

Pierwszy wiersz pliku zawiera dyrektywę strony, która przedstawia się następująco:

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

Dyrektywa zawsze rozpoczyna się znakami specjalnymi `<%@` i kończy znakami `%>`. Głównym zastosowaniem dyrektyw jest dostarczenie kompilatorowi informacji wymaganych do skompilowania strony.

Rysunek 1.3.
Wyświetlanie strony
FirstPage.aspx
w przeglądarce
internetowej



Przykładowo dyrektywa umieszczona w kodzie z pliku *Rozdział01\FirstPage.aspx* wskazuje, że kod zawarty na tej stronie jest kodem w języku Visual Basic .NET (VB.NET). Strona jest więc kompilowana przez kompilator Visual Basic .NET, a nie inny kompilator, na przykład C#.

Następna część strony rozpoczyna się od otwierającego znacznika `<script runat=server">` i kończy zamykającym znacznikiem `</script>`. Wymieniony znacznik `<script>` zawiera element, który nazywamy **blokiem kodu deklaracyjnego**.

Blok kodu deklaracyjnego zawiera wszystkie metody, funkcje i procedury wykorzystywane na stronie. Zawarty w pliku *Rozdział01\FirstPage.aspx* blok kodu deklaracyjnego zawiera pojedynczą procedurę o nazwie `Page_Load()`, która prezentuje się następująco:

```
Sub Page_Load()  
    lblServerTime.Text = DateTime.Now.ToString()  
End Sub
```

Powyższa procedura powoduje przypisanie wartości bieżącej daty i czasu do właściwości `Text` kontrolki `Label` o nazwie `lblServerTime`, która jest umieszczona w głównej części strony.

Procedura `Page_Load()` stanowi przykład obsługi zdarzeń, a ta procedura obsługuje zdarzenie `Page_Load()`. W trakcie każdego wczytania strony wymieniona procedura będzie automatycznie wykonywana i przypisze kontrolce `Label` bieżącą datę oraz czas.

Ostatnia część strony nosi nazwę **bloku generowania strony**. Wymieniony blok generowania strony zawiera to wszystko, co zostaje wygenerowane i przekazane przeglądarce internetowej. W pliku *Rozdział01\FirstPage.aspx* blok generowania strony zawiera wszystkie elementy umieszczone między otwierającym a zamykającym znacznikiem `<html>`.

Większość bloku generowania strony składa się z dobrze znanego nam kodu HTML. Przykładowo strona zawiera standardowe znaczniki `<head>` i `<body>`. W kodzie zawartym

w pliku *Rozdzial01\FirstPage.aspx* znajdują się dwa elementy specjalne, które zostały umieszczone w bloku generowania strony.

W pierwszej kolejności należy zwrócić uwagę, że strona posiada znacznik `<form>` w następującej postaci:

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

Powyższy znacznik jest przykładem kontrolki ASP.NET. Ponieważ znacznik zawiera atrybut `runat="server"`, to przedstawia kontrolkę ASP.NET, która jest wykonywana na serwerze.

Strony ASP.NET są często nazywane stronami **formularzy WWW**, ponieważ prawie zawsze zawierają element `form` wykonywany po stronie serwera.

Blok generowania strony posiada także kontrolkę `Label`. Wymieniona kontrolka `Label` została zadeklarowana za pomocą znacznika `<asp:Label>`. W pliku *Rozdzial01\FirstPage.aspx* kontrolka `Label` jest wykorzystywana do wyświetlania aktualnej daty i czasu.

Kontrolki stanowią serce platformy ASP.NET. Większość atramentu użytego w tej książce poświęcono na opisanie właściwości i funkcji kontroltek ASP.NET.

Kontrolki zostaną wkrótce przeanalizowane znacznie bardziej szczegółowo, jednakże w pierwszej kolejności Czytelnik powinien zrozumieć zasadę działania platformy .NET.



Domyślnie strony ASP.NET są zgodne ze standardem XHTML 1.0 Transitional. Czytelnik zapewne zwrócił uwagę, że strona zawarta w pliku *Rozdzial01\FirstPage.aspx* zawiera deklarację `DOCTYPE XHTML 1.0 Transitional`. Więcej informacji dotyczących tego, w jaki sposób platforma ASP.NET jest zgodna zarówno z XHTML-em, jak i standardami dostępności, znajduje się w artykule „Building ASP.NET 2.0 Web Sites Using Web Standards” w witrynie Microsoft MSDN.

ASP.NET i platforma .NET

ASP.NET jest częścią platformy Microsoft .NET. Do tworzenia stron ASP.NET programista powinien wykorzystać zalety płynące z funkcji platformy .NET. Wspomniana platforma .NET składa się z dwóch części: biblioteki Framework Class Library oraz środowiska Common Language Runtime.

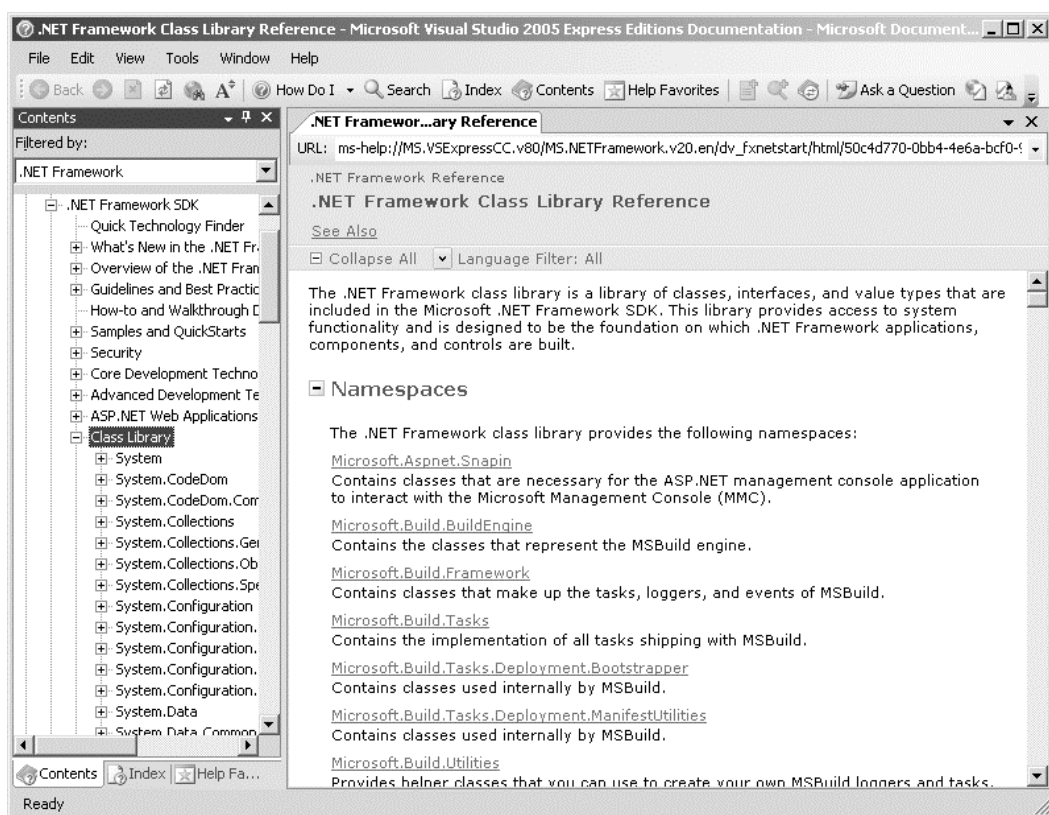
Zrozumienie biblioteki Framework Class Library

Platforma .NET składa się z tysięcy klas, które mogą zostać wykorzystane przez programistę podczas budowania aplikacji. Biblioteka Framework Class Library została zaprojektowana w celu ułatwienia wykonywania najczęstszych zadań programistycznych. Poniżej znajdują się przykłady klas umieszczonych w platformie:

- ♦ **Klasa File** — reprezentuje plik na dysku twardym. Klasa `File` może zostać użyta do sprawdzenia, czy dany plik istnieje, do utworzenia nowego pliku, do usunięcia pliku oraz do wykonania wielu innych zadań związanych z plikami.
- ♦ **Klasa Graphics** — umożliwia programiście pracę z różnymi rodzajami obrazków, na przykład w formacie *GIF*, *PNG*, *BMP* i *JPEG*. Klasa `Graphics` może być również użyta do rysowania na obrazku prostokątów, łuków, elips oraz innych elementów.
- ♦ **Klasa Random** — umożliwia generowanie liczb losowych.
- ♦ **Klasa SmtplibClient** — umożliwia wysyłanie wiadomości e-mail. Programista może użyć klasy `SmtplibClient` do wysyłania wiadomości e-mail, które posiadają załączniki oraz zawartość HTML.

Powyżej zostały przedstawione jedynie cztery klasy z platformy .NET, natomiast platforma .NET składa się z 13 000 klas, które mogą zostać wykorzystane do tworzenia aplikacji.

Wszystkie klasy umieszczone w platformie .NET można przejrzeć po otwarciu dokumentacji Microsoft .NET Framework SDK i rozwinięciu węzła Class Library (zobacz rysunek 1.4). Jeżeli dokumentacja SDK nie została zainstalowana na komputerze, wówczas Czytelnik powinien zapoznać się z ostatnim podrozdziałem bieżącego rozdziału.



Rysunek 1.4. Otworzenie dokumentacji Microsoft .NET Framework SDK



Platforma Microsoft .NET 2.0 zawiera 18 619 typów, 12 909 klas, 401 759 metod typu `public`, 93 105 właściwości typu `public` oraz 30 546 zdarzeń typu `public`.

Każda z klas platformy może zawierać właściwości, metody i zdarzenia. Właściwości, metody i zdarzenia udostępniane przez klasę są jej elementami. Przykładowo poniżej znajduje się częściowa lista elementów klasy `SmtpClient`:

- ◆ Właściwości
 - ◆ `Host` — nazwa lub adres IP serwera wiadomości e-mail
 - ◆ `Port` — numer portu używanego w trakcie wysyłania wiadomości e-mail
- ◆ Metody
 - ◆ `Send` — umożliwia programiście synchroniczne wysyłanie wiadomości e-mail
 - ◆ `SendAsync` — umożliwia programiście asynchroniczne wysyłanie wiadomości e-mail
- ◆ Zdarzenia
 - ◆ `SendCompleted` — wywoływane, gdy zakończy się operacja asynchronicznego wysyłania wiadomości.

Jeżeli programista zna element klasy, wówczas wie wszystko, co może osiągnąć za pomocą danej klasy. Przykładowo klasa `SmtpClient` zawiera dwie właściwości o nazwie `Host` i `Port`, które umożliwiają podanie serwera wiadomości e-mail oraz portu używanego w trakcie wysyłania wiadomości e-mail.

Klasa `SmtpClient` posiada również dwie metody, które mogą być wykorzystane do wysłania wiadomości: `Send()` oraz `SendAsync()`. Metoda `Send()` blokuje dalsze wykonywanie programu, dopóki operacja wysyłania nie zostanie zakończona. Z drugiej strony metoda `SendAsync()` asynchronicznie wysyła wiadomość e-mail. W przeciwieństwie do metody `Send()` metoda `SendAsync()` nie musi czekać z wykonywaniem dalszej części kodu do chwili upewnienia się, że operacja wysyłania wiadomości zakończyła się powodzeniem.

Na koniec klasa `SmtpClient` zawiera także zdarzenie o nazwie `SendCompleted`, które zostaje wywołane, gdy zakończy się operacja asynchronicznego wysyłania wiadomości. Programista może utworzyć obsługę zdarzeń dla zdarzenia `SendCompleted`, w którym będzie wyświetlał komunikat informujący o pomyślnym wysłaniu wiadomości e-mail.

Strona zawarta w pliku `Rozdzial01\SendMail.aspx` w celu wysłania wiadomości e-mail wywołuje metodę `Send()` klasy `SmtpClient`. Pierwszym parametrem jest adres nadawcy, drugim adres odbiorcy, trzecim temat wiadomości, a ostatni parametr to część główna wysyłanej wiadomości e-mail.



Strona z pliku `Rozdzial01\SendMail.aspx` powoduje wysłanie wiadomości e-mail za pomocą lokalnego serwera SMTP. Jeżeli serwer SMTP nie jest włączony, wówczas użytkownik otrzyma komunikat błędu o braku możliwości nawiązania połączenia (*An existing connection was forcibly closed by the remote host*). Włączenie lokalnego serwera SMTP następuje po otwarciu w *Panelu sterowania* elementu *Internetowe usługi informacyjne*, a następnie wybraniu węzła *Domyślny serwer wirtualny SMTP* i wybraniu z menu kontekstowego opcji *Uruchom*.

Zrozumienie przestrzeni nazw

W platformie .NET jest zawartych około 13 000 klas. Jest to oczywiście przytłaczająca liczba. Gdyby firma Microsoft umieściła wszystkie klasy razem, wówczas programista prawdopodobnie nigdy nie znalazłby niczego. Na szczęście Microsoft podzielił klasy platformy na oddzielne przestrzenie nazw.

Przestrzeń nazw jest po prostu kategorią. Przykładowo wszystkie klasy powiązane z operacjami na systemie plików są umieszczone w przestrzeni nazw `System.IO`, natomiast wszystkie klasy powiązane z pracą obejmującą wykorzystanie serwera Microsoft SQL Server zostały umieszczone w przestrzeni nazw `System.Data.SqlClient`.

Zanim klasa będzie mogła zostać użyta na stronie, programista musi wskazać przestrzeń nazw przypisaną danej klasie. Istnieje kilka sposobów realizacji tego zadania.

Po pierwsze, programista może podać pełną nazwę klasy w jej przestrzeni nazw. Przykładowo, ponieważ klasa `File` jest zawarta w przestrzeni nazw `System.IO`, to w celu sprawdzenia, czy dany plik istnieje, można użyć poniższego polecenia:

```
System.IO.File.Exists("DowolnyPlik.txt")
```

Podawanie przestrzeni nazw podczas każdego użycia klasy może bardzo szybko stać się uciążliwe (wymaga to większej ilości kodu do napisania). Inną możliwością jest więc opcja zaimportowania przestrzeni nazw.

Po drugie, programista może dodać na stronie dyrektywę `<%@ Import %>` służącą do wspomnianego powyżej zaimportowania określonej przestrzeni nazw. W pliku *Rozdział01\SendMail.aspx* została zaimportowana przestrzeń nazw `System.Net.Mail`, ponieważ klasa `SmtpClient` jest częścią wymienionej przestrzeni nazw. Na samym początku strony zawartej w pliku *Rozdział01\SendMail.aspx* znajduje się poniższa dyrektywa:

```
<%@ Import Namespace="System.Net.Mail" %>
```

Po zaimportowaniu danej przestrzeni nazw programista może korzystać z wszystkich klas umieszczonych w tej przestrzeni nazw bez konieczności podawania pełnej nazwy klasy.

Uwaga

Plik konfiguracyjny jest plikiem specjalnego typu, który może zostać dodany do aplikacji w celu jej skonfigurowania. Należy pamiętać, że plik konfiguracyjny jest zapisany w formacie XML i dlatego też, wszystkie znaczniki umieszczone w tym pliku różnią wielkość liter. Dołączenie pliku konfiguracyjnego do aplikacji następuje po wybraniu opcji *Add New Item* z menu *Webside*, a następnie zaznaczeniu elementu *Web Configuration File*. Szczegółowe omówienie plików konfiguracyjnych znajdzie się w rozdziale 26.

Na koniec, jeżeli się okaże, że określona przestrzeń nazw jest wykorzystywana na wielu stronach aplikacji, wówczas można tak zmodyfikować konfigurację, aby wszystkie strony aplikacji mogły używać danej przestrzeni nazw.

Jeżeli do aplikacji zostanie dołączony plik konfiguracyjny przedstawiony na listingu 1.1, wówczas na stronie nie trzeba będzie importować przestrzeni nazw `System.Net.Mail` w celu użycia klas z wymienionej przestrzeni nazw. Przykładowo po dołączeniu do projektu poniższego pliku *Web.Config* można będzie usunąć dyrektywę `<%@ Import %>` ze strony zawartej w pliku *Rozdział01\SendMail.aspx*.

Listing 1.1. Web.Config

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <pages>
      <namespaces>
        <add namespace="System.Net.Mail"/>
      </namespaces>
    </pages>
  </system.web>
</configuration>
```

Programista nie musi importować każdej przestrzeni nazw. Platforma ASP.NET domyślnie udostępnia większość najczęściej wykorzystywanych przestrzeni nazw, do których zaliczamy:

- ◆ System
- ◆ System.Collections
- ◆ System.Collections.Specialized
- ◆ System.Configuration
- ◆ System.Text
- ◆ System.Text.RegularExpressions
- ◆ System.Web
- ◆ System.Web.Caching
- ◆ System.Web.SessionState
- ◆ System.Web.Security
- ◆ System.Web.Profile
- ◆ System.Web.UI
- ◆ System.Web.UI.WebControls
- ◆ System.Web.UI.WebControls.WebParts
- ◆ System.Web.UI.HtmlControls

Domyślne przestrzenie nazw zostały podane w elemencie `pages` znajdującym się w nadrzędnym pliku konfiguracyjnym umieszczonym pod następującą ścieżką dostępu:

```
\\WINDOWS\Microsoft.NET\Framework\[wersja]\CONFIG\Web.Config
```

Zrozumienie podzespółów

Podzespół jest rzeczywistym plikiem `.dll` znajdującym się na dysku twardym, w którym jest przechowywana klasa z platformy .NET. Przykładowo klasy zawarte w platformie .NET są umieszczone w podzespole o nazwie *System.Web.dll*.

Mówiąc bardziej precyzyjnie, podzespół jest podstawową jednostką projektowania, bezpieczeństwa i kontrolowania wersji w platformie .NET. Ponieważ podzespół może być rozmieszczony w wielu plikach, to do podzespołu często odnosimy się jako do „logicznej” biblioteki dll.

Uwaga

Platforma .NET w wersji 2.0 posiada 51 podzespołów.

Istnieją dwa rodzaje podzespołów: prywatne i współdzielone. Podzespół prywatny może być używany jedynie przez pojedynczą aplikację. Z drugiej strony podzespół współdzielony może być używany przez wszystkie aplikacje umieszczone na tym samym serwerze.

Podzespoły współdzielone są umieszczone w *Global Assembly Cache* (GAC). Przykładowo podzespół *System.Web.dll* oraz wszystkie pozostałe podzespoły dołączone do platformy .NET są umieszczone w *Global Assembly Cache*.

Uwaga

Global Assembly Cache jest fizycznie umieszczony w katalogu `\WINDOWS\Assembly`. Oddzielna kopia każdego podzespołu znajduje się także w katalogu `\WINDOWS\Microsoft.NET\Framework\v2.0.50727`. Pierwszy z wymienionych zbiorów podzespołów jest wykorzystywany w trakcie uruchamiania aplikacji, natomiast drugi wymieniony zbiór jest używany w trakcie jej kompilowania.

Zanim programista będzie mógł w tworzonej aplikacji użyć klasy znajdującej się w podzespole, musi dodać odniesienie do danego podzespołu. Domyślnie aplikacja ASP.NET posiada odniesienia do większości najczęściej używanych podzespołów znajdujących się w *Global Assembly Cache*:

- ♦ `mscorlib.dll`
- ♦ `System.dll`
- ♦ `System.Configuration.dll`
- ♦ `System.Web.dll`
- ♦ `System.Data.dll`
- ♦ `System.Web.Services.dll`
- ♦ `System.Xml.dll`
- ♦ `System.Drawing.dll`
- ♦ `System.EnterpriseServices.dll`
- ♦ `System.Web.Mobile.dll`

W celu użycia określonej klasy z platformy .NET programista musi wykonać dwie operacje. Po pierwsze, tworzona aplikacja musi posiadać odniesienie do podzespołu, który zawiera daną klasę. Po drugie, aplikacja musi importować przestrzeń nazw, która jest przypisana do danej klasy.

W większości przypadków programista nie musi martwić się o tworzenie odniesienia do niezbędnych podzespołów, ponieważ odniesienia do najczęściej wykorzystywanych podzespołów są tworzone automatycznie. Jednakże, jeśli zachodzi konieczność użycia wyspecjalizowanego podzespołu, należy wyraźnie dodać odniesienie do tego podzespołu.

Przykładowo, jeżeli programista musi pracować z usługą *Active Directory*¹ za pomocą klas z przestrzeni nazw `System.ActiveDirectory`, wtedy w tworzonej aplikacji powinien dodać odniesienie do podzespołu `System.DirectoryServices.dll`.

Każdy klasa opisana w dokumentacji .NET Framework SDK zawiera nazwę podzespołu i przestrzeni nazw przypisanych do danej klasy. Przykładowo po wyszukaniu w dokumentacji klasy `MessageQueue` Czytelnik odkryje, że ta klasa jest umieszczona w przestrzeni nazw `System.Messaging` położonej w podzespole `System.Messaging.dll`.

Jeżeli programista używa pakietu Visual Web Developer, wówczas istnieje możliwość dołączenia odniesienia do podzespołu poprzez wybranie opcji *Add Reference* z menu *Website*, a następnie zaznaczeniu nazwy podzespołu, do którego jest wymagane utworzenie odniesienia. Przykładowo dodanie odniesienia do podzespołu `System.Messaging` powoduje dołączenie do aplikacji pliku konfiguracyjnego przedstawionego na listingu 1.2.

Listing 1.2. Web.Config

```
<?xml version="1.0"?>
<configuration>
  <system.web>
    <compilation>
      <assemblies>
        <add
          assembly="System.Messaging, Version=2.0.0.0,
            Culture=neutral, PublicKeyToken=B03F5F7F11D50A3A"/>
      </assemblies>
    </compilation>
  </system.web>
</configuration>
```

Programiści, którzy nie wykorzystują pakietu Visual Web Developer, mogą dodać odniesienie do podzespołu `System.Messaging` po prostu po przez ręczne utworzenie pliku przedstawionego na listingu 1.2.

Zrozumienie środowiska Common Language Runtime

Drugą częścią platformy .NET jest środowisko Common Language Runtime (CLR). Jest ono odpowiedzialne za wykonanie kodu tworzonej aplikacji.

W trakcie pisania aplikacji platformy .NET za pomocą języka programowania takiego jak Visual Basic .NET lub C# tworzony kod źródłowy nie będzie nigdy bezpośrednio skompilowany do postaci kodu maszynowego. Zamiast tego kompilator Visual Basic .NET lub C# skonwertuje tworzony kod do specjalnego języka o nazwie MSIL (Microsoft Intermediate Language).

¹ Active Directory — usługa katalogowa dla systemów Windows 2000 oraz Windows Server 2003 zgodna ze specyfikacją LDAP 3.0 (choć istnieje kilka odstępstw od tej specyfikacji). Więcej informacji znajduje się na stronach http://pl.wikipedia.org/wiki/Active_Directory oraz <http://www.microsoft.com/windowsserver2003/technologies/directory/activedirectory/default.mspx> (w języku angielskim) — *przyj. tłum.*

Język MSIL jest bardzo podobny do języka assemblerowego zorientowanego obiektowo, jednakże, w przeciwieństwie do typowego języka assemblerowego nie został utworzony pod kątem określonego procesora. MSIL jest więc niezależnym od platformy sprzętowej językiem niskiego poziomu.

Kiedy następuje rzeczywiste wykonanie aplikacji, kod MSIL jest „w danym momencie” kompilowany przez **JITTER** (kompilator typu *Just-In-Time*) do postaci kodu maszynowego. Zazwyczaj cała aplikacja nie jest kompilowana z języka MSIL do postaci kodu maszynowego — zamiast tego następuje kompilowanie metod, które są faktycznie wywoływane w trakcie uruchamiania aplikacji.

W rzeczywistości platforma .NET rozumie tylko jeden język, MSIL, jednakże programista może tworzyć aplikacje, wykorzystując języki takie jak Visual Basic .NET i C#, ponieważ platforma .NET zawiera kompilatory umożliwiające skompilowanie tworzonego w wymienionych językach kodu do postaci MSIL.

Kod dla platformy .NET można tworzyć, używając dziesiątek różnych języków programowania, między innymi:

- ♦ Ada
- ♦ Apl
- ♦ Caml
- ♦ COBOL
- ♦ Eiffel
- ♦ Forth
- ♦ Fortran
- ♦ JavaScript
- ♦ Oberon
- ♦ PERL
- ♦ Pascal
- ♦ PHP
- ♦ Python
- ♦ RPG
- ♦ Scheme
- ♦ Small Talk

Przeważająca większość programistów tworzących aplikacje ASP.NET wykorzystuje w tym celu albo język Visual Basic .NET albo C#. Wiele innych języków wymienionych na powyższej liście możemy zaliczyć do grupy eksperymentów czysto akademickich.

W przeszłości, jeśli Czytelnik chciał zostać programistą, musiał skupić się na biegłej znajomości danego języka programowania. Przykładowo można było zostać programistą C++, programistą COBOL-a lub programistą języka Visual Basic.

Jednakże, kiedy mówimy o platformie .NET, wtedy kwestia znajomości konkretnego języka programowania nie jest tak szczególnie istotna. Wybór wykorzystywanego języka programowania do tworzenia aplikacji .NET jest w dużym stopniu wyborem opierającym się na osobistych preferencjach i upodobaniach. Jeżeli programista lubi nawiasy klamrowe i polecenia rozróżniające wielkość liter, wówczas powinien wybrać C#, natomiast, jeśli jest nieco leniwy w kwestii dotyczącej wielkości liter i nie lubi średników, wtedy powinien pisać kod w języku Visual Basic .NET.

Wszystkie rzeczywiste działania w platformie .NET zachodzą w środowisku Framework Class Library. Jeżeli Czytelnik chciałby zostać dobrym programistą wykorzystującym technologie firmy Microsoft, wówczas powinien się nauczyć, w jaki sposób używać metod, właściwości i zdarzeń tych 13 000 klas dołączonych do platformy .NET. Z punktu widzenia platformy .NET naprawdę nie ma znaczenia, czy dane klasy są używane z poziomu aplikacji Visual Basic .NET czy aplikacji C#.



Wszystkie przykładowe fragmenty kodów przedstawione w tej książce zostały napisane zarówno w języku Visual Basic .NET, jak i C#. Przykłady w języku C# oraz VB .NET znajdują się na płycie CD dołączonej do tej książki.

Zrozumienie kontroltek ASP.NET

Kontrolki ASP.NET są sercem platformy ASP.NET. Kontrolka ASP.NET jest klasą .NET, która jest wykonywana na serwerze i generuje w przeglądarce internetowej określoną zawartość.

Przykładowo na pierwszej stronie ASP.NET, utworzonej na początku bieżącego rozdziału, kontrolka Label została użyta do wyświetlenia bieżącej daty i czasu. Platforma ASP.NET składa się z około 70-ciu kontroltek, które umożliwiają programiście realizację każdego zadania, począwszy od wyświetlania listy rekordów bazy danych aż do wyświetlania cyklicznie zmieniającego się i losowo wybranego banera reklamowego.

W tym podrozdziale przedstawiony zostanie ogólny opis kontroltek dołączonych do platformy ASP.NET. Czytelnik dowie się również, w jaki sposób obsługiwać zdarzenia wywołane przez kontrolki i jak wykorzystać zalety stanu widoku.

Ogólny opis kontroltek ASP.NET

Platforma ASP.NET w wersji 2.0 zawiera około 70 kontroltek. Te kontrolki mogą zostać podzielone na następujących osiem grup:

- ♦ **Kontrolki standardowe** — kontrolki standardowe umożliwiają wygenerowanie standardowych elementów formularza, takich jak przyciski, pola wprowadzania danych oraz etykiety. Kontrolki te zostaną szczegółowo omówione w rozdziale 2.

- ♦ **Kontrolki sprawdzania poprawności danych** — kontrolki sprawdzania poprawności danych umożliwiają weryfikację poprawności danych formularza przed wysłaniem tych danych do serwera. Przykładowo programista może wykorzystać kontrolkę `RequiredFieldValidator` do sprawdzenia, czy użytkownik podał wartość w polu, którego wypełnienie jest wymagane. Te kontrolki zostaną szczegółowo omówione w rozdziale 3.
- ♦ **Kontrolki zaawansowane** — kontrolki wzbogacone umożliwiają wygenerowanie elementów takich jak kalendarze, przyciski przesyłania plików, zmieniające się cyklicznie banery reklamowe i kreatory składające się z wielu etapów. Te kontrolki zostaną szczegółowo omówione w rozdziale 4.
- ♦ **Kontrolki danych** — kontrolki danych umożliwiają pracę z danymi, takimi jak dane pochodzące z bazy danych. Przykładowo programista może użyć tych kontrolki do wysłania nowych rekordów do tabeli bazy danych lub do wyświetlenia listy rekordów bazy danych. Kontrolki te zostaną szczegółowo omówione w trzeciej części tej książki.
- ♦ **Kontrolki nawigacyjne** — kontrolki nawigacyjne umożliwiają wyświetlanie standardowych elementów nawigacyjnych takich jak menu, drzewka widoku oraz elementy typu „jesteś tutaj”. Te kontrolki zostaną szczegółowo omówione w rozdziale 17.
- ♦ **Kontrolki logowania** — kontrolki logowania umożliwiają wyświetlanie formularzy logowania, zmiany hasła oraz formularzy rejestracyjnych. Zostaną one szczegółowo omówione w rozdziale 20.
- ♦ **Kontrolki Web Part** — kontrolki Web Part umożliwiają budowanie spersonalizowanych aplikacji portali internetowych. Kontrolki te zostaną szczegółowo omówione w ósmej części książki.
- ♦ **Kontrolki HTML** — kontrolki HTML umożliwiają konwersję dowolnego znacznika HTML do postaci kontrolki serwerowej. Ta grupa kontrolki zostanie szczegółowo przeanalizowana w kolejnym podrozdziale bieżącego rozdziału.

Za wyjątkiem kontrolki HTML deklarowanie i używanie na stronie wszystkich kontrolki ASP.NET następuje w dokładnie ten sam sposób. Przykładowo, jeżeli programista chce wyświetlić na stronie pole wprowadzania tekstu, wówczas może w następujący sposób zadeklarować kontrolkę `TextBox`:

```
<asp:TextBox id="TextBox1" runat="Server" />
```

Powyższa deklaracja kontrolki wygląda podobnie do deklaracji znacznika HTML, jednakże należy pamiętać, że w przeciwieństwie do kontrolki HTML to powyższa kontrolka jest klasą .NET, która zostanie wykonana na serwerze, a nie w przeglądarce internetowej.

Kiedy kontrolka `TextBox` jest generowana do przeglądarki internetowej, przyjmuje następującą postać:

```
<input name="TextBox1" type="text" id="TextBox1" />
```

Pierwsza część deklaracji kontrolki, prefiks `asp:`, wskazuje przestrzeń nazw dla danej kontrolki. Wszystkie kontrolki standardowe ASP.NET są zawarte w przestrzeni nazw `System.Web.UI.WebControls`. Wymieniony prefiks `asp:` reprezentuje tę przestrzeń nazw.

Następnie deklaracja zawiera nazwę deklarowanej kontrolki. W omawianym przypadku zostaje zadeklarowana kontrolka `TextBox`.

Deklaracja zawiera również atrybut `ID`, który jest używany w celu odniesienia się do danej kontrolki z poziomu kodu. Każda kontrolka musi posiadać unikalny atrybut `ID`.

Uwaga

Należy zawsze przypisywać atrybut `ID` do każdej kontrolki, nawet jeśli nie będzie konieczności uzyskania programowego dostępu do tej kontrolki. Jeśli programista nie dostarczy atrybutu `ID`, wtedy określone funkcje platformy ASP.NET (takie jak dwukierunkowe wiązanie danych) nie będą funkcjonowały.

Powyższa deklaracja kontrolki zawiera także atrybut `runat="Server"`. Wymieniony atrybut wskazuje, że znacznik przedstawia kontrolkę serwerową. Jeśli ten atrybut zostanie pominięty, wówczas kontrolka `TextBox` będzie przekazana bez wykonania do przeglądarki internetowej, która po prostu zignoruje ten znacznik.

Na końcu warto zwrócić uwagę, że kontrolka kończy się ukośnikiem. Wspomniany ukośnik jest skróconym zapisem znacznika zamykającego `</asp:TextBox>`. Programista może oczywiście zadeklarować kontrolkę `TextBox` w sposób następujący:

```
<asp:TextBox id="TextBox1" runat="Server"></asp:TextBox>
```

W takim przypadku znacznik otwierający nie posiada ukośnika, a na końcu jest umieszczony znacznik zamykający.

Zrozumienie kontrolek HTML

Kontrolki HTML są deklarowane w inny sposób niż standardowe kontrolki ASP.NET. Platforma ASP.NET umożliwiła pobranie dowolnego znacznika HTML (rzeczywistego lub wymyślnego) i dołączenie do tego znacznika atrybutu `runat="Server"`. Wymieniony atrybut `runat="Server"` konwertuje znacznik HTML do postaci serwerowej kontrolki ASP.NET.

Znacznik `` znajdujący się na stronie zawartej w pliku *Rozdzial01\HtmlControls.aspx* wygląda podobnie jak zwykła kontrolka HTML ``, za wyjątkiem dodatkowego atrybutu `runat="Server"`.

Ponieważ znacznik `` ze strony zawartej w pliku *Rozdzial01\HtmlControls.aspx* jest kontrolką serwerową HTML, to istnieje możliwość jej programowania. W pliku *Rozdzial01\HtmlControls.aspx* bieżąca data i czas są przypisane znacznikowi `` w metodzie `Page_Load()`.

Kontrolki HTML zostały dołączone do platformy ASP.NET, aby ułatwić konwersję istniejących stron HTML do użycia platformy ASP.NET. W tej książce rzadko będziemy używać kontrolek HTML, ponieważ kontrolki standardowe ASP.NET w zasadzie zapewniają wszystkie te same funkcje, a nawet dostarczają jeszcze więcej możliwości.

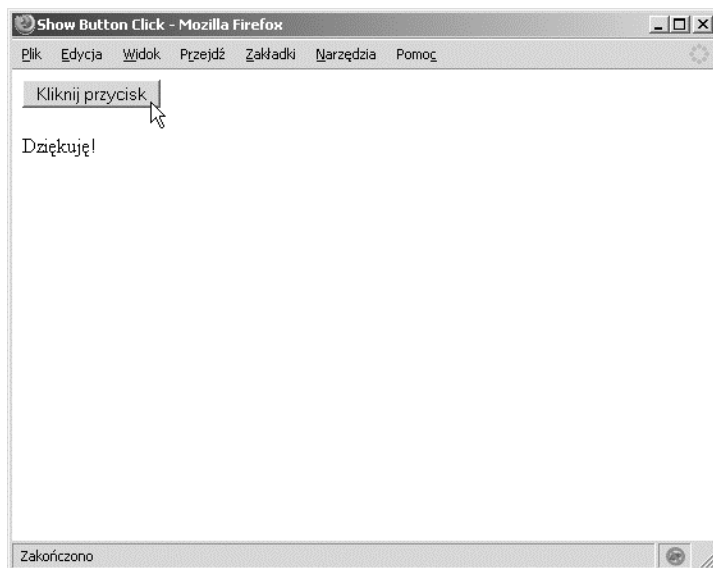
Zrozumienie i obsługa zdarzeń kontroltek

Większość kontroltek ASP.NET obsługuje jedno lub większą ilość zdarzeń. Przykładowo kontrolka ASP.NET Button obsługuje zdarzenie Click. Wspomniane zdarzenie Click jest wywoływane na serwerze po kliknięciu przez użytkownika przycisku, który został wygenerowany w przeglądarce internetowej przez kontrolkę Button.

Strona zawarta w pliku *Rozdzial01\ShowButtonClick.aspx* demonstruje, w jaki sposób można napisać kod, który będzie wykonywany po kliknięciu przez użytkownika przycisku wygenerowanego przez kontrolkę Button. Innymi słowy, przykład pokazuje, jak programista może utworzyć obsługę zdarzeń Click.

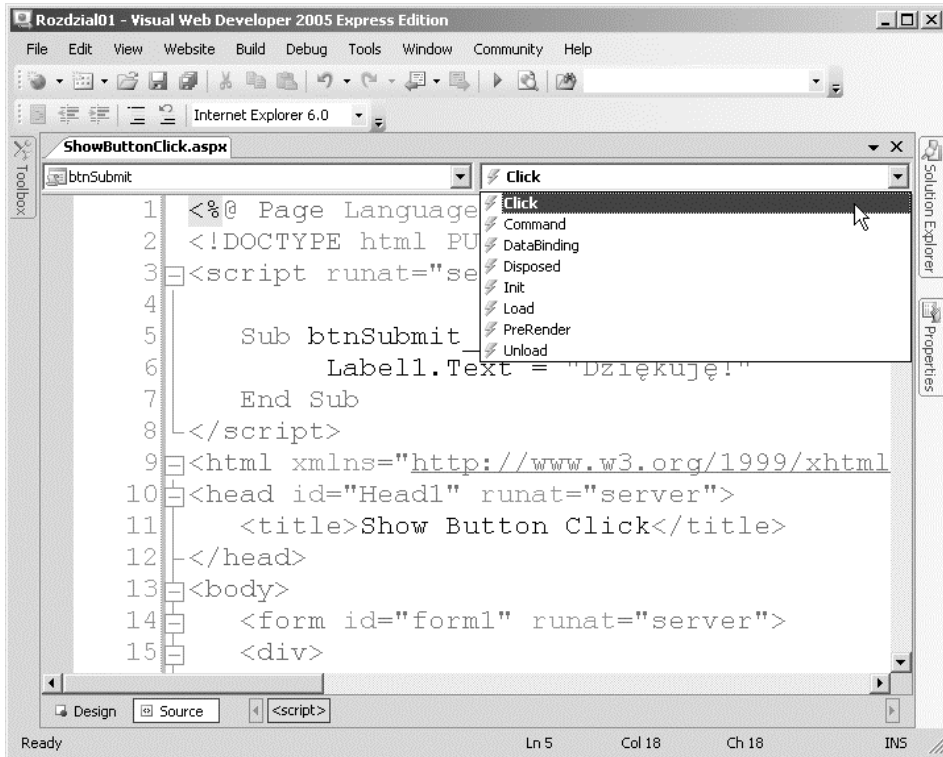
Warto zauważyć, że kontrolka Button zawarta na stronie z pliku *Rozdzial01\ShowButtonClick.aspx* zawiera atrybut `OnClick`. Wymieniony atrybut wskazuje na procedurę o nazwie `btnSubmit_Click()`. Procedura `btnSubmit_Click()` jest obsługą dla zdarzenia Click kontrolki Button. Procedura ta zostanie wykonana, gdy użytkownik kliknie przycisk (zobacz rysunek 1.5).

Rysunek 1.5.
Wywoływanie
zdarzenia Click



W trakcie korzystania z pakietu Visual Web Developer programista może automatycznie dodać do kontrolki obsługę zdarzeń na wiele różnych sposobów. W trybie *Source view* należy zaznaczyć daną kontrolkę z lewej górnej rozwijanej listy, a następnie wybrać odpowiednie zdarzenie z prawej górnej listy rozwijanej. Kod obsługi zdarzeń zostanie automatycznie dołączony do strony (zobacz rysunek 1.6).

Z kolei w trybie *Design view*, aby dodać do kontrolki domyślną obsługę zdarzeń, programista może dwukrotnie kliknąć kontrolkę. Dwukrotne kliknięcie kontrolki spowoduje przejście do trybu *Source view* i dołączenie obsługi zdarzeń.



Rysunek 1.6. Dodawanie obsługi zdarzeń w trybie Source view

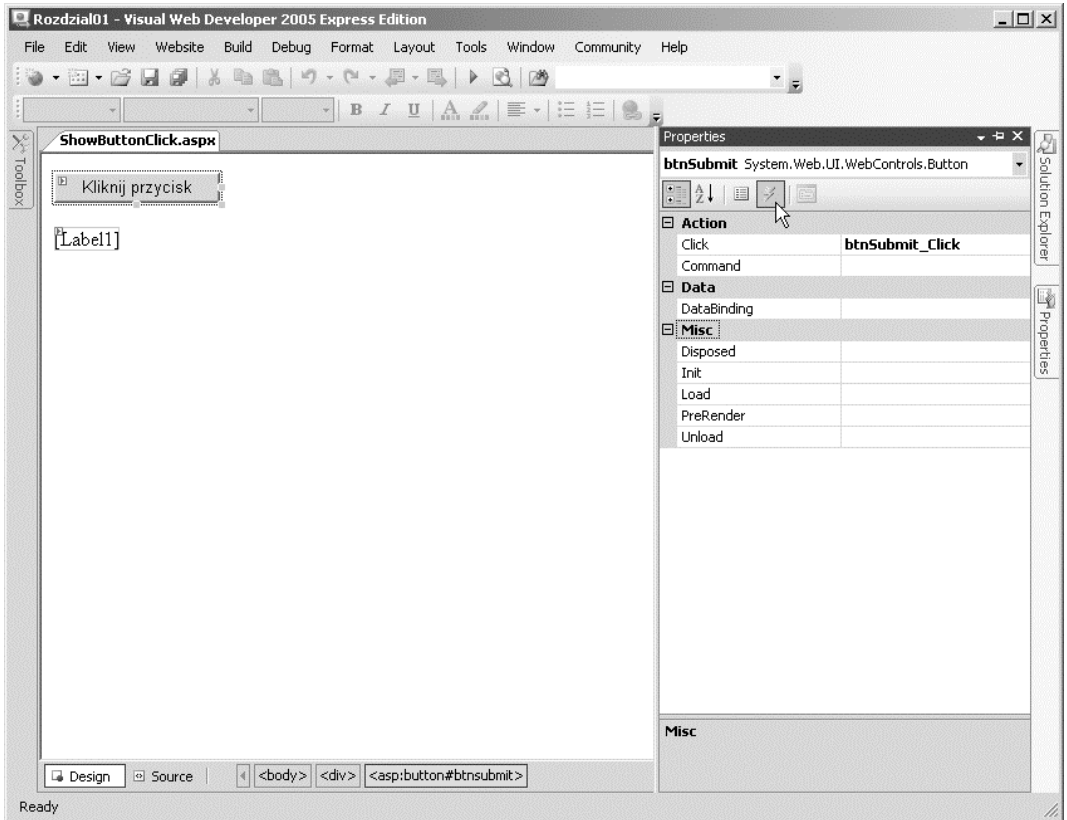
Na koniec w trybie *Design view* po zaznaczeniu kontrolki w oknie roboczym istnieje możliwość dodania obsługi zdarzeń z okna *Properties* poprzez kliknięcie przycisku *Events* (symbol błyskawicy), a następnie dwukrotne kliknięcie obok wybranego zdarzenia (zobacz rysunek 1.7).

Jest bardzo ważne, aby zrozumieć, że wszystkie zdarzenia kontrolki ASP.NET zachodzą po stronie serwera. Przykładowo zdarzenie *Click* nie jest faktycznie wykonywane po kliknięciu przycisku przez użytkownika. Zdarzenie *Click* nie będzie wywołane do chwili, dopóki strona zawierająca kontrolkę *Button* nie zostanie przekazana z powrotem do serwera.

Platforma ASP.NET jest platformą aplikacji sieciowych po stronie serwera. Kod napisany przez programistę jest wykonywany przez platformę .NET na serwerze, a nie w przeglądarce internetowej. Z punktu widzenia ASP.NET, dopóki strona nie zostanie przekazana z powrotem do serwera, na którym platforma .NET będzie mogła wykonać swoje zadania, to nic się nie zdarzyło.

Warto zwrócić uwagę na dwa parametry przekazywane metodzie `btnSubmit_Click()` w kodzie z pliku `Rozdzial01\ShowButtonClick.aspx`. Wszystkie obsługi zdarzeń kontrolki ASP.NET posiadają tę samą ogólną sygnaturę.

Pierwszy parametr, obiekt o nazwie `sender`, reprezentuje kontrolkę, która wywołała zdarzenie. Innymi słowy, w omawianym przypadku będzie to kontrolka *Button* kliknięta przez użytkownika.

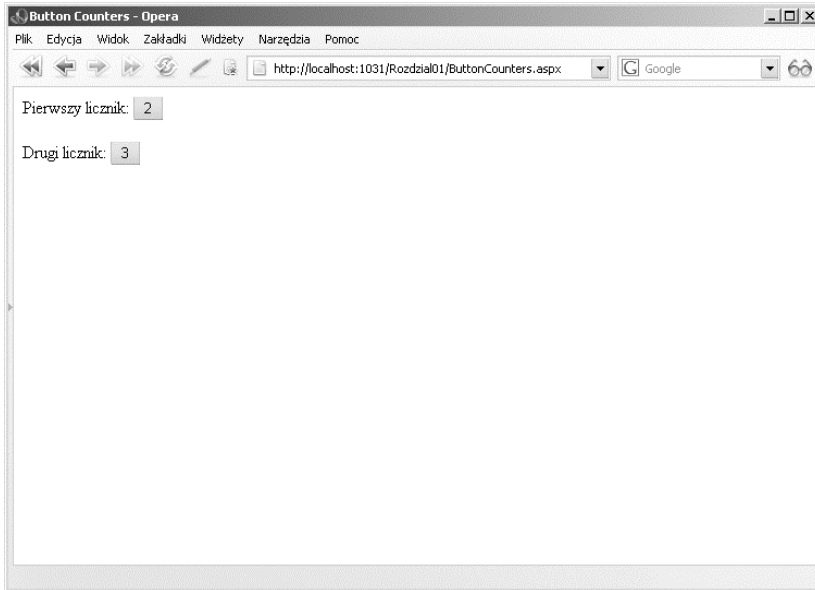


Rysunek 1.7. Dodawanie obsługi zdarzeń za pomocą okna *Properties*

Programista może połączyć wiele kontrolki na stronie z jedną procedurą obsługi zdarzeń, a następnie używać pierwszego parametru w celu określenia kontrolki, która wywołała zdarzenie. Przykładowo strona zawarta w pliku *Rozdział01\ButtonCounters.aspx* zawiera dwie kontrolki Button. Kiedy użytkownik kliknie dowolną kontrolkę Button, wówczas tekst wyświetlany przez kontrolkę Button zostanie uaktualniony (zobacz rysunek 1.8).

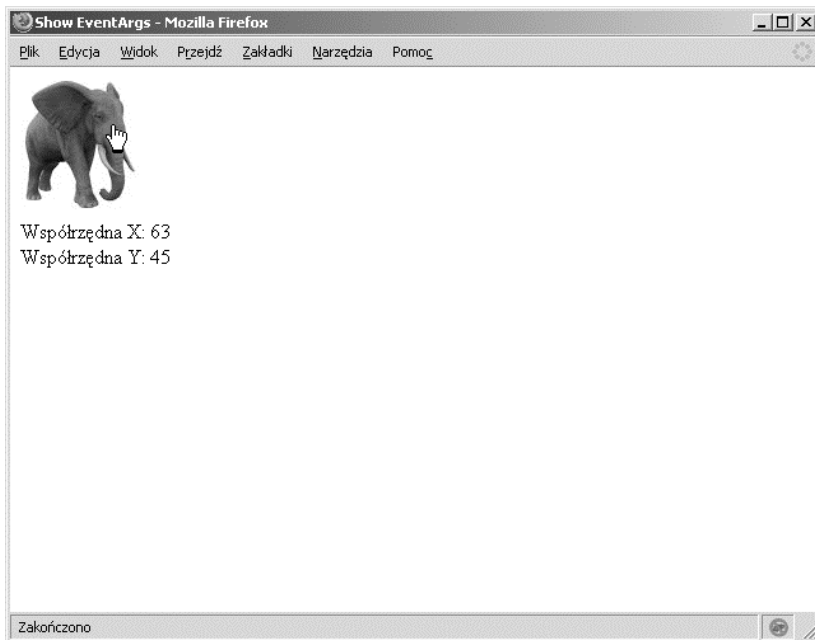
Drugim parametrem przekazywanym do obsługi zdarzeń Click jest parametr EventArgs o nazwie e reprezentujący wszystkie dodatkowe informacje o zdarzeniu, które są z tym zdarzeniem związane. W chwili kliknięcia przycisku nie zostają przypisane do zdarzenia żadne dodatkowe informacje, tak więc ani w pliku *Rozdział01>ShowButtonClick.aspx*, ani w pliku *Rozdział01\ButtonCounters.aspx* drugi parametr nie zawiera żadnych użytecznych informacji.

Z drugiej strony, gdy użytkownik kliknie kontrolkę ImageButton zamiast Button, wówczas do obsługi zdarzeń zostaną przekazane dodatkowe informacje o zdarzeniu. Kiedy użytkownik kliknie kontrolkę ImageButton, to do obsługi zdarzeń zostaną przekazane współrzędne X i Y miejsca, w którym nastąpiło kliknięcie.



Rysunek 1.8. Obsługiwanie dwóch kontrolki Button za pomocą jednej procedury obsługi zdarzeń

Strona zawarta w pliku *Rozdzial01\ShowEventArgs.aspx* zawiera kontrolkę `ImageButton` wyświetlającą obrazek. Kiedy użytkownik kliknie ten obrazek, wtedy w kontrolce `Label` zostaną wyświetlone współrzędne X i Y miejsca, w którym nastąpiło kliknięcie (zobacz rysunek 1.9).



Rysunek 1.9. Klikanie kontrolki `ImageButton`

Warto zauważyć, że drugi parametr przekazywany do metody `btnElephant_Click()` jest parametrem typu `ImageClickEventArgs`. Kiedykolwiek drugi parametr nie jest domyślnym parametrem `EventArgs`, wtedy Czytelnik powinien wiedzieć, że do obsługi zdarzeń są przekazywane dodatkowe informacje dotyczące tego zdarzenia.

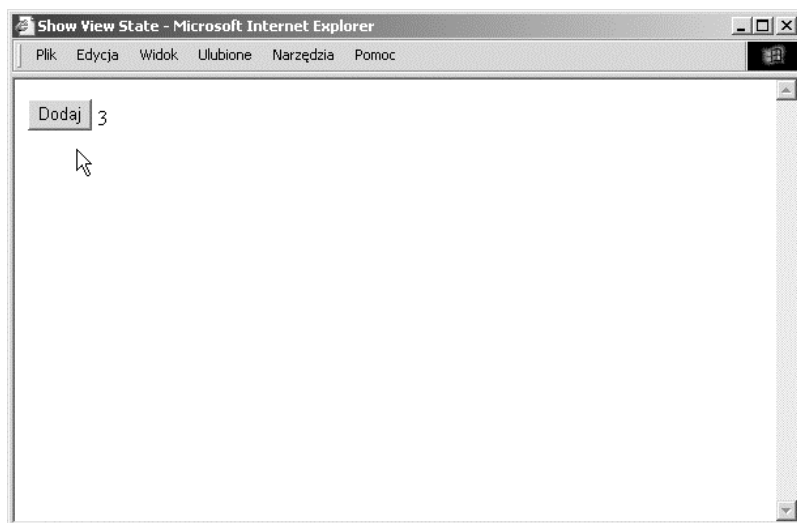
Zrozumienie stanu widoku

Protokół HTTP, który stanowi fundamentalny protokół dla sieci WWW, jest protokołem bezstanowym. Za każdym razem, gdy następuje żądanie wyświetlenia strony internetowej z witryny, to z punktu widzenia witryny internetowej użytkownik jest zupełnie inną osobą.

Jednakże platforma ASP.NET wykracza poza to ograniczenie protokołu HTTP. Przykładowo jeśli programista przypisze wartość właściwości `Text` kontrolki `Label`, to kontrolka `Label` zachowa tę wartość w trakcie wielu żądań wyświetlenia strony.

Zastanówmy się nad przykładem strony zawartej w pliku `Rozdzial01\ShowViewState.aspx`. Ta strona zawiera kontrolki `Button` i `Label`. Za każdym razem, gdy zostanie kliknięta kontrolka `Button`, wartość wyświetlana przez kontrolkę `Label` będzie zwiększona o jednostkę (zobacz rysunek 1.10). W jaki sposób kontrolka `Label` zachowuje swoją wartość w trakcie wielokrotnego przekazywania strony do serwera?

Rysunek 1.10.
Utrzymywanie stanu kontrolki w trakcie wielokrotnego przekazywania strony do serwera



Platforma ASP.NET wykorzystuje sztuczkę o nazwie stan widoku (*View State*). Jeżeli użytkownik otworzy w przeglądarce internetowej stronę zawartą w pliku `Rozdzial01\ShowViewState.aspx` i wybierze opcję *Pokaż źródło* z menu *Widok*, wówczas zauważy, że wygenerowana strona zawiera ukryte pole formularza sieciowego o nazwie `_VIEWSTATE`. Wspomniane pole przedstawia się następująco:

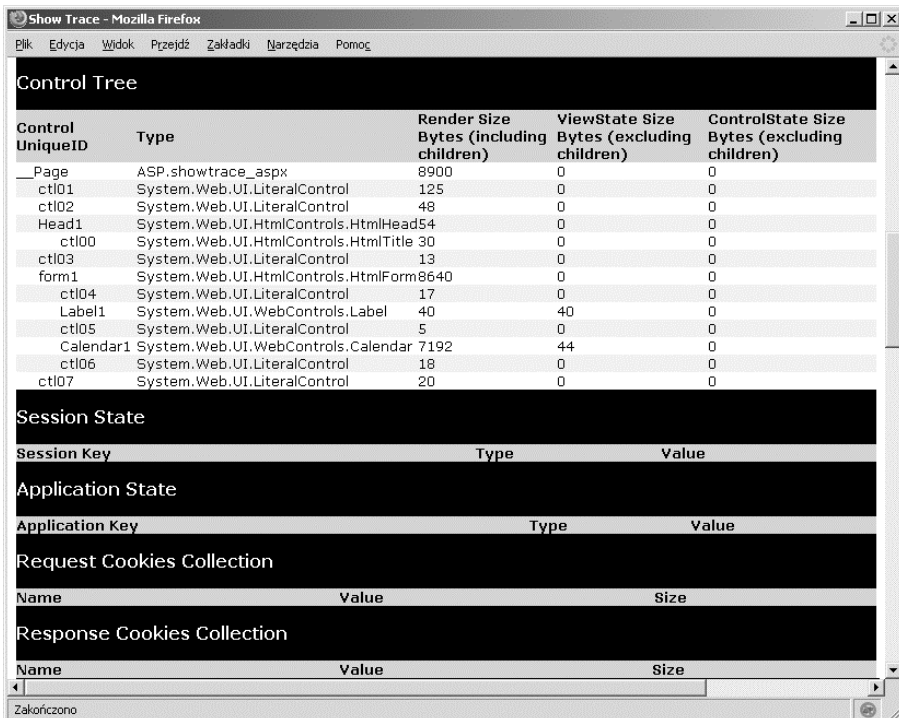
```
<input type="hidden" name="__VIEWSTATE" id="__VIEWSTATE"
value="/wEPDwUKLTc2ODE1OTYxNw9kFgICBA9kFgICAw8PFgIeBFR1eHQFATFkZGT3tMnThg9KZpGak
55p367vfInj1w==" />
```

Powyższe ukryte pole formularza zawiera wartość właściwości `Text` kontrolki `Label` (oraz wartości wszystkich pozostałych właściwości kontrolki, które są przechowywane w stanie widoku). Kiedy strona zostanie przekazana z powrotem do serwera, platforma ASP.NET pobiera te ciągi tekstowe, a następnie ponownie utworzy wartości dla wszystkich właściwości przechowywanych w stanie widoku. W ten sposób platforma ASP.NET zachowuje stan właściwości kontrolki podczas przekazywania strony do serwera.

Domyślnie stan widoku jest dostępny dla każdej kontrolki z platformy ASP.NET. Jeśli programista zmieni kolor tła w kontrolce `Calendar`, nowy kolor zostanie zapamiętany i zachowany podczas przekazywania strony do serwera. Po zmianie zaznaczonego elementu na liście kontrolki `DropDownList` zaznaczony element zostanie zapamiętany i zachowany podczas przekazywania strony do serwera. Wartości tych właściwości są automatycznie przechowywane w stanie widoku.

Stan widoku jest dobrym mechanizmem, ale zdarzają się sytuacje, w których dobrych mechanizmów jest zbyt wiele. Ukryte pole `_VIEWSTATE` formularza WWW może osiągnąć olbrzymie rozmiary. Umieszczenie zbyt wielu danych w stanie widoku może prowadzić do spowolnienia procesu generowania strony, ponieważ zawartość ukrytego pola formularza musi być przekazana pomiędzy serwerem a przeglądarką internetową.

Dzięki włączeniu mechanizmu śledzenia strony programista może określić, jaką część stanu widoku zużywa każda kontrolka umieszczona na stronie (zobacz rysunek 1.11). Strona zawarta w pliku *Rozdział01\ShowTrace.aspx* zawiera w dyrektywie `<%@ Page %>` atrybut `Trace="true"`, który włącza śledzenie strony.



Control UniqueID	Type	Render Size Bytes (including children)	ViewState Size Bytes (excluding children)	ControlState Size Bytes (excluding children)
_Page	ASP.showtrace.aspx	8900	0	0
ctl01	System.Web.UI.LiteralControl	125	0	0
ctl02	System.Web.UI.LiteralControl	48	0	0
Head1	System.Web.UI.HtmlControls.HtmlHead54	0	0	0
ctl00	System.Web.UI.HtmlControls.HtmlTitle	30	0	0
ctl03	System.Web.UI.LiteralControl	13	0	0
form1	System.Web.UI.HtmlControls.HtmlForm8640	0	0	0
ctl04	System.Web.UI.LiteralControl	17	0	0
Label1	System.Web.UI.WebControls.Label	40	44	0
ctl05	System.Web.UI.LiteralControl	5	0	0
Calendar1	System.Web.UI.WebControls.Calendar	7192	0	0
ctl06	System.Web.UI.LiteralControl	18	0	0
ctl07	System.Web.UI.LiteralControl	20	0	0

The screenshot also shows sections for Session State, Application State, Request Cookies Collection, and Response Cookies Collection, each with a table structure for keys and values.

Rysunek 1.11. Wyświetlanie wielkości stanu widoku dla każdej kontrolki

Kiedy użytkownik otworzy w przeglądarce internetowej stronę zawartą w pliku *Rozdział01\ShowTrace.aspx*, wówczas w dolnej części strony zostaną dołączone dodatkowe informacje odnośnie tejże strony. Kontrolka *Tree* wyświetla wielkość stanu widoku zajmowaną przez każdą kontrolkę ASP.NET umieszczoną na danej stronie.

Każda kontrolka ASP.NET posiada właściwość o nazwie *EnableViewState*. Jeśli wymienionej właściwości zostanie przypisana wartość *false*, wtedy stan widoku będzie dla tej kontrolki wyłączony. W takim przypadku, wartości właściwości tej kontrolki nie będą zapamiętane w trakcie przekazywania strony do serwera.

Przykładowo strona zawarta w pliku *Rozdział01\DisableViewState.aspx* zawiera dwie kontrolki *Label* oraz kontrolkę *Button*. Stan widoku dla pierwszej kontrolki został wyłączony, natomiast druga kontrolka posiada włączony stan widoku. Kiedy użytkownik kliknie przycisk, to tylko wartość drugiej kontrolki *Label* zostanie zwiększona o jednostkę.

Czasami programista chciałby wyłączyć stan widoku, nawet jeśli nie obawia się wielkości ukrytego pola *_VIEWSTATE* formularza sieciowego. Przykładowo jeśli kontrolka *Label* jest używana w celu wyświetlania komunikatu błędu dotyczącego sprawdzania poprawności danych, to w trakcie każdego wysyłania strony chcemy, aby kontrolka rozpoczęła działanie w swoim pierwotnym stanie. W takim przypadku wystarczy po prostu wyłączyć stan widoku dla wskazanej kontrolki *Label*.



Platforma ASP.NET w wersji 2.0 zawiera nową funkcję o nazwie stan kontrolki (*Control State*). Wspomniany stan kontrolki jest bardzo podobny do stanu widoku, za wyjątkiem faktu, że jest używany jedynie do zachowania istotnych informacji odnośnie stanu. Przykładowo kontrolka *GridView* używa stanu kontrolki do przechowywania zaznaczonego rekordu. Nawet jeśli stan widoku zostanie wyłączony, to kontrolka *GridView* i tak będzie pamiętała, który rekord został zaznaczony.

Zrozumienie stron ASP.NET

W niniejszym podrozdziale zostaną w bardziej szczegółowy sposób przedstawione strony ASP.NET. Czytelnik dowie się o kompilacji dynamicznej oraz plikach ukrytego kodu. Dokonamy również analizy zdarzeń obsługiwanych przez klasę *Page*.

Zrozumienie zagadnień kompilacji dynamicznej

To dziwne, ale kiedy programista tworzy stronę ASP.NET, wtedy faktycznie tworzy kod źródłowy dla klasy *.NET*. Następuje utworzenie nowego egzemplarza klasy *System.Web.UI*. Cała zawartość strony ASP.NET, włączając w to wszystkie skrypty oraz kod HTML, zostaje skompilowana do postaci klasy *.NET*.

W trakcie żądania wyświetlenia strony ASP.NET platforma ASP.NET poszukuje klasy *.NET* odpowiadającej żądanej stronie. Jeśli odpowiadająca stronie klasa nie istnieje, wówczas platforma automatycznie kompiluje stronę do postaci nowej klasy. Skompilowana klasa (podzespół) jest przechowywana w katalogu *Temporary ASP.NET Files* dostępnym pod następującą ścieżką dostępu:

```
\\WINDOWS\Microsoft .NET\Framework\[version]\Temporary ASP.NET Files
```

Podczas kolejnego żądania wyświetlenia tej samej strony ta strona nie będzie ponownie kompilowana. Zamiast tego nastąpi wykonanie poprzednio skompilowanej klasy, a wyniki zostaną zwrócone do przeglądarki internetowej.

Nawet po fizycznym wyłączeniu serwera sieciowego i przeniesieniu się na trzy lata na wyspę Borneo po ponownym włączeniu serwera sieciowego, jeśli ktokolwiek zgłosi żądanie wyświetlenia tej strony, nie będzie ona ponownie kompilowana. Skompilowana klasa jest aż do chwili zmodyfikowania kodu źródłowego aplikacji przechowywana w katalogu *Temporary ASP.NET Files*.

Kiedy klasa zostaje dodana do katalogu *Temporary ASP.NET Files*, następuje utworzenie zależności plikowej między klasą a oryginalną stroną ASP.NET. Jeżeli strona ASP.NET zostanie zmodyfikowana w jakikolwiek sposób, wówczas odpowiadająca jej klasa .NET będzie usunięta. Podczas kolejnego żądania wyświetlenia strony, platforma automatycznie skompiluje zmodyfikowany kod źródłowy strony do postaci nowej klasy .NET.

Omówiony powyżej proces nosi nazwę **kompilacji dynamicznej**. Wspomniana kompilacja dynamiczna umożliwia aplikacji ASP.NET jednoczesną obsługę tysięcy użytkowników. W przeciwieństwie do na przykład klasycznej strony ASP, strona ASP.NET nie musi być analizowana i kompilowana w trakcie każdego żądania jej wyświetlenia. Strona ASP.NET jest kompilowana jedynie wówczas, gdy nastąpi zmodyfikowanie aplikacji.

Uwaga

Za pomocą narzędzia wiersza poleceń o nazwie *aspnet_compiler.exe* programista może prekompilować całą aplikację ASP.NET. Jeśli aplikacja zostanie prekompilowana, wtedy użytkownicy nie będą narażeni na opóźnienie w trakcie pierwszego żądania wyświetlenia strony, które jest związane z kompilacją tejże strony.

Uwaga

Dzięki atrybutowi `CompilationMode` kompilacja dynamiczna może zostać wyłączona dla pojedynczej strony, dla stron w podanym katalogu lub dla całej aplikacji. Kiedy atrybut `CompilationMode` jest używany z dyrektywą `<%@ Page %>`, umożliwia wyłączenie kompilacji dynamicznej dla pojedynczej strony. Z kolei użycie atrybutu `CompilationMode` w elemencie `pages` umieszczonym w pliku konfiguracyjnym aplikacji umożliwia wyłączenie kompilacji dynamicznej dla całego katalogu lub aplikacji.

Wyłączenie kompilacji jest użyteczne wtedy, gdy w witrynie internetowej posiadamy tysiące stron i nie chcemy, aby dla każdej strony podzespół był wczytywany do pamięci. Po ustawieniu wartości `Never` atrybutowi `CompilationMode` strona nie będzie nigdy kompilowana, a więc dla tej strony nie zostanie wygenerowany podzespół. Interpretacja strony nastąpi w trakcie uruchomienia aplikacji.

Nie istnieje możliwość wyłączenia kompilacji dla stron zawierających kod wykonywany po stronie serwera. W szczególności strona, która nie jest kompilowana, nie może zawierać bloku `<script>...</script>` wykonywanego po stronie serwera. Z drugiej jednak strony nieskompilowana strona może posiadać kontrolki ASP.NET oraz wyrażenia wiązania danych.

Dla zainteresowanych Czytelników w pliku *Rozdzial01\FirstPage.aspx.vb* został przedstawiony kod źródłowy klasy, która odpowiada przedstawionej wcześniej stronie *Rozdzial01\FirstPage.aspx* (kod został odrobinę oczyszczony w celu zmniejszenia jego objętości). Przedstawiony kod został skopiowany z pliku umieszczonego w katalogu *Temporary ASP.NET Files* po wcześniejszym włączeniu debugowania aplikacji.

Klasa zawarta w pliku *Rozdział01\FirstPage.aspx.vb* dziedziczy po klasie `System.Web.UI.Page`. Metoda `ProcessRequest()` jest wywoływana przez platformę ASP.NET w trakcie wyświetlania strony. Wymieniona metoda jest odpowiedzialna za zbudowanie drzewa kontrolek strony, które jest tematem kolejnego punktu.

Zrozumienie drzewa kontrolek

W poprzednim punkcie dowiedzieliśmy się, że strona ASP.NET w rzeczywistości jest kodem źródłowym klasy .NET. Ewentualnie Czytelnik może wyobrazić sobie stronę ASP.NET jako zbiór kontrolek. Jest to dokładniejsze określenie, ponieważ niektóre kontrolki mogą zawierać kontrolki potomne. Stronę ASP.NET możemy więc określić mianem drzewka kontrolek.

Przykładowo strona zawarta w pliku *Rozdział01\ShowControlTree.aspx* zawiera kontrolki `DropDownList` oraz `Button`. Co więcej, ponieważ dyrektywa `<%@ Page %>` posiada atrybut `Trace="true"`, to dla tej strony zostało włączone śledzenie.

Kiedy użytkownik otworzy w przeglądarce internetowej stronę zawartą w pliku *Rozdział01\ShowControlTree.aspx*, wówczas zobaczy drzewko kontrolek strony, które zostało dołączone w dolnej części strony. Wspomniane drzewko jest zaprezentowane poniżej:

```
__Page ASP.showcontroltree_aspx
  ct102 System.Web.UI.LiteralControl
  ct100 System.Web.UI.HtmlControls.HtmlHead
    ct101 System.Web.UI.HtmlControls.HtmlTitle
  ct103 System.Web.UI.LiteralControl
  form1 System.Web.UI.HtmlControls.HtmlForm
    ct104 System.Web.UI.LiteralControl
    DropDownList1 System.Web.UI.WebControls.DropDownList
  ct105 System.Web.UI.LiteralControl
    Button1 System.Web.UI.WebControls.Button
  ct106 System.Web.UI.LiteralControl
  ct107
```

Węzeł nadrzędny w drzewku kontrolek jest stroną właściwą. Strona posiada atrybut `ID` o wartości `__Page`. Klasa strony zawiera wszystkie pozostałe kontrolki w jej zbiorach kontrolek potomnych.

Drzewko kontrolek posiada również egzemplarz klasy `HtmlForm` o nazwie `form1`. Ta kontrolka jest serwerowym znacznikiem `form` umieszczonym na stronie. Wymieniony znacznik zawiera wszystkie pozostałe kontrolki formularza — `DropDownList` oraz `Button` — jako kontrolki podrzędne.

Należy również zwrócić uwagę na kilka kontrolek `LiteralControl`, które funkcjonują w charakterze pośrednika między innymi kontrolkami w drzewku kontrolek. Czym są wspomniane kontrolki?

Trzeba pamiętać, że wszystko na stronie ASP.NET zostaje skonwertowane do postaci klasy .NET, włączając w to kod HTML oraz dowolną zawartość tekstową strony. Klasa `LiteralControl` przedstawia zawartość HTML na stronie (łącznie z obecnymi na stronie między znacznikami znakami nowego wiersza).



Zazwyczaj programista odnosi się do kontrolki umieszczonej na stronie za pomocą jej atrybutu `ID`, jednakże istnieją sytuacje, w których nie jest to możliwe. W takich przypadkach można wykorzystać metodę `FindControl()` klasy `Control` w celu pobrania kontrolki o określonym identyfikatorze `ID`. Metoda `FindControl()` jest podobna do metody JavaScript `getElementById()`.

Używanie stron ukrytego kodu

Platforma ASP.NET (oraz pakiet Visual Web Developer) umożliwia utworzenie dwóch odmiennych rodzajów stron ASP.NET. Istnieje więc możliwość utworzenia stron ASP.NET składających się z pojedynczego pliku lub z dwóch plików.

Wszystkie przykładowe fragmenty kodu przedstawione w tej książce zostały napisane z wykorzystaniem stron ASP.NET składających się z pojedynczego pliku. W stronie ASP.NET pojedynczego pliku kod strony oraz kontrolki strony znajdują się w jednym pliku. Kod strony jest umieszczony wewnątrz znacznika `<script runat="server">`.

Alternatywą dla stron ASP.NET składających się z pojedynczego pliku są strony ASP.NET posiadające dwa pliki. Składająca się z dwóch plików strona ASP.NET nosi nazwę **strony ukrytego kodu**. Na stronie ukrytego kodu kod strony jest umieszczony w oddzielnym pliku.



Strony ukrytego kodu działają zupełnie w inny sposób w platformie ASP.NET 2.0, niż miało to miejsce w przypadku platformy ASP.NET 1.x. W platformie ASP.NET 1.x dwie połówki strony ukrytego kodu były ze sobą powiązane za pomocą dziedziczenia, natomiast w platformie ASP.NET 2.0 dwie połówki strony ukrytego kodu są ze sobą powiązane za pomocą połączenia klas częściowych oraz dziedziczenia.

Przykładowo pliki *Rozdzial01\FirstPageCodeBehind.aspx* oraz *Rozdzial01\FirstPageCodeBehind.aspx.vb* zawierają dwie połówki strony ukrytego kodu.

Uwaga dla użytkowników pakietu Visual Web Developer

W przypadku korzystania z pakietu Visual Web Developer stronę ukrytego kodu można utworzyć po wybraniu opcji *Add New Item* z menu *Website*, a następnie zaznaczeniu elementu *Web Form* oraz pola wyboru *Place Code in Separate File* przed dodaniem strony do aplikacji.

Strona zawarta w pliku *Rozdzial01\FirstPageCodeBehind.aspx* nosi nazwę **strony prezentacyjnej** i zawiera kontrolki `Button` oraz `Label`. Jednakże strona ta nie posiada jakiegokolwiek kodu. Cały kod ją obsługujący został umieszczony w pliku ukrytego kodu.

Uwaga dla użytkowników pakietu Visual Web Developer

Przejdźcie do pliku ukrytego kodu strony następuje po kliknięciu danej strony prawym przyciskiem myszy i wybraniu opcji *View Code*.

Plik ukrytego kodu *Rozdzial01\FirstPageCodeBehind.aspx.vb* zawiera obsługę zdarzeń `Page_Load()` i `Button1_Click()`. Plik ten nie zawiera jednak żadnych kontroltek.

Warto zwrócić uwagę, że strona z pliku *Rozdział01\FirstPageCodeBehind.aspx* zawiera atrybuty `CodeFile` oraz `Inherits` w dyrektywie `<%@ Page %>`. Wymienione atrybuty powodują połączenie strony z jej plikiem ukrytego kodu.

W jaki sposób działa ukryty kod?

W poprzedniej wersji platformy ASP.NET (ASP.NET 1.x) przez stronę ukrytego kodu były generowane dwie klasy. Pierwsza odpowiadała stronie prezentacyjnej, natomiast druga plikowi ukrytego kodu. Te dwie klasy były ze sobą powiązane za pomocą mechanizmu dziedziczenia. Klasa strony prezentacyjnej dziedziczyła po klasie pliku ukrytego kodu.

Problem związany z taką metodą przypisania stron prezentacyjnych do ich plików ukrytego kodu był bardzo bezwzględny — dziedziczenie jest związkiem jednokierunkowym. Wszystko to, co jest prawdą w przypadku matki, jest również prawdą w przypadku córki, ale nie na odwrót. Każda kontrolka zadeklarowana na stronie prezentacyjnej musiała zostać także zadeklarowana w pliku ukrytego kodu. Co więcej, kontrolka powinna zostać zadeklarowana z dokładnie tym samym identyfikatorem ID. W przeciwnym przypadku zależność dziedziczenia nie zostanie dotrzymana i zdarzenia wywoływane przez kontrolkę nie będą mogły być obsługiwane w pliku ukrytego kodu.

W wersji beta ASP.NET 2.0 zastosowano zupełnie inną metodą przypisywania stron prezentacyjnych do ich plików ukrytego kodu. Ta nowa metoda jest znacznie mniej bezwzględna. Dwie połowy strony ukrytego kodu nie są dłużej powiązane za pomocą dziedziczenia, ale przy użyciu nowej technologii o nazwie **klasy częściowe**, która jest obsługiwana przez platformę .NET 2.0.

Uwaga

Szczegółowa analiza klas częściowych zostanie przedstawiona w rozdziale 14.

Klasy częściowe umożliwiają zadeklarowanie klasy nie tylko w jednym fizycznym pliku. W trakcie kompilowania klasy jest ona generowana ze wszystkich klas częściowych. Wszystkie elementy jednej klasy częściowej — włączając w to wszystkie prywatne pola, metody i właściwości — są dostępne w innej klasie częściowej tej samej klasy. Jest to całkiem logiczne rozwiązanie, ponieważ klasy częściowe są ostatecznie łączone w celu utworzenia klasy końcowej.

Zaletą używania klas częściowych jest to, że programista nie musi deklarować kontrolki zarówno na stronie prezentacyjnej, jak i w pliku ukrytego kodu. Wszystkie elementy, które zostaną zadeklarowane na stronie prezentacyjnej, stają się automatycznie dostępne w pliku ukrytego kodu, natomiast wszystko, co zostało zadeklarowane w pliku ukrytego kodu, jest automatycznie dostępne na stronie prezentacyjnej.

Wersja beta platformy ASP.NET 2.0 używała klas częściowych w celu połączenia strony prezentacyjnej z jej plikiem ukrytego kodu, jednakże pewne zaawansowane funkcje platformy ASP.NET 1.x nie zachowały zgodności podczas wykorzystywania klas częściowych. Aby obsłużyć te zaawansowane funkcje, w końcowej wersji platformy ASP.NET zostały zastosowane bardziej złożone metody powiązania stron prezentacyjnych z ich plikami ukrytego kodu.



Platforma ASP.NET 1.x umożliwia programiście utworzenie klasy podstawowej `Page` oraz dziedziczenie każdej strony ASP.NET w danej aplikacji po tej własnej klasie `Page`. Powiązanie stron oraz plików ukrytego kodu z klasami częściowymi spowodowało powstanie konfliktów dotyczących dziedziczenia po własnej klasie bazowej `Page`. W ostatecznej wersji platformy ASP.NET 2.0 programista może ponownie utworzyć własną klasę podstawową `Page`. Przykład utworzenia własnej klasy podstawowej `Page` zostanie przedstawiony w ostatnim podrozdziale rozdziału 5.

W celu powiązania stron prezentacyjnych z plikami ukrytego kodu ostateczna wersja platformy ASP.NET 2.0 używa połączenia dziedziczenia oraz klas częściowych. Gdy programista tworzy plik ukrytego kodu, wtedy platforma ASP.NET 2.0 generuje trzy klasy.

Pierwsze dwie klasy odpowiadają stronie prezentacyjnej. Przykładowo po utworzeniu strony `FirstPageCodeBehind.aspx` w katalogu *Temporary ASP.NET Files* zostaną automatycznie wygenerowane następujące klasy:

```
Partial Public Class FirstPageCodeBehind

    Protected WithEvents Button1 As Global.System.Web.UI.WebControls.Button
    Protected WithEvents Label1 As Global.System.Web.UI.WebControls.Label
    ... miejsce na dodatkowy kod klasy ...

End Class

Public Class firstpagecodebehind_aspx
    Inherits FirstPageCodeBehind

    ... miejsce na dodatkowy kod klasy ...

End Class
```

Trzecia wygenerowana klasa odpowiada plikowi ukrytego kodu. Dla pliku `FirstPageCodeBehind.vb` zostanie więc wygenerowana następująca klasa:

```
Partial Class FirstPageCodeBehind
    Inherits System.Web.UI.Page

    Protected Sub Button1_Click(ByVal sender As Object, ByVal e As System.EventArgs)
        Handles Button1.Click
        Label1.Text = "Dziękuję!"
    End Sub

End Class
```

Klasa `firstpagecodebehind_aspx` jest wykonywana, gdy w przeglądarce internetowej pojawi się żądanie wyświetlenia strony `FirstPageCodeBehind.aspx`. Wymieniona klasa dziedziczy po klasie `FirstPageCodeBehind`, która jest klasą częściową. Klasa zostaje wygenerowana dwukrotnie: raz przez stronę prezentacyjną oraz ponownie przez plik ukrytego kodu.

W celu powiązania stron prezentacyjnych oraz plików ukrytego kodu końcowe wydanie platformy ASP.NET 2.0 używa połączenia klas częściowych oraz dziedziczenia. Ponieważ strona oraz klasy ukrytego kodu są klasami częściowymi, to w przeciwieństwie do poprzedniej wersji ASP.NET programista nie musi dłużej dwukrotnie deklarować kontrolki zarówno na stronie prezentacyjnej, jak i w pliku ukrytego kodu. Każda kontrolka

zadeklarowana na stronie prezentacyjnej jest automatycznie dostępna w pliku ukrytego kodu. Ponieważ klasa strony dziedziczy po klasie ukrytego kodu, to platforma ASP.NET 2.0 nadal obsługuje zaawansowane funkcje platformy ASP.NET, takie jak własne klasy bazowe Page.

Decyzja pomiędzy stronami składającymi się z pojedynczego pliku, a stronami ukrytego kodu

Tak więc powstaje pytanie, kiedy programista powinien korzystać ze stron ASP.NET składających się z pojedynczego pliku, a kiedy powinien wykorzystywać strony ukrytego kodu? Taka decyzja jest pochodną upodobań i osobistych preferencji programisty. W wielu blogach dostępnych w internecie znajdują się mocne argumenty przemawiające zarówno za jednym, jak i za drugim rozwiązaniem.

Pojawiają się także głosy, że strony ukrytego kodu są ważniejsze od stron składających się z pojedynczego pliku, ponieważ strony ukrytego kodu umożliwiają lepsze oddzielenie interfejsu użytkownika od warstwy logicznej aplikacji. Problem dotyczący tego argumentu jest taki, że zwykle usprawiedliwieniem oddzielenia interfejsu użytkownika od logiki aplikacji jest ponowne użycie kodu. Tworzenie stron ukrytego kodu w rzeczywistości nie zachęca do ponownego używania kodu. Lepszym sposobem ponownego używania logiki aplikacji na wielu stronach jest zbudowanie oddzielnych komponentów bibliotek (ten temat zostanie przedstawiony w części czwartej książki).

Wyborem autora tejże książki jest budowanie aplikacji za pomocą stron ASP.NET składających się z pojedynczego pliku, ponieważ takie podejście wymaga zarządzania znacznie mniejszą liczbą plików, jednakże autor tworzy również wiele aplikacji, używając modelu stron ukrytego kodu (na przykład niektóre ASP.NET Starter Kits) bez żadnych strasznych konsekwencji.



Poprzednia wersja Visual Studio .NET nie obsługiwała tworzenia stron ASP.NET składających się z pojedynczego pliku. Jeżeli Czytelnik chce tworzyć strony ASP.NET w postaci pojedynczego pliku za pomocą poprzedniej wersji ASP.NET, to musi wykorzystać alternatywne środowiska programistyczne, takie jak Web Matrix lub na przykład aplikację Notatnik systemu Windows.

Obsługa zdarzeń strony

Podczas każdego żądania wyświetlenia strony ASP.NET zostaje wykonany określony zbiór zdarzeń w ustalonej kolejności. Wspomniana kolejność zdarzeń nosi nazwę **cyklu życia strony**.

Przykładowo w poprzednich przykładach przedstawionych w bieżącym rozdziale zostało wykorzystane zdarzenie Load. Wymienione zdarzenie Load jest zwykle używane w celu inicjalizacji właściwości kontrolki, które są umieszczone na stronie, jednakże zdarzenie Load jest tylko jednym ze zdarzeń obsługiwanych przez klasę Page.

Poniżej została przedstawiona kolejność zdarzeń, które są wywoływane w trakcie żądania wyświetlenia strony:

1. PreInit
2. Init
3. InitComplete
4. PreLoad
5. Load
6. LoadComplete
7. PreRender
8. PreRenderComplete
9. SaveStateComplete
10. Unload

Dlaczego na powyższej liście znajduje się tak wiele zdarzeń? Na kolejnych etapach cyklu życia strony zachodzą różne zdarzenia oraz są udostępniane rozmaite informacje.

Przykładowo stan widoku nie zostanie w pełni wczytany przed wywołaniem zdarzenia `InitComplete`. Dane przekazywane z kontrolki, na przykład `TextBox`, do serwera również nie będą dostępne przed wywołaniem wymienionego zdarzenia.

W dziewięćdziesięciu dziewięciu procentach sytuacji programista nie będzie obsługiwał wymienionych na powyższej liście zdarzeń za wyjątkiem zdarzeń `Load` i `PreRender`. Różnica między wskazanymi zdarzeniami polega na tym, że zdarzenie `Load` występuje przed jakimikolwiek zdarzeniami kontrolki, a zdarzenie `PreRender` występuje po wywołaniu wszystkich zdarzeń kontrolki.

Strona zawarta w pliku *Rozdzial01\ShowPageEvents.aspx* prezentuje różnicę między zdarzeniami `Load` i `PreRender`. Strona zawiera więc trzy procedury obsługi zdarzeń: jedną dla zdarzenia `Load`, drugą dla zdarzenia `Button Click` oraz trzecią dla zdarzenia `PreRender`. Każda obsługa zdarzeń dodaje do kontrolki `Label` własny komunikat (zobacz rysunek 1.12).

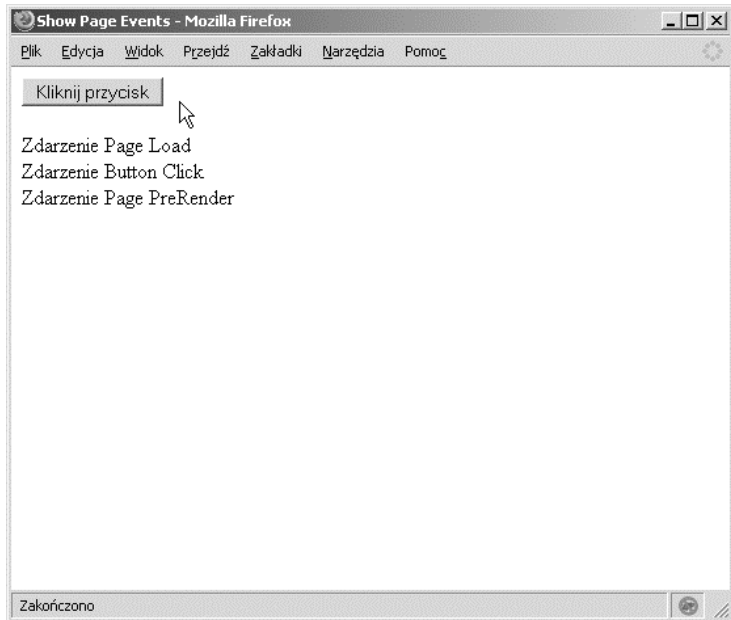
Kiedy użytkownik kliknie kontrolkę `Button`, zdarzenie `Click` zostanie wywołane na serwerze po zdarzeniu `Load`, ale jeszcze przed wystąpieniem zdarzenia `PreRender`.

Inną kwestią odnośnie pliku *Rozdzial01\ShowPageEvents.aspx*, na którą warto zwrócić uwagę, jest sposób połączenia procedur obsługi zdarzeń ze zdarzeniami klasy `Page`. ASP.NET obsługuje włączoną domyślnie funkcję o nazwie `AutoEventWireUp`. Jeżeli procedura będzie miała nazwę `Page_Load()`, wówczas procedura automatycznie obsłuży zdarzenie `Load` klasy `Page`, natomiast nazwa procedury w postaci `Page_PreRender()` spowoduje automatyczną obsługę zdarzenia `PreRender` klasy `Page`.



Funkcja `AutoEventWireUp` nie działa w przypadku każdego zdarzenia strony. Przykładowo funkcja nie działa dla zdarzenia `Page_InitComplete()`.

Rysunek 1.12.
Wyświetlanie kolejności
występowania zdarzeń
strony



Używanie właściwości Page.IsPostBack

Klasa Page posiada właściwość o nazwie IsPostBack, która może zostać użyta w celu określenia, czy strona została już przekazana z powrotem do serwera.

Z powodu istnienia stanu widoku, gdy programista inicjalizuje właściwość kontrolki, nie chce inicjalizować tej właściwości podczas każdego wczytywania strony. Skoro stan widoku zapisuje między wieloma odświeżeniami strony stan właściwości kontrolki, więc programista zwykle jednokrotnie inicjalizuje właściwość kontrolki podczas pierwszego wczytania strony.

W rzeczywistości wiele kontrolki nie działa prawidłowo, jeżeli ich właściwości są ponownie inicjalizowane w czasie każdego wczytywania strony. W takich przypadkach programista musi używać właściwości IsPostBack, aby określić, czy strona została czy nie została odświeżona.

Strona zawarta w pliku *Rozdział01\ShowIsPostBack.aspx* prezentuje technikę używania właściwości Page.IsPostBack podczas dodawania elementów do kontrolki DropDownList.

W pliku *Rozdział01\ShowPageEvents.aspx* kod zawarty w obsłudze zdarzenia Page_Load() jest wykonywany tylko jednokrotnie, podczas pierwszego wczytywania strony. Kiedy strona będzie odświeżona, wówczas właściwość IsPostBack zwróci wartość true, co spowoduje pominięcie wykonywania kodu umieszczonego w zdarzeniu Page_Load().

Jeżeli programista usunie z metody Page_Load() wiersz zawierający sprawdzenie wartości właściwości IsPostBack, wówczas może otrzymać bardzo dziwne wyniki. Kontrolka DropDownList zawsze wyświetli pierwszy element jako zaznaczony. Dołączenie listy

elementów do kontrolki DropDownList spowoduje ponowną inicjalizację kontrolki DropDownList — dlatego też chcemy, aby kontrolka DropDownList została dołączona tylko raz, podczas pierwszego wczytywania strony.

Debugowanie i śledzenie stron ASP.NET

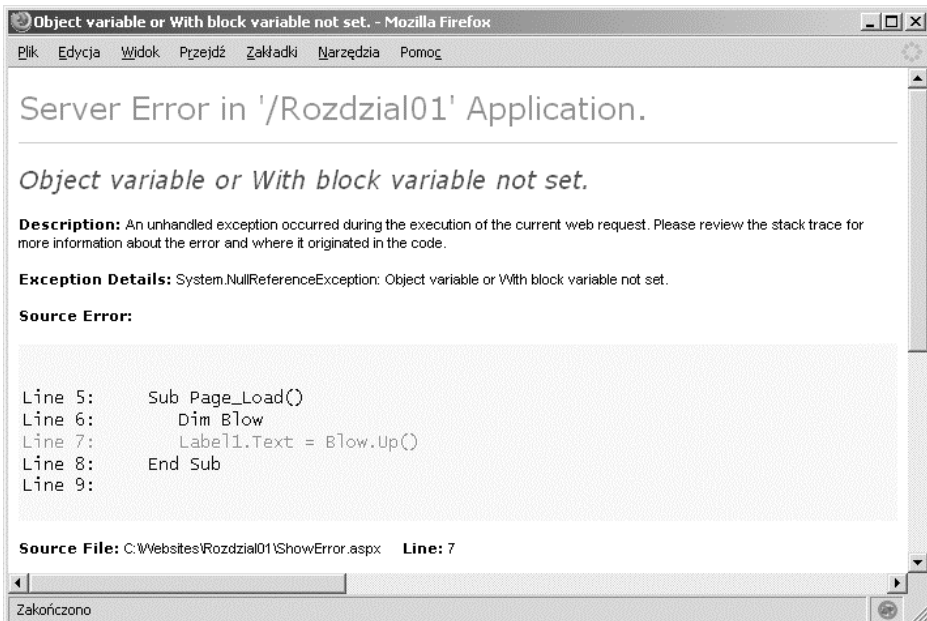
Smutny fakt dotyczący życia jest taki, że większość czasu spędzona w trakcie budowy aplikacji WWW jest wykorzystywana na debugowanie tej aplikacji.

W niniejszym punkcie dowiemy się, w jaki sposób uzyskać szczegółowy komunikat błędu podczas programowania stron ASP.NET. Nauczymy się wyświetlać własne komunikaty śledzenia strony, których możemy użyć w trakcie debugowania strony.

Debugowanie stron ASP.NET

Jeżeli w trakcie wykonywania strony programista musi wyświetlić szczegółowe komunikaty błędów, to w takim przypadku powinno zostać włączone debugowanie dla strony lub dla całej aplikacji. Włączenie debugowania dla podanej strony odbywa się za pomocą atrybutu `Debug="true"` umieszczonego w dyrektywie `<%@ Page %>`. Przykładowo strona zawarta w pliku `Rozdzial01\ShowError.aspx` ma włączone debugowanie.

Kiedy strona z pliku `Rozdzial01\ShowError.aspx` zostanie otwarta w przeglądarce internetowej, wówczas pojawi się szczegółowy komunikat błędu (zobacz rysunek 1.13).



Rysunek 1.13. Wyświetlanie szczegółowego komunikatu błędu



Przed umieszczeniem aplikacji „w produkcji” należy się upewnić o wyłączeniu debugowania. Gdy aplikacja znajduje się w trybie debugowania, kompilator nie może wykonać określonych zadań optymalizacyjnych.

Zamiast włączać debugowanie dla pojedynczej strony, programista może włączyć debugowanie dla całej aplikacji poprzez dodanie do aplikacji przedstawionego na listingu 1.3 pliku konfiguracyjnego.

Listing 1.3. Web.Config

```
<?xml version="1.0"?>
<configuration>
<system.web>
  <compilation debug="true" />
</system.web>
</configuration>
```

Podczas debugowania aplikacji ASP.NET umieszczonej na zdalnym serwerze sieciowym należy zablokować wyświetlanie własnych błędów. Domyślnie, z powodów bezpieczeństwa, platforma ASP.NET nie wyświetla komunikatów błędów, gdy żądanie wyświetlenia strony pochodzi ze zdalnego komputera. Kiedy wyświetlanie komunikatów o własnych błędach zostanie włączone, wtedy nie będą wyświetlane komunikaty błędów powstałych na zdalnym komputerze. Zmodyfikowana wersja pliku konfiguracyjnego aplikacji sieciowej, która wyłącza wyświetlanie komunikatów własnych błędów, została przedstawiona na listingu 1.4.

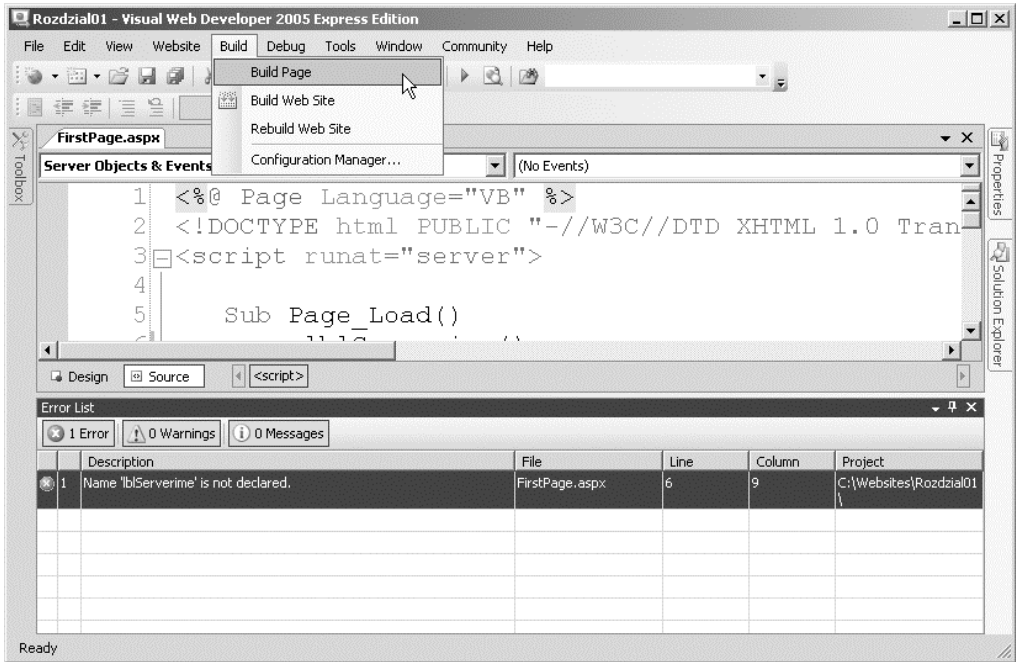
Listing 1.4. Web.Config

```
<?xml version="1.0"?>
<configuration>
<system.web>
  <compilation debug="true" />
  <customErrors mode="Off" />
</system.web>
</configuration>
```

Debugowanie stron za pomocą pakietu Visual Web Developer

Jeżeli programista używa pakietu Visual Web Developer, wówczas może wyświetlić komunikaty błędów kompilacji poprzez zbudowanie strony lub całej aplikacji. W tym celu należy wybrać opcję *Build Page* lub *Build Web Site* z menu *Build*. Lista komunikatów błędów kompilacji oraz ostrzeżeń zostanie wyświetlona w oknie *Error List* (zobacz rysunek 1.14). Po dwukrotnym kliknięciu dowolnego z przedstawionych błędów nastąpi bezpośrednie przejście do fragmentu kodu, który spowodował wystąpienie klikniętego błędu.

Kiedy zachodzi konieczność przeprowadzenia bardziej zaawansowanego debugowania, wtedy można skorzystać z debugera pakietu Visual Web Developer. Wspomniany debugger umożliwia ustawienie punktów kontrolnych i przechodzenie przez kod wiersz po wierszu.



Rysunek 1.14. Proces budowania strony w pakiecie Visual Web Developer

Ustawienie punktu kontrolnego następuje po kliknięciu w trybie *Source view* skrajnej lewej kolumny. Po dodaniu punktu kontrolnego na ekranie pojawi się czerwone kółko (zobacz rysunek 1.15).

Po ustawieniu punktu kontrolnego uruchomienie aplikacji następuje po wybraniu opcji *Start Debugging* z menu *Debug*. Wykonywanie aplikacji zostanie wstrzymane po osiągnięciu ustawionego punktu kontrolnego. W tym miejscu programista może umieścić wskaźnik myszy nad dowolną zmienną lub właściwością kontrolki i poznać bieżącą wartość zmiennej lub właściwości kontrolki.

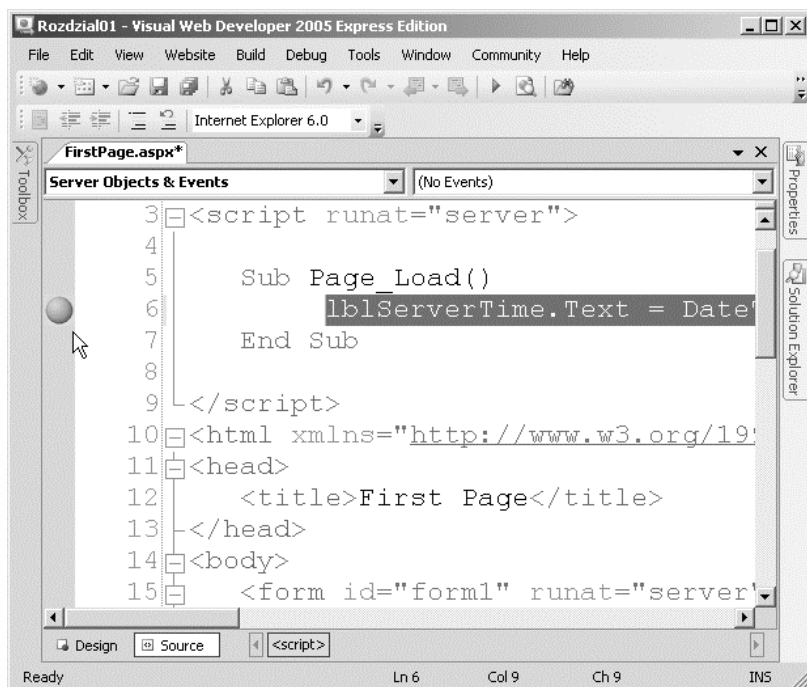


Istnieje możliwość wyznaczenia jednej ze stron aplikacji jako strony początkowej. W ten sposób w trakcie każdego uruchomienia aplikacji zostanie wykonana wskazana strona, niezależnie od tego, która strona była wcześniej otwarta. Ustawienie strony jako strony początkowej następuje po jej kliknięciu w oknie Solution Explorer i wybraniu z menu kontekstowego opcji *Set As Start Page*.

Po osiągnięciu punktu kontrolnego programista może kontynuować wykonywanie aplikacji za pomocą poleceń *Step Into*, *Step Over* i *Step Out* dostępnych w menu *Debug* lub na pasku narzędziowym. Poniżej znajduje się objaśnienie każdej z wymienionych wcześniej opcji:

- ♦ **Step Into** — powoduje wykonanie kolejnego wiersza kodu.
- ♦ **Step Over** — powoduje wykonanie kolejnego wiersza kodu bez opuszczanie bieżącej metody.
- ♦ **Step Out** — powoduje wykonanie kolejnego wiersza kodu oraz powrót do metody, która wywołała bieżącą metodę.

Rysunek 1.15.
Ustawienie punktu kontrolnego



Kiedy debugowanie strony zostanie zakończone, wówczas można kontynuować, zatrzymać lub uruchomić ponownie działanie aplikacji poprzez wybór odpowiedniej opcji z menu *Debug* lub paska narzędziowego.

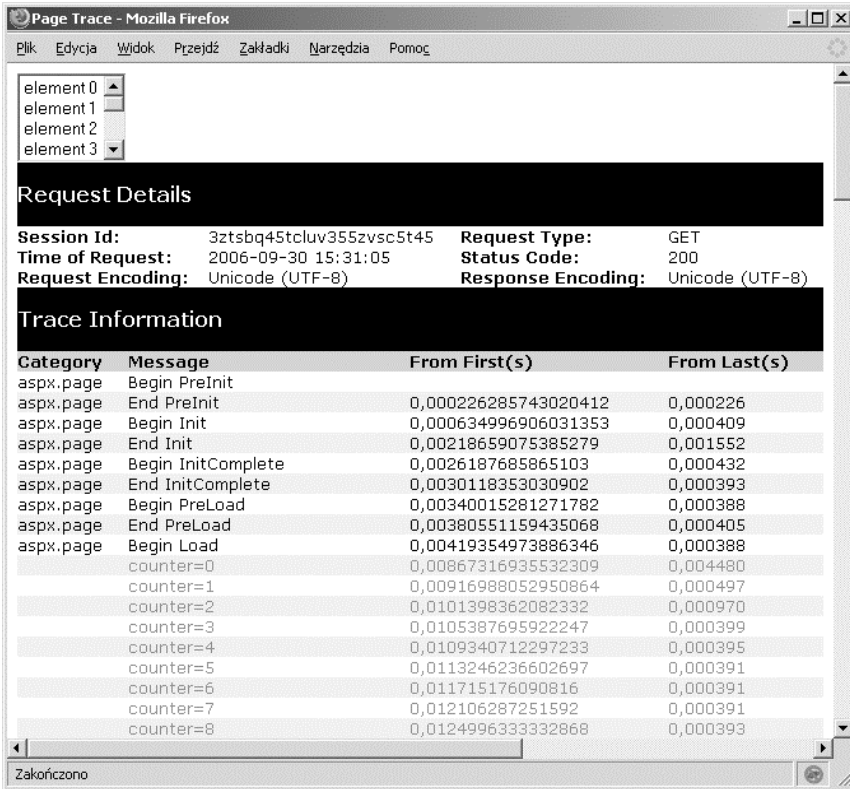
Śledzenie wykonywania strony

Jeżeli programista chce śledzić komunikaty danych wyjściowych podczas wykonywania strony, może włączyć śledzenie danej strony lub całej aplikacji. Platforma ASP.NET obsługuje zarówno śledzenie na poziomie strony, jak i śledzenie na poziomie całej aplikacji.

Strona zawarta w pliku *Rozdział01\PageTrace.aspx* prezentuje, w jaki sposób możemy wykorzystać zalety płynące z procesu śledzenia na poziomie strony.

Warto zwrócić uwagę, że dyrektywa `<%@ Page %>` zawarta w pliku *Rozdział01\PageTrace.aspx* zawiera atrybut `trace="true"`. Atrybut ten powoduje włączenie śledzenia oraz dodanie w dolnej części strony sekcji *Trace Information* (zobacz rysunek 1.16).

Co więcej, warto zauważyć, że obsługa zdarzeń `Page_Load()` używa metody `Trace.Warn()` do zapisania komunikatu w sekcji *Trace Information*. Do sekcji *Trace Information* może zostać przekazany dowolny ciąg tekstowy. Kod zawarty w pliku *Rozdział01\PageTrace.aspx* powoduje wyświetlenie w sekcji *Trace Information* bieżącej wartości zmiennej o nazwie `counter`.



Rysunek 1.16. Wyświetlanie informacji pochodzących ze śledzenia strony

Programista powinien wykorzystać zalety płynące ze śledzenia strony, gdy musi dokładnie wiedzieć, jakie zachodzą zdarzenia w trakcie uruchomienia strony. Metodę `Trace.Warn()` można wywołać w dowolnym miejscu kodu. Ponieważ sekcja *Trace Information* pojawia się w miejscu wystąpienia błędu, to śledzenie strony będzie przydatną techniką diagnozowania powodu powstawiania błędów na stronie.

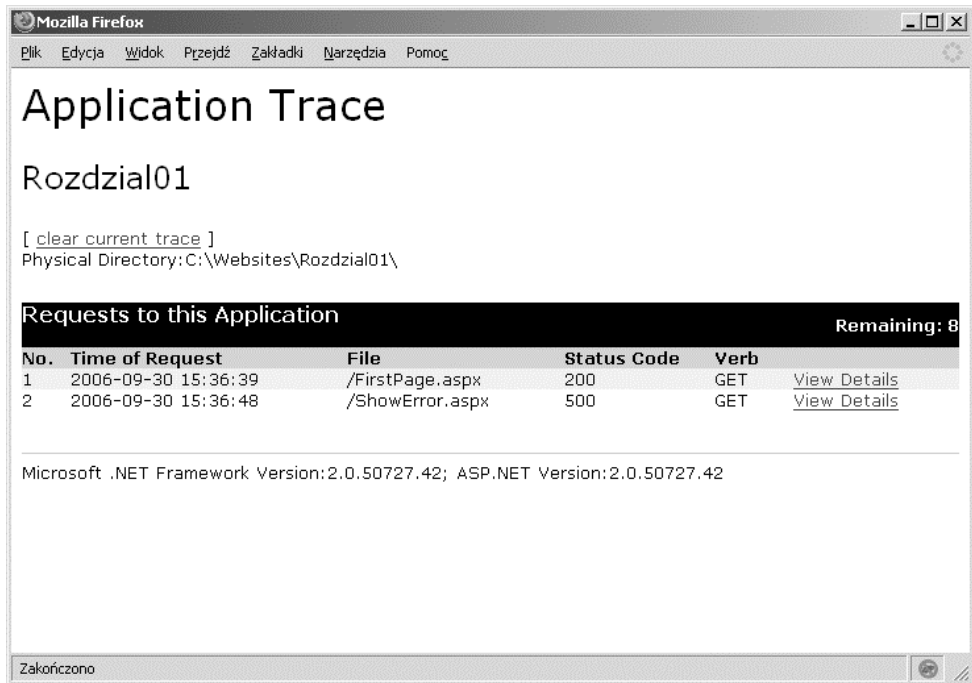
Jedyną wadą śledzenia strony jest fakt, że każda osoba na świecie może zobaczyć informacje pochodzące ze śledzenia strony. Ominięciem tego problemu może być wykorzystanie zalet śledzenia na poziomie aplikacji. Kiedy zostanie włączone śledzenie na poziomie aplikacji, wówczas informacje pochodzące ze śledzenia zostaną wyświetlone jedynie po wystąpieniu żądania wyświetlenia strony o nazwie *Trace.axd*.

W celu włączenia śledzenia na poziomie aplikacji do aplikacji należy dodać plik konfiguracyjny, który jest przedstawiony na listingu 1.5.

Listing 1.5 *Web.Config*

```
<?xml version="1.0"?>
<configuration>
<system.web>
  <trace enabled="true" />
</system.web>
</configuration>
```

Po dodaniu do aplikacji pliku konfiguracyjnego przedstawionego na listingu 1.5 programista może zgłosić żądanie wyświetlenia strony *Trace.axd* w przeglądarce internetowej. Na podanej stronie zostanie wyświetlonych ostatnich 10 żądań strony po włączeniu śledzenia na poziomie aplikacji (zobacz rysunek 1.17).



Rysunek 1.17. Wyświetlanie informacji pochodzących ze śledzenia aplikacji



Domyślnie strona *Trace.axd* nie może być wywołana ze zdalnego komputera. Jeżeli zachodzi potrzeba zdalnego uzyskania dostępu do strony *Trace.axd*, wówczas do elementu *trace* w pliku konfiguracyjnym należy dodać atrybut `localOnly="false"`.

Jeśli użytkownik kliknie łącze *View Details* znajdujące się obok każdego żądania wyświetlenia strony, wówczas będzie mógł poznać wszystkie komunikaty pochodzące ze śledzenia wygenerowane przez daną stronę. Komunikaty zapisywane za pomocą metody `Trace.Warn()` zostaną wyświetlone przez stronę *Trace.axd* nawet wtedy, gdy śledzenie na poziomie strony zostało wyłączone.



Istnieje możliwość wykorzystania nowego atrybutu `writeToDiagnosticsTrace` elementu *trace* w celu zapisania w oknie *Output* pakietu Visual Web Developer wszystkich komunikatów pochodzących ze śledzenia aplikacji po jej uruchomieniu. Atrybut `mostRecent` powoduje wyświetlenie ostatnich dziesięciu żądań strony zamiast dziesięciu żądań strony po włączeniu śledzenia.

Instalowanie platformy ASP.NET

Najłatwiejszym sposobem zainstalowania platformy ASP.NET jest instalacja pakietu Visual Web Developer Express. Pakiet Visual Web Developer Express został umieszczony na płycie CD dołączonej do książki (najnowszą wersję pakietu można również pobrać z witryny <http://www.asp.net/> będącej oficjalną witryną Microsoft ASP.NET).

Instalowanie pakietu Visual Web Developer Express powoduje również zainstalowanie następujących komponentów:

- ◆ Microsoft .NET Framework w wersji 2.0
- ◆ SQL Server Express

Pakiet Visual Web Developer Express jest zgodny z następującymi systemami operacyjnymi:

- ◆ Windows 2000 Service Pack 4
- ◆ Windows XP Service Pack 2
- ◆ Windows Server 2003 Service Pack 1
- ◆ Windows x64
- ◆ Windows Vista

Zaleca się także pobranie pakietu .NET Framework SDK (Software Development Kit). Pakiet SDK zawiera dodatkową dokumentację, przykładowe fragmenty kodu oraz narzędzia do budowania aplikacji ASP.NET. Wspomniany pakiet SDK jest dostępny do pobrania na witrynie internetowej Microsoft MSDN znajdującej się pod adresem <http://msdn.microsoft.com/>.

Instalacja pakietu Visual Web Developer Express może zostać przeprowadzona na komputerze z zainstalowanym pakietem Visual Studio .NET 2003. Obydwa środowiska programistyczne mogą ze sobą koegzystować bez problemów.

Co więcej, ten sam serwer sieciowy może obsługiwać strony utworzone zarówno w technologii ASP.NET 1.1, jak i ASP.NET 2.0. Każda wersja platformy .NET jest zainstalowana w następującym katalogu:

```
C:\WINDOWS\Microsoft.NET\Framework
```

Przykładowo na komputerze autora znajdują się trzy zainstalowane wersje platformy .NET (wersja 1.0, wersja 1.1 oraz wersja 2.0):

```
C:\WINDOWS\Microsoft.NET\Framework\v1.0.3705  
C:\WINDOWS\Microsoft.NET\Framework\v1.1.4322  
C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727
```

Każdy z wymienionych katalogów zawiera narzędzie wiersza polecenia o nazwie *aspnet_regiis.exe*. Jest ono używane do przypisania określonego katalogu wirtualnego na komputerze lokalnym do danej platformy .NET.

Przykładowo uruchomienie z wiersza polecenia poniższego polecenia spowoduje włączenie określonej wersji ASP.NET dla katalogu wirtualnego o nazwie *MojaAplikacja*:

```
aspnet_regiis -s W3SVC/1/ROOT/MojaAplikacja
```

Uruchomienie narzędzia *aspnet_regiis.exe* umieszczonego w różnych katalogach wersji .NET Framework umożliwia programiście mapowanie określonego katalogu wirtualnego do dowolnej wersji platformy ASP.NET.

Podsumowanie

W rozdziale znalazł się wstęp do platformy ASP.NET 2.0. W pierwszej kolejności zbudowaliśmy prostą stronę ASP.NET. Poznaliśmy trzy główne elementy strony ASP.NET: dyrektywy, bloki deklarowania kodu oraz bloki generowania strony.

Następnie dokonaliśmy analizy platformy .NET. Dowiedzieliśmy się przy okazji o istnieniu około 13 000 klas zawartych w bibliotece Framework Class Library oraz o funkcjach środowiska Common Language Runtime.

Rozdział ten zaprezentował również ogólny opis kontrolek ASP.NET. Dowiedzieliśmy się o różnych grupach kontroltek, które zostały umieszczone w platformie .NET. Nauczyliśmy się także sposób obsługiwać zdarzenia kontroltek i wykorzystywać zalety stanu widoku.

Przeanalizowaliśmy również strony ASP.NET i dowiedzieliśmy się, w jaki sposób strony ASP.NET są dynamicznie kompilowane w trakcie pierwszego żądania ich wyświetlenia. Poznaliśmy techniki podziału strony ASP.NET składającej się z pojedynczego pliku na stronę ukrytego kodu, a ponadto, nauczyliśmy się metod debugowania i śledzenia wykonywania strony ASP.NET.

Na końcu zostały przedstawione kwestie związane z uruchomieniem platformy ASP.NET. Dowiedzieliśmy się, w jaki sposób mapować różne katalogi wirtualne do różnych wersji platformy ASP.NET.