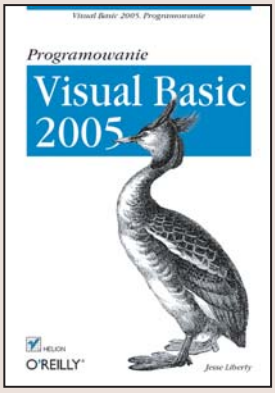
Visual Basic 2005. Programowanie



Autor: Jesse Liberty
Tłumaczenie: Daniel Kaczmarek
ISBN: 83-246-0342-5
Tytuł oryginału: Programming Visual Basic 2005
Format: B5, stron: 568

Naucz się tworzyć aplikacje dla systemu Windows oraz aplikacje WWW

- Projektowanie formularzy i korzystanie z kontrolek
- Komunikacja z bazami danych
- Tworzenie usług sieciowych

Visual Basic to jeden z najpopularniejszych obecnie języków programowania. Szerokie grono programistów opracowuje z jego pomocą aplikacje dla systemu Windows oraz aplikacje internetowe. Integracja z platformą .NET otwiera przed jego użytkownikami jeszcze większe możliwości. Dzięki ogromnej liczbie gotowych kontrolek i czytelnej składni pisanie aplikacji przebiega błyskawicznie. Programista może skoncentrować się na zadaniu, nie poświęcając zbyt wiele czasu na działania drugorzędne.

Książka „Visual Basic 2005. Programowanie" to podręcznik przedstawiający zasady tworzenia aplikacji dla Windows oraz aplikacji sieciowych w Visual Basicu.
Nie znajdziesz tu teoretycznych wywodów i długich opisów. Od pierwszego rozdziału zaczniesz poznawać praktyczne aspekty programowania. Stworzysz aplikację operującą na danych pobieranych z bazy, aplikację internetową oraz usługi sieciowe.
Wszystkie elementy języka Visual Basic poznasz, pracując nad konkretnym projektem.

- Projektowanie formularza i obsługa zdarzeń
- Dostęp do danych zgromadzonych w bazie
- Korzystanie z gotowych kontrolek i tworzenie własnych
- Tworzenie obiektów graficznych za pomocą biblioteki GDI+
- Budowanie aplikacji internetowej
- Strony wzorcowe i mechanizmy nawigacyjne
- Personalizacja aplikacji internetowej
- Korzystanie ze środowiska Visual Studio 2005
- Zasady programowania obiektowego

Jeśli chcesz szybko opanować metody tworzenia aplikacji w języku Visual Basic 2005 — koniecznie sięgnij po tę książkę

Wydawnictwo Helion
ul. Chopina 6
44-100 Gliwice
tel. (32)230-98-63
e-mail: helion@helion.pl

Spis treści

ROZDZIAŁ 8.

Projektowanie aplikacji WWW i tworzenie pierwszych formularzy

W niniejszym rozdziale zaczniemy tworzyć aplikację WWW. Podobnie jak w pierwszej części książki szybko przejdziemy do istoty rzeczy, jednak w tym przypadku, zanim zaczniemy tworzenie aplikacji, niezbędne jest krótkie wprowadzenie. Informacje zawarte we wprowadzeniu będą stanowić podstawę wszystkich dalszych czynności, lecz postaram się, by było ono jak najkrótsze.

W procesie tworzenia nowej aplikacji wyróżnia się pięć podstawowych, zachodzących na siebie faz: *analizę*, *projektowanie*, *implementację*, *testowanie* i *wdrożenie*. W rozdziale 1. opisano wszystkie wymienione fazy, które (oprócz wdrożenia) przebiegają identycznie bez względu na to, czy tworzona jest aplikacja WWW, czy aplikacja dla systemu Windows.

Kluczowa różnica między aplikacją dla systemu Windows a aplikacją WWW to przebieg fazy wdrożenia. Aplikacje wdrażane w sieci WWW nie muszą być dystrybuowane wśród klientów, lecz wystarczy wdrożyć je na „serwerze produkcyjnym" (czyli na komputerze, do którego będą łączyć się klienci) — wówczas aplikacja stanie się powszechnie dostępna.

Części I i II są od siebie zasadniczo niezależne. Nie trzeba czytać najpierw części pierwszej, by zrozumieć materiał zamieszczony w części drugiej. Jednak w przypadku podobnych elementów w tekście znajdzie się odniesienie do zagadnień, które były już opisywane we wcześniejszych rozdziałach. Dzięki temu unikniemy powtarzania tych samych informacji.

W niniejszym rozdziale zostaną zdefiniowane wymagania dla prawdziwej aplikacji WWW, natomiast dalej będziemy się zajmować zagadnieniami ściśle związanymi z implementacją. Decyzje dotyczące konstrukcji aplikacji będą podejmowane z biegiem czasu. Ponadto będziemy się trzymać podstawowego założenia, by uruchomić dany mechanizm i utrzymywać jego stałą gotowość do pracy.

Wymagania dla aplikacji WWW będą podobne — choć nie takie same — jak wymagania dla aplikacji dla systemu Windows z pierwszej części książki. Celem jest zademonstrowanie sposobu, w jaki sieć WWW zarówno rozszerza, jak i ogranicza rzeczywiste możliwości aplikacji. Przekonamy się, że implementacja będzie w niektórych przypadkach uderzająco podobna do implementacji aplikacji dla systemu Windows, zaś w niektórych przypadkach wystąpią daleko idące różnice.

Niniejsza książka nie opisuje języka HTML ani projektowania stron WWW (doskonałą książką na temat projektowania stron WWW w języku HTML jest *HTML i XHTML: Przewodnik encyklopedyczny, Helion 2001, Gliwice*).

Nie będziemy poświęcać czasu na przedstawienie sposobów tworzenia czytelnych, użytecznych i atrakcyjnych stron w języku HTML, strony generowane przez naszą aplikację *celowo* będą bardzo skromne. Tak naprawdę będą one zaledwie stanowić obszar zablokowany dla profesjonalnego interfejsu użytkownika, który należy zaprojektować, co z kolei wykracza już poza zakres książki. W kręgu naszego zainteresowania będzie zatem leżeć nie wygląd stron, ale ich funkcje.

Sposób działania formularzy WWW

Zanim rozpoczniemy tworzenie aplikacji, przede wszystkim niezbędne jest krótkie przedstawienie architektury aplikacji ASP.NET. Jej zrozumienie pozwoli pisać działający kod w bardzo krótkim czasie.

Kluczowym elementem aplikacji ASP.NET jest tworzenie i interakcja między formularzami WWW. Formularze WWW implementują model programistyczny, w którym strony WWW są generowane dynamicznie na serwerze WWW w celu dostarczenia ich do przeglądarki za pośrednictwem sieci internet. Dzięki formularzom WWW ASP.NET można tworzyć strony ASPX z mniej lub bardziej statyczną zawartością, zawierające kod HTML oraz kontrolki WWW. Można również pisać kod w języku Visual Basic 2005, który będzie dodawał zawartość dynamiczną. Kod w języku Visual Basic 2005 *działa na serwerze*, a generowane przez niego dane są łączone z zadeklarowanymi obiektami strony i tworzą razem stronę HTML, która zostaje przesłana do przeglądarki.

W powyższym akapicie znajdują się trzy kluczowe zagadnienia, o których należy pamiętać przez cały czas lektury niniejszego rozdziału:

- Strony WWW mogą zawierać zarówno kod HTML, jak i kontrolki WWW (które zostaną opisane w późniejszym czasie).
- Przetwarzanie jest w całości wykonywane na serwerze (można oczywiście przenieść część przetwarzania na klienta przy użyciu języków skryptowych, jednak nie stanowią one części ASP.NET i dlatego nie będą opisywane w książce).
- Jeśli używane są kontrolki WWW ASP.NET, przeglądarka nadal będzie widzieć tylko kod HTML (od tej reguły istnieją jednak wyjątki: do niektórych nowocześniejszych przeglądarek może być wysyłany również kod skryptu). Oznacza to, że pomimo faktu, iż kontrolki WWW ASP.NET stanowią zupełnie nowe narzędzie tworzenia aplikacji WWW, przeglądarka otrzymuje strony wyłącznie w języku HTML.

Formularze WWW ASP.NET 2.0 są następcami niezwykle popularnych formularzy WWW ASP.NET 1.x, które z kolei zastąpiły strony ASP. ASP.NET utworzono po to, by pracochłonność kodowania zmniejszyć w stosunku do ASP 1.x o 70%. Oznacza to, że *programowanie* aplikacji WWW ma charakter coraz bardziej *deklaratywny*, a nie programistyczny — to znaczy kontrolki na stronach WWW się deklaruje, a nie trzeba pisać (i przepisywać) czystego kodu.

Programista nadal może pisać kod (pisanie kodu jest zawsze możliwe), jednak w przypadku znakomitej większości zadań programistycznych dla sieci WWW programista będzie pisał w ASP.NET 2.0 o wiele mniej kodu niż w ASP.NET 1.x.

Formularze WWW zaprojektowano w taki sposób, by można je było przeglądać w dowolnej przeglądarce — kod HTML odpowiedni dla danej przeglądarki jest generowany przez serwer. Logikę formularza WWW można zaprogramować w dowolnym języku .NET. My oczywiście będziemy używać języka Visual Basic 2005. Ponieważ Visual Studio zdecydowanie ułatwia proces projektowania i testowania formularzy WWW, w niniejszej książce do tworzenia aplikacji WWW będziemy używać wyłącznie Visual Studio 2005.

W formularzach WWW interfejs użytkownika jest dzielony na dwie części: część widoczną, albo interfejs użytkownika (ang. *user interface* — UI), oraz logikę formularza. Konstrukcja ta jest bardzo podobna do konstrukcji formularzy Windows. Podział między plikiem, który zawiera interfejs użytkownika, oraz odpowiadającym mu plikiem z kodem źródłowym to tak zwana `separacja kodu`. Kod języka Visual Basic 2005 można umieszczać w tym samym pliku, w którym znajduje się interfejs użytkownika (na przykład w języku HTML).

W ASP.NET w wersji 2.0 Visual Studio wykorzystuje klasy częściowe, dzięki którym strony z odseparowanym kodem tworzy się znacznie łatwiej niż w ASP.NET 1.x. Dzięki temu, że odseparowany kod oraz strona deklaracji stanowią części tej samej klasy, Visual Studio może ukryć kod inicjujący w oddzielnym pliku.

Strona z interfejsem użytkownika jest przechowywana w pliku z rozszerzeniem *.aspx*. Gdy strona jest wywoływana przez przeglądarkę, serwer uruchamia kod powiązany ze stroną i generuje kod HTML, który zostaje odesłany do przeglądarki klienta. Aplikacja ASP.NET korzysta z bogatych kontrolek WWW znajdujących się w przestrzeniach nazw `System.Web` i `System.Web.UI` biblioteki .NET Framework Class Library (FCL).

Trudno wyobrazić sobie prostsze programowanie formularzy WWW niż w Visual Studio 2005. Wystarczy otworzyć formularz, przeciągnąć na niego kontrolki i napisać kod obsługujący zdarzenia. I to wszystko! W ten sposób powstaje kompletna aplikacja WWW.

Z drugiej strony, napisanie wydajnej i bogatej aplikacji WWW nawet w Visual Studio 2005 może być trudnym zadaniem. Formularze WWW mogą oferować bardzo rozbudowany interfejs użytkownika, ponadto istnieje wiele kontrolek WWW, których zadaniem jest usprawnienie pracy, lecz na początku tak duża różnorodność może przytłaczać.

Zdarzenia formularza WWW

Podobnie jak formularze Windows, które tworzyliśmy w części I, formularze WWW również są sterowane zdarzeniami. Nastąpienie *zdarzenia* oznacza, że „coś się stało".

Zdarzenie jest generowane (lub *wywoływane*), gdy użytkownik naciska przycisk, wybiera pozycję w polu listy albo w inny sposób wchodzi w interakcję z interfejsem użytkownika. Zdarzenia mogą być również generowane przez system rozpoczynający lub kończący jakieś zadanie. Na przykład, użytkownik może otworzyć plik do odczytu, a w momencie wczytania całego pliku do pamięci system wywołuje odpowiednie zdarzenie.

Metoda, która odpowiada na zdarzenie, to tak zwana *procedura obsługi zdarzenia*. Procedury obsługi zdarzeń pisze się w języku Visual Basic 2005 i są one powiązane z kontrolkami znajdującymi się na stronie HTML za pośrednictwem atrybutów kontrolek.

Zgodnie z konwencją procedury obsługi zdarzeń ASP.NET są procedurami **Sub** (a nie funkcjami) i wymagają podania dwóch parametrów. Pierwszy parametr reprezentuje obiekt, który wywołał zdarzenie. Drugi parametr natomiast, tak zwany *argument zdarzenia*, zawiera informacje na temat danego zdarzenia. W większości przypadków argument zdarzenia jest typu `EventArgs`, który nie udostępnia żadnych właściwości i pełni tak naprawdę jedynie rolę obszaru zablokowanego. Jednak w przypadku niektórych kontrolek argument zdarzenia może być typu potomnego po `EventArgs`, który udostępnia właściwości charakterystyczne dla zdarzenia danego typu.

Na przykład, w momencie wiązania wiersza do siatki `GridView` (rozdział 10.) wywoływane jest zdarzenie, które przekazuje obiekt `GridViewRowEventArgs` potomny po `EventArgs`. Obiekt `GridViewRowEventArgs` posiada właściwość `Rows`, która zwraca kolekcję obiektów `GridViewRow`. Z kolei obiekty `GridViewRow` udostępniają wszystkie atrybuty odpowiadających wierszy, w tym także obiekt danych (`DataItem`) z danymi, które znajdują się w wierszu.

W aplikacjach WWW zdarzenia są obsługiwane na serwerze i wymagają komunikacji dwukierunkowej (ang. *roundtrip*). Wyjątkiem jest obsługa zdarzeń skryptów klienckich, która nie jest opisywana w niniejszej książce. ASP.NET obsługuje jedynie ograniczony zbiór zdarzeń, takich jak kliknięcie przycisku czy zmiana tekstu. Są to bowiem zdarzenia, po których użytkownik może się spodziewać, że spowodują istotne zmiany na stronie, w przeciwieństwie do szerokiego zbioru zdarzeń obsługiwanych w aplikacjach dla systemu Windows (które były tematem części I), takich jak zdarzenia myszy wywoływane wiele razy w trakcie wykonywania przez użytkownika pojedynczego zadania.

Model zdarzeń

Wyróżnia się dwa następujące modele wykonania programów: *linearny* oraz *sterowany zdarzeniami*. Obydwa modele nie wykluczają się wzajemnie.

W programach linearnych najpierw wykonywany jest krok 1., potem krok 2. i kolejne, aż wykonane zostaną wszystkie kroki. Przebieg wykonania programu może być zmieniany przez struktury sterujące przebiegiem, takie jak pętle, instrukcje **`If`**, wywołania funkcji albo procedur. Jednak zasadniczo po uruchomieniu programu ani czynności wykonywane przez użytkownika, ani wykonywane przez system nie mają już wpływu na przebieg wykonania programu. Przed powstaniem środowisk graficznych większość programów komputerowych miała charakter linearny.

W przeciwieństwie do programów linearnych, programy sterowane zdarzeniami odpowiadają na sytuacje, w których coś się dzieje (na przykład użytkownik klika przycisk). Zdarzenia są najczęściej generowane przez czynności użytkownika, mogą także powstawać w wyniku rozpoczęcia albo zakończenia jakiegoś zadania przez system. Na przykład, system może wywołać zdarzenie, gdy plik otwierany do odczytu zostanie wczytany do pamięci albo gdy zacznie się wyczerpywać moc baterii.

W ASP.NET obiekty mogą wywoływać zdarzenia, a innym obiektom można przypisywać procedury obsługi zdarzeń. Na przykład, przycisk może wywoływać zdarzenie `Click`, a strona może posiadać metodę, która obsługuje zdarzenie polegające na kliknięciu przycisku (czyli `Button_Click`). Procedura obsługi zdarzenia odpowiada na kliknięcie przycisku w sposób właściwy dla danej aplikacji.

Zdarzenia ASP.NET

W klasycznym ASP wyróżniano sześć zdarzeń, z których w powszechnym użyciu znajdowały się tylko cztery. Były to następujące zdarzenia:

`Application_OnStart`
: Zdarzenie było wywoływane w momencie uruchomienia aplikacji.

`Application_OnEnd`
: Zdarzenie było wywoływane w momencie zakończenia aplikacji.

`Session_OnStart`
: Zdarzenie było wywoływane na początku każdej sesji.

`Session_OnEnd`
: Zdarzenie było wywoływane w momencie zakończenia każdej sesji.

Z kolei w ASP.NET występują dosłownie tysiące zdarzeń. Zdarzenia generuje aplikacja, każda sesja generuje zdarzenia, strona i większość kontrolek serwera również generują zdarzenia. Wszystkie procedury obsługi zdarzenia ASP.NET są wykonywane na serwerze. Niektóre zdarzenia od razu powodują wysłanie żądania do serwera, inne natomiast są przechowywane do momentu, gdy strona zostanie przesłana na serwer, i dopiero wtedy następuje ich wykonanie.

Ze względu na to, że zdarzenia ASP.NET są wykonywane na serwerze, różnią się one nieco od zdarzeń w tradycyjnych aplikacjach klienckich, w których zarówno zdarzenie, jak i procedura obsługi zdarzenia znajdują się na kliencie. W aplikacjach ASP.NET zdarzenie jest zazwyczaj generowane na kliencie (na przykład w wyniku kliknięcia przez użytkownika przycisku wyświetlanego przez przeglądarkę), natomiast obsługa zdarzenia odbywa się na serwerze.

Wyobraźmy sobie klasyczną stronę WWW ASP zawierającą kontrolkę przycisku. W momencie naciśnięcia przycisku generowane jest zdarzenie `Click`. Zdarzenie jest obsługiwane przez klienta (czyli przeglądarkę), która reaguje, wysyłając formularz na serwer. W tym przypadku na serwerze zdarzenie nie jest obsługiwane.

Teraz wyobraźmy sobie stronę WWW ASP.NET z podobną kontrolką przycisku. Różnica między kontrolką przycisku ASP.NET a klasyczną kontrolką przycisku w języku HTML jest taka, że przycisk ASP.NET posiada atrybut `runat=server`, dzięki któremu programista może implementować przetwarzanie na serwerze dodatkowe względem standardowych funkcji przycisku HTML.

Gdy generowane jest zdarzenie `Click`, przeglądarka obsługuje zdarzenie po stronie klienta, przesyłając stronę na serwer. Tym razem jednak do serwera jest przesyłany również komunikat zdarzenia. Serwer ustala, czy zdarzeniu `Click` przypisano procedurę obsługi. Jeśli tak, procedura obsługi zdarzenia zostaje wykonana na serwerze.

Komunikat zdarzenia jest przesyłany na serwer za pośrednictwem żądania HTTP POST. ASP.NET autogamicznie (to takie techniczne określenie) obsługuje wszystkie mechanizmy odpowiedzialne za przechwycenie zdarzenia, przesłanie go na serwer i przetworzenie. Zadanie programisty sprowadza się jedynie do utworzenia procedur obsługi zdarzeń.

Wiele zdarzeń, takich jak `MouseOver`, nie nadaje się do przetwarzania na serwerze, ponieważ drastycznie zmniejszyłoby to wydajność. Przetwarzanie na serwerze wymaga zwrotnej komunikacji (ang. *postback* — przesłanie danych do serwera i z powrotem), a przecież nie chcemy, by strona była wysyłana z powrotem na serwer za każdym razem, gdy nastąpi zdarzenie `MouseOver`. Jeżeli takie zdarzenia w ogóle są obsługiwane, to tylko na kliencie (przy użyciu skryptu) i poza zasięgiem ASP.NET.

Zdarzenia aplikacji i sesji

ASP.NET obsługuje zdarzenia aplikacji i sesji analogiczne do zdarzeń w ASP. Zdarzenie `Application_OnStart` jest wywoływane, gdy następuje uruchomienie aplikacji. Jest to dobry moment na zainicjowanie zasobów, które będą używane przez aplikację, takich jak ciągi połączeń z bazą danych (lecz nie samo połączenie z bazą). Zdarzenie `Application_OnEnd` jest wywoływane wówczas, gdy aplikacja zostaje zakończona. W tym momencie można zamknąć zasoby i wykonać inne niezbędne czynności porządkowe. Warto zwrócić uwagę, że mechanizm odśmiecania sam zatroszczy się o zwolnienie pamięci, lecz jeśli zaalokowane zostaną zasoby, które nie są zarządzane, na przykład komponenty napisane w języku niezgodnym z .NET Framework, programista sam będzie musiał je usunąć.

Podobnie rzecz ma się ze zdarzeniami sesji. Sesja zaczyna się w momencie, gdy użytkownik po raz pierwszy zażąda strony od aplikacji, i kończy się, gdy aplikacja zamknie sesję lub czas ważności sesji dobiegnie końca. Zdarzenie `Session_OnStart` jest wywoływane w momencie rozpoczęcia sesji; w tym też czasie można zainicjować zasoby, które sesja będzie wykorzystywać, na przykład otworzyć połączenie z bazą danych. W momencie zakończenia sesji zachodzi zdarzenie `Session_OnEnd`.

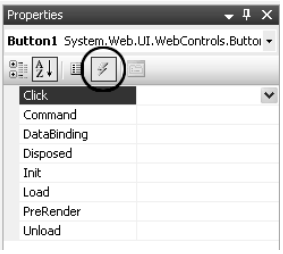
Zdarzenia strony są automatycznie obsługiwane przez metody o następujących nazwach:

- `Page_AbortTransaction`
- `Page_CommitTransaction`
- `Page_DataBinding`
- `Page_Dispose`
- `Page_Error`
- `Page_Init`
- `Page_InitComplete`
- `Page_Load`
- `Page_LoadComplete`
- `Page_PreInit`
- `Page_PreLoad`
- `Page_PreRender`
- `Page_PreRenderComplete`
- `Page_SaveStateComplete`
- `Page_Unload`

Zdarzenia w Visual Studio .NET

Zintegrowane środowisko programistyczne Visual Studio .NET potrafi automatycznie obsłużyć większość czynności wymaganych do zaimplementowania zdarzeń w ASP.NET. Na przykład, Visual Studio udostępnia listę wszystkich zdarzeń dostępnych dla poszczególnych kontrolek, a gdy programista zdecyduje się na zaimplementowanie zdarzenia, wystarczy wpisać nazwę procedury obsługi zdarzenia. W odpowiedzi IDE utworzy szablon niezbędnego kodu i połączy go z odpowiednim obiektem.

Dodawane kontrolki będą posiadać własne zdarzenia, które programista może samodzielnie obsługiwać. Po dodaniu kontrolki można sprawdzić dostępne zdarzenia, klikając ją myszą, a następnie klikając w oknie *Properties* przycisk zdarzeń (przycisk z błyskawicą). Przykładowe zdarzenia dla przycisku są widoczne na rysunku 8.1.



Rysunek 8.1. Zdarzenia przycisku

W polu obok zdarzenia można wpisać nazwę metody. Można również dwukrotnie kliknąć w tym miejscu — spowoduje to automatyczne dodanie procedury obsługi zdarzenia. Na ekranie pojawi się wówczas okno z kodem źródłowym, a kursor będzie znajdował się w procedurze obsługi zdarzenia gotowej do zaimplementowania przez programistę.

Każda kontrolka posiada zdarzenie domyślne, którym jest zwykle najczęściej implementowane zdarzenie kontrolki. Nietrudno zgadnąć, że domyślnym zdarzeniem przycisku jest zdarzenie `Click`. W celu utworzenia domyślnej procedury obsługi zdarzenia wystarczy dwukrotnie kliknąć kontrolkę. Jeżeli więc procedura `Button1_Click` nie została jeszcze utworzona przez którąś z wcześniej wspomnianych metod, można przejść do widoku projektowego i dwukrotnie kliknąć przycisk. Efekt będzie identyczny: utworzona zostanie procedura obsługi zdarzenia o nazwie `Button1_Click`, a programista zostanie przeniesiony do kodu procedury i będzie od razu mógł przystąpić do implementacji metody.

Domyślne zdarzenia najczęściej używanych kontrolek WWW przedstawiono w tabeli 8.1.

Tabela 8.1. Domyślne zdarzenia dla niektórych kontrolek ASP.NET

Kontrolka	Zdarzenie domyślne
`Button`	`Click`
`Calendar`	`SelectionChanged`
`CheckBox`	`CheckedChanged`
`CheckBoxList`	`SelectedIndexChanged`
`DataGrid`	`SelectedIndexChanged`
`DataList`	`SelectedIndexChanged`
`DropDownList`	`SelectedIndexChanged`
`HyperLink`	`Click`
`ImageButton`	`Click`
`Label`	Brak
`LinkButton`	`Click`
`ListBox`	`SelectedIndexChanged`
`RadioButton`	`CheckedChanged`
`RadioButtonList`	`SelectedIndexChanged`
`Repeater`	`ItemCommand`

Przypisywanie procedury obsługi zdarzenia więcej niż jednej kontrolce

Istnieje możliwość, by jedna procedura obsługi zdarzenia obsługiwała zdarzenia wywoływane przez kilka różnych kontrolek. Można na przykład utworzyć ogólną procedurę obsługi zdarzenia `Click`, która będzie obsługiwać kliknięcia wszystkich przycisków znajdujących się na formularzu. Przycisk, który wywołał zdarzenie, można ustalić, sprawdzając wartość parametru `sender`. W poniższym fragmencie kodu procedura obsługi zdarzenia `Click` rzutuje obiekt `sender` (czyli kontrolkę, która wywołała zdarzenie) na typ `Button`, po czym przypisuje właściwość `ID` przycisku zmiennej typu **`String`**.

```
Protected Sub GenericButton_Click( _
ByVal sender As Object, _
ByVal e As System.EventArgs) Handles btnOrder.Click
   Dim b As Button = CType(sender, Button) 'rzutowanie obiektu na przycisk
   Dim buttonID As String = b.ID           'przypisanie identyfikatora przycisku
End Sub
```

W ten sposób można zaoszczędzić znaczną ilość czasu, jaką trzeba by poświęcić na napisanie powtarzającego się kodu. Ponadto dzięki temu program staje się bardziej czytelny i łatwiejszy w utrzymaniu.

Zdarzenia wymagające komunikacji zwrotnej a pozostałe zdarzenia

Zdarzenia wymagające komunikacji zwrotnej z serwerem (ang. *postback events*) to takie zdarzenia, które powodują natychmiastowe przesłanie formularza z powrotem na serwer. Są to zdarzenia podobne do kliknięcia, na przykład kliknięcie przycisku. W przeciwieństwie do nich, wiele zdarzeń (zwykle zdarzenia zmian) nie wymaga komunikacji zwrotnej, to znaczy po ich wystąpieniu formularz nie jest od razu przesyłany z powrotem do serwera. Takie zdarzenia są zapisywane w pamięci podręcznej kontrolki i pozostają tam do momentu, gdy nastąpi komunikacja zwrotna z serwerem.

Na kontrolkach, które nie wymagają komunikacji zwrotnej, można wymusić, by zachowywały się jak zdarzenia wymagające takiej komunikacji. W tym celu właściwości `AutoPostBack` zdarzeń trzeba przypisać wartość **`True`**.

Stan

Stan aplikacji WWW to bieżąca wartość wszystkich kontrolek i wszystkich zmiennych dla bieżącego użytkownika i bieżącej sesji. Jednak sieć WWW z natury jest środowiskiem „bezstanowym". Oznacza to, że standardowo stan każdego żądania wysłanego do serwera jest tracony, chyba że programista zada sobie trochę trudu i zapisze informacje o sesji. Na szczęście ASP.NET obsługuje mechanizm, który utrzymuje stan sesji użytkownika.

Za każdym razem gdy strona zostaje przesłana na serwer, ten przed odesłaniem strony z powrotem do przeglądarki odtwarza ją od nowa. ASP.NET posiada mechanizm, który automatycznie utrzymuje stan kontrolek serwera (właściwość `ViewState`) niezależny od sesji HTTP.

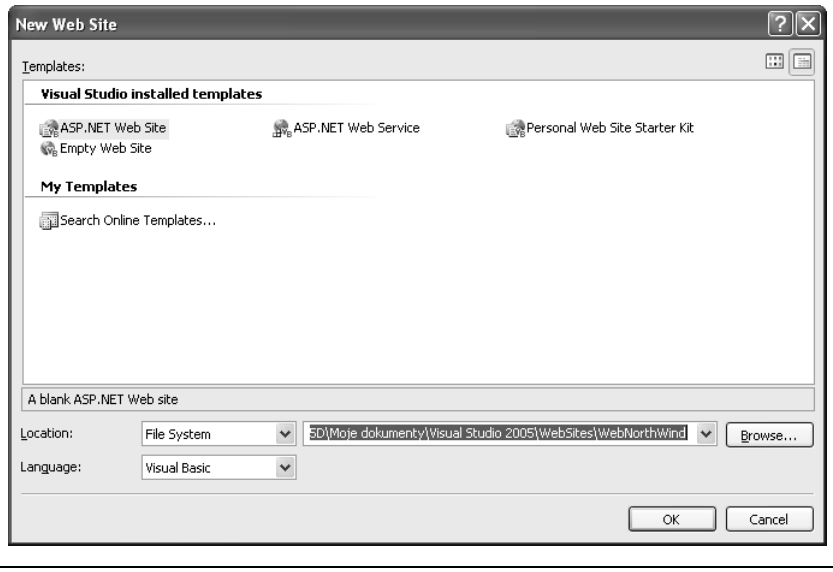
Dzięki temu, gdy użytkownik wybierze pozycję z listy, jego wybór zostanie zapamiętany w momencie przesyłania strony na serwer i odtworzony na kliencie.

Stany innych obiektów (które nie są kontrolkami) nie są automatycznie zapamiętywane przez `ViewState` i muszą być zapisywane przez programistę we właściwości `ViewState` albo `SessionState`, które zostaną opisane w dalszych punktach.

Sesja HTTP daje złudne wrażenie istnienia połączenia między użytkownikiem i aplikacją WWW mimo tego, że sama sieć WWW jest środowiskiem bezstanowym, w którym żadne połączenia nie istnieją.

Pierwsze kroki

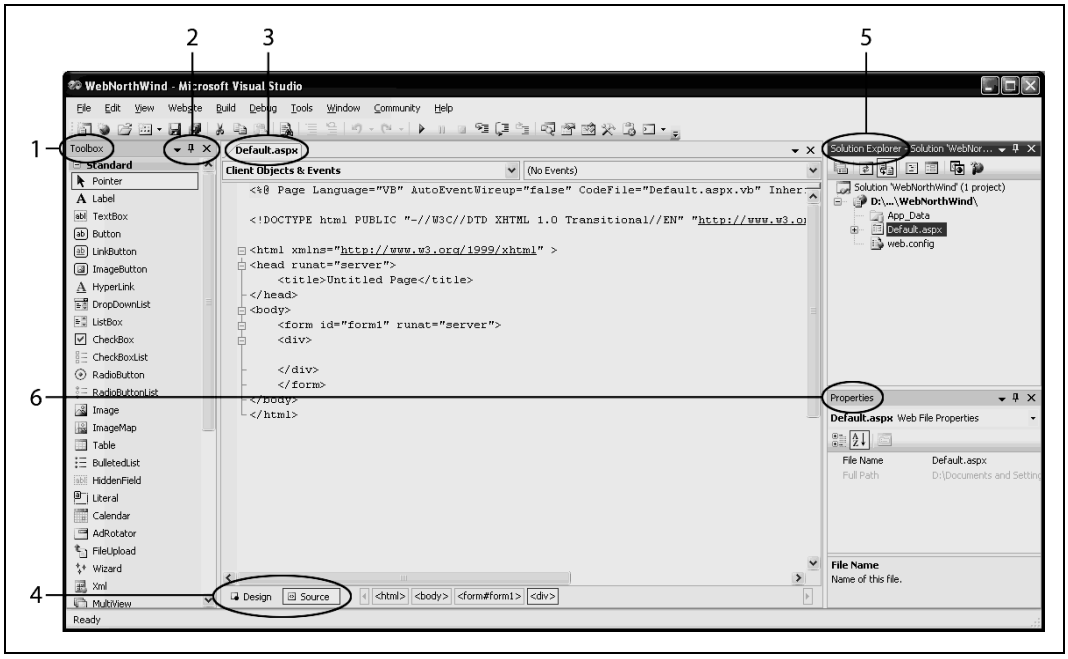
Na początek utworzymy aplikację WWW o nazwie `WebNorthWind`. W tym celu trzeba otworzyć Visual Studio 2005, wybrać polecenie *New Web Site* i na liście rozwijanej wskazać *File System*, a także wskazać lokalizację pliku. Jako język należy wskazać Visual Basic, jak na rysunku 8.2.



Rysunek 8.2. Tworzenie nowej witryny WWW

Visual Studio 2005 utworzy witrynę WWW znajdującą się w systemie plików (to znaczy nie będzie ona widoczna w narzędziu *Internetowe Usługi Informacyjne*). Utworzony zostanie również plik *Default.aspx* reprezentujący pierwszy formularz ASP.NET. Nastąpi otwarcie edytora i widoczne stanie się okno narzędziowe *Toolbox* z kontrolkami WWW (jeśli okno nie będzie widoczne, można je wyświetlić, wybierając polecenie z menu *View*). Podobnie jak wszystkie inne okna, okno narzędziowe można zadokować, klikając ikonę pinezki.

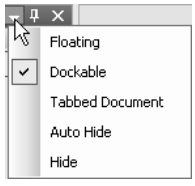
Zależnie od sposobu, w jaki skonfigurowano Visual Studio, najprawdopodobniej wyświetlony zostanie widok kodu źródłowego zawierający okno z zakładkami, dzięki którym będzie można się przełączać do trybu projektowania WYSIWYG (ang. *What You See Is What You Get*). Ilustruje to rysunek 8.3.



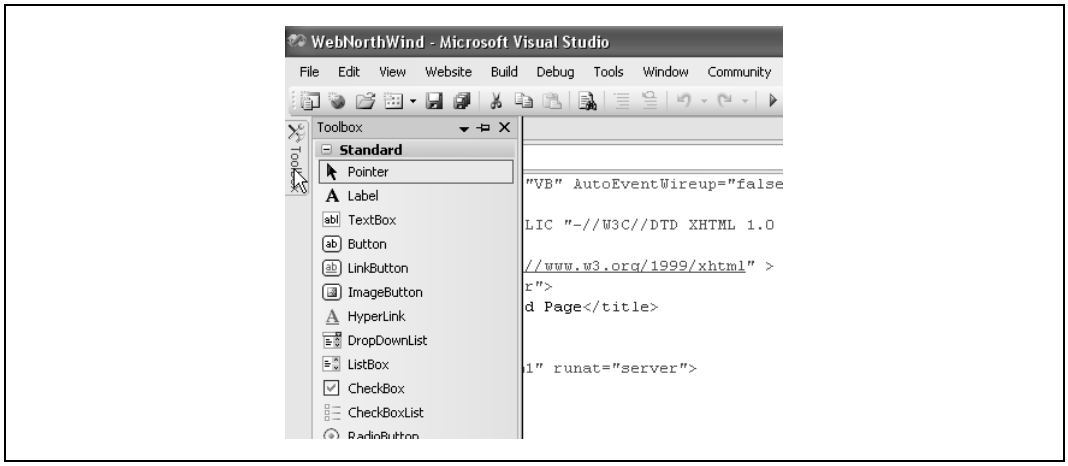
Rysunek 8.3. Edytor tworzenia stron WWW

Na rysunku oznaczono cyframi sześć elementów:

1. Okno narzędziowe *Toolbox* zawierające kilka rozwijalnych kolekcji kontrolek, które można dodawać do aplikacji (na rysunku widoczna jest kolekcja standardowa). Dowolną kontrolkę można kliknąć prawym przyciskiem myszy i posortować kolekcję alfabetycznie, jak na rysunku.
2. Kontrolki operacji na oknie. Kliknięcie strzałki w dół pozwoli na zmianę umiejscowienia okna, co ilustruje rysunek 8.4. Po kliknięciu przycisku pinezki okno zostanie otwarte w tym samym miejscu, a jeśli w momencie kliknięcia okno jest już otwarte, włączony zostanie tryb autoukrywania i obok okna edycyjnego utworzona zostanie zakładka. Wówczas umieszczenie kursora myszy na zakładce spowoduje wyświetlenie okna, a po usunięciu kursora z obszaru okna okno *Toolbox* ponownie się ukryje. Sytuację tę ilustruje rysunek 8.5. (Ponowne kliknięcie przycisku myszy spowoduje, że okno znów zostanie zadokowane). Kliknięcie przycisku *X* prowadzi do zamknięcia okna *Toolbox* (można je ponownie otworzyć w menu *View*).



Rysunek 8.4. Sposób umiejscowienia okna



Rysunek 8.5. Autoukrywanie okna Toolbox

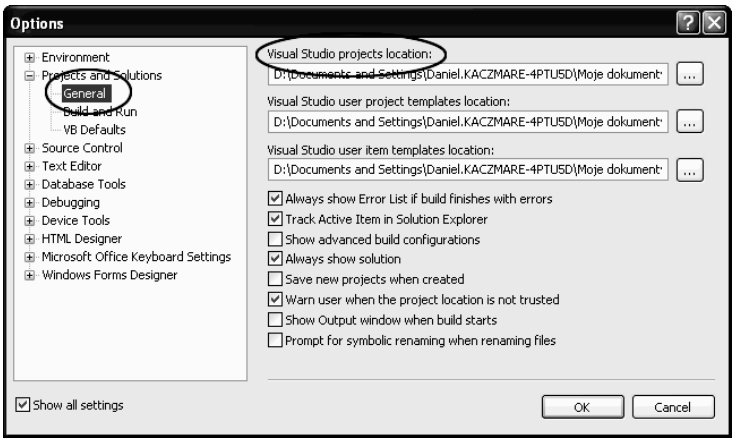
3. Zakładka wskazująca formularz, z którym aktualnie pracuje programista (w danym momencie może być otwartych wiele formularzy i każdy z nich będzie miał własną zakładkę).
4. Zakładka, dzięki której można przełączać się między widokiem kodu źródłowego *Source* i widokiem projektowania Design. Z okna *Toolbox* można przeciągać kontrolki bezpośrednio do obu widoków.
5. W oknie *Solution Explorer* widoczne są nazwy projektów i pliki wchodzące w skład poszczególnych projektów rozwiązania WWW. Rozwiązanie jest kolekcją projektów i każdy projekt jest zwykle kompilowany do postaci podzespołu.
6. Okno właściwości *Properties*. Po kliknięciu kontrolki (lub formularza) zawartość okna *Properties* ulegnie zmianie i będzie przedstawiać właściwości (lub zdarzenia) danej kontrolki.

We wskazanym katalogu Visual Studio utworzy folder o nazwie *WebNorthWind*. W folderze utworzone zostaną strony *Default.aspx* (z interfejsem użytkownika), *Default.aspx.vb* (z kodem źródłowym) oraz katalog *App_Data* (na razie pusty, zwykle używany do przechowywania plików *.mdb* lub innych plików z danymi).

Visual Studio nie tworzy już projektów dla aplikacji WWW, jednak nadal utrzymywane są pliki rozwiązania, dzięki czemu programista szybko może wrócić do implementowanej witryny WWW lub aplikacji. Pliki rozwiązania znajdują się w katalogu, który można wskazać w oknie dialogowym wywoływanym poleceniem *Tools/Options* widocznym na rysunku 8.6.

Pliki z kodem źródłowym

Przyjrzyjmy się bliżej plikom *.aspx* i z kodem źródłowym (ang. *code-behind files*), utworzonym przez Visual Studio. Najpierw trzeba zmienić nazwę pliku *Default.aspx* na *Welcome.aspx*. W tym celu wystarczy kliknąć nazwę pliku w oknie *Solution Explorer* i zmienić ją.



Rysunek 8.6. Ustawienia lokalizacji projektu

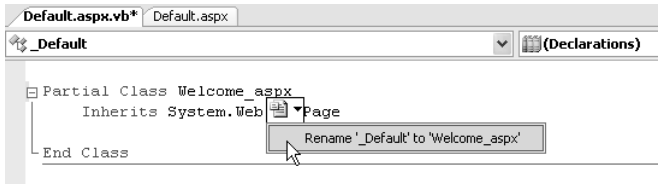
Uwaga dla programistów ASP.NET 1.1: model plików z kodem źródłowym ASP.NET uległ zmianie.

W wersji 1.x plik z kodem źródłowym definiował klasę potomną po klasie `Page`. Klasa ta zawierała zmienne reprezentujące wszystkie kontrolki znajdujące się na stronie oraz jawne wiązania zdarzeń tworzone przez delegaty i stronę *.aspx* potomną po klasie z pliku z kodem źródłowym.

W wersji 2.0 ASP.NET generuje pojedynczą klasę na podstawie strony *.aspx* oraz częściowej definicji klasy w pliku z kodem źródłowym.

ASP.NET może rozpoznawać kopie kontrolek i w trakcie kompilacji wywodzić wiązania zdarzeń. Dzięki temu nowy plik z kodem źródłowym zawiera tylko niezbędny kod aplikacji, w tym procedury obsługi zdarzeń, nie wymaga natomiast obecności zmiennych reprezentujących kontrolki ani jawnych wiązań zdarzeń. Nowe pliki z kodem źródłowym są prostsze, łatwiejsze w utrzymaniu i zawsze synchronizują się ze stroną *.aspx*.

W kolejnym kroku trzeba zmienić nazwę klasy, to znaczy kliknąć prawym przyciskiem myszy stronę *.aspx* i wybrać polecenie *View Code*. Jako nową nazwę klasy należy wpisać `Welcome_aspx`. Obok nazwy pojawi się krótka linia. Jej kliknięcie spowoduje otwarcie taga inteligentnego, w którym będzie można zmienić nazwę klasy we wszystkich miejscach, w których jest ona używana. Nazwę `Default_aspx` należy zmienić na `Welcome_aspx`, a resztę pracy wykona Visual Studio, zmieniając każde wystąpienie `Default_aspx` na nową nazwę. Sytuację tę przedstawiono na rysunku 8.7.



Rysunek 8.7. Zmiana nazwy klasy

Niestety, nazwa klasy nie zostanie zmieniona w dyrektywie strony w pliku *Welcome.aspx*. Należy zatem przejść do pliku *Welcome.aspx* i zmienić wartość atrybutu **Inherits** dyrektywy strony na `Welcome_aspx`.

```
<%@ Page Language="VB" AutoEventWireup="false" CodeFile="Default.aspx.vb"
Inherits="Welcome_aspx" %>
```

W widoku HTML strony *Welcome.aspx* można zauważyć, że w sekcji body strony przy użyciu standardowego znacznika języka HTML zdefiniowany został formularz:

```
<form id="Form1" runat="server">
```

ASP.NET przyjmuje założenie, że do zarządzania interakcją z użytkownikiem potrzebny jest co najmniej jeden formularz, dlatego tworzy go w momencie otwierania projektu. Atrybut `runat="server"` jest kluczem do magii przetwarzania na serwerze. Każdy znacznik, który posiada taki atrybut, jest traktowany jak kontrolka serwerowa, która musi być wykonana przez platformę .NET na serwerze. Wewnątrz formularza Visual Studio umieszcza znaczniki `div`, by ułatwić programiście umieszczanie kontrolek i tekstu.

Uruchomienie aplikacji

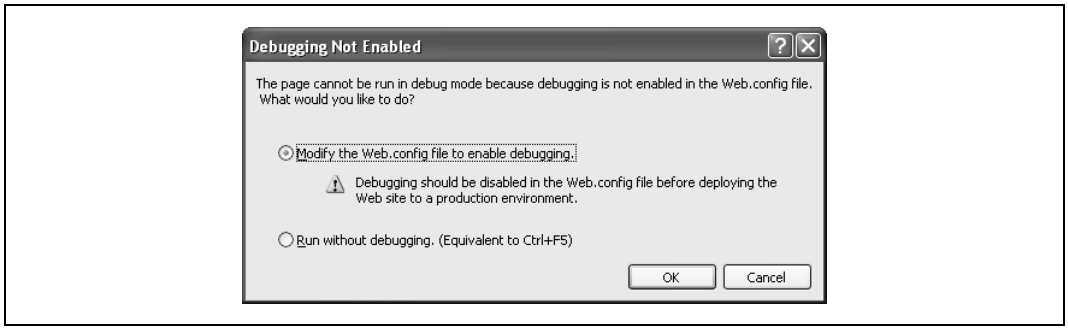
Po utworzeniu pustego formularza warto najpierw dodać na stronie jakiś tekst. Po przełączeniu się do widoku kodu źródłowego *Source* bezpośrednio do pliku można dodać skrypt oraz kod HTML (tak samo jak w klasycznym ASP). Dopisanie poniższego wiersza do sekcji `<body>` strony HTML spowoduje, że strona wyświetli pozdrowienie oraz aktualną datę i godzinę:

```
Witaj świecie! Teraz mamy <% =DateTime.Now.ToString()%>
```

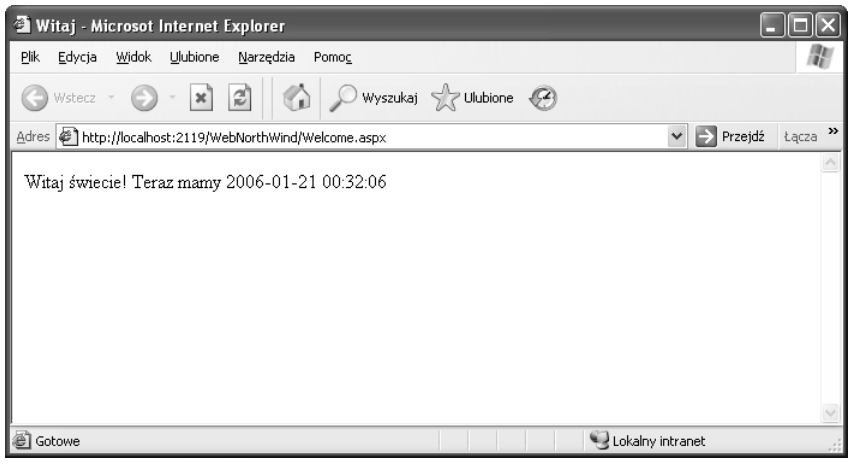
Znaczniki `<%` i `%>` wskazują, że między nimi znajduje się kod źródłowy (w tym przypadku kod języka Visual Basic 2005). Znak = zaraz za znacznikiem otwierającym powoduje, że ASP.NET wyświetli wartość wyrażenia, tak jakby wywołano `Response.Write`. Równie dobrze można by napisać:

```
Witaj świecie! Teraz mamy
<% Response.Write(DateTime.Now.ToString())%>
```

Stronę można uruchomić, naciskając *F5*. Visual Studio rozpozna, że nie zostało włączone debugowanie aplikacji i wyświetli okno dialogowe z propozycją włączenia debugowania. Okno dialogowe przedstawiono na rysunku 8.8. Po kliknięciu *OK* powinien ukazać się ciąg znaków wyświetlony w przeglądarce, widoczny na rysunku 8.9.



Rysunek 8.8. Włączenie debugowania



Rysunek 8.9. Strona ASP.NET Witaj świecie

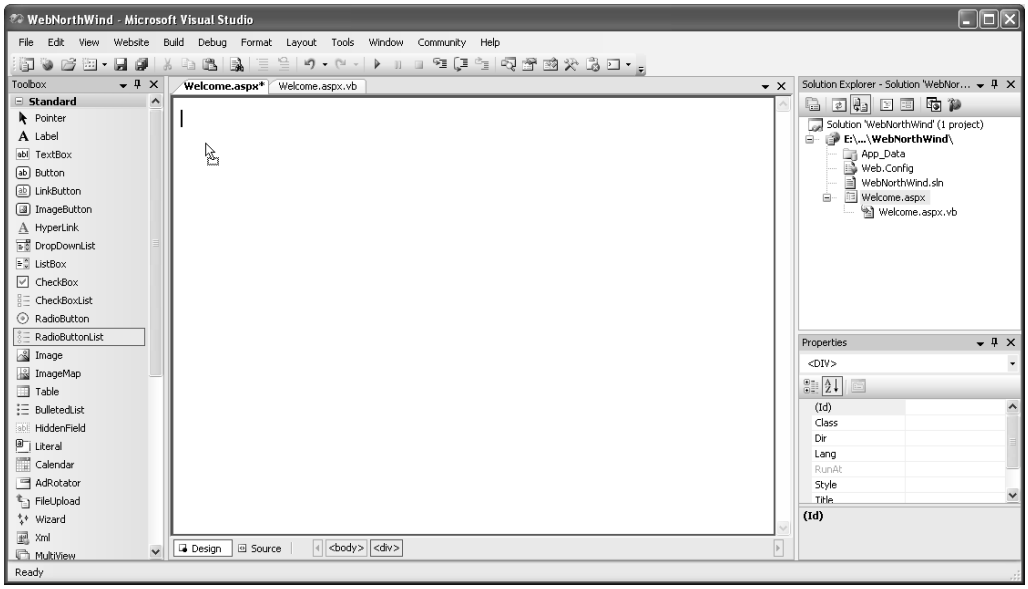
Dodawanie kontrolek

Zanim przejdziemy dalej, powinniśmy usunąć ze strony aspx wiersz informujący o aktualnej dacie i godzinie, aby rozpocząć tworzenie nowej strony od zera. Tak naprawdę była to ostatnia sytuacja, w której kod HTML współistniał z kodem języka Visual Basic 2005. Od teraz kontrolki będą dodawane do strony aspx, a kod do pliku z kodem źródłowym (z rozszerzeniem *.aspx.vb*).

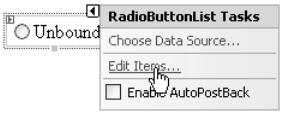
Kontrolki serwerowe można dodawać do formularza WWW na trzy sposoby: przeciągając kontrolki z okna narzędziowego *Toolbox* na stronę w trybie projektowania, wpisując odpowiedni kod HTML w widoku *Source* strony lub programistycznie, dodając kontrolki w fazie wykonania. Załóżmy na przykład, że użytkownikowi trzeba udostępnić przyciski opcji, dzięki którym będzie mógł wybrać jedną z trzech firm wysyłkowych dostępnych w bazie danych `Northwind`. W tym celu należy kliknąć widok projektowania *Design* i przeciągnąć na formularz kontrolkę `RadioButtonList`, jak na rysunku 8.10.

Gdy kontrolka `RadioButtonList` znajdzie się już na formularzu, należy kliknąć jej tag inteligentny i wybrać polecenie *Edit Items*, by dodać odpowiednie pozycje do listy. Ilustruje to rysunek 8.11.

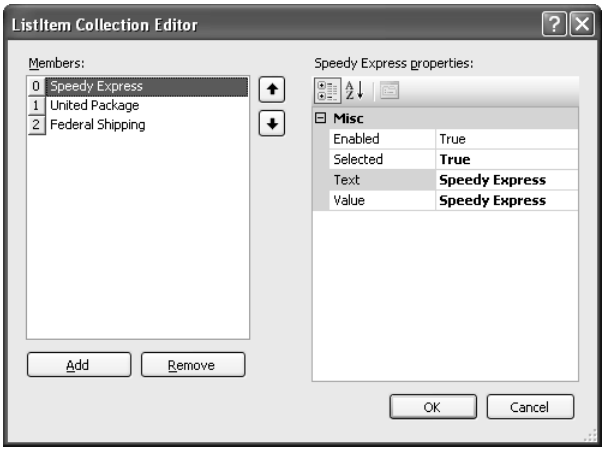
Otwarte zostanie okno dialogowe *ListItem Collection Editor*. W oknie należy kliknąć przycisk *Add*, aby dodać element *ListItem*. Właściwości `Text` należy przypisać nazwę firmy wysyłkowej (na przykład „Speedy Express"), a właściwości `Selected` pierwszej pozycji na liście trzeba dodatkowo przypisać wartość **`True`** (aby zaznaczyć pierwszy przycisk wyboru znajdujący się na liście). Następnie trzeba dodać kolejne dwie pozycje, „United Package" i „Federal Shipping", jak na rysunku 8.12.



Rysunek 8.10. Przeciąganie kontrolki RadioButtonList na formularz

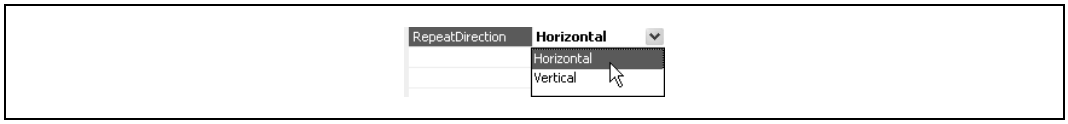


Rysunek 8.11. Edycja pozycji na liście RadioButtonList



Rysunek 8.12. Edytor list ListItem Collection Editor

Po kliknięciu przycisku *OK* w grupie pojawią się trzy przyciski. Teraz trzeba kliknąć grupę i przejść do okna właściwości *Properties*. Jako nazwę grupy należy wpisać `rblShipper` (ustawiając jej właściwość `ID`) i przejrzeć pozostałe właściwości dostępne dla grupy. Warto zwrócić szczególną uwagę na właściwość `RepeatDirection` (widoczną na rysunku 8.13), dzięki której można ustawiać przyciski w poziomie albo w pionie.



Rysunek 8.13. Właściwość RepeatDirection

Można teraz przejść do trybu *Source* i przeanalizować kod HTML, który został wygenerowany przez edytor trybu *Design*:

```
<asp:RadioButtonList ID="rblShipper" runat="server">
    <asp:ListItem Selected="True">Speedy Express</asp:ListItem>
    <asp:ListItem>United Package</asp:ListItem>
    <asp:ListItem>Federal Shipping</asp:ListItem>
</asp:RadioButtonList> </div>
```

Oczywiście taki sam kod można było wpisać samodzielnie, lecz użycie edytora jest znacznie prostsze i chroni przed ewentualnymi błędami literowymi. Chętni czytelnicy mogą ręcznie wpisać dodatkowe kontrolki `ListItem`, a wprowadzone zmiany zostaną uwzględnione w trybie projektowania.

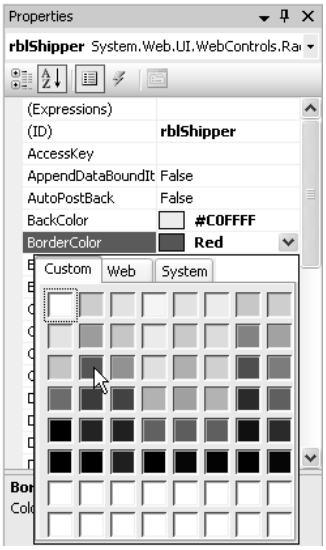
Kontrolki można dodawać do strony w jednym z dwóch trybów. Trybem domyślnym jest `FlowLayout`. W trybie tym kontrolki są dodawane do strony od góry do dołu, jak w standardowym dokumencie HTML. Alternatywą dla trybu `FlowLayout` jest tryb `GridLayout`, w którym kontrolki są układane w przeglądarce przy użyciu pozycjonowania bezwzględnego (to znaczy na podstawie współrzędnych x i y).

Aby zmienić tryb `FlowLayout` na `GridLayout` i odwrotnie, należy w Visual Studio .NET zmienić wartość właściwości `PageLayout` dokumentu. W niniejszej książce zawsze będziemy używali trybu `FlowLayout`, ponieważ w trybie `GridLayout` informacje dotyczące pozycji kontrolek rozszerzają kod strony i znacznie ograniczają jego czytelność.

Należy powrócić do trybu *Design* i kliknąć kontrolkę `RadioButtonList`. W oknie *Properties* właściwość `BackColor` należy ustawić na jasnoniebieski, a `BorderColor` na czerwony, jak na rysunku 8.14.

Odpowiedni kolor można wpisać bezpośrednio w polu właściwości jako wartość szesnastkową albo wybrać go z palety kolorów. W polu właściwości `BorderColor` można nawet wpisać słowo **`Red`**, dzięki czemu wybrany zostanie standardowy czerwony kolor. (Aby ujrzeć obramowanie, należy zmienić wartość właściwości `BorderStyle` z domyślnej wartości pustej na przykład na wartość `Solid`). Można przejść teraz do trybu *Source* — będzie można zauważyć, że kod HTML został odpowiednio rozszerzony:

```
<asp:RadioButtonList
ID="rblShipper"
runat="server"
BackColor="#COFFFF"
BorderColor="Red"
BorderStyle="Solid">
```



Rysunek 8.14. Ustawienia kontrolki przycisków opcji

Kontrolki serwerowe

Dla formularzy WWW dostępne są dwa typy kontrolek serwerowych. Pierwszy typ to serwerowe kontrolki HTML. Są to kontrolki HTML oznaczane atrybutem `runat="Server"`.

Drugim sposobem oznaczenia kontrolek HTML jako kontrolek serwerowych jest użycie kontrolek WWW ASP.NET, nazywanych również kontrolkami ASP. Kontrolki WWW zaprojektowano po to, by rozszerzyć i zastąpić standardowe kontrolki HTML. Kontrolki WWW posiadają bardziej spójny model obiektowy oraz nazewnictwo atrybutów. Na przykład, w przypadku kontrolek HTML istnieje wiele różnych sposobów obsługi danych wejściowych:

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

Każda z powyższych kontrolek zachowuje się w inny sposób i wymaga podania różnych atrybutów. W kontrolkach WWW podjęto próbę znormalizowania zbioru kontrolek, używając spójnego nazewnictwa atrybutów w całym modelu obiektowym kontrolek. Powyższym kontrolkom HTML odpowiadają następujące serwerowe kontrolki WWW:

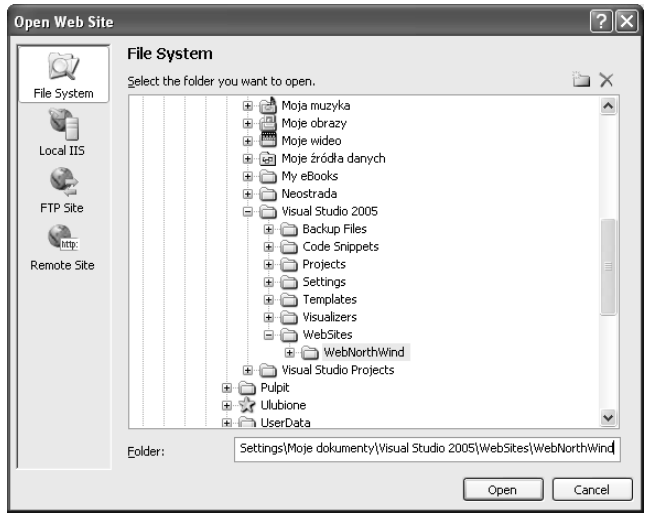
```
<asp:RadioButton>
<asp:CheckBox>
<asp:Button>
<asp:TextBox rows="1">
<asp:TextBox rows="5">
```

W pozostałej części książki używane będą kontrolki WWW.

Dodawanie kontrolek i zdarzeń

Zanim przejdziemy dalej, utworzymy nową aplikację, w której wykorzystamy zebrane dotychczas informacje. W tym celu należy utworzyć nową aplikację WWW i nadać jej nazwę `NorthWindASP`.

Aby skopiować pliki z już istniejącej witryny WWW do nowej witryny, należy wybrać polecenie *WebSite/Copy WebSite*. Otwarta zostanie strona *Copy Web Site*. W jej lewym górnym rogu znajduje się lista rozwijana *Connect to:*, a obok niej niebieski przycisk. Przycisk należy kliknąć, aby wyświetlić okno dialogowe *Open Web Site* widoczne na rysunku 8.15.



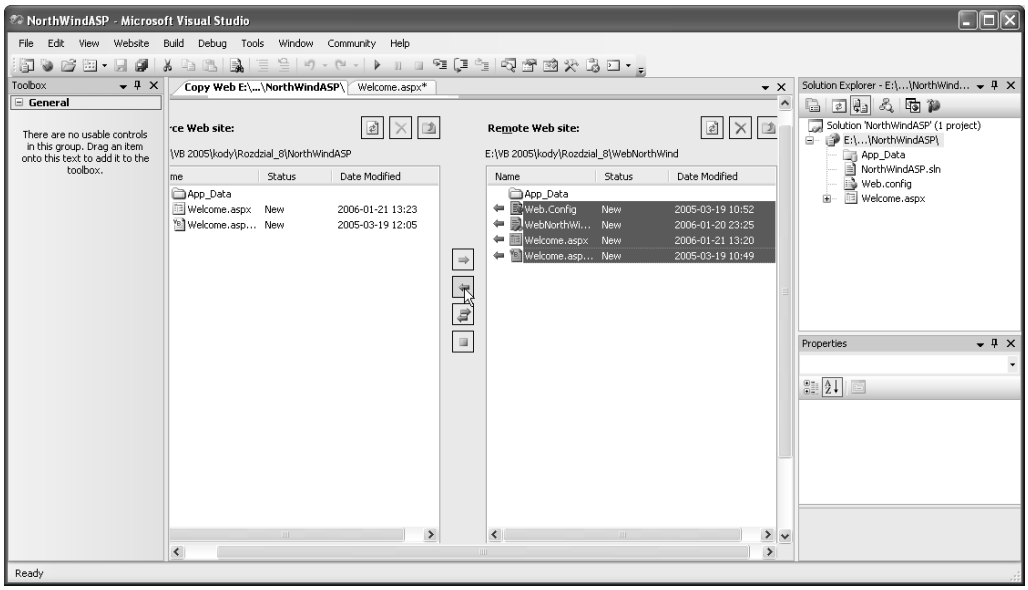
Rysunek 8.15. Otwarcie witryny WWW, która ma zostać skopiowana

W oknie dialogowym należy wskazać witrynę, która ma zostać skopiowana, i kliknąć przycisk *Open*. Dzięki temu okno dialogowe będzie już gotowe do transferu plików. Należy zaznaczyć wszystkie pliki zdalnej witryny WWW i kliknąć strzałkę transferu, jak na rysunku 8.16.

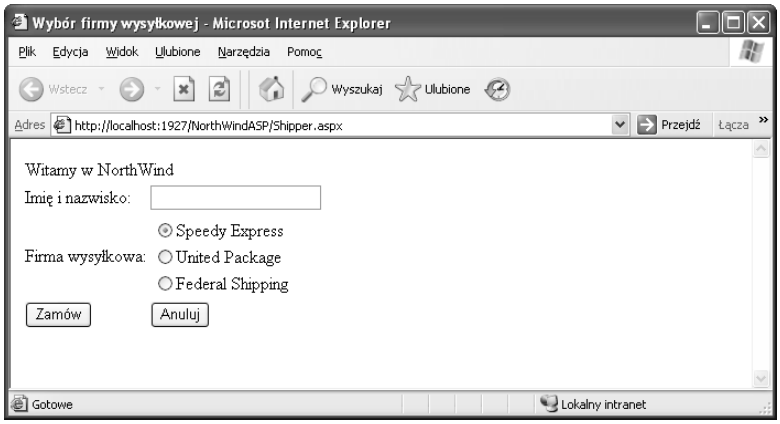
Po wykonaniu operacji można zamknąć stronę *Copy Web* (duży przycisk *X* w prawym górnym rogu). Stronę WWW *Default.aspx* można usunąć (klikając ją prawym przyciskiem myszy w oknie *Solution Explorer* i wybierając polecenie *Delete*) oraz ustawić stronę *Welcome.aspx* jako stronę startową aplikacji (w tym celu trzeba kliknąć stronę *Welcome.aspx* w oknie *Solution Explorer* i wybrać polecenie *Select As Start Page*). W ten sposób nowa witryna WWW będzie stanowić duplikat witryny dotychczasowej. Można ją uruchomić, by upewnić się, że wszystko działa poprawnie.

Dodawanie strony wysyłkowej

Przez dodanie tylko kilku dodatkowych kontrolek można utworzyć kompletny formularz, z którego będą mogli korzystać użytkownicy. W tym celu dodamy bardziej odpowiednie powitanie („Witamy w NorthWind"), pole tekstowe do wpisywania nazwiska użytkownika, dwa nowe przyciski (*Zamów* i *Anuluj*) oraz tekst stanowiący informację dla użytkownika. Ukończony formularz przedstawiono na rysunku 8.17.



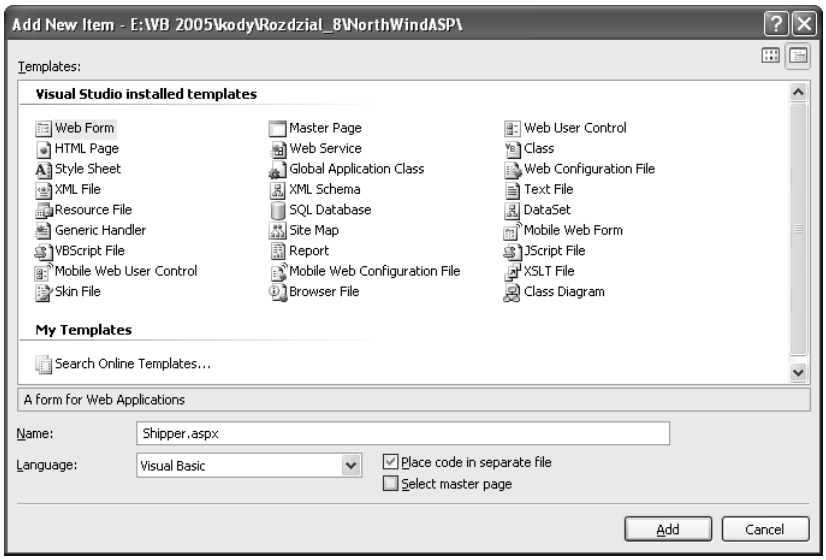
Rysunek 8.16. Kopiowanie wszystkich plików docelowej witryny WWW do nowej witryny WWW



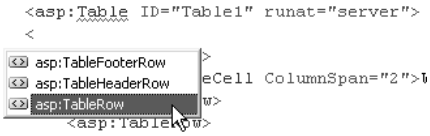
Rysunek 8.17. Ukończony formularz wysyłkowy

Aplikację należy kliknąć prawym przyciskiem myszy i wybrać polecenie *Add New Item*. W oknie dialogowym *Add New Item* trzeba kliknąć formularz WWW i nadać nowemu formularzowi nazwę *Shipper.aspx*, jak na rysunku 8.18.

Nowe kontrolki najlepiej jest rozmieścić w tabeli, a najłatwiej można tego dokonać, przeciągając tabelę z okna narzędziowego *Toolbox* do widoku kodu źródłowego *Source* (wewnątrz znaczników `<div>` w formularzu). Gdy tabela będzie już na swoim miejscu, bez trudu można do niej dodawać wiersze, wpisując w niej otwierający nawias kątowy. Wówczas uruchomiony zostanie mechanizm IntelliSense, który pomoże utworzyć znacznik `ASP:TableRow` (wraz ze znacznikiem zamykającym). Sytuację tę prezentuje rysunek 8.19.

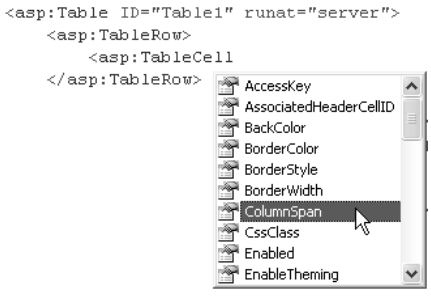


Rysunek 8.18. Dodanie nowej strony WWW



Rysunek 8.19. Dodawanie nowego wiersza tabeli

W pierwszym wierszu należy wstawić komórkę tabeli. Również to zadanie znacznie ułatwi IntelliSense. Po dodaniu komórki należy nacisnąć klawisz spacji — wówczas wyświetlone zostaną wszystkie atrybuty komórki. Atrybutowi ColumnSpan komórki należy przypisać wartość `2`, jak na rysunku 8.20.

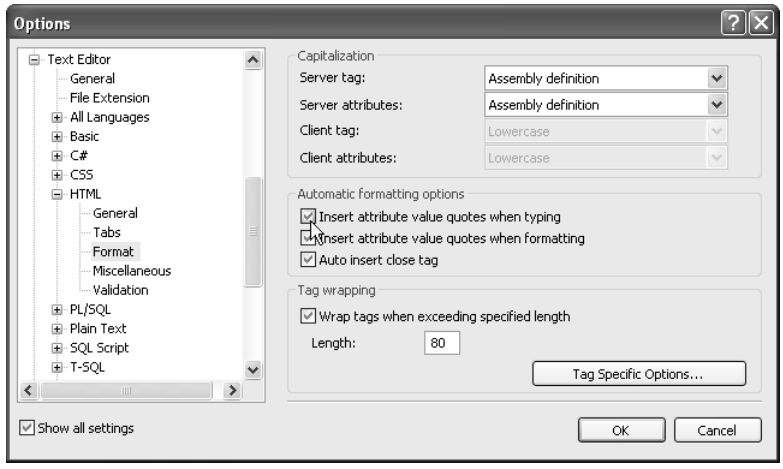


Rysunek 8.20. Definiowanie atrybutu ColumnSpan

W deklaracji komórki `TableCell` można wpisać następujący tekst powitania:

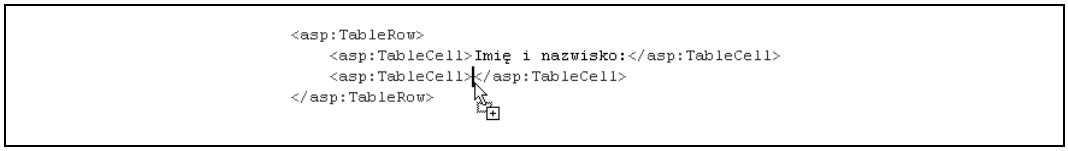
```
<asp:TableCell ColumnSpan="2">Witamy w NorthWind</asp:TableCell>
```

HTML zgodny z XHTML wymaga, by wartości atrybutów znajdowały się w cudzysłowach. Visual Studio 2005 może pomóc w spełnieniu tego wymogu. Należy wybrać w menu polecenie *Tools/Options*, otworzyć sekcję *Text Editor*, a następnie sekcję HTML. W sekcji HTML trzeba wybrać pozycję Format i w grupie *Automatic formatting options* zaznaczyć pole wyboru *Insert attribute value quotes when typing*, jak na rysunku 8.21.



Rysunek 8.21. Ustawianie automatycznego formatowania kodu HTML

W kolejnym kroku należy dodać drugi wiersz z dwiema kolumnami. W pierwszej kolumnie będzie się znajdować tekst, a do drugiej kolumny z okna *Toolbox* trzeba przeciągnąć pole tekstowe, jak na rysunku 8.22.

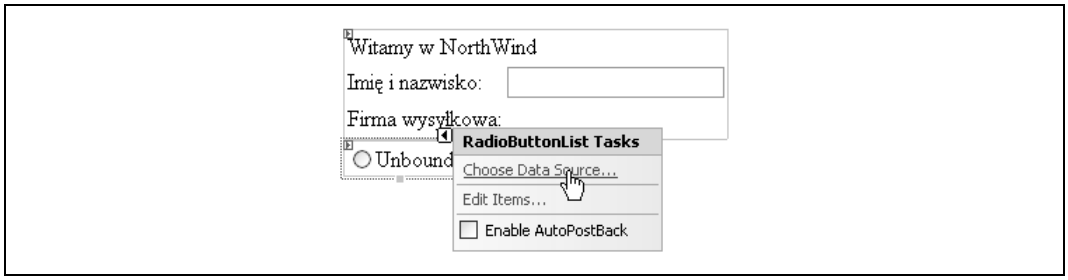


Rysunek 8.22. Przeciąganie kontrolki TextBox do komórki tabeli

W dalszej kolejności trzeba utworzyć trzeci wiersz z listą `RadioButtonList`, którą wcześniej utworzyliśmy na stronie powitalnej. Najpierw należy wstawić wiersz z dwiema kolumnami (w pierwszej kolumnie znajdować się będzie tekst *Firma wysyłkowa*).

Druga kolumna jest przeznaczona na nową kontrolkę `RadioButtonList`, którą najpierw utworzymy, a dopiero po jej właściwym skonfigurowaniu umieścimy w kolumnie. Należy przejść do widoku *Design* i przeciągnąć kontrolkę `RadioButtonList` na formularz, poniżej tabeli. Warto zauważyć, że kontrolka zostanie oznaczona jako `Unbound`. Zamiast wypełniania jej pozycjami o wartościach wpisanych jawnie (jak miało to miejsce poprzednio), lista zostanie powiązana z danymi pochodzącymi z bazy danych.

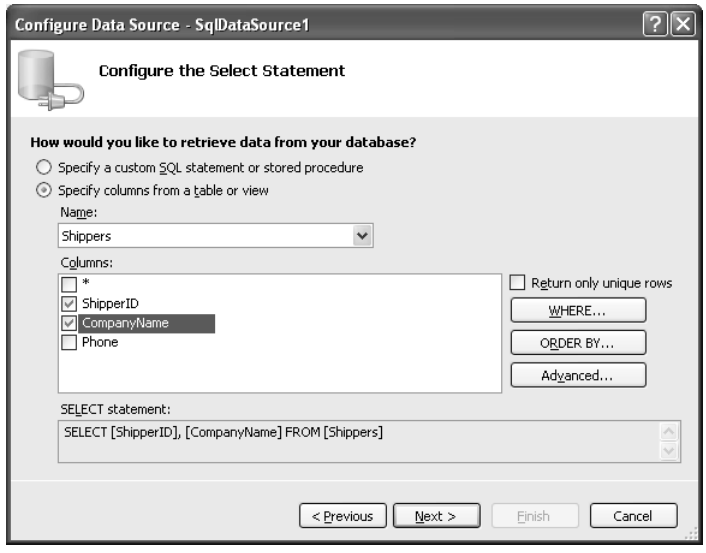
W tym celu należy kliknąć tag inteligentny kontrolki i wybrać polecenie *Choose Data Source*, jak na rysunku 8.23.



Rysunek 8.23. Wybór źródła danych dla kontrolki RadioButtonList

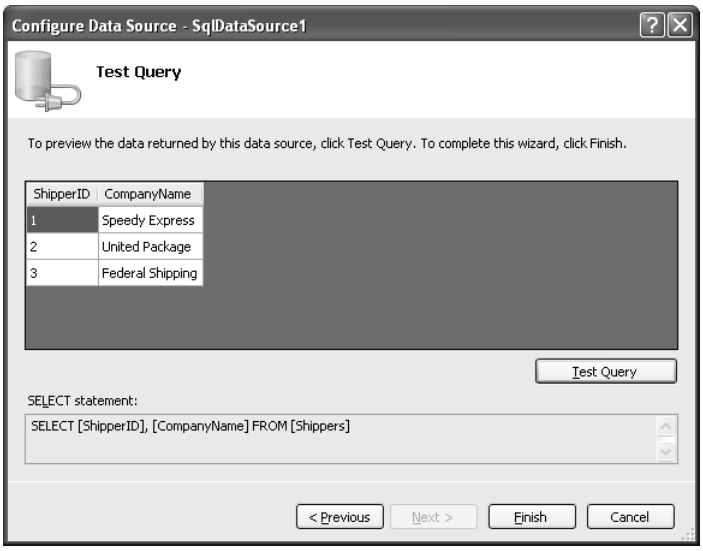
W oknie *Data Source Configuration Wizard* należy wybrać pozycję <*New Data Source…*>. W odpowiedzi kreator wyświetli listę różnych źródeł danych, spośród których można dokonać wyboru. Należy wybrać *SQL Database*, a kreator zaproponuje nazwę nowego źródła `SqlDataSource1`. Należy kliknąć przycisk *OK*. Kreator poprosi o wybranie istniejącego połączenia lub utworzenie nowego połączenia. Należy utworzyć nowe połączenie do bazy danych `Northwind` jak w poprzednich rozdziałach, po czym kliknąć przycisk *Next*. Połączenie należy zapisać pod nazwą `NorthWindConnectionString` i kliknąć przycisk *Next*.

Kolejny krok kreatora polega na wybraniu pól, które mają zostać odczytane ze wskazanej tabeli. Wybierzemy pola `ShipperID` i `CompanyName` tabeli `Shippers`, jak na rysunku 8.24.



Rysunek 8.24. Definiowanie instrukcji SELECT

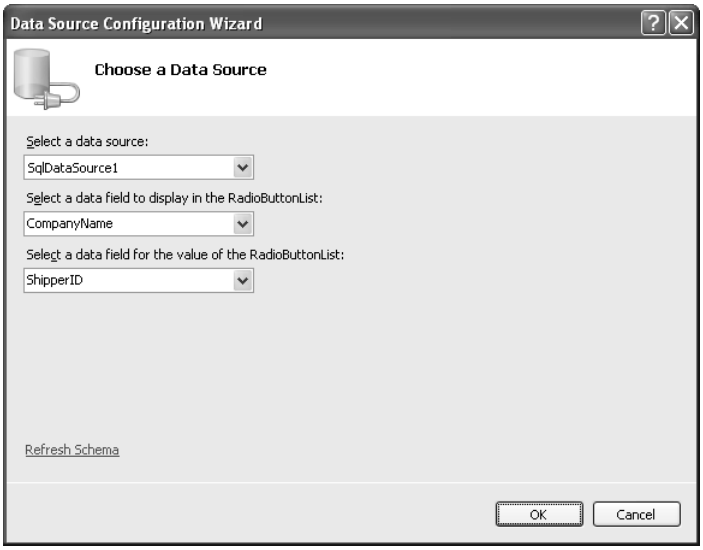
Po kliknięciu przycisku *Next* będzie można sprawdzić działanie zapytania, jak na rysunku 8.25.



Rysunek 8.25. Sprawdzanie działania zapytania

Zwróćmy uwagę, że pole `CompanyName` zawiera interesujące nas dane, które mają być wyświetlane. Wartości `ShipperID` powinny znajdować się w polu `Value` pozycji listy, by móc jednoznacznie zidentyfikować wybraną firmę wysyłkową.

Należy kliknąć przycisk *Finish*, co spowoduje powrót do kreatora *Data Source Configuration Wizard*. Teraz można wskazać pole, które ma być wyświetlane, oraz drugie pole, którego wartości mają być wartościami pozycji na liście `RadioButtonList`. Widać to na rysunku 8.26.



Rysunek 8.26. Wskazanie pola do wyświetlenia oraz pola stanowiącego wartość pozycji na liście

Gdy kontrolka `RadioButtonList` i powiązane z nią źródło danych `SqlDataSource` zostaną skonfigurowane, należy powrócić do widoku kodu źródłowego i przesunąć obydwa elementy do przeznaczonej do tego komórki tabeli.

```
<asp:TableRow>
    <asp:TableCell>Firma wysyłkowa:</asp:TableCell>
    <asp:TableCell>
        <asp:RadioButtonList
        DataSourceID="SqlDataSource1"
        DataTextField="CompanyName"
        DataValueField="ShipperID"
        ID="RadioButtonList1" runat="server"/>

        <asp:SqlDataSource
        ConnectionString="<%$ ConnectionStrings:NorthwindConnectionString %>"
        ID="SqlDataSource1" runat="server"
        SelectCommand="SELECT [ShipperID], [CompanyName] FROM [Shippers]"/>
    </asp:TableCell>
</asp:TableRow>
```

Visual Studio 2005 nie utworzy samozamykających się znaczników kontrolki. Znaczniki te dodano ręcznie, by zaoszczędzić miejsce i zwiększyć czytelność kodu.

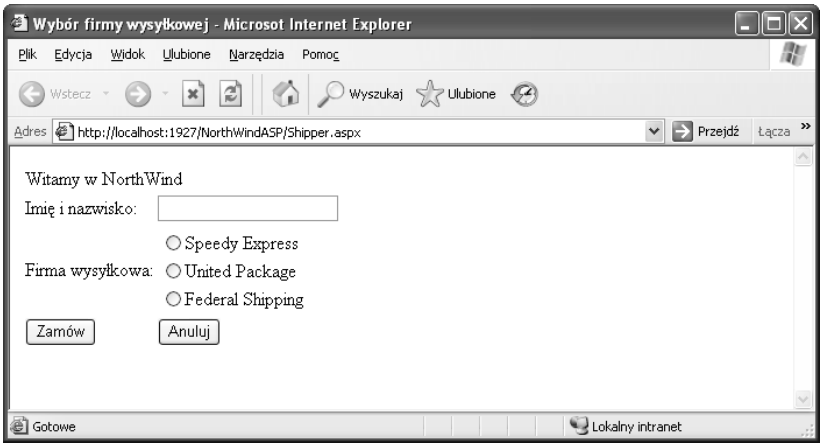
Na koniec należy dodać jeszcze dwa wiersze: jeden z dwoma przyciskami i drugi z etykietą, która nie będzie posiadać tekstu:

```
<asp:TableRow>
    <asp:TableCell>
        <asp:Button ID="btnOrder" runat="server" Text="Zamów"/>
    </asp:TableCell>
    <asp:TableCell>
        <asp:Button ID="btnCancel" runat="server" Text="Anuluj"/>
    </asp:TableCell>
</asp:TableRow>
<asp:TableRow>
    <asp:TableCell ColumnSpan="2">
        <asp:Label ID="lblMsg" runat="server"></asp:Label>
    </asp:TableCell>
</asp:TableRow>
```

Stronę *Shipper.aspx* należy ustawić jako stronę startową aplikacji i uruchomić aplikację. Powinna ona wyglądać podobnie jak na rysunku 8.27.

Nasz formularz nie wygra oczywiście głównej nagrody w konkursie na najbardziej atrakcyjną witrynę, lecz ilustruje szereg kluczowych mechanizmów formularzy WWW. Gdy użytkownik kliknie przycisk *Zamów*, aplikacja sprawdzi, czy użytkownik podał swoje imię i nazwisko, i czy wybrał firmę wysyłkową. Warto zwrócić uwagę, że w trakcie projektowania nie będą znane nazwy firm wysyłkowych (zostaną one dopiero odczytane z bazy danych), dlatego trzeba będzie sprawdzić nazwę i identyfikator wybranej pozycji na liście `RadioButtonList`.

Aby wypełnić to zadanie, trzeba przejść do trybu *Design* i dwukrotnie kliknąć przycisk *Zamów*. Visual Studio wyświetli stronę z kodem źródłowym i utworzy procedurę obsługi zdarzenia `Click` przycisku.



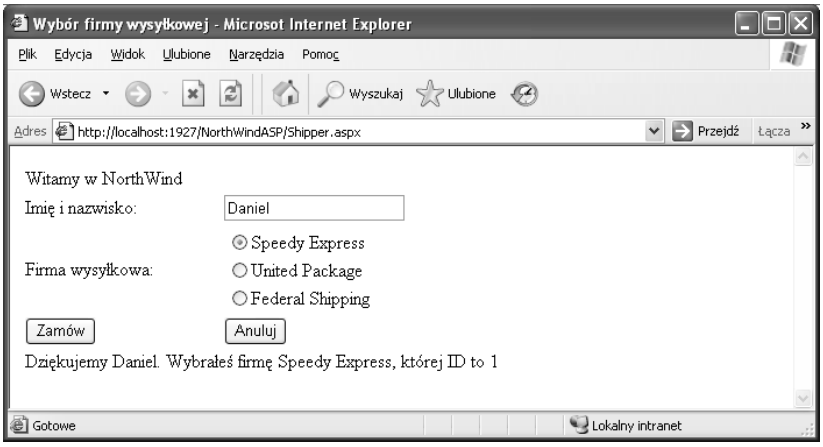
Rysunek 8.27. Ukończony formularz wysyłkowy

Procedura obsługi zdarzenia powinna przypisać etykiecie tekst złożony z wartości pola tekstowego oraz tekstu i wartości pozycji wybranej na liście `RadioButtonList`. Odpowiedni kod znajduje się na listingu 8.1.

Listing 8.1. Procedura obsługi zdarzenia Click przycisku Zamów

```
Protected Sub btnOrder_Click( _
ByVal sender As Object, _
ByVal e As System.EventArgs) Handles btnOrder.Click
   lblMsg.Text = "Dziękujemy " + TextBox1.Text.Trim() + ".  Wybrałeś firmę " + _
      RadioButtonList1.SelectedItem.Text.ToString() + " której ID to " + _
      RadioButtonList1.SelectedValue.ToString()
End Sub
```

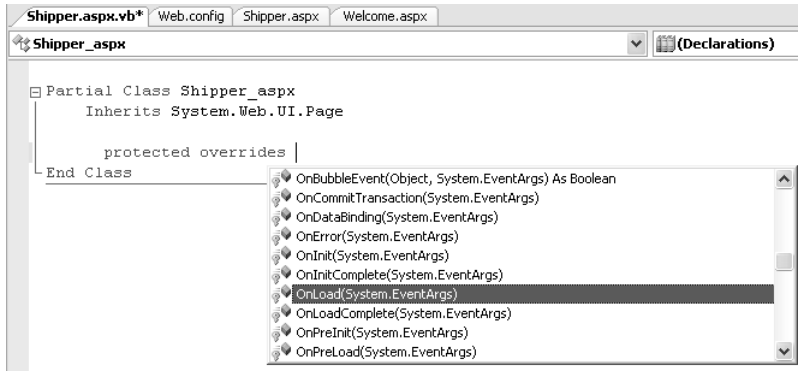
Po uruchomieniu programu można kliknąć jeden z przycisków opcji, wypełnić pole tekstowe i kliknąć przycisk Zamów. Etykieta zostanie wypełniona tekstem, jak na rysunku 8.28.



Rysunek 8.28. Test strony wysyłkowej

Należy zatrzymać program i ponownie go uruchomić. Da się wtedy zauważyć, że żaden przycisk opcji nie jest zaznaczony. W czasie dowiązywania listy nie została wskazana pozycja domyślna. Problem można rozwiązać na kilka sposobów, z których najprostszy polega na pokryciu procedury obsługi zdarzenia `OnLoad` i wybraniu w nim pierwszego przycisku opcji.

Trzeba zatem wrócić do strony *Shipper.aspx.vb* i umieścić kursor wewnątrz klasy, ale nie wewnątrz istniejącej procedury **Sub**. Następnie trzeba wpisać **Protected Overrides**. Na ekranie pojawi się przewijana lista wszystkich metod, właściwości itd., które można pokryć, co widać na rysunku 8.29.



Rysunek 8.29. Pokrywanie zdarzenia OnLoad

Warto zwrócić uwagę, że w kodzie nie zastosowano właściwej wielkości liter. Jednak po wybraniu metody Visual Studio 2005 sam poprawi wielkość liter, to znaczy wstawi literę `P` na początku słowa `protected` i `O` na początku słowa `overrides`.

Procedurę `OnLoad` można wskazać na liście albo zacząć wpisywać nazwę **OnLoad**, a po jej wyróżnieniu na liście nacisnąć klawisz tabulacji. W efekcie utworzony zostanie szkielet metody, zawierający jeden wiersz kodu. Należy dopisać drugi wiersz kodu, tak by pełen kod procedury **Sub** wyglądał następująco:

```
Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)
   MyBase.OnLoad(e)
   RadioButtonList1.SelectedIndex = 0
End Sub
```

W pierwszym wierszu wywoływana jest metoda `OnLoad()` klasy bazowej (`System.Web.UI.Page`), aby wykonać wszystkie operacje niezbędne do załadowania strony. Następnie wykonywany jest dodatkowy wiersz kodu, w którym wybierana jest pierwsza pozycja na liście `RadioButtonList`.

W tym rozwiązaniu istnieje pewien drobny problem. Gdy aplikacja zostanie uruchomiona, zaznaczony będzie pierwszy przycisk opcji. Kiedy jednak użytkownik wybierze przycisk drugi albo trzeci i naciśnie przycisk *Zamów*, ponownie zaznaczony będzie przycisk pierwszy. Mogłoby to sugerować, że użytkownik zawsze wybiera pierwszą opcję. Przyczyną jest to, że za każdym razem gdy strona jest ładowana wywoływane jest zdarzenie `OnLoad`, a procedura obsługi tego zdarzenia resetuje indeks pozycji wybranej na liście.

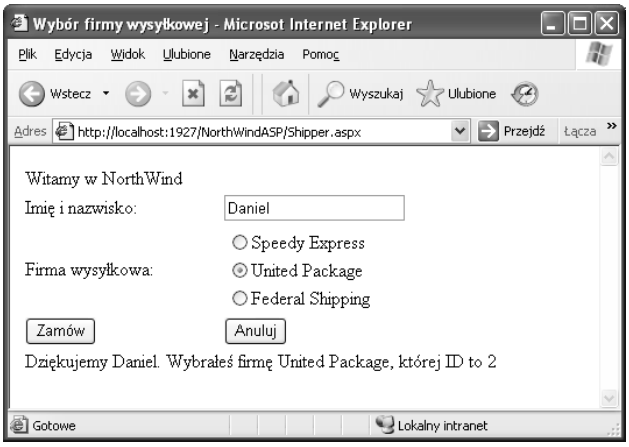
Pierwszy przycisk powinien być wybierany tylko wówczas, gdy strona jest ładowana po raz pierwszy, a nie za każdym razem gdy strona jest przesyłana z powrotem na serwer w wyniku kliknięcia przycisku *Zamów*.

Aby rozwiązać problem, wiersz ustawiający pole opcji należy umieścić w instrukcji **If** sprawdzającej, czy strona została wysłana na serwer.

```
If IsPostBack = False Then RadioButtonList1.SelectedIndex = 0
```

Jeśli cała instrukcja **If** zostanie umieszczona w jednym wierszu, nie trzeba jej kończyć instrukcją **End If**.

Po uruchomieniu strony sprawdzona zostanie wartość właściwości `IsPostBack`. Przy pierwszym wyświetleniu strony właściwość będzie mieć wartość **False** i zaznaczony zostanie pierwszy przycisk opcji. Gdy użytkownik kliknie przycisk opcji, a następnie przycisk Zamów, strona zostanie przesłana na serwer w celu jej przetworzenia (na serwerze wykonana zostanie procedura `btnOrder_Click`), po czym serwer odeśle ją do użytkownika. Tym razem właściwość `IsPostBack` będzie już mieć wartość **True**, a więc kod wewnątrz instrukcji **If** nie zostanie wykonany i zachowany zostanie wcześniejszy wybór dokonany przez użytkownika. Widać to na rysunku 8.30.



Rysunek 8.30. Po odesłaniu strony przez serwer wybór użytkownika zostanie zachowany

Zmieniona strona będzie pamiętać, który przycisk został kliknięty pomimo tego, że sama sieć WWW jest bezstanowa. W tym przypadku odpowiednie informacje są utrzymywane dzięki stanowi widoku, o którym więcej w następnym punkcie.

Stan

Stan to bieżąca wartość wszystkich kontrolek i zmiennych dla aktualnego użytkownika w bieżącej sesji. Sieć WWW jest ze swej natury środowiskiem *bezstanowym*, co oznacza, że za każdym razem, gdy strona jest przesyłana na serwer i odsyłana z powrotem do przeglądarki, jest ona tworzona od nowa. Jeśli stan wszystkich kontrolek nie zostanie jawnie zachowany

przed wysłaniem strony na serwer, zostanie utracony i wszystkie kontrolki zostaną odtworzone z wartościami domyślnymi. Jedną z istotnych zalet ASP.NET jest to, że automatycznie utrzymuje on stan wszystkich kontrolek serwerowych — zarówno HTML, jak i ASP. W niniejszym punkcie opisany zostanie mechanizm realizujący tę funkcję, a także sposób, w jaki programiści mogą wykorzystać obecny w ASP.NET mechanizm zarządzania stanami.

ASP.NET zarządza trzema typami stanów. Są to:

- Stan widoku (przechowywany w zasobniku stanów);
- Stan aplikacji;
- Stan sesji.

Tabela 8.2 zawiera porównanie wszystkich trzech typów stanów.

Tabela 8.2. Typy stanów

Zmienne	Stan widoku	Stan aplikacji	Stan sesji
Używają zasobów serwera	Nie	Tak	Tak
Używają przepustowości	Tak	Nie	Zależy
Mają czas ważności	Nie	Nie	Tak
Wpływają na bezpieczeństwo	Tak	Nie	Zależy
Zoptymalizowane dla typów, które nie są prymitywne	Nie	Tak	Tak
Dostępne dla dowolnych danych	Tak	Tak	Tak
Dostępne programistycznie	Tak	Tak	Tak
Zasięg	Strona	Aplikacja	Sesja
Są zachowywane po restarcie	Tak	Nie	Zależy

W kolejnych punktach zostaną opisane poszczególne typy stanów.

Stan widoku

Stan widoku to stan strony i wszystkich jej kontrolek. Stan widoku jest automatycznie utrzymywany między poszczególnymi żądaniami platformy ASP.NET. Gdy strona jest wysyłana do serwera, następuje odczyt stanu widoku, natomiast tuż przed odesłaniem strony do przeglądarki stan widoku jest odtwarzany.

Stan widoku jest przechowywany w zasobniku stanów (opisanym w następnym punkcie) przy użyciu ukrytych pól strony, które zawierają stan zakodowany w zmiennych będących ciągami znaków. Stan widoku jest utrzymywany w standardowych polach HTML formularza, dzięki czemu działa we wszystkich przeglądarkach.

Jeśli utrzymywanie stanu widoku strony nie jest konieczne, można zwiększyć wydajność, wyłączając stan widoku dla strony. Na przykład, jeśli strona nie odwołuje się do samej siebie albo jeżeli jedyna kontrolka, która wymaga utrzymania swego stanu, jest wypełniana danymi z bazy danych odczytywanymi każdorazowo w trakcie komunikacji z serwerem, utrzymywanie stanu widoku dla strony nie jest konieczne. Aby wyłączyć stan widoku dla strony, należy do dyrektywy `Page` dodać atrybut `EnableViewState` o wartości **`False`**:

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

Domyślnie atrybut `EnableViewState` ma wartość **`True`**.

Stan widoku można wyłączyć w całej aplikacji. Wystarczy w tym celu przypisać właściwości `EnableViewState` wartość **`False`** w sekcji `<pages>` pliku konfiguracyjnego *machine.config* lub *Web.config*.

Stan widoku można utrzymywać albo wyłączać również dla pojedynczych kontrolek. Służy do tego właściwość `Control.EnableViewState`, która przechowuje wartość typu **`Boolean`** i domyślnie ma wartość **`True`**. Tak samo jak w przypadku strony, wyłączenie stanu widoku dla pojedynczej kontrolki nieco zwiększy wydajność. Wyłączenie stanu widoku dla kontrolki jest zasadne na przykład wówczas, gdy kontrolka jest wypełniania danymi z bazy danych każdorazowo w trakcie ładowania strony. W takiej sytuacji zawartość kontrolki jest nadpisywana przez dane zwrócone przez zapytanie do bazy danych, zatem nie ma potrzeby, by utrzymywać stan takiej kontrolki, zwłaszcza jeśli zawiera ona znaczną ilość danych.

Istnieją przypadki, w których stan widoku nie jest najlepszym mechanizmem przechowywania danych. Jeśli trzeba przechowywać znaczną ilość danych, stan widoku nie będzie rozwiązaniem najbardziej wydajnym, ponieważ wraz z każdym przesłaniem strony do serwera i z powrotem przesyłany będzie również cały zbiór danych. Jeśli ze względu na charakter danych trzeba szczególnie zatroszczyć się o ich bezpieczeństwo i nie są one wyświetlane na stronie, wówczas przechowywanie ich w widoku stanu tylko zwiększa niebezpieczeństwo odczytania danych. Ponadto stan widoku jest zoptymalizowany wyłącznie dla ciągów znaków, liczb całkowitych, wartości logicznych, tablic, list i słowników. Inne typy .NET można serializować i utrzymywać w stanie widoku, lecz spowoduje to obniżenie wydajności i zwiększy przestrzeń zajmowaną przez stan widoku.

W niektórych wymienionych przypadkach lepszym rozwiązaniem może być wykorzystanie stanu sesji. Z drugiej strony, stan widoku w ogóle nie zużywa zasobów serwera i nie wygasa czas jego ważności, jak ma to miejsce w przypadku stanu sesji.

Zasobnik stanów

Jeśli istnieją wartości, które nie są powiązane z żadną kontrolką, a trzeba je przechowywać między kolejnymi kontaktami z serwerem, wówczas można takie wartości przechowywać w zasobniku stanów strony. *Zasobnik stanów* (ang. *state bag*) to struktura danych zawierająca pary atrybut-wartość, przechowywane w postaci ciągów znaków powiązanych z obiektami. Poprawne obiekty mają typy prymitywne, to znaczy typy **`Integer`**, **`Byte`**, **`String`**, **`Boolean`** i tak dalej. Zasobnik stanów jest implementowany przy użyciu klasy `StateBag`, która jest obiektem słownikowym. Podobnie jak w przypadku innych obiektów słownikowych, elementy usuwa się i dodaje do zasobnika stanów przez przypisanie „kluczowi" wartości. Mechanizm ten zostanie opisany w dalszej części punktu.

Zasobnik stanów jest utrzymywany przy użyciu tych samych pól ukrytych, które są wykorzystywane dla celów stanu widoku. Wartości przechowywane w zasobniku stanów można definiować i odczytywać przy użyciu słowa kluczowego `ViewState`. Aby sprawdzić jego działanie, dodamy do naszej aplikacji kolejną stronę (można też utworzyć nową aplikację z nową stroną) o nazwie *StateBagDemo.aspx*.

Między znacznikami `<div>` należy wpisać słowo **`Licznik:`** i przeciągnąć na formularz etykietę. Jako `ID` etykiety należy wpisać `lblCounter`, trzeba również usunąć jej atrybut `Text` (albo wpisać pustą wartość w oknie *Properties*).

Na formularz należy przeciągnąć przycisk, jego `ID` zdefiniować jako `btn`, a jako tekst wpisać **`Zwiększ licznik`**. Po wykonaniu tych czynności kod HTML w widoku *Source* powinien być taki sam jak na listingu 8.2.

Listing 8.2. Kod źródłowy licznika

```
<div>
   Licznik:
   <asp:Label ID="lblCounter" runat="server"></asp:Label>
   <asp:Button ID="btn" runat="server" Text="Zwiększ licznik" />
</div>
```

Wartość licznika będzie przechowywana we właściwości. Należy ją dodać w pliku z kodem źródłowym, a odpowiedni kod przedstawiono na listingu 8.3.

Listing 8.3. Właściwość Counter

```
Public Property Counter() As Integer
   Get
      If (ViewState("intCounter") IsNot Nothing) Then
         Return CInt(ViewState("intCounter")) 'odczytanie ze stanu widoku
      Else
         Return 0
      End If
   End Get

   Set(ByVal value As Integer)
      ViewState("intCounter") = value 'dodanie do stanu widoku
   End Set
End Property
```

`ViewState` jest słownikiem, `"int Counter"` to klucz, a `value` to wartość związana z kluczem. Metodę `OnLoad` należy pokryć (w sposób przedstawiony wcześniej), dodając do niej wiersze kodu zapisane na listingu 8.4 pogrubioną czcionką.

Listing 8.4. Potomna procedura obsługi zdarzenia OnLoad formularza StateBagDemo

```
Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)
   MyBase.OnLoad(e)
   lblCounter.Text = Counter.ToString()
   Counter += 1
End Sub
```

Formularz *StateBagDemo.aspx* ustawia licznik `Counter`, który jest utrzymywany przez cały czas aktywności sesji. Za każdym razem gdy użytkownik naciska przycisk *Zwiększ licznik*, strona jest przeładowywana i wartość licznika ulega zwiększeniu.

Aby umożliwić prawidłowe działanie formularza, między kolejnymi odwołaniami do serwera trzeba utrzymywać stan. Jednym z możliwych rozwiązań jest przechowywanie wartości licznika w zasobniku stanów. ASP.NET udostępnia zasobnik stanów w kolekcji `ViewState`, do której należy się odwoływać przy użyciu nazwy właściwości. W naszym przykładzie dostęp do kolekcji `ViewState` uzyskuje się poprzez właściwość `Counter`. Metoda dostępowa **`Get`** właściwości rzutuje wartość przechowywaną w kolekcji `ViewState` na liczbę całkowitą, ponieważ `ViewState` przechowuje obiekty.

Za każdym razem gdy strona jest ładowana, wyświetlana jest wartość właściwości `Counter` i jest ona zwiększana o `1`:

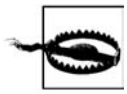
```
Protected Overrides Sub OnLoad(ByVal e As System.EventArgs)
   MyBase.OnLoad(e)
   lblCounter.Text = Counter.ToString()
   Counter += 1
End Sub
```

W bloku **`Get`** właściwości `Counter` następuje sprawdzenie, czy zasobnik stanów noszący nazwę `intCounter` zawiera jakąś wartość:

```
If (ViewState("intCounter") IsNot Nothing) Then
```

Jeśli zasobnik stanów `intCounter` nie jest pusty, zwracana jest przechowywana wartość. W przeciwnym razie zwracana jest wartość `0`. Aby odczytać wartość, trzeba się do niej odwołać przy użyciu nazwy. Aby jednak móc użyć odczytanej wartości w programie, trzeba ją zrzutować na typ **`Integer`**.

```
If (ViewState("intCounter") IsNot Nothing) Then
   Return CInt(ViewState("intCounter"))
Else
   Return 0
End If
```



W tym i we wszystkich programach zawartych w tej książce dyrektywa **`Option`** `Strict` ma zawsze wartość **`True`**. Niestety, jej domyślną wartością jest **`False`**, lecz zasady dobrego programowania obiektowego i zachowania bezpieczeństwa typów nakładają wymóg, by dyrektywa miała wartość **`True`**. Dzięki temu kompilator zostanie zaangażowany do pomocy w wyszukiwaniu błędów powstałych w trakcie rzutowania typów.

Stan sesji

W trakcie pracy aplikacji uruchamianych może być wiele *sesji*. Sesja to seria żądań wysyłanych przez jedną przeglądarkę klienta w sposób mniej lub bardziej ciągły. Jeśli przez określoną ilość czasu (czas ważności sesji) klient nie wyśle żadnego żądania, sesja zostanie zakończona. Domyślnie czas ważności to 20 minut.

Jak już wcześniej wspomniano, ze swej natury sieć WWW jest środowiskiem bezstanowym. Protokół HTTP nie potrafi zidentyfikować, które żądania powinny być grupowane razem w ramach jednej sesji. Sesja musi być definiowana poza protokołem HTTP. ASP.NET przechowuje stan sesji, który posiada następujące cechy:

- Działa z przeglądarkami, w których wyłączono obsługę *cookies*.
- Rozpoznaje, czy żądanie należy do istniejącej sesji.
- Przechowuje dane będące w zasięgu sesji, używane przez różne żądania. Dane tego rodzaju są utrzymywane nawet po restartach serwera i są wykorzystywane w środowiskach wieloprocesorowych (ang. *web garden*) i wieloserwerowych (ang. *web farm*), jak również w środowiskach jednoprocesorowych i przez pojedyncze serwery.
- Automatycznie zwalnia zasoby sesji w momencie zakończenia sesji lub jej wygaśnięcia.

Stan sesji jest przechowywany w pamięci serwera poza procesem ASP.NET. Oznacza to, że jeśli proces ASP.NET ulegnie załamaniu lub zostanie zrestartowany, stan sesji nie zostanie utracony. Stan sesji może być również przechowywany na dedykowanym, oddzielnym komputerze albo na komputerze współużytkowanym, a nawet w bazie danych.

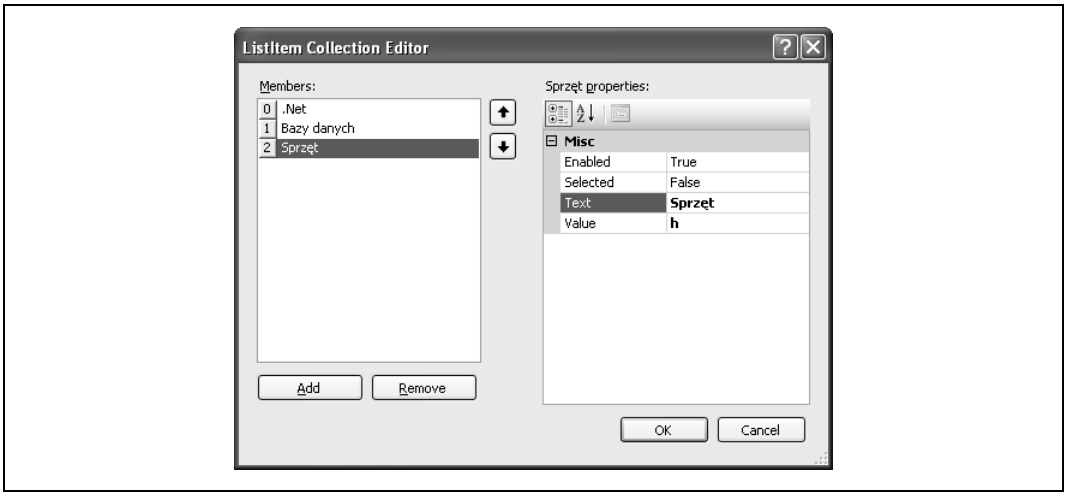
Identyfikacja i śledzenie sesji odbywa się na podstawie 120-bitowego identyfikatora sesji `SessionID` przekazywanego przez klienta na serwer i z powrotem za pośrednictwem pliku *cookie* HTTP lub zmodyfikowanego adresu URL — zależy to od sposobu, w jaki skonfigurowano aplikację. Identyfikator `SessionID` jest obsługiwany automatycznie przez .NET Framework i nie trzeba manipulować nim w kodzie programu. `SessionID` zawiera znaki ASCII dozwolone w adresie URL i posiada dwie ważne cechy:

- Jest unikatowy, dzięki czemu dwie sesje nie będą mieć identycznego `SessionID`.
- Jest losowy, dlatego trudno jest odgadnąć wartość identyfikatora `SessionID` innej sesji na podstawie wartości identyfikatora istniejącej sesji.

Stan sesji zaimplementowano przy użyciu kolekcji `Contents` klasy `HttpSessionState`. Kolekcja `Contents` jest słownikiem par klucz-wartość i zawiera wszystkie obiekty słownikowe stanu sesji, które zostały dodane bezpośrednio w kodzie źródłowym.

Aby zobaczyć, jak działa sesja, należy utworzyć nową stronę *SessionState.aspx* (w aplikacji już istniejącej lub w nowej aplikacji) i wskazać ją jako stronę startową.

Na stronę należy przeciągnąć kontrolkę `RadioButtonList`, jako jej `ID` wpisać `rbl` i przy użyciu taga inteligentnego przeprowadzić edycję pozycji na liście. Do listy należy dodać trzy pozycje. Tekstem pierwszej powinno być słowo `.NET`, a wartością `n`; druga pozycja powinna mieć tekst `Bazy danych` i wartość `d`, zaś tekstem trzeciej powinno być słowo `Sprzęt`, a wartością `h`, jak na rysunku 8.31.



Rysunek 8.31. Dodanie trzech pozycji do listy RadioButtonList

Należy również dodać przycisk i jako jego `ID` wpisać `btn`, a właściwości `Text` przypisać wartość `Zatwierdź`. Na formularz trzeba przeciągnąć też kontrolkę etykiety (`ID=lblMsg`) oraz listę rozwijaną z `ID` o wartości `ddl`. Właściwości `Visible` listy należy przypisać wartość **`False`**.

Wszystkie kontrolki zostały już ustawione, zatem można przejść do strony z kodem źródłowym i zaimplementować procedurę obsługi zdarzenia `Click` przycisku oraz procedurę obsługi zdarzenia `SelectedIndexChanged` kontrolki `RadioButtonList`. Pełen kod źródłowy obydwu metod znajduje się na listingu 8.5.

Listing 8.5. SessionState.aspx.vb

```
Partial Class SessionState_aspx
    Inherits System.Web.UI.Page
    Protected Sub btn_Click( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles btn.Click

        If (rbl.SelectedIndex = -1) Then
            lblMsg.Text = "Należy wybrać kategorię książek."
        Else
            Dim sb As StringBuilder = New StringBuilder()
            sb.Append("Wybrałeś kategorię ")
            sb.Append(CStr(Session("cattext")))
            sb.Append(", której kod to """)
            sb.Append(CStr(Session("catcode")))
            sb.Append(""".")

            lblMsg.Text = sb.ToString()

            ddl.Visible = True

            Dim CatBooks() As String = CType(Session("books"), String())

            ' wypełnienie listy rozwijanej
            Dim i As Integer
            ddl.Items.Clear()
            For i = 0 To CatBooks.GetLength(0) - 1
                ddl.Items.Add(New ListItem(CatBooks(i)))
            Next
        End If
    End Sub

    Protected Sub rbl_SelectedIndexChanged( _
    ByVal sender As Object, _
    ByVal e As System.EventArgs) Handles rbl.SelectedIndexChanged
        If (rbl.SelectedIndex <> -1) Then

            Dim Books(3) As String

            Session("cattext") = rbl.SelectedItem.Text
            Session("catcode") = rbl.SelectedItem.Value

            Select Case (rbl.SelectedItem.Value)
                Case "n"
                    Books(0) = "Visual Basic 2005. Programowanie"
                    Books(1) = "ASP.NET. Programowanie"
                    Books(2) = "Visual C# 2005. Zapiski programisty"
                Case "d"
                    Books(0) = "Oracle Database 10g. Nowe możliwości"
                    Books(1) = "SQL. Almanach. Opis poleceń języka"
                    Books(2) = "Transact-SQL. Czarna księga"
                Case "h"
                    Books(0) = "PC hardware. Almanach. Wydanie III"
                    Books(1) = "Rozbudowa i naprawa laptopów"
                    Books(2) = "To tylko awaria, czyli katastrofy i wpadki z pecetem"
```

```
            End Select
            Session("books") = Books
        End If

    End Sub
End Class
```

Przeanalizujmy najpierw procedurę `rbl_SelectedIndexChanged` obsługującą kontrolkę `RadioButtonList`. Procedura sprawdza najpierw, czy wybrana została kategoria książek. Następnie `rbl_SelectedIndexChanged` definiuje tablicę ciągów znaków przechowującą listę książek z każdej kategorii, po czym wartości `Text` i `Value` zaznaczonej pozycji zostają przypisane dwóm obiektom słownikowym sesji `Session`.

```
Session("cattext") = rbl.SelectedItem.Text
Session("catcode") = rbl.SelectedItem.Value
```

W pierwszym wierszu tekst zaznaczonej pozycji jest zapisywany w stanie sesji przy użyciu klucza `"cattext"`.

W kolejnym kroku `rbl_SelectedIndexChanged` wykonuje instrukcję **`Select Case`**, która wypełnia wcześniej zadeklarowaną tablicę ciągów znaków (`Books`) listą książek z wybranej kategorii. Na końcu metoda przypisuje tablicę ciągów znaków obiektowi słownikowemu Session.

```
Session("books") = Books
```

W przedstawionym przykładzie obiekty słownikowe `Session` przechowują jedynie ciągi znaków i tablicę. W rzeczywistości w obiektach słownikowych można przechowywać dowolne obiekty potomne po `ISerializable`. Dotyczy to wszystkich typów prymitywnych, tablic wartości o typach prymitywnych, a także obiektów `DataSet`, `DataTable`, `HashTable` i `Image`. Dzięki temu w stanie sesji można przechowywać na przykład wyniki zapytań albo kolekcje elementów składające się na implementację koszyka na zakupy.

Druga procedura obsługi zdarzenia, czyli `btn_Click`, jest wywoływana każdorazowo po kliknięciu przez użytkownika przycisku *Zatwierdź*. Procedura sprawdza najpierw, czy zaznaczono któryś z przycisków opcji. Jeśli nie, kontrolka `Label` wyświetla komunikat z ostrzeżeniem.

```
If (rbl.SelectedIndex = -1) Then
    lblMsg.Text = "Należy wybrać kategorię książek."
```

Kluczowy mechanizm strony znajduje się jednak w klauzuli **`Else`** instrukcji **`If`**. W klauzuli odczytywane są obiekty słownikowe `Session` i przy użyciu klasy `StringBuilder` następuje połączenie ciągów znaków w jeden ciąg, który zostanie wyświetlony przez kontrolkę `Label`.

```
Dim sb As StringBuilder = New StringBuilder()
sb.Append("Wybrałeś kategorię ")
sb.Append(CStr(Session("cattext")))
sb.Append(", której kod to """)
sb.Append(CStr(Session("catcode")))
sb.Append("""."")

lblMsg.Text = sb.ToString()
```

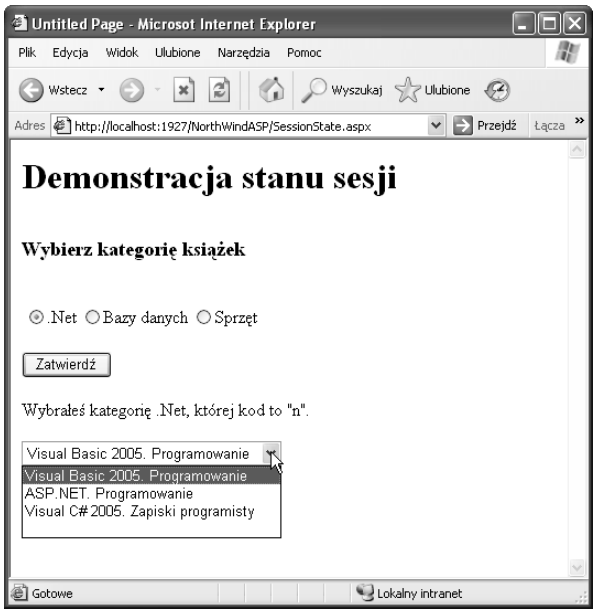
Metoda `btn_Click` odkrywa również kontrolkę `DropDownList`, która została osadzona w kodzie HTML strony i ukryta. Następnie metoda odczytuje tablicę ciągów znaków z obiektu słownikowego `Session` i wypełnia listę rozwijaną `DropDownList`.

```
ddl.Visible = True

Dim CatBooks() As String = CType(Session("books"), String())

'   wypełnienie listy rozwijanej
Dim i As Integer
ddl.Items.Clear()
For i = 0 To CatBooks.GetLength(0) - 1
    ddl.Items.Add(New ListItem(CatBooks(i)))
Next
```

Ponieważ dyrektywa `Page` VB.NET zawiera instrukcję `Strict="true"`, konieczne jest jawne zrzutowanie obiektu słownikowego `Session` zawierającego tablicę ciągów znaków na tablicę ciągów znaków przy użyciu funkcji **CType**. Wyniki działania programu ilustruje rysunek 8.32.



Rysunek 8.32. Demonstracja stanu sesji

Patrząc na przedstawiony przykład, Czytelnik może się zastanawiać, jaka korzyść płynie z użycia stanu sesji i dlaczego zamiast niego nie odczytuje się wartości kontrolek w kodzie źródłowym. Otóż ze względu na prostotę przykładu żadna korzyść nie występuje. Jednak w prawdziwych aplikacjach złożonych z wielu różnych stron stan sesji stanowi łatwe narzędzie przekazywania wartości i obiektów z jednej strony do drugiej, zapewniając jednocześnie uzyskanie korzyści opisanych na początku niniejszego punktu.

Konfigurowanie stanu sesji

Konfiguracja stanu sesji jest kontrolowana oddzielnie na poszczególnych stronach na podstawie ustawień dyrektywy `Page` znajdujących się na początku każdej strony. Konfiguracja stanu sesji dla całej aplikacji jest definiowana w pliku *Web.config*, który zwykle znajduje się w wirtualnym głównym katalogu aplikacji.

Domyślnie stan sesji jest włączony. Stan sesji można włączać również dla konkretnych stron przez dodanie do dyrektywy `Page` atrybutu `EnableSessionState`, jak w poniższej dyrektywie `Page` VB:

```
<%@ Page Language="VB" Strict="true" EnableSessionState="true"%>
```

Aby wyłączyć stan sesji dla strony, należy użyć dyrektywy `Page` w następującej postaci:

```
<%@ Page Language="VB" Strict="true" EnableSessionState="false"%>
```

Aby włączyć stan sesji w trybie tylko do odczytu, to znaczy w taki sposób, by wartości można było odczytywać, lecz nie zmieniać, atrybutowi `EnableSessionState` należy przypisać wartość **`ReadOnly`**:

```
<%@ Page Language="VB" Strict="true" EnableSessionState="ReadOnly"%>
```

(Wielkość liter w wartościach atrybutu `EnableSessionState` nie ma znaczenia). Powodem, dla którego wyłącza się stan sesji lub włącza jedynie w trybie tylko do odczytu, jest wydajność. Jeśli z góry wiadomo, że stan sesji nie będzie wykorzystywany na stronie, można zwiększyć wydajność jej działania przez wyłączenie stanu sesji.

Jeśli proces ASP.NET ulegnie załamaniu lub zostanie zrestartowany, stan sesji nie zostanie utracony. Dodatkowo ASP.NET można skonfigurować w taki sposób, by co jakiś czas następował restart każdego procesu — na przykład po wykonaniu określonej liczby żądań albo po upływie określonego czasu. Ma to na celu podniesienie dostępności i stabilności, a odpowiednie ustawienia konfiguracyjne wprowadza się w plikach *machine.config* i (lub) *Web.config*.

Plik *Web.config* jest plikiem XML, dlatego musi on być prawidłowo ukształtowany. W wartościach znajdujących się w pliku ważna jest wielkość liter, a sam plik składa się z sekcji oddzielonych od siebie znacznikami. Ustawienia konfiguracyjne stanu sesji znajdują się w sekcji `<system.web>`, która z kolei wchodzi w skład sekcji `<configuration>`. Zatem kod wyznaczający typową konfigurację stanu sesji będzie mieć postać podobną do przedstawionej na listingu 8.6.

Listing 8.6. Fragment zawartości pliku Web.config

```
<?xml version="1.0" encoding="utf-8" ?>
<configuration>

 <system.web>
.
.
.
  <sessionState
          mode="InProc"
          cookieless="false"
          timeout="20"
          stateConnectionString="tcpip=127.0.0.1:42424"
          sqlConnectionString="data source=127.0.0.1;userid=sa;password="
/>
```

W sekcji `sessionState` może występować pięć następujących atrybutów:

`mode`
: Wskazuje, czy stan sesji jest wyłączony dla wszystkich stron kontrolowanych przez dany plik *Web.config*, a jeśli stan sesji jest włączony, to gdzie jest przechowywany. W tabeli 8.3 przedstawiono możliwe wartości atrybutu `mode`.

Tabela 8.3. Możliwe wartości atrybutu mode

Wartość	Opis
`Off`	Stan sesji jest wyłączony.
`InProc`	Stan sesji jest przechowywany w procesie na serwerze lokalnym. Jest to wartość domyślna.
`StateServer`	Stan sesji jest przechowywany na zdalnym serwerze. Jeśli użyty został atrybut `StateServer`, wówczas należy zdefiniować również atrybut `stateConnectionString`, który wskazuje serwer przechowujący stan sesji.
`SqlServer`	Stan sesji jest przechowywany na serwerze SQL Server. W przypadku użycia tego atrybutu należy zdefiniować również atrybut `sqlConnectionString`, który definiuje sposób połączenia z serwerem SQL Server. SQL Server może znajdować się na komputerze lokalnym lub zdalnym.

Przechowywanie stanu sesji w procesie na serwerze lokalnym (atrybut `InProc`) jest rozwiązaniem, które działa najszybciej i jest najlepiej dopasowane do wymiany niewielkich ilości zmieniających się danych. Ustawienie to jest jednak także najbardziej podatne na załamania komputera i nie powinno się go używać w środowiskach wieloserwerowych (*web farms*) ani w środowiskach wieloprocesorowych (*web garden*), czyli na pojedynczym serwerze posiadającym większą liczbę procesorów. W takich sytuacjach powinno się używać trybu `StateServer` lub `SqlServer`. Tryb `SqlServer` jest najbardziej odporny na załamania systemu i restarty komputerów.

`cookieless`

Pliki *cookies* są używane przez stan sesji do przechowywania identyfikatora sesji `SessionID`, dzięki któremu serwer rozpoznaje, do której sesji jest podłączony. Atrybut `cookieless` może mieć wartość **`True`** lub **`False`**, przy czym **`False`** jest wartością domyślną. Inaczej mówiąc, domyślnym rozwiązaniem jest użycie plików *cookies*. Jeśli jednak przeglądarka klienta nie obsługuje *cookies* lub ich obsługa została wyłączona przez użytkownika, wówczas każda próba zapisania lub odczytania stanu sesji zakończy się niepowodzeniem. W takich przypadkach atrybutowi `cookieless` należy przypisać wartość **`True`**.

Jeśli `cookieless` ma wartość **`True`**, wówczas wartość `SessionID` jest utrzymywana przez dodawanie jej do adresu URL.

`timeout`

Wskazuje liczbę minut bezczynności, po których upłynięciu sesja wygasa i zostaje porzucona przez serwer. Domyślną wartością jest `20`.

`stateConnectionString`

Wskazuje serwer i port służący do zapisywania stanu sesji. Zdefiniowanie atrybutu jest konieczne, jeśli wartością atrybutu `mode` jest `StateServer`. Zapisywanie stanu sesji na dedykowanym serwerze znacznie ułatwia i zwiększa efektywność zarządzania stanem sesji w środowiskach wieloprocesorowych i wieloserwerowych. Atrybut `stateConnectionString` może mieć na przykład wartość:

```
stateConnectionString="tcpip=127.0.0.1:42424"
```

W powyższym przykładzie stan sesji będzie przechowywany na serwerze o adresie IP `127.0.0.1`, czyli w tym przypadku na serwerze `localhost`, a więc na komputerze lokalnym. Używanym portem będzie port `42424`. Aby takie ustawienie zadziałało prawidłowo, na wskazanym serwerze musi być uruchomiony proces *ASP.NET State Service* (dostępny w narzędziu *Panel sterowania/Narzędzia administracyjne/Usługi*). Ponadto na serwerze musi być otwarty port, przez który będzie przebiegać komunikacja (to znaczy port ten nie może być zamknięty ani zablokowany przez zaporę sieciową ani inne narzędzie zabezpieczające).

sqlConnectionString
: Definiuje ciąg połączenia z uruchomioną kopią SQL Servera. Atrybut musi być zdefiniowany, jeśli atrybut `mode` ma wartość `SqlServer`. Podobnie jak w przypadku `stateConnectionString`, ustawienia takiego najlepiej używać w środowiskach wieloserwerowych i wieloprocesorowych. Ponadto stan sesji przechowywany na SQL Serverze zostanie utrzymany nawet po załamaniu systemu i restarcie komputera. Stan sesji jest zapisywany w tabelach SQL, w których indeksem jest `SessionID`.

Obiekty aplikacji w zasięgu sesji

Dodatkowym sposobem utrzymywania informacji w ramach sesji jest użycie obiektów statycznych, które definiuje się w pliku *global.asax*. Gdy obiekty statyczne zostaną zadeklarowane z atrybutem `Scope` o wartości `Session`, staną się dostępne dla sesji w każdym miejscu kodu źródłowego aplikacji po podaniu nazwy obiektu.

Cykl życia

Użytkownik uruchamia przeglądarkę i wpisuje adres URL. Pojawia się strona WWW z tekstem, obrazkami, przyciskami i tak dalej. Użytkownik wpisuje dane w polu tekstowym i klika przycisk. Co się wówczas dzieje?

Każde żądanie do serwera WWW inicjuje serię kroków. Cały ciąg od kroku początkowego do końcowego tworzy *cykl życia* strony.

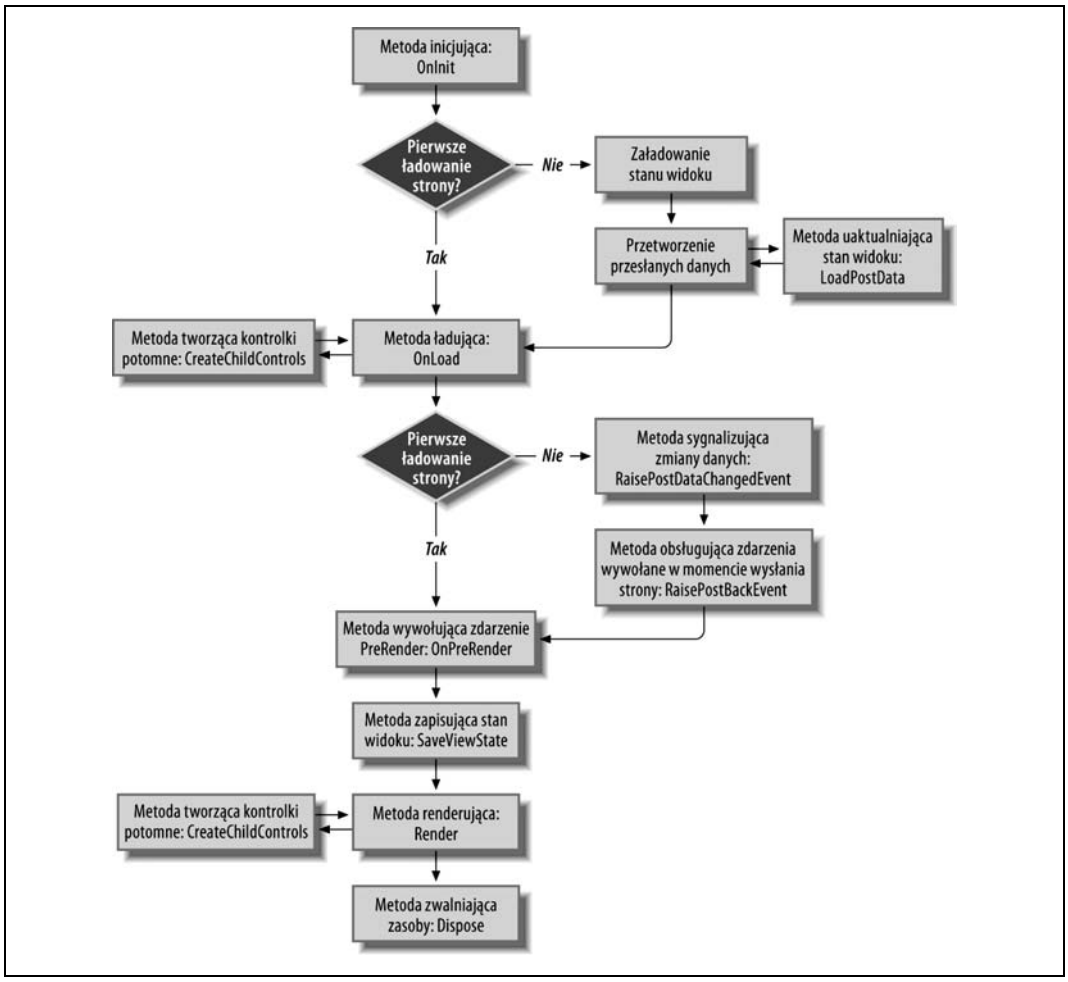
Po wywołaniu strony następuje jej załadowanie, przetworzenie, wysłanie do użytkownika i wyładowanie. W każdym cyklu życia strony jej celem jest poprawne wykonanie renderowania kodu HTML i innych danych wyjściowych oraz przesłanie ich do przeglądarki, która wysłała żądanie. W każdym kroku dostępne są metody i zdarzenia, dzięki którym można pokrywać domyślne zachowanie strony lub rozszerzać je, pisząc własny kod źródłowy.

Aby w pełni zrozumieć cykl życia strony i jej kontrolek, trzeba najpierw zdawać sobie sprawę, że klasa `Page` tworzy hierarchiczne drzewo wszystkich kontrolek znajdujących się na stronie. Wszystkie komponenty strony z wyjątkiem dyrektyw `Page` (które zaraz zostaną opisane) stanowią część takiego *drzewa kontroli*. Drzewo kontroli dla każdej strony można ujrzeć po dodaniu do dyrektywy `Page` atrybutu `Trace="true"`.

Sama strona jest głównym węzłem drzewa. Wszystkie nazwane kontrolki wchodzą w skład drzewa i są oznaczane przez identyfikator kontrolki. Tekst statyczny, w tym znaki niewidoczne, znaki nowego wiersza i znaczniki HTML są reprezentowane w drzewie przez kontrolki `LiteralControl`. Kontrolki są ułożone w drzewie w porządku ściśle hierarchicznym. Na danym poziomie hierarchii porządek kontrolek znajdujących się w drzewie jest wyznaczany przez kolejność, w jakiej pojawiają się w pliku strony.

Komponenty WWW, w tym `Page`, przechodzą cały cykl życia za każdym razem gdy strona jest ładowana. Zdarzenia są najpierw wywoływane na obiekcie `Page`, a następnie rekurencyjnie na każdym obiekcie znajdującym się w drzewie kontrolek.

Poniżej znajduje się szczegółowy opis kolejnych faz cyklu życia formularza WWW. Istnieją dwie nieznacznie różniące się od siebie sekwencje zdarzeń występujących w cyklu życia. Jedna sekwencja dotyczy pierwszego ładowania strony, zaś druga występuje w trakcie każdego kolejnego przesłania strony do serwera. Schemat cyklu życia przedstawiono na rysunku 8.33.



Rysunek 8.33. Cykl życia strony WWW

W przypadku pierwszego ładowania strony cykl życia przedstawia się następująco:

Inicjalizacja

Faza *inicjalizacji* jest pierwszą fazą cyklu życia każdej strony i kontrolki. W fazie inicjalizacji tworzone jest drzewo kontrolek. W fazie tej można zainicjować wartości, które będą używane w trakcie żądania.

Fazę inicjalizacji można zmodyfikować, obsługując zdarzenie `Init` w metodzie `OnInit`.

Ładowanie

Następuje uruchomienie kodu użytkownika i kontrolki formularza wyświetlają dane klienta.

Fazę ładowania można zmodyfikować, obsługując zdarzenie `Load` w metodzie `OnLoad`.

Faza PreRender
: Faza poprzedzająca renderowanie danych wyjściowych. W razie potrzeby następuje wywołanie metody `CreateChildControls`, która tworzy i inicjuje kontrolki serwerowe w drzewie kontrolek. Modyfikacji fazy dokonuje się poprzez zdarzenie `PreRender` w metodzie `OnPreRender`.

Zapisanie stanu widoku
: Widok stanu jest zapisywany w ukrytej zmiennej strony i przybiera postać obiektu **`String`**, kończącego przesyłanie danych do klienta. Modyfikacji fazy dokonuje się w metodzie `SaveViewState`.

Renderowanie
: Strona i jej kontrolki są renderowane do postaci kodu HTML. Modyfikacji fazy dokonuje się w metodzie `Render`. W razie potrzeby w metodzie `Render` wywoływana jest metoda `CreateChildControls`, która tworzy i inicjuje kontrolki serwerowe w drzewie kontrolek.

Zwolnienie zasobów
: Jest to ostatnia faza cyklu życia. W jej trakcie programista może wyczyścić zasoby i zwolnić odwołania do zasobów kosztownych, takich jak połączenia z bazą danych. Faza odgrywa ważną rolę zwłaszcza w zakresie skalowalności. Jej przebieg można zmodyfikować w metodzie `Dispose`.

W przypadku przesłania strony do serwera cykl życia strony prezentuje się następująco:

Inicjalizacja
: Faza przebiega tak samo jak przy pierwszym załadowaniu strony.

Załadowanie stanu widoku
: Właściwość `ViewState` jest odczytywana z ukrytej zmiennej znajdującej się na stronie zgodnie z opisem zawartym w punkcie „Stan widoku" we wcześniejszej części niniejszego rozdziału. Przebieg fazy można zmodyfikować, pokrywając metodę `LoadViewState`.

Załadowanie przesłanych danych
: W trakcie tej fazy następuje przetworzenie danych przesłanych do serwera w ramach żądania `POST`. Wszelkie zmiany stanu widoku wymuszone przez przesłane dane są wykonywane przez metodę `LoadPostData`.

Ładowanie
: Faza przebiega tak samo jak przy pierwszym załadowaniu strony.

Wywołanie zdarzeń zmian
: Jeśli między stanem bieżącym i stanem poprzednim nastąpiły jakiekolwiek zmiany stanu, metoda `RaisePostDataChangedEvent` wywołuje zdarzenia zmian. Zdarzenia są wywoływane na kontrolkach w kolejności zgodnej z kolejnością pojawiania się kontrolek w drzewie kontrolek.

Obsługa zdarzeń wywołanych przy wysłaniu strony
: Wysłanie strony jest wynikiem wykonania przez użytkownika jednej czynności. Czynność ta jest obsługiwana w niniejszej fazie, po obsłużeniu wszystkich zdarzeń zmian. Początkowe zdarzenie, które zaszło po stronie klienta i spowodowało przesłanie strony do serwera, jest obsługiwane przez metodę `RaisePostBackEvent`.

Dyrektywy

Dyrektywy służą do przekazywana opcjonalnych ustawień do stron ASP.NET i kompilatorów. Dyrektywy mają zwykle następującą składnię:

```
<%@ dyrektywa atrybut=wartość [atrybut=wartość] %>
```

Istnieje wiele poprawnych typów dyrektyw, które zostaną szerzej opisane w kolejnych punktach. Każda dyrektywa może posiadać jedną lub więcej par atrybut-wartość, chyba że zostanie wskazane inaczej. Pary atrybut-wartość są oddzielane od siebie znakiem spacji. Należy zwrócić szczególną uwagę, by *nie* wstawiać znaku spacji przed ani po znaku równości (=) rozdzielającym atrybut i jego wartość.

Dyrektywy umieszcza się zwykle na początku odpowiedniego pliku, choć nie jest to wymóg konieczny. Na przykład dyrektywy `Application` znajdują się na początku pliku *global.asax*, a dyrektywy `Page` na początku plików *.aspx*.

Dyrektywa Application

Dyrektywa `Application` służy do definiowania atrybutów dotyczących aplikacji. Zwykle dyrektywa znajduje się w pierwszym wierszu pliku *global.asax*.

Oto przykładowa dyrektywa `Application`:

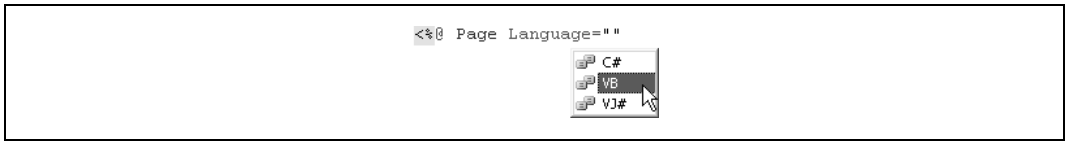
```
<%@ Application Language="VB" %>
```

Dyrektywa `Application` może posiadać trzy atrybuty przedstawione w tabeli 8.4.

Tabela 8.4. Atrybuty dostępne dla dyrektywy Application

Atrybut	Opis
`Inherits`	Nazwa klasy, po której ma nastąpić dziedziczenie.
`Description`	Tekstowy opis aplikacji. Atrybut jest ignorowany przez parser i kompilator.
`Language`	Wskazuje język używany we wszystkich blokach kodu. Dozwolonymi wartościami są `C#`, `VB` i `VJ#`. W miarę rozszerzania zakresu języków obsługiwanych przez .NET Framework lista dozwolonych wartości będzie się poszerzać.

Mechanizm IntelliSense ułatwi wybór jednej z wartości wyliczeniowych, co widać na rysunku 8.34.



Rysunek 8.34. Ustawianie wartości dyrektywy Application Language

Dyrektywa Assembly

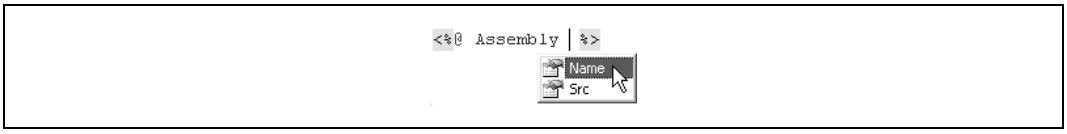
Dyrektywa **`Assembly`** łączy w fazie parsowania podzespół (ang. *assembly*) z aplikacją lub stroną. Dyrektywa działa analogicznie do przełącznika `/reference:` używanego w wierszu poleceń przez kompilatory wiersza poleceń VB.NET.

Dyrektywa **Assembly** jest umieszczana w pliku *global.asax* (obowiązuje wówczas w całej aplikacji) lub w pliku strony (*.aspx*) albo kontrolki (*.ascx*), przez co obowiązuje tylko na danej stronie albo w danej kontrolce. W każdym pliku może się znajdować więcej niż jedna dyrektywa **Assembly**. Ponadto każda dyrektywa **Assembly** może posiadać więcej niż jedną parę atrybut-wartość.

Podzespoły, które znajdują się w podkatalogu *\bin* wirtualnego katalogu głównego aplikacji, są łączone z aplikacją automatycznie i nie trzeba ich dołączać dyrektywą **Assembly**. Dyrektywa może posiadać dwa atrybuty, opisane w tabeli 8.5 i widoczne na rysunku 8.35.

Tabela 8.5. Atrybuty dyrektywy Assembly

Atrybut	Opis
`Name`	Nazwa podzespołu, który ma zostać dołączony do aplikacji lub strony. Nazwa nie zawiera rozszerzenia pliku. Podzespoły mają zazwyczaj rozszerzenie *.dll* (choć mogą również mieć rozszerzenie *.exe*).
`Src`	Ścieżka do pliku źródłowego, który należy dynamicznie dołączyć i skompilować.



Rysunek 8.35. Ustawianie dyrektywy Assembly

Pozostałe dyrektywy zostaną opisane w dalszych częściach książki, w których ich użycie okaże się zasadne.