helion.pl
KSIĘGARNIA INTERNETOWA · WYDAWNICTWO

IDŹ DO
PRZYKŁADOWY ROZDZIAŁ
SPIS TREŚCI

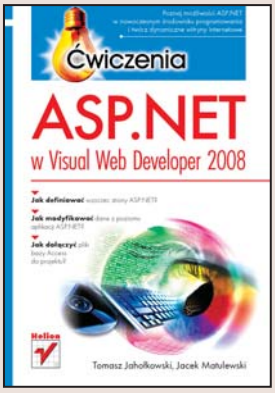
KATALOG KSIĄŻEK
KATALOG ONLINE
ZAMÓW DRUKOWANY KATALOG

TWÓJ KOSZYK
DODAJ DO KOSZYKA

CENNIK I INFORMACJE
ZAMÓW INFORMACJE O NOWOŚCIACH
ZAMÓW CENNIK

CZYTELNIA
FRAGMENTY KSIĄŻEK ONLINE

ASP.NET w Visual Web Developer 2008. Ćwiczenia



Autor: Tomasz Jahołkowski, Jacek Matulewski
ISBN: 978-83-246-1290-1
Format: B5, stron: 160

Poznaj możliwości ASP.NET w nowoczesnym środowisku programowania i twórz dynamiczne witryny internetowe

- Jak definiować wzorzec strony ASP.NET?
- Jak modyfikować dane z poziomu aplikacji ASP.NET?
- Jak dołączyć pliki bazy Access do projektu?

ASP.NET to technologia tworzenia dynamicznych stron internetowych i usług sieciowych po stronie serwera, działająca w oparciu o platformę NET. Narzędzie to doskonale sprawdza się w nowoczesnym i bezpłatnym środowisku programistycznym Visual Web Developer 2008. ASP.NET umożliwia korzystanie z dowolnego języka dostępnego na platformie .NET. Wprowadzany kod jest kompilowany, co oznacza zwiększenie wydajności i jednocześnie daje możliwość sprawdzenia jego syntaktycznej poprawności przed publikacją.

Książka „ASP.NET w Visual Web Developer 2008. Ćwiczenia" przeznaczona jest dla początkujących programistów, pragnących tworzyć witryny ASP.NET, którzy zetknęli się już z jakimkolwiek językiem programowania. Dzięki temu podręcznikowi nauczysz się korzystać z opisanych narzędzi, działających w nowoczesnym środowisku Visual Web Developer 2008. Dowiesz się, jak zaprojektować interfejs, zaprogramować silnik strony, tworzyć arkusze stylów oraz bazy danych. Na konkretnym przykładzie strony domowej nauczyciela poznasz mechanizmy autoryzacji w ASP.NET, a także sposoby dodawania i edycji danych dla poszczególnych użytkowników.

- Tworzenie interfejsu
- Programowanie silnika strony ASP.NET
- Budowanie i stosowanie arkuszy stylów
- Tworzenie bazy danych na serwerze SQL Server 2005
- Tworzenie bazy danych Access
- Technologia LINQ to SQL
- ASP.NET, ADO.NET i LINQ
- Mechanizmy autoryzacji ASP.NET
- Publikowanie aplikacji

Budowanie dynamicznych witryn internetowych jest prostsze, niż myślisz!

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

Spis treści

Wstęp 5

Rozdział 1. Szybki start 9

Tworzenie projektu pierwszej strony ASP.NET 10
Projektowanie interfejsu strony 11
Edycja kodu w pliku .aspx 15
Programowanie silnika strony ASP.NET 17
Walidacja danych 20

Rozdział 2. Praktyka projektowania stron ASP.NET 25

Wzorzec strony (master page) 25
Site map i komponent SiteMapPath 31
Dwa słowa na temat kaskadowych arkuszy stylów 36
Czas życia sesji i aplikacji ASP.NET 40
Sesja i dane sesji 41
Pliki cookies 45
Dane aplikacji 46
AJAX dla ASP.NET 49
Częściowa aktualizacja strony 50
AJAX Control Toolkit 57

Rozdział 3. ASP.NET, ADO.NET i LINQ 67

Moc ADO.NET 68
SQL Server 2005 70
Microsoft Access 76
Bardzo krótki wstęp do języka SQL 80
Modyfikacje danych z poziomu aplikacji ASP.NET 82
LINQ to SQL 92

Rozdział 4. Studium przypadków: strona domowa nauczyciela 103
Dane witryny i konta użytkowników 109
Baza danych 111
Autoryzacja 114
Strony nauczyciela 118
Prezentacja ocen 134
Księga gości 136
Rozdział 5. Publikowanie aplikacji ASP.NET 143
Przygotowanie serwera IIS 144
Publikowanie witryn ASP.NET na serwerze IIS 149



Praktyka projektowania stron ASP.NET

Wzorzec strony (master page)

W przypadku witryny zawierającej kilka stron warto posłużyć się wzorcem. **Wzorzec** (ang. *master page*) to zwykła strona ASP.NET zapisana do pliku z rozszerzeniem *.master*. Wyróżnia ją jednak to, że zawiera komponenty `ContentPlaceHolder`, które rezerwują miejsce do wypełnienia przez strony korzystające ze wzorca. Poza tym wzorzec może zawierać zwykłe elementy HTML, jak i komponenty ASP.NET.

Wzorzec służy jako szablon pozostałych stron projektu. Tworząc nowe strony projektu, możemy wskazać ich wzorzec, a wówczas w widoku projektowania, zamiast edytować całą stronę, będziemy edytować jedynie te miejsca, które we wzorcu zostały zarezerwowane komponentami `ContentPlaceHolder`.

W najprostszym przypadku można posłużyć się wzorcem do ujednolicenia nagłówków i stopek wszystkich stron witryny — wówczas wzorzec zawiera tylko jeden komponent `ContentPlaceHolder`. I właśnie na takim przykładzie nauczymy się teraz tworzenia wzorców i korzystania z nich.

ĆWICZENIE

2.1 Przygotowywanie projektu

Dodanie wzorca do projektu i zastosowanie go w odniesieniu do istniejących stron jest możliwe, ale wymaga edycji szablonu strony. Odłożymy to zatem na później, a teraz utworzymy zupełnie nowy projekt, w którym pierwszą czynnością będzie usunięcie domyślnie utworzonej strony *Default.aspx*. Następnie zdefiniujemy wzorzec i utworzymy korzystające z niego strony.

1. Tworzymy nowy projekt:
   a) z menu *File* wybieramy *New Web Site…*,
   b) zaznaczamy pozycję *ASP.NET Web Site*,
   c) z rozwijanej listy *Location* wybieramy *File System* (wartość domyślna),
   d) a z rozwijanej listy *Language — Visual C#*,
   e) klikamy *OK*.
2. Z projektu usuwamy stronę *Default.aspx*:
   a) zaznaczamy ją w oknie projektu (podokno o nazwie *Solution Explorer*),
   b) rozwijamy menu kontekstowe,
   c) wybieramy z niego polecenie *Delete*,
   d) pojawi się pytanie o potwierdzenie, w którym klikamy przycisk *OK*.

To usunie plik strony nie tylko z projektu, ale także z dysku. W tym miejscu umieścimy nową wersję strony o nazwie *Default.aspx*, ale korzystającą ze wzorca. Wcześniej musimy oczywiście przygotować wzorzec. Nie będziemy w tym zbyt wymyślni — zdefiniujemy prosty nagłówek oraz stopkę strony i zadowolimy się jednym komponentem `Content` ↪`PlaceHolder`.

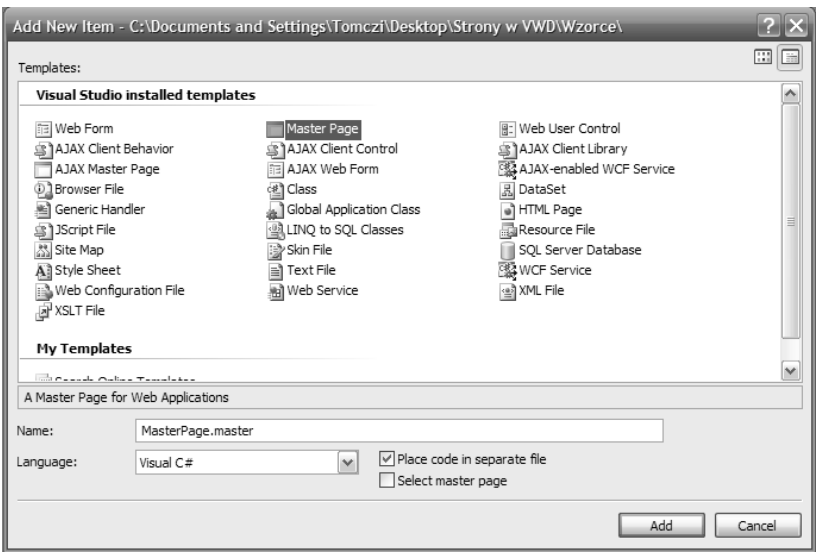


Nie należy mylić usuwania pliku (także z dysku), a więc polecenia *Delete*, z usuwaniem pliku z projektu, tj. z poleceniem *Exclude From Project*.

ĆWICZENIE

2.2 Definiowanie wzorca

1. W oknie projektu zaznaczamy główną gałąź reprezentującą projekt całej witryny (a nie katalog *App_Data*).
2. Z menu *File* wybieramy *New File*….
3. W oknie *Add New Item* (rysunek 2.1) zaznaczamy pozycję *Master Page*.



Rysunek 2.1. *Polecenie dodawania pliku jest czułe na zaznaczoną pozycję w oknie projektu — aby zobaczyć wszystkie możliwe rodzaje plików, należy zaznaczyć pozycję odpowiadającą całemu projektowi*

4. Klikamy *Add*. Do projektu zostanie dodany nowy plik *MasterPage.master*. W edytorze zobaczymy jego kod. Zawiera on dwa komponenty `ContentPlaceHolder`: jeden w nagłówku (w znaczniku `head`), drugi w ciele strony (w znaczniku `body`). My będziemy się teraz interesować przede wszystkim tym drugim.
5. Przejdźmy do widoku projektowania nowej strony (zakładka *Design* w dole okna). Zobaczymy na niej komponent klasy `ContentPlaceHolder` — ten zdefiniowany w znaczniku `body`.

Rezerwuje on miejsce, które będzie zapełniane przez strony korzystające z naszego wzorca. Jeżeli chcemy dodać więcej komponentów `ContentPlaceHolder`, znajdziemy je na zakładce *Standard* w *Toolbox*.

6. Umieśćmy powyżej i poniżej tego komponentu jakiś tekst pełniący rolę nagłówka i stopki stron naszej witryny. Przykład widoczny jest na rysunku 2.2.



Rysunek 2.2. *Osadzanie komponentu ContentPlaceHolder we wzorcu witryny*

ĆWICZENIE

2.3 Strona korzystająca ze wzorca

Nasz wzorzec jest prosty, żeby nie powiedzieć prymitywny, ale nie o estetykę teraz chodzi, a o ideę. Stworzymy zatem przykładowe strony, które będą z tego wzorca korzystać.

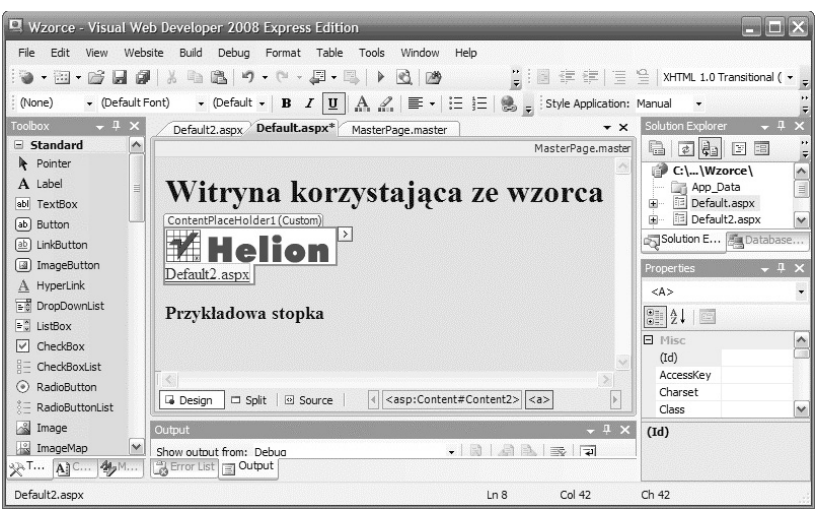
1. Z menu *File* wybieramy ponownie pozycję *New File*.
2. Tym razem zaznaczamy ikonę *Web Form*.
3. Koniecznie musimy zaznaczyć pole opcji *Select master page*. Tylko w momencie tworzenia strony można wskazać jej wzorzec.

4. Zalecam również zaznaczenie opcji *Place code in separate file*, dzięki której ewentualne metody zdarzeniowe będą umieszczone w osobnym pliku.
5. Musimy wskazać jeszcze nazwę pliku strony — domyślnie jest to *Default.aspx* — oraz język użyty do programowania metod zdarzeniowych. Jak już się pewnie Czytelnik zorientował, w tej książce zalecamy używanie C#.
6. Wreszcie klikamy *Add*.
7. Natychmiast pojawi się okno *Select a Master Page*. Wskazujemy w nim stronę *MasterPage.master* i klikamy *OK*.



Ciekawe możliwości daje zagnieżdżanie tworzonej strony wzorca w innych wzorcach. Można tego użyć między innymi do bardziej elastycznej kontroli wyglądu stron z różnych działów jednej witryny.

Po utworzeniu strony znajdziemy się w edytorze kodu. Przejdźmy niezwłocznie do widoku projektowania. Zobaczymy w nim stronę wzorca, ale poza obszarem wyznaczonym przez komponent `ContentPlaceHolder` jest ona niedostępna do edycji (rysunek 2.3). Miejscem, w którym możemy umieszczać nasze komponenty, jest wyłącznie miejsce zarezerwowane wcześniej we wzorcu.



Rysunek 2.3. *Edycja stron korzystających ze wzorca ogranicza się do obszarów wyznaczonych we wzorcu przez komponent ContentPlaceHolder*

Wypełnijmy miejsce przeznaczone na stronę jakąś przykładową zawartością. Umieśćmy w nim np. komponent `HyperLink`. Za pomocą okna *Properties* do jego właściwości `ImageUrl` przypiszmy adres *http://helion.pl//img/logo162_35.gif*, natomiast do właściwości `NavigateUrl` adres *http://helion.pl*. Na podglądzie powinniśmy zobaczyć natychmiast logo Wydawnictwa Helion (por. rysunek 2.3). Możemy stworzyć teraz kolejne strony korzystające z tego samego wzorca, który ujednolici ich wygląd. Wzorzec poprawia zatem spójność całej witryny. Nie do przecenienia jest fakt, że stopkę i nagłówek wszystkich stron witryny kontrolujemy z jednego pliku, zatem jeżeli chcemy coś do nich dodać lub je zmienić, wystarczy edytować tylko plik wzorca.

ĆWICZENIE

2.4 Stosowanie wzorca w istniejącej stronie

Może się zdarzyć, że zechcemy dodać do rozwijanego od pewnego czasu projektu witryny wzorzec ujednolicający wygląd stron. Wówczas stajemy przed problemem użycia nowego wzorca w istniejących już stronach. Wymaga to od programisty modyfikacji kodu w pliku *.aspx*, ale jest jak najbardziej wykonalne. Załóżmy zatem, że projekt zawiera wzorzec i stronę nie korzystającą ze wzorca oraz że chcemy ową stronę w ten wzorzec wtłoczyć. Wówczas należy:

1. Przejść do pliku *.aspx* strony (widok *Source*).
2. W dyrektywie `Page` umieścić atrybut `MasterPageFile`, podając nazwę pliku wzorca.
3. Usunąć wszystkie niepotrzebne znaczniki, a pozostawić tylko te, które stanowią ciało strony (samo wnętrze znacznika `body`).
4. Sprecyzować, w którym kontenerze umieścimy ciało strony. W tym celu wszystkie znaczniki oprócz `Page` trzeba umieścić w znaczniku `asp:Content`, dla którego należy ustawić trzy atrybuty:
   a) `ID` z dowolną wartością,
   b) `ContentPlaceHolderID` — tutaj przy ustawianiu wartości automatycznie otrzymamy do wyboru wszystkie kontenery dostępne we wzorcu,
   c) `runat` z wartością `serwer`.
5. Jeżeli w kodzie strony znajduje się znacznik `form` z atrybutem `runat="server"`, należy go także usunąć.

Listing 2.1 prezentuje kod strony, która została stworzona bez wzorca, a następnie przystosowana do pokazania swojej zawartości (przycisk z etykietą) w komponencie `ContentPlaceHolder1` używanego dotąd w tym rozdziale wzorca.

Listing 2.1. *Kod przekreślony został usunięty z domyślnego kodu strony bez wzorca. Kod z szarym tłem dodano, aby strona współpracowała ze wzorcem*

```
<%@ Page Language="C#" AutoEventWireup="true" CodeFile="Default.aspx.cs"
➥Inherits="_Default" MasterPageFile="~/MasterPage.master" %>

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

<html xmlns="http://www.w3.org/1999/xhtml">
<head runat="server">
    <title>Untitled Page</title>
</head>
<body>
<asp:Content ID="dowolna" ContentPlaceHolderID="ContentPlaceHolder1"
➥runat="server">
    <form id="form1" runat="server">
    <div>

        //przykładowa zawartość strony
        Etykieta:<br />
        <asp:Button ID="Button1" runat="server" onclick="Button1_Click"
            Text="Hello World!" />

    </div>
    </form>
</asp:Content>
</body>
</html>
```

Site map i komponent SiteMapPath

Pozostańmy przy tym samym projekcie i dorzućmy do niego jeszcze kilka stron. Jeżeli witryna ma więcej stron (mowa o kilkunastu, kilkudziesięciu), warto pomyśleć o *site map* — mapie witryny. W ASP.NET przygotowanie takiej mapy polega na utworzeniu pliku XML o nazwie *Web.sitemap*, w którym znajduje się zhierarchizowana grupa elementów `siteMapNode`. W atrybutach każdego z nich wskazujemy adres strony

z witryny, jej tytuł i ewentualnie opis. Struktura znaczników ma odzwierciedlać logiczną strukturę strony, wskazując strony nadrzędne i ich podstrony. Ilość stopni hierarchii jest w zasadzie dowolna.

W naszym projekcie jest tylko kilka stron (załóżmy, że trzy: *Default.aspx*, *Default2.aspx* i *Default3.aspx*), ale i my zdefiniujemy plik *Web.sitemap*. Przyjmijmy, że *Default.aspx* jest stroną tytułową jakiegoś działu witryny o nazwie „Łącza do ważnych stron", a *Default2.aspx* i *Default3.aspx* są zwykłymi stronami tego działu.

ĆWICZENIE

2.5 Tworzenie mapy witryny (site map)

1. Z menu *File* wybieramy pozycję *New File…* i w oknie *Add New Item* wskazujemy pozycję *Site Map*.
2. Powstanie plik *Web.sitemap*. Uzupełniamy go według wzoru z listingu 2.2.

Listing 2.2. *Plik Web.sitemap to plik XML opisujący logiczną strukturę witryny ASP.NET na potrzeby komponentów nawigacyjnych*

```
<?xml version="1.0" encoding="utf-8" ?>
<siteMap xmlns="http://schemas.microsoft.com/AspNet/SiteMap-File-1.0" >
  <siteMapNode url="" title="Nazwa witryny"  description="Opis witryny">
    <siteMapNode url="Default.aspx" title="Linki do stron"
    ➥description="Linki do stron związanych z projektem" >
      <siteMapNode url="Default2.aspx" title="Link do ASP.NET"
      description="Linki do oficjalnej strony ASP.NET" />
      <siteMapNode url="Default3.aspx" title="Link do MSDN"
      ➥description="Link do strony dokumentacji MSDN2" />
    </siteMapNode>
  <siteMapNode url="" title=""  description="" />
  </siteMapNode>
</siteMap>
```



W elemencie `siteMap` może być tylko jeden element `siteMapNode`, więc ewentualną rozbudowę powyższej struktury należy zacząć od trzeciego poziomu zagnieżdżenia elementów XML.

Plik *Web.sitemap* może być źródłem danych dla komponentów umieszczanych na stronach, które będą pozwalać internaucie na zorientowanie się w pozycji oglądanej strony w strukturze całej witryny i umożliwią szybkie przejście do innych jej stron. Na początek przyjrzyjmy się komponentowi `SiteMapPath`.

ĆWICZENIE

2.6 Informacja o pozycji bieżącej strony w strukturze witryny

1. Przejdźmy do widoku projektowania wzorca *MasterPage.master*.
2. Umieśćmy na nim komponent `SiteMapPath` z zakładki *Navigation*. Komponent ten pokazuje ścieżkę aktualnej strony w strukturze zdefiniowanej w pliku *Web.sitemap* (rysunek 2.4).



Rysunek 2.4. *Działanie komponentu SiteMapPath jest uwarunkowane obecnością pliku Web.sitemap. Zdefiniowane w nim opisy pojawiają się w okienkach podpowiedzi*

3. Zwróćmy uwagę na mały trójkącik widoczny po prawej stronie górnej krawędzi nowego komponentu, w sytuacji gdy ten jest zaznaczony. Jeżeli go klikniemy, pojawi się lista typowych zadań dotyczących tego komponentu (rysunek 2.5). W tym przypadku składa się ona z dwóch pozycji *Auto Format* oraz *Edit Templates*. Pierwsze polecenie służy do niemal automatycznego konfigurowania wyglądu komponentu. Wybierzmy np. szablon *Colorful*.



Rysunek 2.5. *Podręczna lista zadań to zbiór najczęściej wykorzystywanych kreatorów związanych z komponentem*

ĆWICZENIE

2.7 Menu strony

Innym zastosowaniem mapy witryny jest automatycznie tworzone menu i drzewo zawierające strony uwzględnione w tym pliku. Zacznijmy od menu. Dodamy je do wzorca — tym razem ponad komponentem rezerwującym miejsce dla stron.

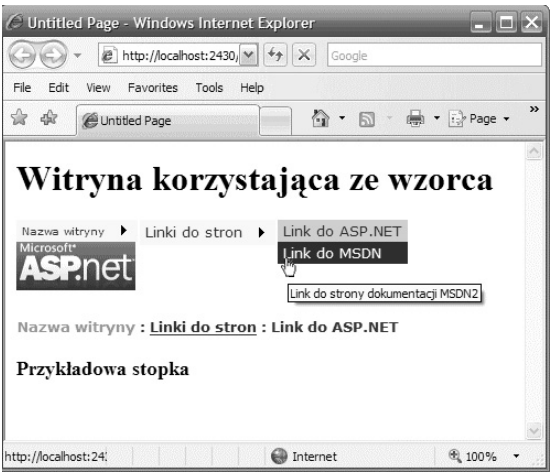
1. Przechodzimy do widoku projektowania wzorca *MasterPage.master*.
2. Ponad komponentem `ContentPlaceHolder` umieszczamy komponent `Menu` z zakładki *Navigation*.
3. W liście podręcznej z rozwijanej listy *Choose Data Source* wybieramy *<New data source…>*.
4. Pojawi się kreator *Data Source Configuration Wizard* pozwalający na wybór źródła danych, na podstawie których zostanie utworzone menu. Może ono zostać zbudowane w oparciu o dowolny plik XML lub nasz gotowy plik *Web.sitemap*. Wybieramy oczywiście tę drugą możliwość, zaznaczając ikonę *Site Map*. Klikamy *OK*. Utworzony zostanie komponent `SiteMapDataSource1`.
5. Nam pozostaje tylko sformatować wygląd menu. Proponuję również tym razem wybrać szablon *Colorful* (rysunek 2.6).

ĆWICZENIE

2.8 Drzewo pokazujące strukturę stron w witrynie

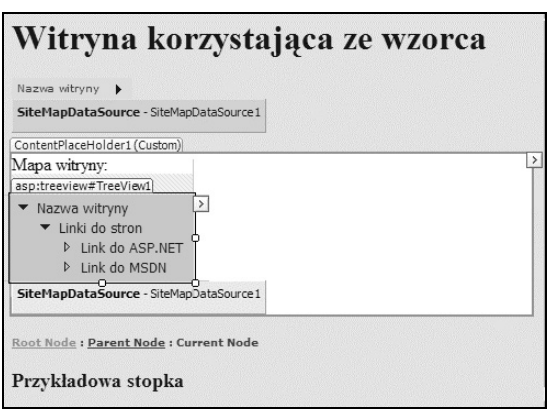
Na zakładce *Navigation* jest jeszcze jeden komponent, na który też warto zwrócić uwagę. Jest to drzewo `TreeView`, które prezentuje strukturę witryny. Nadaje się bardziej na osobną stronę niż do umieszczenia w nagłówku lub stopce stron.

1. Z menu *File* wybieramy *New File*.
2. W oknie *Add New Item* zaznaczamy *Web Form*, wybierając opcję *Select master page*, i zmieniamy nazwę pliku na *MapaWitryny.aspx*.
3. Klikamy *Add*. W nowym oknie wybieramy wzorzec i klikamy *OK*.
4. Przechodzimy do widoku projektowania nowej strony.
5. Na dostępnym obszarze umieszczamy komponent `TreeView` z zakładki *Navigation*.



Rysunek 2.6. *Użycie tego samego stylu do formatowania menu i ścieżki pokazującej pozycję w strukturze witryny to zalążek estetycznego i spójnego wizerunku wszystkich stron witryny*

6. Postępując identycznie jak w przypadku menu, tworzymy źródło danych korzystające z mapy witryny (niestety, nie można użyć gotowego komponentu `SiteMapDataSource1` widocznego w obszarze wzorca).
7. Formatujemy drzewo, wybierając z podręcznej listy zadań pozycję *Auto Format…*. Proponuję użyć stylu *Arrows 2* (rysunek 2.7).



Rysunek 2.7. *Automatycznie generowana mapa witryny w widoku projektu*

8. Nową stronę warto dopisać do mapy witryny, tworząc węzeł równorzędny z *Default.aspx*.



Poza zwykłą nawigacją, jaką umożliwia komponent `TreeView`, można definiować metody zdarzeniowe związane z kliknięciem różnych pozycji drzewa (zdarzenie `SelectedNodeChanged`).

Dwa słowa na temat kaskadowych arkuszy stylów

Wiemy już, że w projektach ASP.NET możemy w znacznym stopniu, w zasadzie nawet całkowicie, odseparować kod C# od szablonu HTML strony. W ten sposób oddzielony zostaje kod odpowiedzialny za statyczny wygląd stron witryny od metod kształtujących ich dynamikę. Do tych dwóch etapów projektowania dochodzi trzeci, w którym za pomocą kaskadowych arkuszy stylów wpływamy na estetykę witryny. Podobnie jak w przypadku kodu C#, także arkusze stylu mogą być całkowicie odseparowane w plikach *.css*, a przez to ich rozwój, podobnie jak kodu C#, może być z łatwością powierzony innym osobom niż te, które rozwijają kod z plików *.aspx*.

W kilku poniższych przykładach przedstawię podstawowe narzędzia służące do budowania kaskadowych arkuszy stylów.

ĆWICZENIE

2.9 Tworzenie arkuszy stylów

Kaskadowe arkusze stylów (ang. *cascade style sheet*) to kolejne obok wzorca narzędzie ujednolicenia stron witryny, a jednocześnie zcentralizowania kontroli nad ich wyglądem. I w tym przypadku wsparcie ze strony VWD jest godne pochwały.

1. Z menu *File* wybieramy *New File*….
2. W oknie *Add New Item* zaznaczamy pozycję *Style Sheet* i jeżeli odczuwamy taką potrzebę, zmieniamy nazwę pliku; potwierdzamy kliknięciem *OK*.

W edytorze zobaczymy niemal pusty plik, który zawiera jedynie tekst:

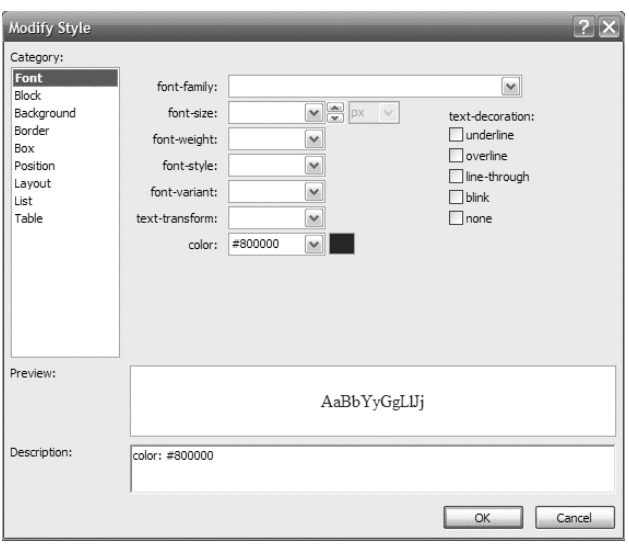
```
body {
}
```

To zalążek stylu związanego ze zwykłym tekstem umieszczonym na stronie (tekst między znacznikami `BODY` w kodzie HTML). Na szczęście nie musimy znać się na formacie arkuszy stylów, bo VWD zawiera proste narzędzie pozwalające na ich definiowanie. Zacznijmy od rozbudowania reguły formatowania dotyczącej prostego tekstu.

ĆWICZENIE

2.10 Formatowanie tekstu na stronie

1. W edytorze ustawiamy kursor (edycji, nie myszy) między nawiasami istniejącej reguły stylu.
2. Klikamy na pasku narzędzi przycisk *Build Style…* (dostępny jest również z menu kontekstowego).
3. W oknie *Modify Style* (rysunek 2.8) możemy wybrać format i kolor czcionki, tła, list i innych elementów umieszczonych na stronie. My ograniczymy się do sformatowania czcionki, dlatego klikamy pozycję *Font* na liście zakładek widocznej z lewej strony okna.



Rysunek 2.8. *Definiowanie reguły stylu dla znacznika body*

4. Rozwijamy listę zatytułowaną *color* i klikamy *More Colors…*:
   a) w oknie *More Colors* odnajdujemy np. kolor *maroon* (jeden z brązowych), który będzie dobrze współgrał ze stylem formatowania wybranym w menu i w innych komponentach nawigacyjnych; dla większej elastyczności kolory reprezentowane są przez liczby szesnastkowe[1];
   b) klikamy *OK*, aby zamknąć kreator stylu.
5. Następnie, korzystając z ikony *Add Style Rule* na pasku narzędzi (lub analogicznego polecenia menu kontekstowego), tworzymy nowy styl dla znacznika `A` (tj. dla umieszczonych na stronie łączy):
   a) ponownie uruchamiamy kreator stylów (przycisk *Build Style…* na pasku narzędzi) i zakładkę *Font*;
   b) tym razem wybieramy kolor *Black*;
   c) w części *text-decoration* zaznaczamy pole *None*.
6. Po tym zdefiniujmy jeszcze jedną regułę formatowania dla `A:hover` (łącze po naprowadzeniu na niego kursora), w którym kolor ustalamy na *Red*, a w części *text-decoration* (por. rysunek 2.8) włączamy opcję *Underline*.

Po tych czynnościach plik kaskadowego arkusza stylów (plik z rozszerzeniem *.css*) powinien wyglądać jak na listingu 2.3.

Listing 2.3. *Zawartość pliku kaskadowych arkuszy stylów*

```
body
{
    color: #800000;
}
A
{
    color: #000000;
    text-decoration: none;
}
A:hover
{
    color: #FF0000;
    text-decoration: underline;
}
```

[1] Bardziej egzotyczne nazwy kolorów mogą być niezrozumiałe dla starszych przeglądarek.

ĆWICZENIE

2.11 Stosowanie arkuszy stylów

Czas, aby arkusz wykorzystać do formatowania stron naszej przykładowej witryny:

1. Przejdźmy na stronę *Default.aspx*.
2. W widoku projektowania dodajmy do niej prosty tekst (wpisując go w polu `Content`) oraz komponent `HyperLink` ze skonfigurowaną właściwością `NavigateUrl` i etykietą (właściwość `Text`).
3. Przejdźmy do widoku projektowania pliku wzorca *MastepPage.master*.
4. Przeciągnijmy z okna projektu (*Solution Explorer*) utworzony plik *.css*. Do kodu strony dodany zostanie element `<link href="StyleSheet.css" rel="stylesheet" type="text/css" />`, dzięki któremu wzorzec i wszystkie używające go strony będą korzystały z arkusza i zdefiniowanych w nim stylów.

W podglądzie wzorca i podglądzie stron, które z niego korzystają, zobaczymy zmianę — tekst zmieni kolor na brązowy, łącza na jasnobrązowy. Ponadto łącza pozbawione zostały podkreślenia. Jeżeli obejrzymy stronę w przeglądarce, to zobaczymy, że kolor łączy zmienia się na pomarańczowy, jeżeli naprowadzić na nie kursor myszy, oraz że pojawia się wówczas pod nimi podkreślenie.



Jeżeli witryna nie ma wzorca, arkusz należy dodać do każdej strony osobno. To samo dotyczy stron w naszej witrynie, które nie korzystają ze wzorca.

Można również edytować indywidualny styl poszczególnych komponentów na stronach. W ich menu kontekstowym znajduje się pozycja *Style…*, która uruchamia okno *Style Builder*, widoczne na rysunku 2.8. Jeżeli natomiast chcemy użyć istniejących klas stylu, w oknie *Properties* odnajdujemy właściwość `CssClass` i tam wpisujemy nazwę klasy zdefiniowanej w arkuszu stylu. Klasy można definiować w pliku *.css*, dodając regułę i zaznaczając opcję *Class name*.

Czas życia sesji i aplikacji ASP.NET

Po pierwszym wywołaniu witryny na serwerze WWW uruchamiana jest odpowiedzialna za nią aplikacja ASP.NET i od tego momentu najczęściej nie przestaje działać aż do zakończenia pracy serwera. Jednocześnie „w obrębie aplikacji" powstaje sesja, która związana jest z tym pierwszym żądaniem. Kolejne żądania od innych przeglądarek-klientów powodują tworzenie następnych sesji. Po opuszczeniu przez internautę witryny sesje kończą pracę, ale aplikacja wciąż jest na posterunku. To zasadnicza zmiana w porównaniu do starszych technologii rozszerzeń serwerów WWW, w których aplikacja uruchamiana była tylko po to, aby przetworzyć otrzymane od przeglądarki żądanie i wygenerować nowy kod HTML. W ASP.NET mamy do czynienia ze stale pracującą aplikacją, która przechowuje swój stan. W tym jest podobna do technologii ***Java Server Pages*** firmy Sun. W szczególności ciągłość pracy aplikacji powoduje, że możemy zapamiętać jakieś dane na jednej stronie witryny, a wykorzystać na innej. Do tego służy zbiór danych (zmiennych) sesji. Nie ma także problemu, aby utworzyć zmienną przechowywaną nawet po zamknięciu sesji. Wystarczy umieścić ją w zbiorze danych aplikacji. A co w przypadku zamknięcia serwera WWW? Oczywiście oba zbiory zostaną utracone. Jednak dane mogą być przechowywane w plikach. Możemy do tego wykorzystać stare dobre *cookies*, szczególnie jeżeli dane dotyczą konkretnego użytkownika — w tym przypadku dane przechowywane są po stronie klienta; można również wykorzystać pliki XML lub wręcz jedną z baz danych ADO.NET. To już oczywiście po stronie serwera. Mamy tu pełną swobodę działania.

Aby lepiej prześledzić cykl życia aplikacji ASP.NET i jej sesji, stwórzmy metody zdarzeniowe związane z kluczowymi momentami życia aplikacji i sesji. W tym celu do aplikacji, którą stworzyliśmy w poprzednim podrozdziale, dodamy specjalny plik *Global.asax*.

ĆWICZENIE

2.12 Tworzenie pliku Global.asax

1. Z menu *File* wybieramy *New File*.
2. W dobrze nam już znanym oknie zaznaczamy pozycję *Global Application Class*.

3. Z rozwijanej listy *Language* wybieramy *Visual C#* i klikamy *Add*.

Zwróćmy uwagę, że rozszerzenie pliku to *.asax*, a nie *.aspx*. To podmienione „a" bierze się od *application*, podczas gdy „p" oznaczało *page* — stronę. Zwróćmy także uwagę na to, że nie mieliśmy możliwości zaznaczenia opcji *Place code in seperate file*, co oznacza, że metody, których potrzebujemy, zostaną zanurzone wprost w kodzie pliku *.asax*.

Po utworzeniu pliku *Global.asax* znajdziemy w nim pięć metod zdarzeniowych, których nazwy dobrze wskazują na zdarzenia, z którymi są związane. Dwie pierwsze to `Application_Start` i `Application_End`. Pierwsza wykonywana jest tuż po uruchomieniu aplikacji ASP.NET, a druga w momencie jej „zdrowego" zamknięcia. Trzecia to `Application_Error` wywoływana w chwili, gdy w aplikacji wystąpi nieobsłużony wyjątek — może być bardzo przydatna w kryzysowych sytuacjach. Kolejne dwie to `Session_Start` i `Session_End`. Ich przeznaczenia nie trzeba się długo domyślać. Warto jednak wiedzieć, że zdarzenie związane z zakończeniem sesji nie jest wywoływane w każdym przypadku — to zależy od trybu pracy sesji. Nie jest też wywoływane natychmiast po opuszczeniu witryny; o szczegółach niżej.

Sesja i dane sesji

Załóżmy, że chcemy naszą stronę uczynić przyjazną. Umieścimy na stronie powitalnej *Default.aspx* pole tekstowe i przycisk z etykietą zachęcającą do wpisania imienia. W domyślnej metodzie zdarzeniowej przycisku umieśćmy instrukcje umieszczające odczytane z pola tekstowego imię w zbiorze zmiennych sesji. Wykorzystaemy do tego obiekt `Session`, który jest składową klasy `Page`, tj. klasy reprezentującej stronę ASP.NET. Metodą `Add` obiektu `Session` możemy dodać nowy obiekt do zbioru danych sesji. Jego identyfikacja możliwa jest na podstawie łańcucha będącego pierwszym argumentem metody `Add` — dla uproszczenia będziemy o nim mówili „nazwa zmiennej".

ĆWICZENIE

2.13 Dodawanie zmiennych do zbioru danych sesji

1. Na stronie umieszczamy pole tekstowe oraz przycisk.
2. Własność `Text` przycisku zmieniamy np. na `Zapamiętaj imię`.

3. Tworzymy domyślną metodę zdarzeniową przycisku i umieszczamy w niej kod z listingu 2.4.

Listing 2.4. *Umieszczanie łańcucha w zbiorze danych sesji i przeładowanie strony*

```
protected void Button1_Click(object sender, EventArgs e)
{
    Session.Add("imie", TextBox1.Text);
    Response.Redirect("Default.aspx");
}
```

Stan obiektu w zbiorze można oczywiście zmienić. Wystarczy kolejne wywołanie metody `Add` z tą samą nazwą zmiennej. Zmienną można także usunąć, służy do tego metoda `Remove`. Można również usunąć wszystkie zmienne, korzystając z metody `RemoveAll`.

ĆWICZENIE

2.14 Weryfikowanie obecności zmiennej w danych sesji i pobieranie jej wartości

Jak sprawdzić, czy zmienna o znanej nazwie znajduje się w zbiorze danych sesji? Można do tego wykorzystać indeksator, czyli nawiasy klamrowe umieszczone zaraz po nazwie obiektu `Session`, wewnątrz których podajemy nazwę zmiennej. Jeżeli zmiennej nie ma, to zwracana przez indeksator wartość równa jest `null`. Wykorzystajmy tę sposobność do ukrycia komponentów wykorzystanych do wpisania imienia przy ponownym załadowaniu strony. W tym celu:

1. Do metody `Page_Load` dodajmy polecenia z listingu 2.5.

Listing 2.5. *Sprawdzamy, czy zmienna o identyfikatorze „imie" jest zdefiniowania*

```
protected void Page_Load(object sender, EventArgs e)
{
    if (Session["imie"] != null)
    {
        Label1.Visible = false;
        TextBox1.Visible = false;
        Button1.Visible = false;
    }
}
```

A jak możemy odczytać wartość przechowywanej w sesji zmiennej? Ano, korzystając z dokładnie tego samego indeksatora. Pamiętajmy

jednak, że zwracana przez niego referencja jest typu `HttpSessionState` i należy ją zrzutować na właściwy typ zmiennej (w naszym przypadku na typ `String`).

Jak możemy wykorzystać przechowywane imię? Możemy je np. wyświetlić w nagłówku strony. Niektórzy lubią, gdy pamięta się ich imię.

2. Przejdźmy zatem do edycji wzorca *MasterPage.master* i umieśćmy na nim komponent `Label`.
3. Jego właściwość `Text` zmieniamy na np. *Wpisz swoje imię na <A HREF="Default.aspx">stronie głównej</A> witryny*. Będzie to tekst widoczny w razie braku zmiennej z imieniem w zbiorze danych sesji. Natomiast jeżeli zmienna `imie` jest tam obecna, to etykietę tę zamieniamy na *Masz na imię* z dodanym przechowywanym w sesji imieniem.

   No cóż, zdaję sobie sprawę z tego, że niezbyt mądry jest to tekst, ale w końcu chodzi o nauczenie się przechowywania zmiennych w sesji, a nie o wymyślanie ich błyskotliwych zastosowań.

   W pliku *MasterPage.master* nie ma metody `Page_Load`, musimy ją zatem utworzyć i wykorzystamy do tego sposób, którego wcześniej nie używaliśmy.
4. Przejdźmy do edytora kodu pliku *MasterPage.master*. Zwróćmy uwagę na dwie rozwijane listy widoczne nad podoknem edytora.
5. Z lewej wybierzmy pozycję `Page`. Wówczas w prawej możemy wskazać zdarzenie `Load`. Tym samym utworzymy metodę `Page_Load`, która będzie wywoływana przy każdym wczytaniu strony.
6. Umieśćmy w niej polecenia widoczne na listingu 2.6.

Listing 2.6. *Odtwarzanie wartości zmiennej przy załadowaniu wzorca do przeglądarki*

```
<script runat="server">

    protected void Page_Load(object sender, EventArgs e)
    {
        String imie = (String)Session["imie"];
        if (imie!=null) Label1.Text = "Masz na imię <FONT COLOR=black>"
        ↪+ imie + "</FONT>.";
    }

</script>
```

ĆWICZENIE

2.15 Wyświetlanie adresu IP zdalnego komputera

Nieco mniej przyjazne będzie wyświetlenie adresu komputera, przy którym siedzi internauta. To jakby mu powiedzieć: „Uważaj, co robisz, wiem, skąd jesteś, obserwuję Cię". Mimo to zróbmy to. Adres odczytamy tuż po utworzeniu sesji w metodzie `Session_Start` z pliku *Global.asax*.

1. Dodajmy do niej polecenie widoczne na listingu 2.7.

Listing 2.7. *Tu z kolei „permanentna inwigilacja"*

```
void Session_Start(object sender, EventArgs e)
{
    Session.Add("adresIP", Request.UserHostAddress);
}
```

Zamiast pobierać z właściwości `Request.UserHostAddress` adres IP, można również odczytać nazwę DNS komputera, na którym uruchomiona jest przeglądarka (właściwość `Request.UserHostName`), o ile nazwa ta jest zdefiniowana i dostępna. W podobny sposób można odczytać także kod używanej przez internautę przeglądarki (`Request.UserAgent`).

Pozostaje tylko umieścić adres IP na tym samym komponencie `Label` co imię.

2. Przechodzimy zatem do strony wzorca i do metody z listingu 2.6 dodajemy polecenia wyróżnione w listingu 2.8.

Listing 2.8. *Metoda Page_Load uruchamiana jest przy każdym „przeładowaniu" strony. Jest jeszcze Page_Init*

```
protected void Page_Load(object sender, EventArgs e)
{
    String imie = (String)Session["imie"];
    if (imie != null) Label1.Text = "Masz na imię <FONT COLOR=black>" +
    ↪imie + "</FONT>.";
    String adresIP = (String)Session["adresIP"];
    if (adresIP != null) Label1.Text += "<BR>Twój komputer to <FONT
    ↪COLOR=black>" + adresIP + "</FONT>.";
}
```

Pliki cookies

No dobrze, ale czy nie byłoby fajnie, gdyby witryna zapamiętywała imię nawet po jej opuszczeniu i powrocie następnego dnia? Oczywiście, że tak. A jak to zrobić? Otóż mamy do wyboru dwa sposoby. Możemy zapisać w bazie danych na serwerze adres IP komputera i podane imię. Wówczas przy każdym żądaniu aplikacja mogłaby sprawdzać adres IP, skąd nadeszło żądanie, porównywać z tymi w bazie i w przypadku znalezienia odczytywać zapisane z nim imię. Jak widać, dużo z tym zachodu, a do tego metoda nie jest pewna, bo z jednego komputera może korzystać kilku niezależnych użytkowników. Dlatego lepiej i prościej jest wykorzystać pliki *cookies*, czyli, swojsko mówiąc, ciasteczka. Są one przechowywane wśród danych użytkownika po stronie klienta. To zapewnia pełną personalizację i do tego dyskrecję przechowywanych danych. Internauta ma bowiem pełną kontrolę nad swoimi plikami cookies.

ĆWICZENIE

2.16 Przechowywanie danych w plikach cookie

1. Do metody zdarzeniowej `Button1_Click` z listingu 2.4 dodajmy polecenie zapisujące imię do pliku *cookie*. Odpowiednie modyfikacje pokazuje listing 2.9.

Listing 2.9. *Termin przydatności do spożycia naszego ciasteczka minie za trzy dni*

```
protected void Button1_Click(object sender, EventArgs e)
{
    Session.Add("imie", TextBox1.Text);
    HttpCookie ciasteczko = new HttpCookie("imie", TextBox1.Text);
    ciasteczko.Expires=DateTime.Today.AddDays(3);
    Response.Cookies.Add(ciasteczko);
    Response.Redirect("Default.aspx");
}
```

ĆWICZENIE

2.17 Odczytywanie danych z plików cookie

Po wpisaniu imienia zostanie ono zachowane nie tylko w ulotnych zmiennych sesji, ale również w bardziej trwałych plikach *cookies*. Po ponownym wejściu na witrynę imię powinno zostać odczytane z *cookies*

i wyświetlone w jej nagłówku. Zapewnią to zmiany w metodzie `Session_Start` widoczne na listingu 2.10. Odczytane z *cookie* imię zapisywane jest do zmiennej sesji `imie`, dzięki czemu wykorzystywany jest przygotowany wcześniej mechanizm ukrywania komponentów i wyświetlania imienia.

Listing 2.10. *Przy wejściu na witrynę imię odczytywane jest z pliku cookie na komputerze internauty*

```
void Session_Start(object sender, EventArgs e)
{
    Session.Add("adresIP", Request.UserHostAddress);
    if (Request.Cookies["imie"] != null)
        Session.Add("imie", Request.Cookies["imie"].Value);
}
```



Przykłady wykorzystania plików XML i bazy danych do przechowywania informacji po stronie serwera znajdzie Czytelnik w dwóch następnych rozdziałach.

Dane aplikacji

Czym by tu jeszcze umilić życie zabłąkanego na naszą witrynę internauty? Czasami podawana jest na stronie liczba osób odwiedzających w danej chwili witrynę. Aby stworzyć taki licznik, wystarczy w zmiennych aplikacji zdefiniować zmienną, której wartość będzie zwiększana za każdym razem, gdy rozpoczynana będzie nowa sesja, a zmniejszana przy jej zakończeniu. Zamknięcie sesji nie jest jednak tak oczywiste jak jej otwarcie. Internauta może opuścić witrynę, przechodząc na inną stronę. Potem może chcieć do niej wrócić i miło byłoby mu zastać ją w takim stanie, w jakim ją zostawił. Z myślą o tym aplikacja utrzymuje jeszcze sesję przez pewien czas po opuszczeniu witryny. Internauta może też po prostu zamknąć przeglądarkę, a wówczas do aplikacji nie dociera żaden sygnał, że sesja ma być zakończona. Dlatego właśnie sesja utrzymywana jest tylko „przez pewien czas". Mianowicie jeżeli przez dwadzieścia minut nic w takiej porzuconej sesji się nie wydarzy, jest ona automatycznie kończona. Czas ten można zmienić, edytując plik *Web.config*, a dokładnie atrybut `timeout` elementu `sessionState`, który należy umieścić wewnątrz elementu `system.web`. Poniższy element zmniejsza czas życia porzuconej sesji do trzech minut:

```
<system.web>
    <sessionState timeout="3" />
    ...
```

ĆWICZENIE

2.18 Licznik osób odwiedzających witrynę

Przejdźmy jednak do tworzenia naszego licznika.

1. W pliku *Global.asax* zmodyfikujmy nieco metody `Application_Start`, `Session_Start` i `Session_End`, dodając nową zmienną w zbiorze danych aplikacji. Pokazuje to listing 2.11. Zauważmy, że inaczej niż w przypadku zmiennych sesji nową wartość zmiennej aplikacji ustala się za pomocą metody `Set` (wyróżnienie w metodzie `Session_Start`).

Listing 2.11. *Inicjowanie i zmiany wartości zmiennej iloscOsob*

```
void Application_Start(object sender, EventArgs e)
{
    Application.Add("ileOsob", 0);
}

void Session_Start(object sender, EventArgs e)
{
    Session.Add("adresIP", Request.UserHostAddress);
    if (Request.Cookies["imie"] != null) Session.Add("imie",
Request.Cookies["imie"].Value);
    if (Application["ileOsob"] != null) Application.Set("ileOsob",
    ↪(int)Application["ileOsob"]+1);
}

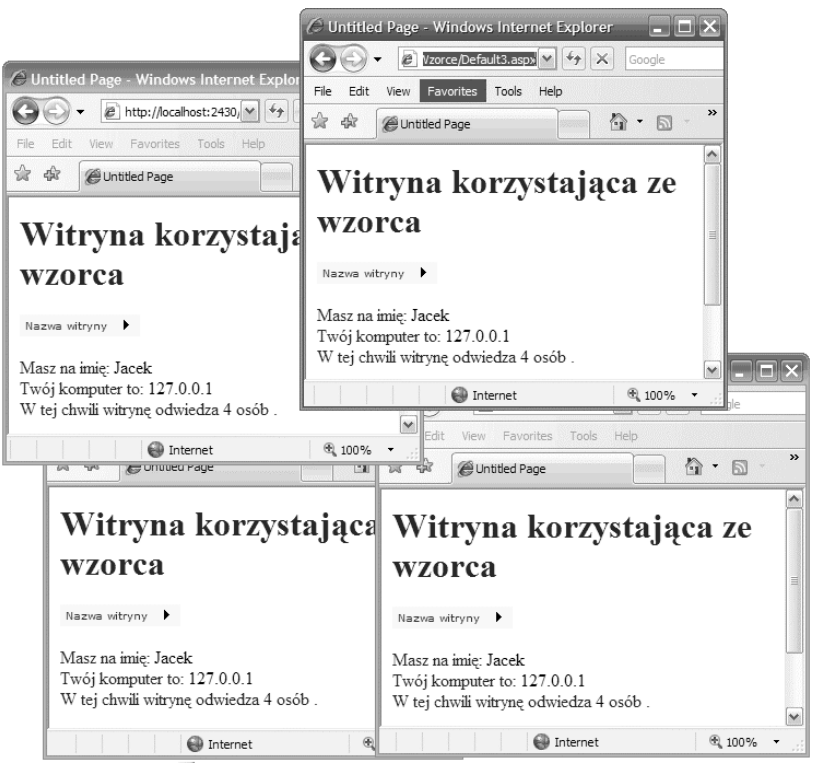
void Session_End(object sender, EventArgs e)
{
    if (Application["ileOsob"] != null) Application.Set("ileOsob",
    ↪(int)Application["ileOsob"] - 1);
}
```

Dobrze sobie także uświadomić, kiedy tworzone są sesje z punktu widzenia komputera-klienta i uruchomionej na nim przeglądarki. Jeżeli w Internet Explorerze wybierzemy adres witryny ASP.NET — z całą pewnością powstanie pierwsza sesja. Jednak jeżeli w tej przeglądarce naciśniemy klawisze *Ctrl+N*, czyli otworzymy nowe okno przeglądarki z załadowaną stroną witryny, to nowa sesja wcale nie powstanie. Aplikacja wykryje, że oba żądania pochodzą od tej samej instancji przeglądarki.

To samo dotyczy otwarcia nowej zakładki w IE7. Jednak jeżeli nowe okno przeglądarki otworzymy, korzystając z menu *Start* lub ikony na pasku narzędzi szybkiego uruchamiania, to utworzymy jej nową, całkowicie niezależną instancję, dla której powstanie nowa sesja, jeżeli wczytamy do niej dowolną stronę omawianej tu witryny.



Ważnym zagadnieniem dotyczącym całej witryny, którego tu nie omówiłem, jest uwierzytelnianie użytkowników. Sporo informacji na ten temat znajdzie Czytelnik w rozdziale 4.



Rysunek 2.9. *Aby zaktualizować licznik widoczny w nagłówku wzorca, strona musi być przeładowana*

AJAX dla ASP.NET

Aplikacje ASP.NET oddzielają wyraźnie zadania przeglądarki i serwera. Przeglądarka udostępnia użytkownikowi jedynie interfejs graficzny, dzięki któremu może sterować działaniem aplikacji na zdalnym serwerze. Wszelkie zmiany na stronie wymagają wysłania z przeglądarki żądania do serwera, tam zostaje ono przetworzone przez aplikację ASP.NET, która wygeneruje i odeśle do przeglądarki nową wersję kodu HTML. Skrypty wykonywane po stronie serwera, może poza procesem walidacji, ingerują w działanie aplikacji w minimalnym stopniu. Dzięki takiemu rozwiązaniu możliwości aplikacji ASP.NET są ogromne, poznaliśmy już ich wiele. Płacimy jednak za to szybkością aplikacji, która musi przy każdej zmianie wysyłać i odbierać dane przez sieć. Nie bez znaczenia jest również obciążenie serwera, który w przypadku popularnej witryny musi jednocześnie obsłużyć wiele żądań.

Stąd wziął się pomysł na AJAX, tj. *Asynchronous JavaScript And XML* (asynchroniczny JavaScript i XML). Jego zasadniczą ideą jest dodanie trzeciej warstwy do klasycznego modelu klient-serwer, jaki wykorzystywany jest w aplikacjach ASP.NET. Warstwa ta to silnik AJAX (ang. *AJAX engine*), który umiejscowiony jest po stronie klienta. Dzięki temu kontakt przeglądarki z nim nie obciąża sieci (pamiętajmy jednak, że konieczne jest przesyłanie skryptów do klienta). Celem nowej warstwy jest odciążenie serwera przez przejęcie części jego zadań i w ten sposób ograniczenie ilości koniecznych z nim kontaktów. Jednak w praktyce używa się AJAX przede wszystkim do ożywienia i uatrakcyjnienia witryny przez dodawanie różnego typu „wodotrysków". A ponieważ to obciąża jedynie komputer klienta, możemy to robić bez obaw o „zatkanie" serwera. Funkcja uatrakcyjnienia witryny, choć w pierwszej chwili kojarzy się tylko z zabawą, jest bardzo istotna. Funkcjonalności wprowadzane dzięki AJAX mogą zdecydować o tym, czy internauta zwiąże się z naszą witryną, czy wybierze inną — bardziej atrakcyjną.

Pozostała część rozdziału w całości poświęcona będzie możliwościom oferowanym przez technologię AJAX, która w Visual Studio 2008 jest integralną częścią ASP.NET (nie trzeba jej już instalować osobno). Zacznę od przedstawienia komponentów z zakładki *AJAX Extensions*, a dokładnie tych kontrolek, które pozwalają na częściową aktualizację strony. Następnie zaprezentuję AJAX Control Toolkit (ACT) — zbiór

kontrolek i rozszerzeń, które implementują najróżniejszego rodzaju „wodotryski": przesuwane panele, komponenty widoczne zawsze nad zawartością strony, wyskakujące okna i wiele innych.

Częściowa aktualizacja strony

ĆWICZENIE

2.19 Komponenty ScriptManager i UpdatePanel

1. Stwórzmy projekt typu *ASP.NET Web Site*.
2. Przejdźmy do widoku projektowania domyślnie utworzonej strony *Default.aspx* i umieśćmy na niej kontrolkę `ScriptManager` z zakładki *AJAX Extensions*.

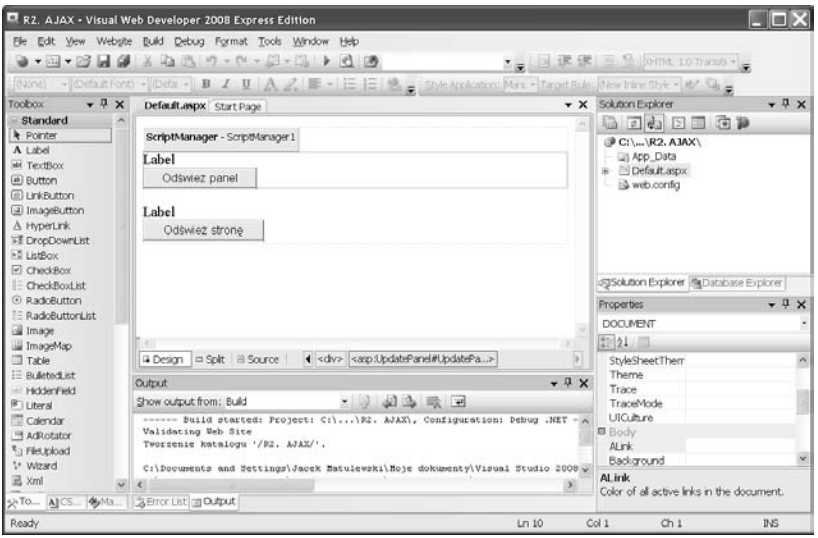


Na każdej stronie korzystającej z AJAX musi znajdować się jeden (i tylko jeden) komponent `ScriptManager`. Jego zadaniem jest zarządzanie skryptami *JavaScript* wysyłanymi do przeglądarki (klienta). Po wyposażeniu strony w menadżera skryptów możemy rozpocząć jej „uajaksowianie". Na początek zbadajmy kontrolkę `UpdatePanel`. Jest ona podstawą ważnego w AJAX mechanizmu częściowego aktualizowania strony.

3. Umieśćmy na stronie komponent `UpdatePanel` z zakładki *AJAX Extensions*. Jest to pojemnik, w którym można umieszczać wszystkie kontrolki biblioteki *Web Forms*.
4. Wstawmy do niego dla przykładu komponent `Label` z zakładki *Standard*. Drugi umieśćmy poza tym pojemnikiem (rysunek 2.10).
5. Do metody `Page_Load` w pliku *Default.aspx.cs* dodajmy polecenia ustalające etykiety obu komponentów `Label` zgodnie ze wzorem z listingu 2.12.

Listing 2.12. *Do etykiet przypisujemy aktualną datę i czas*

```
protected void Page_Load(object sender, EventArgs e)
{
    Label1.Text = "Aktualna data: " + DateTime.Now.ToString();
    Label2.Text = "Wczytanie strony: " + DateTime.Now.ToString();
}
```

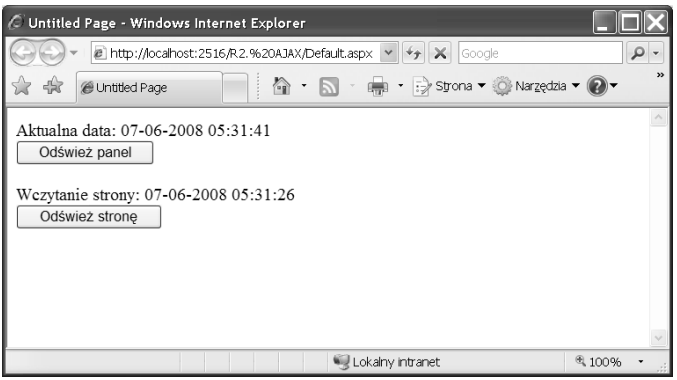


Rysunek 2.10. *UpdatePanel na stronie aplikacji ASP.NET korzystającej z AJAX*

6. Następnie dodajmy do strony dwa przyciski — jeden wewnątrz kontrolki `UpdatePanel`, a drugi poza nią, jak pokazuje rysunek 2.10.

Wiemy, że ze względu na właściwość `PostBack` ustawioną domyślnie na `true` kliknięcie przycisku powoduje przeładowanie strony w każdym przypadku, bez względu na to, czy związana jest z tym zdarzeniem metoda. I zgodnie z tymi oczekiwaniami działa przycisk umieszczony poza pojemnikiem `UpdatePanel`. Jednak gdy klikniemy przycisk umieszczony wewnątrz niewidocznego w przeglądarce komponentu `Update` ➥`Panel`, okaże się, że data zmieni się tylko w komponencie `Label`, który również znajduje się wewnątrz tego panelu (rysunek 2.11). A jeszcze dziwniejsze jest to, że wygląda na to, iż strona wcale nie została przy tym przeładowana. A przynajmniej nie cała.

Jak to działa? Nie dajmy się omamić. Za odświeżoną zawartość komponentu `Label` nie jest odpowiedzialny żaden skrypt uruchomiony w przeglądarce. Nawet jeżeli nie widać tego po zachowaniu przeglądarki, to jednak wysłała ona żądanie do aplikacji ASP.NET na serwerze (*postback*), a ta wykonała wszystkie metody związane z tworzeniem strony, między innymi wykonała metodę `Page_Load` i odesłała do przeglądarki odpowiedź w postaci nowego kodu HTML. W przeglądarce



Rysunek 2.11. *Różnica czasu wynika z tego, że odświeżona została tylko zawartość pojemnika UpdatePanel*

nie doszło jednak do pełnego przeładowania strony, a jedynie do aktualizacji jej części wyznaczonej przez `UpdatePanel`. Dzięki temu strona sprawia wrażenie, jakby działała bez kontaktu z serwerem. Należy podkreślić, że jeżeli zmiany wprowadzone w wykonanej na serwerze metodzie dotyczą komponentów spoza panelu `UpdatePanel` (np. aktualizacja daty na `Label2`), to nie staną się one widoczne po częściowej aktualizacji strony, tj. wówczas, gdy żądanie odświeżenia wysłane było na skutek kliknięcia przycisku wewnątrz panelu.

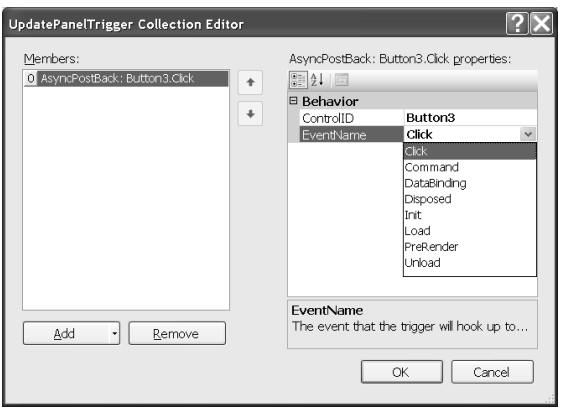
O tym, czy kontrolki umieszczone na `UpdatePanel` powodują częściowe odświeżenie strony, decyduje jego własność `ChildrenAsTriggers`. Aby to sprawdzić, w widoku projektowania strony zaznaczmy `UpdatePanel1` i w oknie *Properties* przełączmy jego właściwość `ChildrenAsTriggers` na `false`, a `UpdateMode` na `Conditional`. Po uruchomieniu aplikacji okaże się, że klikanie przycisku wewnątrz pojemnika nie daje żadnego efektu. Nie dochodzi do częściowej aktualizacji strony. Przywróćmy jednak pierwotne ustawienia.

ĆWICZENIE

2.20 Wyzwalanie częściowej aktualizacji przez komponent spoza UpdatePanel

Można również sprawić, aby aktualizację zawartości `UpdatePanel1` powodował komponent nieumieszczony na tym panelu. Wystarczy dodać go do kolekcji wyzwalaczy zebranych we właściwości `Triggers` komponentu `UpdatePanel1`:

1. Umieśćmy jeszcze jeden przycisk (`Button3`) na stronie poza pojemnikiem `UpdatePanel1`.
2. Zaznaczmy obiekt `UpdatePanel1` i uruchommy edytor jego właściwości `Triggers`. Pojawi się okno z tytułem *UpdatePanelTrigger Collection Editor* widoczne na rysunku 2.12.



Rysunek 2.12. *Konfigurowanie wyzwalacza „zewnętrznego”*

3. Klikamy przycisk *Add*. Do listy *Members* dodana zostanie pozycja *AsyncPostBack*.
4. W prawej części okna pojawiły się właściwości dodanego obiektu. Musimy w nich wskazać komponent wyzwalający — robimy to, wybierając z rozwijanej listy przy właściwości `ControlID` komponent `Button3` i wskazując w polu `EventName` zdarzenie, które ma powodować aktualizację. W przypadku przycisku powinno to być zdarzenie `Click`.

Uruchommy teraz aplikację, aby przekonać się, że kliknięcie nowego przycisku odświeża panel `UpdatePanel1` bez przeładowania całej strony. I tylko panel; jego kliknięcie nie powoduje już odświeżenia drugiego z obiektów `Label` ani przeładowania całej strony. „AsyncPostBack” oznacza, że aktualizacja wykonana zostanie asynchronicznie (tzn. w czasie dogodnym dla aplikacji ASP.NET) — tu wykorzystana zostanie zasadnicza zaleta AJAX, która optymalizuje wykorzystanie sieci i szybkość działania aplikacji.

Możliwe jest również wymuszenie natychmiastowego odświeżenia. Zauważmy, że przy przycisku *Add* jest trójkąt, którego kliknięcie odsłania menu. Poza naszym wyzwalaczem asynchronicznym jest tam również zwykły, powodujący natychmiastowe odświeżenie panelu.

Sygnalizowanie częściowej aktualizacji

Idea częściowej aktualizacji strony polega na ukryciu przed użytkownikiem, że następuje proces aktualizacji, tj. wysłania żądania do działającej na serwerze aplikacji i odebrania nowej wersji kodu HTML. Niestety, mechanizm ten w żaden sposób nie przyspiesza działania serwera, więc od działania użytkownika do zobaczenia przez niego efektów minie niemal tyle samo czasu co w przypadku pełnej aktualizacji strony. Różnica będzie brała się jedynie z czasu poświęconego na renderowanie całego kodu HTML, co zazwyczaj nie trwa długo. Nie widząc przeładowania, użytkownik naszej aplikacji ASP.NET może czuć się zdezorientowany — do momentu odświeżenia panelu strona zachowuje się bowiem, jakby ignorowała jego polecenia. Dlatego czasem warto sygnalizować trwanie przedłużającej się częściowej aktualizacji.

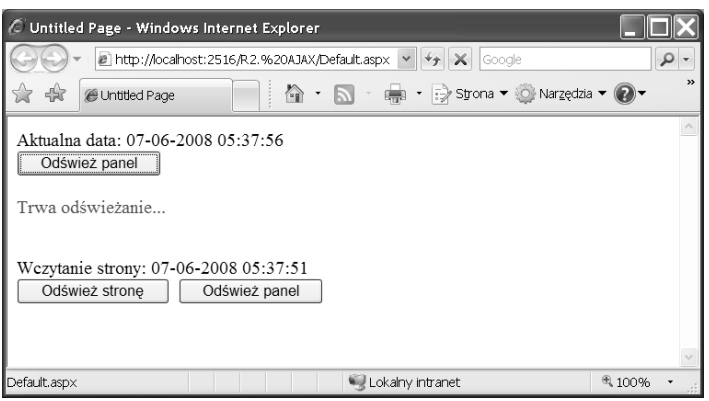
Na stronie umieszczamy komponent `UpdateProgress`, którego zadaniem jest informowanie o trwającym częściowym odświeżaniu strony. Komponent ten to podobnie jak `UpdatePanel` pojemnik, w który można wstawiać inne komponenty Web Forms. My jednak ograniczymy się do prostego napisu, np. „Trwa odświeżanie…”. Aby komunikat był lepiej widoczny, możemy go umieścić w tabeli z jedną komórką, której tło będzie kolorowe, a tekst biały, lub wyróżnić w inny sposób (rysunek 2.13). Przyjrzyjmy się teraz właściwościom tego komponentu. Najważniejsza z nich to `AssociatedApdatePanelID`, w której z rozwijanej listy możemy wybrać komponent `UpdatePanel`. W trakcie jego aktualizacji umieszczony przed chwilą na stronie `UpdateProgress` ma stać się widoczny. Ponieważ na stronie umieściliśmy jak na razie tylko jeden `UpdatePanel`, nie mamy wielkiego wyboru i z nim wiążemy nowy komponent. Drugą ważną właściwością jest `DisplayAfter`, zawierającą liczbę milisekund, po których — o ile w tym czasie serwer nie odeśle kodu HTML — `UpdateProgress` się ujawni. Domyślne ustawienie to 500 milisekund, czyli pół sekundy. To całkiem rozsądny czas w przypadku serwerów, które nie pracują lokalnie. W przypadku lokalnego serwera pół sekundy zwykle wystarczy do odświeżenia i dlatego nigdy nie zobaczylibyśmy komunikatu. Musimy albo skrócić to opóźnienie do

zera, albo posunąć się do drobnego oszustwa i rozmyślnie opóźnić reakcję aplikacji ASP.NET. W tym celu można do metody `Page_Load` z listingu 2.12 wstawić polecenie

```
if (Request.UserHostAddress == "127.0.0.1")
➥System.Threading.Thread.Sleep(3000);
```

usypiające bieżący wątek aplikacji na trzy sekundy. Po przekonaniu się, że komunikat o odświeżeniu rzeczywiście się pokazuje, możemy to polecenie usunąć lub „zakomentować".

Kliknięcie „zewnętrznego przycisku" (`Button3`), pomimo że prowadzi do częściowej aktualizacji, nie powoduje pojawienia się informującego o niej komunikatu. Komunikat nie zostanie również pokazany po kliknięciu przycisku `Button2` i pełnego przeładowania strony.



Rysunek 2.13. *Informacja o trwającym częściowym odświeżaniu strony*

ĆWICZENIE

2.21 Cykliczne wyzwalanie aktualizacji

1. A teraz dodajmy do panelu `UpdatePanel1` komponent `Timer`, również z zakładki *AJAX Extensions*. Jeżeli tylko jego właściwość `Enabled` ustawiona jest na `true`, to co tyle milisekund, na ile wskazuje właściwość `Interval`, będzie wymuszał odświeżanie strony. A ponieważ umieszczony jest w `UpdatePanel`, to aktualizacja ograniczona będzie tylko do niego. W efekcie zobaczymy na `Label1` stale aktualizowaną datę i czas. W naszym przypadku wystarczy aktualizacja co sekundę, a więc właściwość `Interval`

powinna być ustawiona na `1000`. Do tego należy oczywiście wyłączyć instrukcję opóźniającą załadowanie strony w metodzie `Page_Load`.

Powinniśmy sobie jednak zdawać sprawę z niebezpieczeństwa tego rozwiązania. Co sekundę strona będzie wysyłała żądanie do aplikacji ASP.NET, ta będzie wykonywała wszystkie metody związane z odświeżeniem strony, między innymi `Page_Load`, i nawet jeżeli aktualizowany będzie tylko drobny fragment strony, to efekt może nie być warty wywołanego w ten sposób transferu i obciążenia serwera. Szczególnie że prosty zegarek można zrealizować za pomocą JavaScriptu wyłącznie po stronie przeglądarki. Właśnie dlatego domyślna wartość właściwości `Interval` to aż sześćdziesiąt tysięcy milisekund, a więc minuta.

2. Aby móc kontrolować działanie timera, umieśćmy na stronie pole opcji `CheckBox`, zmieńmy jego właściwość `AutoPostBack` na `true` i kliknijmy je dwukrotnie, tworząc dzięki temu metodę zdarzeniową, wywoływaną w momencie zmiany stanu pola opcji. Umieśćmy w niej polecenie kontrolujące właściwość `Enabled` timera: `Timer1.Enabled = CheckBox1.Checked;`.

ĆWICZENIE

2.22 Aktualizacja warunkowa

1. Umieśćmy na stronie drugi `UpdatePanel`, a w nim jeszcze jeden komponent `Label` i przycisk `Button`.
2. Do metody `Page_Load` dodajmy kolejne polecenie umieszczające na etykiecie nowego komponentu `Label3` aktualną datę. Z nowym komponentem `UpdatePanel` możemy związać osobny komponent `UpdateProgress` z odpowiednim komunikatem w środku.
3. Uruchommy aplikację i, klikając przycisk w nowym panelu, wymuśmy jego aktualizację. Zobaczymy, że zaktualizowane zostały obydwa komponenty `UpdatePanel` (pomimo że pojawił się komunikat tylko z tego `UpdateProgress`, z którym związany jest panel z kliknięty przyciskiem).

   Taka bezwarunkowa aktualizacja nie zawsze jest wskazana. Spróbujmy zatem tak skonfigurować panele, aby drugi był aktualizowany niezależnie od pierwszego, ale żeby aktualizacja pierwszego dokonywana była zawsze, także wtedy gdy kliknięty zostanie przycisk w drugim panelu.

4. Okazuje się, że wystarczy przełączyć poznaną już wcześniej właściwość `UpdateMode` kontrolki `UpdatePanel1` na `Conditional` i uzyskamy pożądane działanie. Właściwość `UpdateMode` drugiej pozostawiamy ustawioną na `Always`.

AJAX Control Toolkit

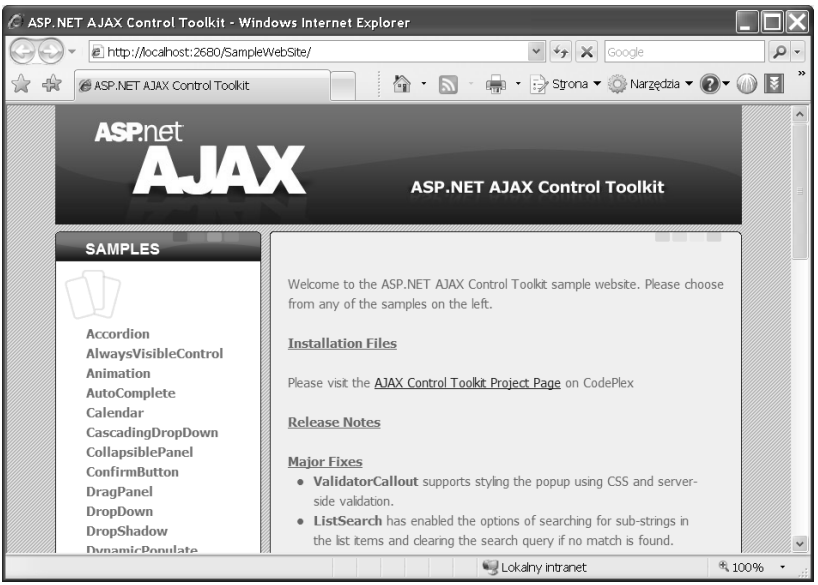
Częściowe odświeżanie strony to miła rzecz, ale naprawdę dobra zabawa rozpoczyna się dopiero po zainstalowaniu AJAX Control Toolkit (ACT). Pakiet ten można pobrać ze strony *http://www.codeplex.com/AjaxCon ➥trolToolkit*[2]. Na tej stronie, na zakładce *Release* dostępne są dwie wersje pakietu. Nas interesuje wersja 3.0.x przeznaczona dla Visual Studio 2008 (w tym dla VWD). Do pobrania są dwa pliki, ja proponuję pobrać większy (niecałe 3 MB) o nazwie *AjaxControlToolkit-Framework3.5.zip*, czyli ten zawierający dodatkowo kody źródłowe. Pobrany plik ZIP należy rozpakować, np. do katalogu *c:\Program Files\Microsoft Visual Studio 9.0\ACT*.

Przegląd ACT

ACT to zbiór komponentów ASP.NET korzystających z AJAX, które potrafią doskonale urozmaicić stronę WWW. Aby lepiej zrozumieć, z jakim typem kontrolek mamy do czynienia, rozpocznijmy naszą przygodę z ACT od wczytania do Visual Web Developera strony ASP.NET znajdującej się w katalogu *c:\Program Files\Microsoft Visual Studio 9.0\ ACT\SampleWebSite* (można też uruchomić ją w IIS — zob. rozdział 5.).

Zobaczymy stronę z rysunku 2.14. W menu z lewej strony wymienione są na niej komponenty i projekty, które możemy podziwiać. Pierwszy to `Accordion` (akordeon). Jest to w pewnym sensie rozwinięcie idei komponentu `MultiView`. Jednak prezentuje się znacznie lepiej. Nagłówki wszystkich paneli są widoczne jednocześnie, ale tylko jeden jest rozwinięty i widać jego zawartość. Przełączanie oglądanych paneli wiąże się z miłą dla oka animacją. Kontrolka `SlideShow` to obraz z cyklicznie zmieniającą się zawartością. Pośród wielu komponentów warto zwrócić także uwagę na `AlwaysVisibleControl` (zawsze widoczna kontrolka) — jest to komponent, który pozostaje zawsze ponad innymi elementami

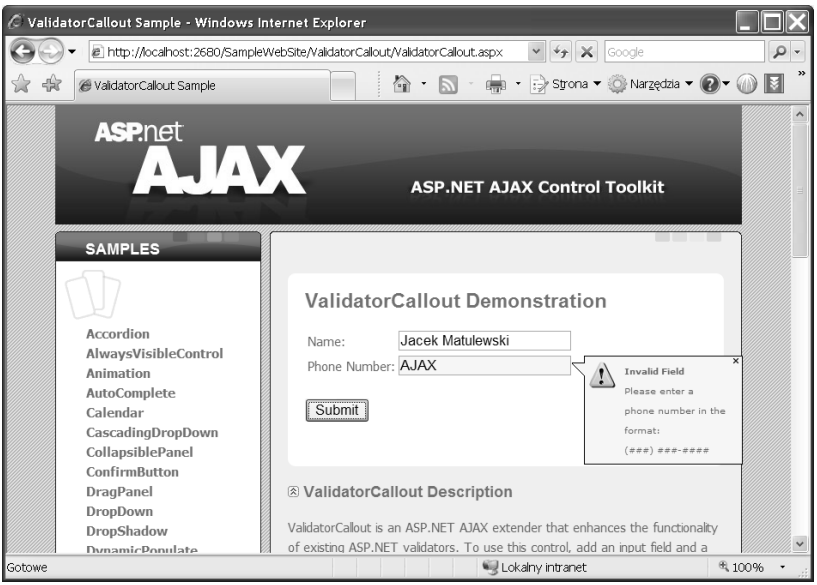
[2] *CodePlex.com* to strona Microsoft zawierająca projekty OpenSource.



Rysunek 2.14. *Strona prezentująca kontrolki ACT*

strony. Doskonały np. do przygotowywania reklam. Podobny komponent prezentowany jest w pozycji `Animation`. Innym rodzajem pseudookna jest `ModalPopup`. W momencie jego wywołania pozostała część strony staje się przygaszona i nie ma do niej dostępu (wygląd tła można zresztą w pełni kontrolować, przypisując własny styl CSS do własności `BackgroundCssClass` kontrolki). Z kolei rozmiar kontrolki `ResizableControl` może być kontrolowany myszą, jak w przypadku zwykłych okien. Są jeszcze suwaki, których brakuje w bibliotece *Web Forms* (kontrolka `Slider`), panel z zakładkami (`Tabs`) i `ValidatorCallout` — kontrolka analogiczna do komponentu `ValidationSummary`, tyle że w jej przypadku informacje o błędach pokazywane są w atrakcyjnych dymkach (rysunek 2.15). Jest też komponent `ConfirmButton`, który wyświetla prosty alert JavaScript z pytaniem o potwierdzenie wykonania akcji wywołanej kliknięciem przycisku.

Składniki biblioteki ACT dzielą się na samodzielne kontrolki i rozszerzenia kontrolek. Samodzielny jest np. akordeon. Natomiast `Always` ↪`VisibleControl`, `Slider` czy `ConfirmButton` są jedynie rozszerzeniami (w nazwie ich klas na końcu dodane jest „Extender”). Rozszerzenia zmieniają lub wzbogacają możliwości zwykłych komponentów. Dla



Rysunek 2.15. *Walidacja z atrakcyjnym komunikatem*

przykładu klasa `ConfirmButtonExtender` związana z przyciskiem `Button` powoduje, że po jego kliknięciu pojawia się pytanie z prośbą o potwierdzenie. Klasa `SliderExtender` związana z kontrolką `TextBox` powoduje, że ta ostatnia zastąpiona jest na stronie przez suwak. `AlwaysVisible` ↪`ControlExtender` sprawia z kolei, że wskazany w tej klasie komponent jest zawsze widoczny i nie zmienia pozycji mimo przewijania strony.

Używanie kontrolek ACT we własnych projektach

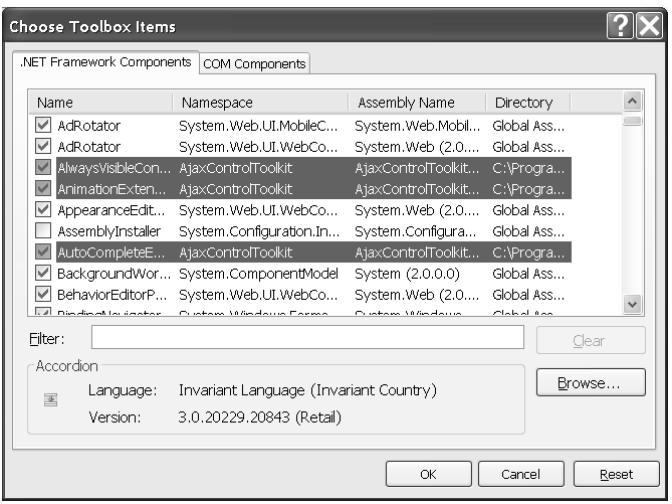
ĆWICZENIE

2.23 Instalacja kontrolek ACT w środowisku VWD

W katalogu witryny *SampleWebSite*, którą przed chwilą przeglądaliśmy, znajdziemy podkatalog *Bin* z zarządzaną biblioteką DLL. Znajdują się w niej podziwiane przed chwilą kontrolki i rozszerzenia. Możemy kontrolki z tej biblioteki zainstalować na zakładkach podokna *Toolbox*. W tym celu:

1. Zamykamy bieżący projekt, wybierając z menu *File* polecenie *Close project*.

2. W menu kontekstowego podokna *Toolbox* wybieramy polecenie *Add Tab*, dzięki czemu dodamy nową zakładkę — nazwijmy ją *AJAX Control Toolkit* lub po prostu *ACT*.
3. Następnie z menu kontekstowego nowej zakładki wybieramy polecenie *Choose Items…*.
4. Pojawi się okno z listą zainstalowanych komponentów (rysunek 2.16), w którym klikamy przycisk *Browse*.



Rysunek 2.16. *Instalowanie kontrolek ACT w środowisku VWD*

5. Za pomocą typowego okna dialogowego odnajdujemy bibliotekę *AjaxControlToolkit.dll* z katalogu *SampleWebSite/Bin* umieszczonego w miejscu, w którym rozpakowaliśmy pakiet ACT.
6. Następnie klikamy przycisk *OK* w oknie *Choose Toolbox Item*.

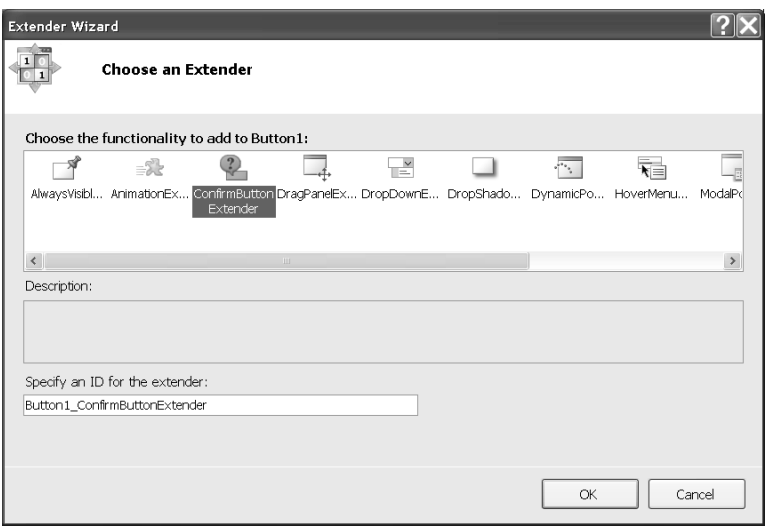
Do listy komponentów zostaną dodane nowe kontrolki, które poznaliśmy w przykładowej aplikacji prezentującej ACT. Wprawdzie nie zobaczymy ich od razu po zamknięciu okna dodawania kontrolek, ale pojawią się, gdy wczytamy projekt witryny.

ĆWICZENIE

2.24 Użycie rozszerzenia ConfirmButtonExtender

Spróbujmy teraz uruchomić kontrolkę `ConfirmButton` we własnym projekcie:

1. Stwórzmy nowy projekt witryny.
2. Na stronie *Default.aspx* w widoku projektowania umieszczamy komponent `ScriptManager` z zakładki *AJAX Extensions* oraz zwykły przycisk `Button` z zakładki *Standard*.
3. Po instalacji ACT w podręcznej liście zadań przycisku pojawiło się polecenie *Add Extender*. Po jego wybraniu zobaczymy listę rozszerzeń, o jakie można ten przycisk wzbogacić. Zaznaczmy wśród nich `ConfirmButtonExtender` i kliknijmy *OK* (rysunek 2.17).



Rysunek 2.17. *Wiązanie rozszerzeń ACT z kontrolkami Web Forms*



Po tej czynności do katalogu projektu widocznego w *Solution Explorer* dodany został katalog *Bin* zawierający zainstalowaną przed chwilą bibliotekę.

4. Nowy komponent (rozszerzenie) nie jest w żaden sposób reprezentowany w widoku projektowania. Dlatego, aby zmienić jego własności, musimy pofatygować się do edytora kodu pliku *Default.aspx* (przycisk *Source* na dole okna VWD). Znajdziemy

tam znacznik `cc1:ConfirmButtonExtender`, a w nim atrybut `ConfirmText`, którego wartość zmieniamy na pytanie w stylu: „Czy jesteś pewien?" (listing 2.13).

Listing 2.13. *Kod elementu ConfirmButtonExtender*

```
<cc1:ConfirmButtonExtender ID="Button1_ConfirmButtonExtender"
➥runat="server"
   ConfirmText="Czy jesteś pewien?" Enabled="True"
   ➥TargetControlID="Button1">
</cc1:ConfirmButtonExtender>
```



Jeżeli w trakcie edycji kodu na zakładce *Source* kursor edycji znajdzie się w obrębie znacznika `cc1:ConfirmButtonExtender`, w podoknie *Properties* pojawia się lista jego atrybutów obejmująca też `ConfirmText` — to pozwoli skonfigurować rozszerzenie osobom uzależnionym od korzystania z okna własności.

5. W tym prostym przykładzie niech kliknięcie przycisku zmieni jego etykietę. Musimy więc utworzyć odpowiednią metodę zdarzeniową, klikając przycisk dwukrotnie w widoku projektowania. Prezentuje ją listing 2.14.

Listing 2.14. *Dla rozszerzenia ConfirmButtonExtender nie ma zupełnie znaczenia, co ma się stać po kliknięciu przycisku — to wydarzy się dopiero po stronie serwera, a on pracuje przed wysłaniem żądania*

```
protected void Button1_Click(object sender, EventArgs e)
{
    Button1.Text = DateTime.Now.ToString();
}
```

Nie pozostaje nam nic innego, jak uruchomić projekt i kliknąć przycisk. Pojawi się okno potwierdzenia (rysunek 2.18). Jeżeli klikniemy *OK*, nastąpi przeładowanie strony, a wówczas metoda z listingu 2.14 zostanie wykonana i etykieta przycisku się zmieni. Jeżeli natomiast klikniemy *Anuluj* — nie stanie się nic. To oznacza, że jeżeli internauta rozmyśli się już po kliknięciu przycisku, unikniemy niepotrzebnego postbacku.

Możemy teraz tę stronę jeszcze bardziej „uajaksować". Proponuję Czytelnikowi dodanie komponentu `UpdatePanel` i przeniesienie do niego przycisku. Dzięki temu przeładowanie strony w momencie kliknięcia nie będzie się tak rzucać w oczy.



Rysunek 2.18. *Efekt użycia komponentu ConfirmButtonExtender*

Jak to jest zrobione?

Jeżeli zajrzymy do plików projektu ACT, a więc do katalogu *AjaxControlToolkit* znajdującego się w katalogu, do którego zainstalowaliśmy ACT, to znajdziemy tam plik rozwiązania (ang. *solution*) projektu całej biblioteki. Możemy go wczytać do pełnej wersji Visual Studio i samodzielnie rozwijać. W interesującym nas w tej chwili najbardziej podkatalogu *ConfirmButton* znajdziemy trzy pliki, które można obejrzeć nawet bez Visual Studio. Zasadniczy z nich to *confirmButtonExtender.cs* zawierający kod źródłowy klasy `ConfirmButtonExtender`. To tu znajdziemy definicję własności `ConfirmText`. Nie znajdziemy jednak w tym pliku kodu C#, który byłby odpowiedzialny za wyświetlenie okna dialogowego. Nic dziwnego, w końcu nie ma to być zwykły komponent wykonywany po stronie serwera. Dlatego większość kodu napisana jest w JavaScripcie i umieszczona w pliku *confirmButtonBehavior.js*. Znajdziemy w nim między innymi funkcję JavaScript `_onClick`. Po jej krótkiej analizie okaże się, że po spełnieniu kilku warunków wywołuje ona funkcję pomocniczą o nazwie `_displayConfirmDialog`, która do wyświetlenia okna dialogowego korzysta z metody JavaScript `confirm`, należącej do predefiniowanego obiektu `window` reprezentującego bieżące okno przeglądarki. Metoda ta powoduje wyświetlenie okna dialogowego z dwoma przyciskami *OK* i *Anuluj* oraz komunikatem wskazanym przez argument tej metody (rysunek 2.17). W zależności od kliknięty przycisku zwracana przez metodę wartość to prawda lub fałsz. Jeżeli klikniemy *OK*, funkcja nie wykona żadnego zadania — po prostu zakończy pracę, a po jej zakończeniu nastąpi wysłanie żądania i przeładowanie strony. Natomiast jeżeli klikniemy *Anuluj*, wykonana zostanie

metoda `preventDefault` argumentu, którym jest obiekt zdarzenia. Wywołanie tej metody powoduje anulowanie zdarzenia, w efekcie czego kliknięcie przycisku nie pociągnie za sobą żadnych widocznych skutków.

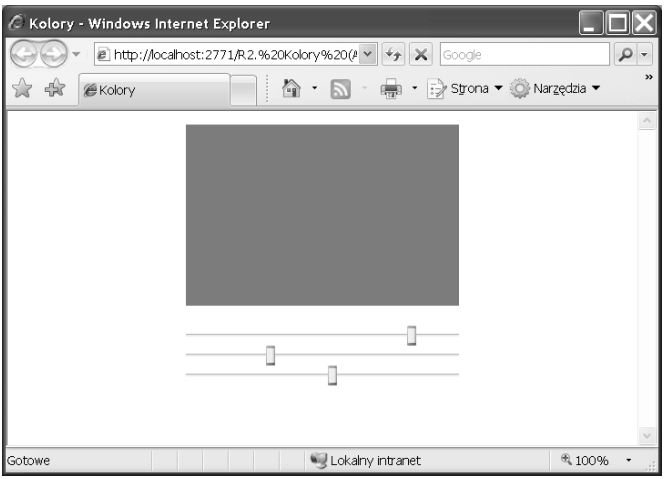
ĆWICZENIE

2.25 Suwaki

Aby pokazać, jak kontrolki ACT mogą uczynić nasze aplikacje przyjaźniejszymi i łatwiejszymi w użyciu, zmodyfikujmy projekt *Kolory* opisany w pierwszym rozdziale. Wyposażmy ją w trzy komponenty `Slider` pozwalające na kontrolę kolor panelu w bardziej ergonomiczny sposób.

1. Wczytajmy projekt *Kolory*.
2. Do strony *Default.aspx* dodajmy `ScripManager` i `UpdatePanel`. Przenieśmy całą dotychczasową zawartość strony (panel i trzy pola tekstowe) do `UpdatePanel1`.
3. Następnie z każdym z trzech pól tekstowych, korzystając z polecenia *Add Extender* w bocznej liście zadań, zwiążmy komponenty `SliderExtension`.
4. Przejdźmy do kodu źródłowego i, zaznaczając poszczególne komponenty rozszerzeń `cc1:SliderExtender`, zmieńmy ich właściwości `Maximum` na `255` i `Length` na `300`. Pierwsza wyznacza zakres wartości zwracanych przez suwak i przypisywanych właściwości `Text` związanego z nim pola `TextBox`, a druga określa długość suwaka.
5. Właściwość `Orientation` możemy ustawić na `Vertical`, jeżeli chcemy, żeby suwaki były ustawione pionowo.
6. Kompilujemy i uruchamiamy projekt (*F5*).

Dzięki powyższym zmianom strona wygląda znacznie lepiej — pola tekstowe zostały zastąpione przez suwaki, dzięki którym kontrolowanie koloru panelu jest znacznie wygodniejsze (rysunek 2.19). Niestety suwaki tworzone są z pewnym opóźnieniem, już po załadowaniu strony i pokazaniu pól tekstowych. Przez to po każdej zmianie pozycji suwaka i w konsekwencji częściowym przeładowaniu strony na chwilę pojawiają się pola tekstowe. To nieco psuje cały efekt.



Rysunek 2.19. *Aplikacja Kolory po „uajaksowieniu”*



Oryginalne kontrolki `TextBox` stały się niewidoczne. Jeżeli poza suwakami chcielibyśmy zobaczyć również bieżące wartości, możemy wykorzystać inny komponent i związać go z suwakami poprzez ich właściwość `BoundControlID`.

ĆWICZENIE

2.26 Reklama

W drugim przykładzie wykorzystamy komponent rozszerzenia `AlwaysVisibleControlExtender` do stworzenia szablonu reklamy pokazywanej nad właściwą zawartością strony. Bez względu na przewijanie strony reklama zawsze będzie w centrum okna przeglądarki.

1. Tworzymy projekt nowej witryny. Jego stronę *Default.aspx* wyposażamy w `ScriptManager`.
2. Dodajemy do niej komponent `Panel` z zakładki *Standard*. Zmieńmy jego szerokość na 200 pikseli.
3. W jego wnętrzu umieszczamy komponenty, które będą pasowały do naszej reklamy. Zwykle jest to jakiś rysunek — ja użyłem logo Helionu, wskazując we własności `ImageUrl` komponentu `HyperLink` adres *http://helion.pl/img/logo162_35.gif*.
4. Minimum przyzwoitości wymaga, aby na panelu znalazł się element, którego kliknięcie ukryje reklamę, tj. przestawi

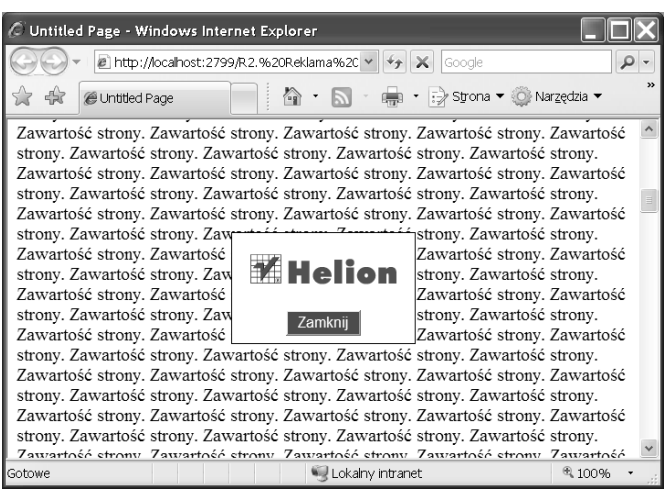
właściwość `Visible` panelu i właściwość `Enabled` rozszerzenia na `False`. Ja użyłem zwykłego przycisku, ale może to być obraz z typowym „x”.

5. Następnie z panelem wiążemy rozszerzenie `AlwaysVisibleControlExtender`.
6. Przechodzimy do edytora kodu i ustawiamy właściwości rozszerzenia: `VerticalSide` na `Middle` i `HorizontalSide` na `Center` — tak, aby reklama była w centrum i dzięki temu stała się jak najbardziej widoczna (i denerwująca). Kod elementu rozszerzenia widoczny jest na listingu 2.15.

Listing 2.15. *Element kontrolki AlwaysVisibleControlExtender1*

```
<cc1:AlwaysVisibleControlExtender
➥ID="Panel1_AlwaysVisibleControlExtender"
   runat="server" Enabled="True" HorizontalSide="Center"
   ➥TargetControlID="Panel1"
   VerticalSide="Middle">
</cc1:AlwaysVisibleControlExtender>
```

I w ten sposób uzyskaliśmy najprostszą, ale już irytującą reklamę (rysunek 2.20). Przy użytych ustawieniach (punkt 6.) okno reklamy będzie zawsze pozostawać w środku bez względu na zmiany rozmiaru okna przeglądarki i przewijanie tekstu znajdującego się pod spodem.


Untitled Page - Windows Internet Explorer
http://localhost:2799/R2.%20Reklama%2C
Google
Untitled Page
Strona
Narzędzia
Zawartość strony. Zawartość strony. Zawartość strony.