Az ASP.NET weboldal-kódolási modellje

Az ASP.NET-weboldalakat kétféle módon készíthetjük el. Létrehozhatunk egyetlen `*.aspx` fájlt, amely a kiszolgálóoldali kód és a HTML keveréke (hasonlóan a klasszikus ASP-hez). Az egyfájlosoldal-modell segítségével a kiszolgálóoldali kódot a `<script>` hatókörébe helyezzük, de maga a kód valójában *nem* szkriptkód (például VBScript/JScript). A `<script>` blokkban található utasításokat e helyett valamilyen .NET-nyelven írjuk (C#, Visual Basic stb.).

Ha nagyon kevés kódot (de sok HTML-t) tartalmazó oldalt készítünk, az egyfájlos oldal modell használata kényelmesebb, mivel a kódot és a markupot egy `*.aspx` fájlban láthatjuk. Emellett, ha az forráskódot és a HTML-markupot egyetlen `*.aspx` fájlba helyezzük, további előnyökre tehetünk szert:

- Az egyfájlos modellel írt oldalakat valamelyest egyszerűbben lehet telepíteni, és másik fejlesztőnek elküldeni.
- Mivel a fájlok között nincs függőség, az egyfájlos oldal átnevezése könnyebb.
- A fájlokat egyszerűbben lehet kezelni egy verziókezelő rendszerben, mivel az összes művelet egyetlen fájlban történik.

A Visual Studio 2008 alapértelmezés szerint a mögöttes kódként ismert módszert alkalmazza új webhelymegoldások létrehozásakor, amely révén a forráskódot különválaszthatjuk a HTML megjelenítési logikától, két külön fájl használatával. Ezt a modellt akkor célszerű használni, amikor az oldalaink nagy mennyiségű kódot tartalmaznak, vagy több fejlesztő dolgozik ugyanazon a webhelyen. A mögötteskód-modellnek szintén több előnye van:

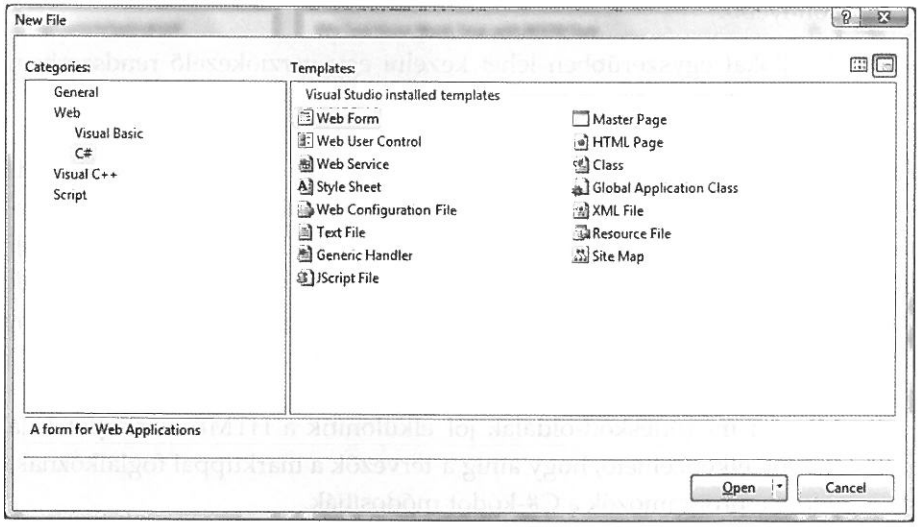
- Mivel a mögötteskód-oldalak jól elkülönítik a HTML-markupot és a -kódot, elképzelhető, hogy amíg a tervezők a markuppal foglalkoznak, addig a programozók a C#-kódot módosítják.
- Az oldaltervezők vagy akik csak az oldal markupjával foglalkoznak nem férhetnek hozzá a kódokhoz (ahogy azt gondolhattuk, a HTML-tervezők nem mindig szeretnének a C#-kódban gyönyörködni).
- A kódfájlokat több `*.aspx` fájlban is használhatjuk.

Függetlenül attól, hogy melyik módszert választjuk, jó, ha tudjuk, hogy teljesítmény szempontjából *nincs* közöttük különbség. Valójában sok ASP.NET-webalkalmazás előnyére válik, ha olyan oldalakat készítünk, amelyek mindkét módszert alkalmazzák.

Adatközpontú egyfájlos tesztoldal készítése

Először is, vizsgáljuk meg az egyfájlos oldal modellt. A célunk, hogy olyan `*.aspx` fájlt készítsünk, amely a 22. fejezetben létrehozott AutoLot adatbázis Inventory tábláját jeleníti meg. Amíg ezt az oldalt kizárólag Jegyzettömb segítségével is elkészíthetnénk, a Visual Studio 2008 egyszerűsítheti a dolgokat az IntelliSense, a kód kiegészítés és egy vizuális laptervező segítségével.

Kezdésként nyissuk meg a Visual Studio 2008-at, és hozzunk létre új Web Formot a File >- New>- File menüpont segítségével (lásd a 31.9. ábrát). Ha ezzel elkészültünk, mentsük a fájlt Default.aspx néven a merevlemezre, egy új C:\CodeTests\SinglePageModel könyvtárba.



31.9. ábra: *Új *.aspx fájl létrehozása*

Az AutoLotDAL.dll fájl manuális hivatkozása

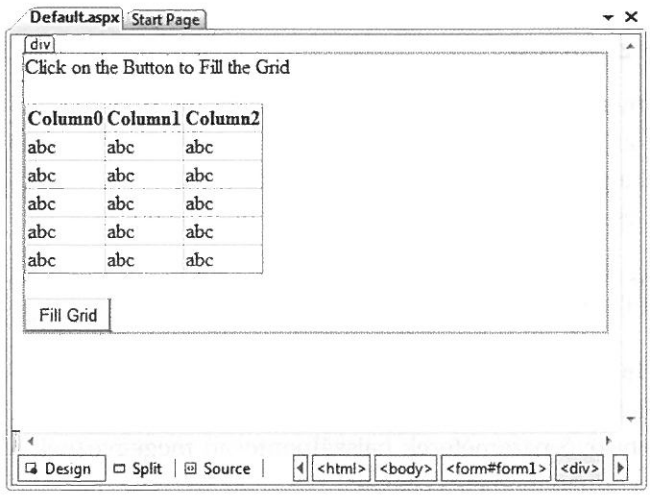
Most hozzunk létre egy „bin" nevű alkönyvtárat a SinglePageModel mappában, a Windows Intéző segítségével. A speciális nevű bin alkönyvtár egy regisztrált név az ASP.NET-futtatómotorban. A webhelyek gyökerének \bin

mappájába a webalkalmazás bármilyen privát szerelvényét telepíthetjük. Ebben a példában az `AutoLotDAL.dll` (lásd a 22. fejezetet) másolatát helyezzük a C:\CodeTests\SinglePageModel\bin mappába.

Megjegyzés Ahogy az a fejezet későbbi részéből kiderül, amikor a Visual Studio 2008 segítségével hozunk létre teljes ASP.NET-webalkalmazást, a \bin mappát az IDE tartja karban helyettünk.

A felhasználói felület tervezése

Válasszuk a Visual Studio 2008 Toolbox eszköztárának Standard fülét, és húzzunk át egy Button, egy Label és egy GridView vezérlőelemet az oldaltervezőbe (a GridView a Toolbox Data fülén található). Nyugodtan állítsuk be a Properties ablak (vagy a HTML IntelliSense) segítségével a különböző felhasználóifelület-tulajdonságokat, és az ID tulajdonság révén minden vezérlőnek adjunk megfelelő nevet. A 31.10. ábra egy lehetséges megoldást ábrázol. (A példa megjelenése és működése szándékosan nem túl bonyolult, hogy a lehető legkevesebb vezérlőelem-markupot kelljen generálni, de nyugodtan alakítsuk át ízlésünk szerint.)



31.10. ábra: *A Default.aspx felhasználói felülete*

Most keressük meg az oldal `<form>` szakaszát. Figyeljük meg, hogyan definiáltuk az egyes webes vezérlőelemeket egy `<asp:>` címke segítségével. A lezáró címke előtt egy név/érték pár sorozatot találunk, amely a Properties ablak beállításaival áll összhangban:

```
<form id="form1" runat="server">
<div>
  <asp:Label ID="lblInfo" runat="server"
       Text="Click on the Button to Fill the Grid">
  </asp:Label>
  <br />
  <br />
  <asp:GridView ID="carsGridView" runat="server">
  </asp:GridView>
  <br />
  <asp:Button ID="btnFillData" runat="server" Text="Fill Grid" />
</div>
</form>
```

Az ASP.NET webes vezérlőelemeit a 32. fejezetben fogjuk részletesen tanulmányozni. Annyit már most tudnunk kell, hogy a webes vezérlőelemek a webkiszolgálón feldolgozott objektumok, amelyek a HTML-ábrázolásukat automatikusan visszaadják a kimenő HTTP-válaszban (vagyis a HTML-t nem mi programozzuk!). Ezen a nagyszerű előnyön túl, az ASP.NET webes vezérlőelemei az asztali programozási modellt utánozzák, mivel a tulajdonságok, a metódusok és az események elnevezései rendre a Windows Forms/WPF megfelelőikre hasonlítanak.

Adatelérési logika hozzáadása

Kezeljük a kattintási eseményét a gombnak vagy a Visual Studio Properties ablakának „villám" ikonjával, vagy a tervezőablak felső részén található legördülő listák segítségével. Ezzel a gomb definícióját kiegészítettük egy `OnClick` attribútummal, amelyet a `Click` esemény kezelőjének nevéhez rendeltünk hozzá:

```
<asp:Button ID="btnFillData" runat="server"
     Text="Fill Grid" OnClick="btnFillData_Click"/>
```

Emellett egy üres `<script>` blokkot is kapunk, amellyel elkészíthetjük a kiszolgálóoldali kattintási esemény kezelőjét. Adjuk hozzá a következő kódot, és figyeljük meg, hogy bejövő paraméterek hajszálpontosan megegyeznek a `System.EventHandler` metódusreferencia céljával:

```
<script runat="server">
  void btnFillData_Click(object sender, EventArgs args)
  {
  }
</script>
```

A következő lépés a GridView feltöltése az AutoLotDAL.dll szerelvény segítségével. Ehhez határozzuk meg a <%@ Import %> direktíva segítségével, hogy az AutoLotConnectedLayer névteret használjuk. Továbbá értesítenünk kell az ASP.NET-futtatókörnyezetet, hogy ez az egyfájlos oldal az AutoLotDAL.dll szerelvényre hivatkozik az <%@ Assembly %> direktíva segítségével (hamarosan többet megtudhatunk a direktívákról). Íme, a Default.aspx fájl megmaradt, releváns logikája (a kapcsolatsztringet szükség szerint módosítanunk kell):

```
<%@ Page Language="C#" %>
<%@ Import Namespace = "AutoLotConnectedLayer" %>
<%@ Assembly Name ="AutoLotDAL" %>

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

<script runat="server">
  void btnFillData_Click(object sender, EventArgs args)
  {
    InventoryDAL dal = new InventoryDAL();
    dal.OpenConnection(@"Data Source=(local)\SQLEXPRESS;" +
        "Initial Catalog=AutoLot;Integrated Security=True");
    carsGridView.DataSource = dal.GetAllInventory();
    carsGridView.DataBind();
    dal.CloseConnection();
  }
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
...
</html>
```

Mielőtt elmélyülnénk az *.aspx fájl formátumának részleteiben, teszteljük az oldalt. Először mentsük el az *.aspx fájlt. Ha manuálisan szeretnénk használni a WebDev.WebServer.exe-t, nyissunk meg egy .NET-parancssort, és futtassuk a WebDev.WebServer.exe segédprogramot. Győződjünk meg arról, hogy helyesen adtuk meg a tárolt Default.aspx fájl elérési útját, például a következőképpen (jelen esetben egy tetszőleges 12345 portot adtunk meg):

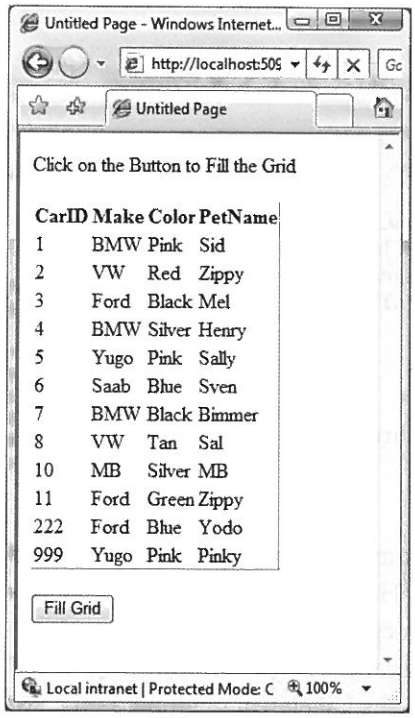
```
webdev.webserver.exe
      /port:12345 /path:"C:\CodeTests\SinglePageModel"
```

Most írjuk be a böngészőnkbe a következő URL-t:

```
http://localhost:12345/
```

Amikor az oldal betöltődik, kezdetben egy címkét és egy gombot látunk. Azonban, amikor a gombra kattintunk, adatokat küldünk vissza a webkiszolgálóra, és a webes vezérlőelemek ekkor a megfelelő HTML-címkéiket jelenítik meg.

A `WebDev.WebServer.exe` segédprogramot közvetett módon a Visual Studio 2008-ból is elindíthatjuk. Csak kattintsunk a jobb gombbal arra az oldalra, amelyet szeretnénk megtekinteni, majd válasszuk a View In Browser menüpontot. A 31.11. ábrán mindkét eset kimenetét láthatjuk, miután a Fill Grid gombra kattintottunk.



31.11. ábra: *Webalapú adatelérés*

Ez egyszerű volt, nem? Bár, ahogy mondják, az ördög a részletekben rejlik, ezért ássuk bele jobban magunkat az `*.aspx` fájl felépítésébe. Kezdjük az *oldaldirektívák* szerepének vizsgálatával.

Az ASP.NET-direktívák szerepe

Az első dolog, amellyel tisztában kell lennünk, hogy az `*.aspx` fájlok általában számos direktívával kezdődnek. Az ASP.NET-direktívákra a `<%@ ... %>` jelölők utalnak, és különböző attribútumok minősíthetik, amelyek utasítják az ASP.NET-futtatókörnyezetet, hogyan kell az adott attribútumot feldolgoznia.

Minden `*.aspx` fájlnak legalább egy `<%@Page%>` direktívát kell tartalmaznia, amely definiálja az oldalon alkalmazott felügyelt nyelvet (a `language` attribútum révén). A `<%@Page%>` direktíva emellett a kapcsolódó mögöttes kódfájl nevét is definiálhatja (ha létezik), engedélyezheti a nyomkövetés támogatást, és így tovább. A 31.2. táblázat a legfontosabb `<%@Page%>` központú attribútumok közül mutat be néhányat.

Attribútum	Jelentés
`CodePage`	Ez az attribútum a kapcsolódó mögöttes kódfájl nevét határozza meg.
`CompilerOptions`	Ennek az attribútumnak a segítségével definiálhatjuk a parancssori kapcsolókat (egyetlen sztring formájában jelennek meg), amelyeket az oldal feldolgozása során a fordítónak átadunk.
`EnableTheming`	Ez az attribútum megadja, hogy az `*.aspx` oldal vezérlőelemei támogatják-e az ASP.NET-témákat.
`EnableViewState`	Ez az attribútum jelzi, hogy a nézet állapotot (view state-et) a rendszer az oldalkérések között megőrzi (a tulajdonságról a 33. fejezetben találhatunk további részleteket).
`Inherits`	Ez az attribútum egy osztályt definiál a mögöttes kódban található oldal részére, amelyből az `*.aspx` fájl származik. Ez bármilyen `System.Web.UI.Page`-ből származtatott osztály lehet.
`MasterPageFile`	Ez az attribútum az aktuális `*.aspx` oldal mesteroldalát adja meg.
`Trace`	Ez az attribútum jelzi, hogy engedélyeztük-e a nyomkövetést.

31.2. táblázat: *A <%@Page%> direktíva néhány attribútuma*

Egy adott *.aspx fájl a <%@Page%> direktíván kívül különféle <%@Import%> direktívákat is megadhat az aktuális oldal és a <%@Assembly%> direktívák által igényelt névterek explicit kifejezésére, hogy a webhely által használt külső kódkönyvtárakat meghatározza (amelyek általában a webhely \bin mappájában találhatóak).

Ebben a példában meghatároztuk, hogy az AutoLotDAL.dll szerelvény AutoLotConnectedLayer névterének típusait használjuk. Ahogy rájöhettünk, ha további .NET-névterekre van szükségünk, egyszerűen csak több <%@Import%>/<%@Assembly%> direktívát kell megadnunk.

A jelenlegi .NET-tudásunkkal elcsodálkozhatunk azon, hogy ez az *.aspx fájl hogyan kerülhette el azt, hogy további <%@Import%> direktívákat határozzunk meg benne ahhoz, hogy hozzáférjen a System névtér System.Object és System.EventHandler típusaihoz (többek között). Ez annak köszönhető, hogy az *.aspx fájlok automatikusan hozzáférnek a .NET platform telepítési útvonalán található machine.config fájlban definiált kulcsfontosságú névterek készletéhez. Ebben az XML-alapú fájlban több automatikusan importált névteret találunk:

```
<pages>
  <namespaces>
    <add namespace="System"/>
    <add namespace="System.Collections"/>
    <add namespace="System.Collections.Specialized"/>
    <add namespace="System.Configuration"/>
    <add namespace="System.Text"/>
    <add namespace="System.Text.RegularExpressions"/>
    <add namespace="System.Web"/>
    <add namespace="System.Web.Caching"/>
    <add namespace="System.Web.SessionState"/>
    <add namespace="System.Web.Security"/>
    <add namespace="System.Web.Profile"/>
    <add namespace="System.Web.UI"/>
    <add namespace="System.Web.UI.WebControls"/>
    <add namespace="System.Web.UI.WebControls.WebParts"/>
    <add namespace="System.Web.UI.HtmlControls"/>
    ...
  </namespaces>
</pages>
```

A szkriptblokk elemzése

Az egyfájlos modellben egy *.aspx fájl kiszolgálóoldali szkriptkódot tartalmazhat, amelyet a rendszer a webkiszolgálón futtat. Emiatt *kulcsfontosságú*, hogy az összes kiszolgálóoldali kódblokkot a runat="server" attribútummal definiáljuk, hogy az a kiszolgálón fusson. Ha a runat="server" attribútumot nem határozzuk meg, a futtatókörnyezet azt felételezi, hogy *ügyféloldali* szkriptblokkot írtunk, amelyet a kimenő HTTP-válaszba továbbítunk:

```
<script runat="server">
  void btnFillData_Click(object sender, EventArgs args)
  {
    InventoryDAL dal = new InventoryDAL();
    dal.OpenConnection(@"Data Source=(local)\SQLEXPRESS;" +
        "Initial Catalog=AutoLot;Integrated Security=True");
    carsGridView.DataSource = dal.GetAllInventory();
    carsGridView.DataBind();
    dal.CloseConnection();
  }
</script>
```

Ezen segédmetódus szignatúrája furcsán ismerősnek tűnhet. Emlékezzünk vissza a Windows Forms (vagy jelen esetben WPF-) tanulmányainkra, hogy egy adott vezérlőelem eseménykezelőjének egyeznie kell a kapcsolódó .NET-metódusreferenciában definiált mintával. Ugyanúgy, mint a Windows Forms esetében, ha egy kiszolgálóoldali kattintási eseményt szeretnénk kezelni, a kérdéses metódusreferencia a `System.EventHandler`, amely – ha emlékszünk rá – csak azokat a metódusokat hívja, amelyek első paramétere a `System.Object`, a második pedig a `System.EventArgs`.

Az ASP.NET vezérlőelem-deklarációinak áttekintése

A példa utolsó figyelemre méltó pontja a Button, a Label és a GridView webes vezérlőelemek deklarációja. A klasszikus ASP-hez és a nyers HTML-hez hasonlóan, az ASP.NET webes vezérlőelemei is a `<form>` elemek hatókörében találhatók. Most azonban a nyitó `<form>` elemet a `runat="server"` attribútummal jelöltük meg. Ez szintén rendkívül fontos, mivel ez a címke tájékoztatja az ASP.NET-futtatókörnyezetet arról, hogy mielőtt a HTML-t továbbítaná a válaszfolyamba, a tárolt ASP.NET-vezérlők még alakíthatják HTML-megjelenésüket:

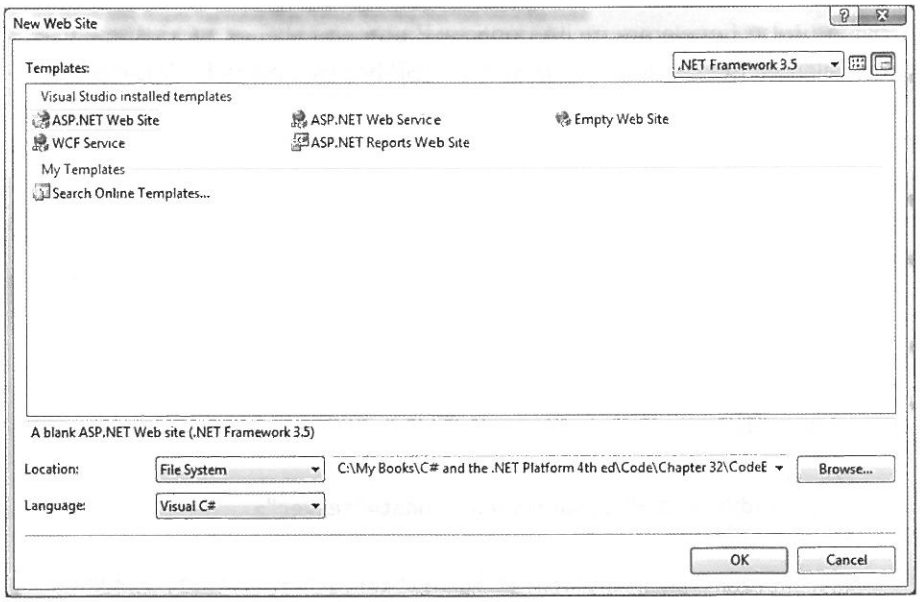
```
<form id="form1" runat="server">
<div>
  <asp:Label ID="lblInfo" runat="server"
       Text="Click on the Button to Fill the Grid">
  </asp:Label>
  <br />
  <br />
  <asp:GridView ID="carsGridView" runat="server">
  </asp:GridView>
  <br />
  <asp:Button ID="btnFillData" runat="server" Text="Fill Grid" />
</div>
</form>
```

Figyeljük meg, hogy az ASP.NET webes vezérlőelemeket <asp> és </asp> címkékkel deklaráltuk, és a runat="server" attribútummal is megjelöltük őket. A nyitó címkében a webes űrlap vezérlőelem nevét és tetszőleges számú név/érték párt adhatunk meg, amelyeket a megfelelő HTML megjelenítésére használunk futásidőben.

Forráskód A SinglePageModel kódfájlt a forráskódkönyvtár 31. fejezetének alkönyvtára tartalmazza. A forráskódkönyvtárról lásd a Bevezetés xlv. oldalát.

A mögötteskód-modell használata

A mögötteskód-modell szemléltetésére hozzuk létre újra az előző példát a Visual Studio 2008 Web Site sablonja segítségével. (Tudnunk kell, hogy a Visual Studio 2008 nem követeli meg a mögöttes kód használatát; azonban ez az új webhelyek alapértelmezett viselkedése.) Válasszuk ki a File > New > Web Site menüpontot, és a 31.12. ábrán látható módon válasszuk az ASP.NET Web Site sablont.



31.12. ábra: *A Visual Studio 2008 ASP.NET Web Site sablonja*

Az ASP.NET weboldal-kódolási modellje

Az ASP.NET-weboldalakat kétféle módon készíthetjük el. Létrehozhatunk egyetlen `*.aspx` fájlt, amely a kiszolgálóoldali kód és a HTML keveréke (hasonlóan a klasszikus ASP-hez). Az egyfájlosoldal-modell segítségével a kiszolgálóoldali kódot a `<script>` hatókörébe helyezzük, de maga a kód valójában *nem* szkriptkód (például VBScript/JScript). A `<script>` blokkban található utasításokat e helyett valamilyen .NET-nyelven írjuk (C#, Visual Basic stb.).

Ha nagyon kevés kódot (de sok HTML-t) tartalmazó oldalt készítünk, az egyfájlos oldal modell használata kényelmesebb, mivel a kódot és a markupot egy `*.aspx` fájlban láthatjuk. Emellett, ha az forráskódot és a HTML-markupot egyetlen `*.aspx` fájlba helyezzük, további előnyökre tehetünk szert:

- Az egyfájlos modellel írt oldalakat valamelyest egyszerűbben lehet telepíteni, és másik fejlesztőnek elküldeni.
- Mivel a fájlok között nincs függőség, az egyfájlos oldal átnevezése könnyebb.
- A fájlokat egyszerűbben lehet kezelni egy verziókezelő rendszerben, mivel az összes művelet egyetlen fájlban történik.

A Visual Studio 2008 alapértelmezés szerint a mögöttes kódként ismert módszert alkalmazza új webhelymegoldások létrehozásakor, amely révén a forráskódot különválaszthatjuk a HTML megjelenítési logikától, két külön fájl használatával. Ezt a modellt akkor célszerű használni, amikor az oldalaink nagy mennyiségű kódot tartalmaznak, vagy több fejlesztő dolgozik ugyanazon a webhelyen. A mögötteskód-modellnek szintén több előnye van:

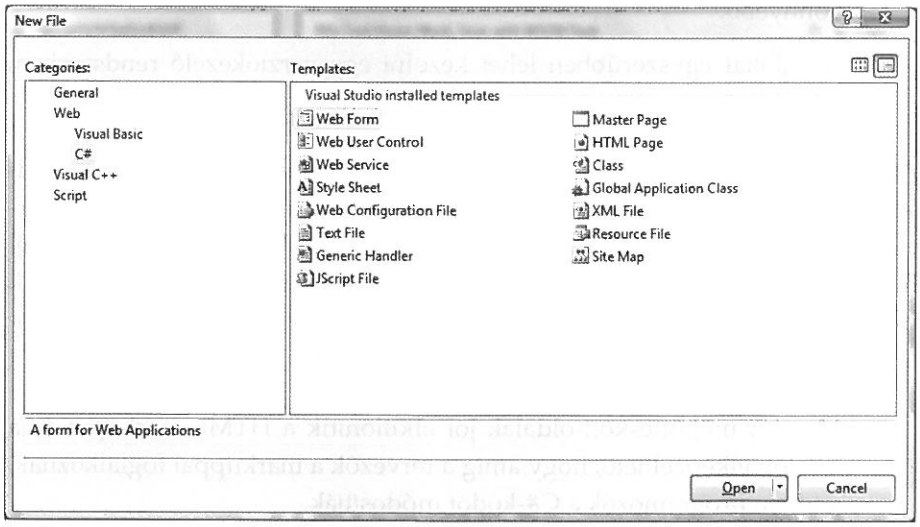
- Mivel a mögötteskód-oldalak jól elkülönítik a HTML-markupot és a -kódot, elképzelhető, hogy amíg a tervezők a markuppal foglalkoznak, addig a programozók a C#-kódot módosítják.
- Az oldaltervezők vagy akik csak az oldal markupjával foglalkoznak nem férhetnek hozzá a kódokhoz (ahogy azt gondolhattuk, a HTML-tervezők nem mindig szeretnének a C#-kódban gyönyörködni).
- A kódfájlokat több `*.aspx` fájlban is használhatjuk.

Függetlenül attól, hogy melyik módszert választjuk, jó, ha tudjuk, hogy teljesítmény szempontjából *nincs* közöttük különbség. Valójában sok ASP.NET-webalkalmazás előnyére válik, ha olyan oldalakat készítünk, amelyek mindkét módszert alkalmazzák.

Adatközpontú egyfájlos tesztoldal készítése

Először is, vizsgáljuk meg az egyfájlos oldal modellt. A célunk, hogy olyan `*.aspx` fájlt készítsünk, amely a 22. fejezetben létrehozott AutoLot adatbázis Inventory tábláját jeleníti meg. Amíg ezt az oldalt kizárólag Jegyzettömb segítségével is elkészíthetnénk, a Visual Studio 2008 egyszerűsítheti a dolgokat az IntelliSense, a kód kiegészítés és egy vizuális laptervező segítségével.

Kezdésként nyissuk meg a Visual Studio 2008-at, és hozzunk létre új Web Formot a File >- New>- File menüpont segítségével (lásd a 31.9. ábrát). Ha ezzel elkészültünk, mentsük a fájlt Default.aspx néven a merevlemezre, egy új C:\CodeTests\SinglePageModel könyvtárba.



31.9. ábra: *Új *.aspx fájl létrehozása*

Az AutoLotDAL.dll fájl manuális hivatkozása

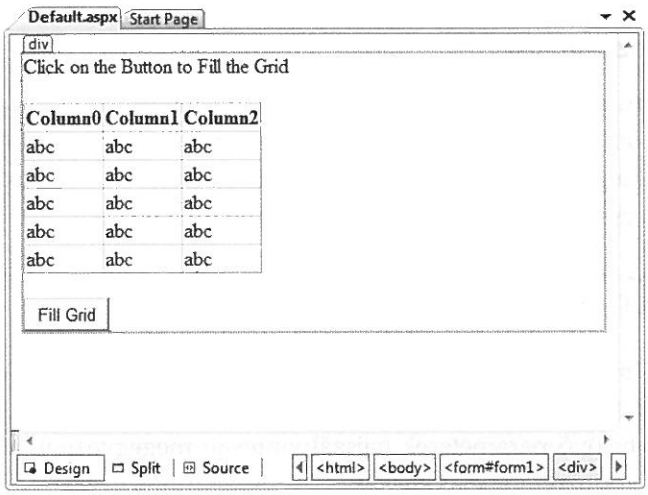
Most hozzunk létre egy „bin" nevű alkönyvtárat a SinglePageModel mappában, a Windows Intéző segítségével. A speciális nevű bin alkönyvtár egy regisztrált név az ASP.NET-futtatómotorban. A webhelyek gyökerének \bin

mappájába a webalkalmazás bármilyen privát szerelvényét telepíthetjük. Ebben a példában az AutoLotDAL.dll (lásd a 22. fejezetet) másolatát helyezzük a C:\CodeTests\SinglePageModel\bin mappába.

Megjegyzés Ahogy az a fejezet későbbi részéből kiderül, amikor a Visual Studio 2008 segítségével hozunk létre teljes ASP.NET-webalkalmazást, a \bin mappát az IDE tartja karban helyettünk.

A felhasználói felület tervezése

Válasszuk a Visual Studio 2008 Toolbox eszköztárának Standard fülét, és húzzunk át egy Button, egy Label és egy GridView vezérlőelemet az oldaltervezőbe (a GridView a Toolbox Data fülén található). Nyugodtan állítsuk be a Properties ablak (vagy a HTML IntelliSense) segítségével a különböző felhasználóifelület-tulajdonságokat, és az ID tulajdonság révén minden vezérlőnek adjunk megfelelő nevet. A 31.10. ábra egy lehetséges megoldást ábrázol. (A példa megjelenése és működése szándékosan nem túl bonyolult, hogy a lehető legkevesebb vezérlőelem-markupot kelljen generálni, de nyugodtan alakítsuk át ízlésünk szerint.)



31.10. ábra: *A Default.aspx felhasználói felülete*

Most keressük meg az oldal <form> szakaszát. Figyeljük meg, hogyan definiáltuk az egyes webes vezérlőelemeket egy <asp:> címke segítségével. A lezáró címke előtt egy név/érték pár sorozatot találunk, amely a Properties ablak beállításaival áll összhangban:

```
<form id="form1" runat="server">
<div>
  <asp:Label ID="lblInfo" runat="server"
       Text="Click on the Button to Fill the Grid">
  </asp:Label>
  <br />
  <br />
  <asp:GridView ID="carsGridView" runat="server">
  </asp:GridView>
  <br />
  <asp:Button ID="btnFillData" runat="server" Text="Fill Grid" />
</div>
</form>
```

Az ASP.NET webes vezérlőelemeit a 32. fejezetben fogjuk részletesen tanulmányozni. Annyit már most tudnunk kell, hogy a webes vezérlőelemek a webkiszolgálón feldolgozott objektumok, amelyek a HTML-ábrázolásukat automatikusan visszaadják a kimenő HTTP-válaszban (vagyis a HTML-t nem mi programozzuk!). Ezen a nagyszerű előnyön túl, az ASP.NET webes vezérlőelemei az asztali programozási modellt utánozzák, mivel a tulajdonságok, a metódusok és az események elnevezései rendre a Windows Forms/WPF megfelelőikre hasonlítanak.

Adatelérési logika hozzáadása

Kezeljük a kattintási eseményét a gombnak vagy a Visual Studio Properties ablakának „villám" ikonjával, vagy a tervezőablak felső részén található legördülő listák segítségével. Ezzel a gomb definícióját kiegészítettük egy `OnClick` attribútummal, amelyet a `Click` esemény kezelőjének nevéhez rendeltünk hozzá:

```
<asp:Button ID="btnFillData" runat="server"
     Text="Fill Grid" OnClick="btnFillData_Click"/>
```

Emellett egy üres `<script>` blokkot is kapunk, amellyel elkészíthetjük a kiszolgálóoldali kattintási esemény kezelőjét. Adjuk hozzá a következő kódot, és figyeljük meg, hogy bejövő paraméterek hajszálpontosan megegyeznek a `System.EventHandler` metódusreferencia céljával:

```
<script runat="server">
  void btnFillData_Click(object sender, EventArgs args)
  {
  }
</script>
```

A következő lépés a GridView feltöltése az AutoLotDAL.dll szerelvény segítségével. Ehhez határozzuk meg a <%@ Import %> direktíva segítségével, hogy az AutoLotConnectedLayer névteret használjuk. Továbbá értesítenünk kell az ASP.NET-futtatókörnyezetet, hogy ez az egyfájlos oldal az AutoLotDAL.dll szerelvényre hivatkozik az <%@ Assembly %> direktíva segítségével (hamarosan többet megtudhatunk a direktívákról). Íme, a Default.aspx fájl megmaradt, releváns logikája (a kapcsolatsztringet szükség szerint módosítanunk kell):

```
<%@ Page Language="C#" %>
<%@ Import Namespace = "AutoLotConnectedLayer" %>
<%@ Assembly Name ="AutoLotDAL" %>

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

<script runat="server">
  void btnFillData_Click(object sender, EventArgs args)
  {
    InventoryDAL dal = new InventoryDAL();
    dal.OpenConnection(@"Data Source=(local)\SQLEXPRESS;" +
        "Initial Catalog=AutoLot;Integrated Security=True");
    carsGridView.DataSource = dal.GetAllInventory();
    carsGridView.DataBind();
    dal.CloseConnection();
  }
</script>
<html xmlns="http://www.w3.org/1999/xhtml" >
...
</html>
```

Mielőtt elmélyülnénk az *.aspx fájl formátumának részleteiben, teszteljük az oldalt. Először mentsük el az *.aspx fájlt. Ha manuálisan szeretnénk használni a WebDev.WebServer.exe-t, nyissunk meg egy .NET-parancssort, és futtassuk a WebDev.WebServer.exe segédprogramot. Győződjünk meg arról, hogy helyesen adtuk meg a tárolt Default.aspx fájl elérési útját, például a következőképpen (jelen esetben egy tetszőleges 12345 portot adtunk meg):

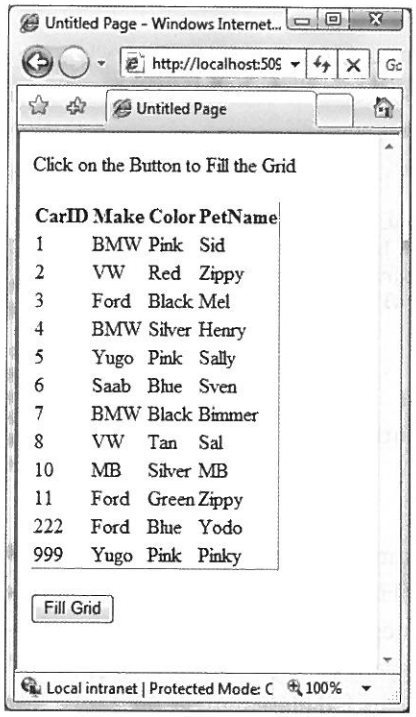
```
webdev.webserver.exe
      /port:12345 /path:"C:\CodeTests\SinglePageModel"
```

Most írjuk be a böngészőnkbe a következő URL-t:

```
http://localhost:12345/
```

Amikor az oldal betöltődik, kezdetben egy címkét és egy gombot látunk. Azonban, amikor a gombra kattintunk, adatokat küldünk vissza a webkiszolgálóra, és a webes vezérlőelemek ekkor a megfelelő HTML-címkéiket jelenítik meg.

A `WebDev.WebServer.exe` segédprogramot közvetett módon a Visual Studio 2008-ból is elindíthatjuk. Csak kattintsunk a jobb gombbal arra az oldalra, amelyet szeretnénk megtekinteni, majd válasszuk a View In Browser menüpontot. A 31.11. ábrán mindkét eset kimenetét láthatjuk, miután a Fill Grid gombra kattintottunk.



31.11. ábra: *Webalapú adatelérés*

Ez egyszerű volt, nem? Bár, ahogy mondják, az ördög a részletekben rejlik, ezért ássuk bele jobban magunkat az `*.aspx` fájl felépítésébe. Kezdjük az *oldaldirektívák* szerepének vizsgálatával.

Az ASP.NET-direktívák szerepe

Az első dolog, amellyel tisztában kell lennünk, hogy az `*.aspx` fájlok általában számos direktívával kezdődnek. Az ASP.NET-direktívákra a `<%@ ... %>` jelölők utalnak, és különböző attribútumok minősíthetik, amelyek utasítják az ASP.NET-futtatókörnyezetet, hogyan kell az adott attribútumot feldolgoznia.

Minden `*.aspx` fájlnak legalább egy `<%@Page%>` direktívát kell tartalmaznia, amely definiálja az oldalon alkalmazott felügyelt nyelvet (a `language` attribútum révén). A `<%@Page%>` direktíva emellett a kapcsolódó mögöttes kódfájl nevét is definiálhatja (ha létezik), engedélyezheti a nyomkövetés támogatást, és így tovább. A 31.2. táblázat a legfontosabb `<%@Page%>` központú attribútumok közül mutat be néhányat.

Attribútum	Jelentés
`CodePage`	Ez az attribútum a kapcsolódó mögöttes kódfájl nevét határozza meg.
`CompilerOptions`	Ennek az attribútumnak a segítségével definiálhatjuk a parancssori kapcsolókat (egyetlen sztring formájában jelennek meg), amelyeket az oldal feldolgozása során a fordítónak átadunk.
`EnableTheming`	Ez az attribútum megadja, hogy az `*.aspx` oldal vezérlőelemei támogatják-e az ASP.NET-témákat.
`EnableViewState`	Ez az attribútum jelzi, hogy a nézet állapotot (view state-et) a rendszer az oldalkérések között megőrzi (a tulajdonságról a 33. fejezetben találhatunk további részleteket).
`Inherits`	Ez az attribútum egy osztályt definiál a mögöttes kódban található oldal részére, amelyből az `*.aspx` fájl származik. Ez bármilyen `System.Web.UI.Page`-ből származtatott osztály lehet.
`MasterPageFile`	Ez az attribútum az aktuális `*.aspx` oldal mesteroldalát adja meg.
`Trace`	Ez az attribútum jelzi, hogy engedélyeztük-e a nyomkövetést.

31.2. táblázat: *A <%@Page%> direktíva néhány attribútuma*

Egy adott *.aspx fájl a <%@Page%> direktíván kívül különféle <%@Import%> direktívákat is megadhat az aktuális oldal és a <%@Assembly%> direktívák által igényelt névterek explicit kifejezésére, hogy a webhely által használt külső kódkönyvtárakat meghatározza (amelyek általában a webhely \bin mappájában találhatóak).

Ebben a példában meghatároztuk, hogy az AutoLotDAL.dll szerelvény AutoLotConnectedLayer névterének típusait használjuk. Ahogy rájöhettünk, ha további .NET-névterekre van szükségünk, egyszerűen csak több <%@Import%>/<%@Assembly%> direktívát kell megadnunk.

A jelenlegi .NET-tudásunkkal elcsodálkozhatunk azon, hogy ez az *.aspx fájl hogyan kerülhette el azt, hogy további <%@Import%> direktívákat határozzunk meg benne ahhoz, hogy hozzáférjen a System névtér System.Object és System.EventHandler típusaihoz (többek között). Ez annak köszönhető, hogy az *.aspx fájlok automatikusan hozzáférnek a .NET platform telepítési útvonalán található machine.config fájlban definiált kulcsfontosságú névterek készletéhez. Ebben az XML-alapú fájlban több automatikusan importált névteret találunk:

```
<pages>
  <namespaces>
    <add namespace="System"/>
    <add namespace="System.Collections"/>
    <add namespace="System.Collections.Specialized"/>
    <add namespace="System.Configuration"/>
    <add namespace="System.Text"/>
    <add namespace="System.Text.RegularExpressions"/>
    <add namespace="System.Web"/>
    <add namespace="System.Web.Caching"/>
    <add namespace="System.Web.SessionState"/>
    <add namespace="System.Web.Security"/>
    <add namespace="System.Web.Profile"/>
    <add namespace="System.Web.UI"/>
    <add namespace="System.Web.UI.WebControls"/>
    <add namespace="System.Web.UI.WebControls.WebParts"/>
    <add namespace="System.Web.UI.HtmlControls"/>
    ...
  </namespaces>
</pages>
```

A szkriptblokk elemzése

Az egyfájlos modellben egy *.aspx fájl kiszolgálóoldali szkriptkódot tartalmazhat, amelyet a rendszer a webkiszolgálón futtat. Emiatt *kulcsfontosságú*, hogy az összes kiszolgálóoldali kódblokkot a runat="server" attribútummal definiáljuk, hogy az a kiszolgálón fusson. Ha a runat="server" attribútumot nem határozzuk meg, a futtatókörnyezet azt felételezi, hogy *ügyféloldali* szkriptblokkot írtunk, amelyet a kimenő HTTP-válaszba továbbítunk:

```
<script runat="server">
  void btnFillData_Click(object sender, EventArgs args)
  {
    InventoryDAL dal = new InventoryDAL();
    dal.OpenConnection(@"Data Source=(local)\SQLEXPRESS;" +
        "Initial Catalog=AutoLot;Integrated Security=True");
    carsGridView.DataSource = dal.GetAllInventory();
    carsGridView.DataBind();
    dal.CloseConnection();
  }
</script>
```

Ezen segédmetódus szignatúrája furcsán ismerősnek tűnhet. Emlékezzünk vissza a Windows Forms (vagy jelen esetben WPF-) tanulmányainkra, hogy egy adott vezérlőelem eseménykezelőjének egyeznie kell a kapcsolódó .NET-metódusreferenciában definiált mintával. Ugyanúgy, mint a Windows Forms esetében, ha egy kiszolgálóoldali kattintási eseményt szeretnénk kezelni, a kérdéses metódusreferencia a `System.EventHandler`, amely – ha emlékszünk rá – csak azokat a metódusokat hívja, amelyek első paramétere a `System.Object`, a második pedig a `System.EventArgs`.

Az ASP.NET vezérlőelem-deklarációinak áttekintése

A példa utolsó figyelemre méltó pontja a Button, a Label és a GridView webes vezérlőelemek deklarációja. A klasszikus ASP-hez és a nyers HTML-hez hasonlóan, az ASP.NET webes vezérlőelemei is a `<form>` elemek hatókörében találhatók. Most azonban a nyitó `<form>` elemet a `runat="server"` attribútummal jelöltük meg. Ez szintén rendkívül fontos, mivel ez a címke tájékoztatja az ASP.NET-futtatókörnyezetet arról, hogy mielőtt a HTML-t továbbítaná a válaszfolyamba, a tárolt ASP.NET-vezérlők még alakíthatják HTML-megjelenésüket:

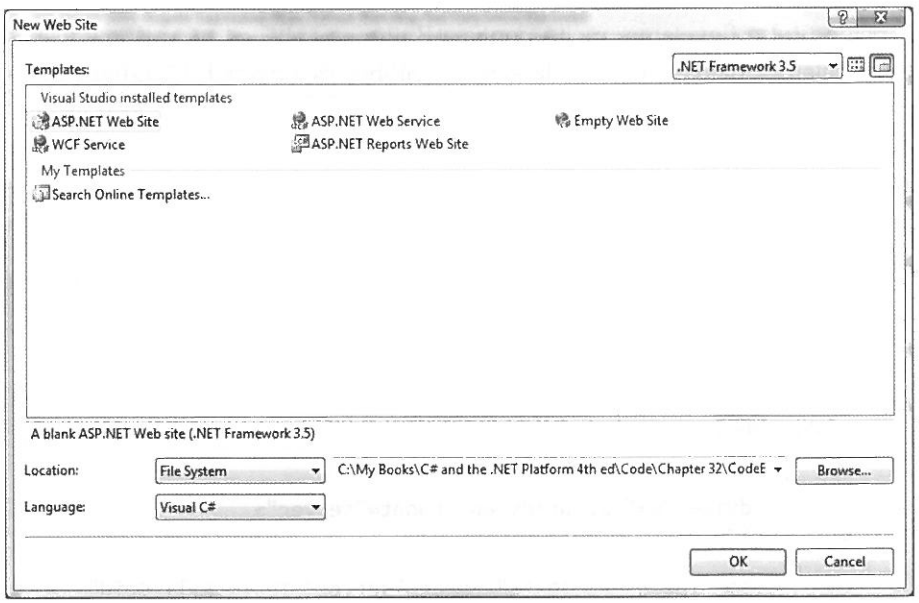
```
<form id="form1" runat="server">
<div>
  <asp:Label ID="lblInfo" runat="server"
       Text="Click on the Button to Fill the Grid">
  </asp:Label>
  <br />
  <br />
  <asp:GridView ID="carsGridView" runat="server">
  </asp:GridView>
  <br />
  <asp:Button ID="btnFillData" runat="server" Text="Fill Grid" />
</div>
</form>
```

Figyeljük meg, hogy az ASP.NET webes vezérlőelemeket <asp> és </asp> címkékkel deklaráltuk, és a runat="server" attribútummal is megjelöltük őket. A nyitó címkében a webes űrlap vezérlőelem nevét és tetszőleges számú név/érték párt adhatunk meg, amelyeket a megfelelő HTML megjelenítésére használunk futásidőben.

Forráskód A SinglePageModel kódfájlt a forráskódkönyvtár 31. fejezetének alkönyvtára tartalmazza. A forráskódkönyvtárról lásd a Bevezetés xlv. oldalát.

A mögötteskód-modell használata

A mögötteskód-modell szemléltetésére hozzuk létre újra az előző példát a Visual Studio 2008 Web Site sablonja segítségével. (Tudnunk kell, hogy a Visual Studio 2008 nem követeli meg a mögöttes kód használatát; azonban ez az új webhelyek alapértelmezett viselkedése.) Válasszuk ki a File > New > Web Site menüpontot, és a 31.12. ábrán látható módon válasszuk az ASP.NET Web Site sablont.



31.12. ábra: *A Visual Studio 2008 ASP.NET Web Site sablonja*

Figyeljük meg a 31.12. ábrán, hogy kiválaszthatjuk az új webhely tárhelyét. Ha a File Systemet választjuk, a tartalomfájlok lokális könyvtárba kerülnek, az oldalakat pedig a `WebDev.WebServer.exe` szolgáltatja. Ha az FTP vagy a HTTP lehetőséget választjuk, a webhelyet egy új IIS-beli virtuális könyvtár hosztolja. Ebben a példában nincs különbség a két lehetőség között, de az egyszerűség kedvéért azt javaslom, válasszuk a File System opciót, és adjuk meg a C:\CodeBehindPageModel nevű új könyvtárat.

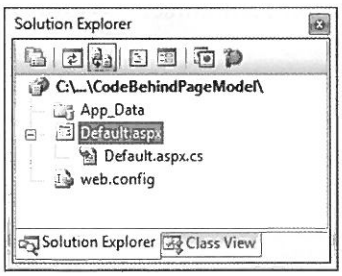
A tervező segítségével hozzunk létre felhasználói felületet, amely egy Label, egy Button és egy GridView vezérlőelemet tartalmaz, majd alakítsuk ki ízlésünk szerint a felhasználói felületet a Properties ablakkal.

Megjegyzés Ha meglévő webhelyet szeretnénk megnyitni a Visual Studio 2008-ban, válasszuk a File > Open > Web Site menüpontot, majd a webtartalmat tartalmazó mappát (vagy az IIS-beli virtuális könyvtárat).

Figyeljük meg, hogy a `<%@Page%>` direktíva néhány új attribútummal egészült ki:

```
<%@ Page Language="C#" AutoEventWireup="true"
        CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

A `CodeFile` attribútummal azt a kapcsolódó külső fájlt adjuk meg, amely az oldal forráskódját tartalmazza. Alapértelmezés szerint ezek a mögöttes kódfájlok úgy kapnak nevet, hogy a `.cs` utótagot adjuk az `*.aspx` fájl nevéhez (jelen esetben `Default.aspx.cs`). Ha megnézzük a Solution Explorert, észreveszszük, hogy ez a mögöttes kódfájl a Web Form ikon egyik alcsomópontján látható (lásd a 31.13. ábrát).



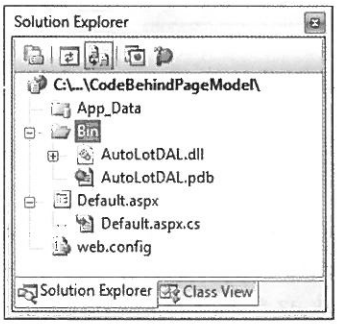
31.13. ábra: *Adott *.aspx fájlhoz rendelt mögöttes kódfájl*

Ha megnyitnánk a mögöttes kódfájlunkat, olyan részleges osztályt találnánk, amely a `System.Web.UI.Page` osztályból származik és implementálja a `Load` esemény kezelését. Figyeljük meg, hogy az osztály neve (`_Default`) megegyezik a `<%@Page%>` direktíva `Inherits` attribútumával:

```
public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
  }
}
```

Hivatkozás az AutoLotDAL.dll szerelvényre

Ahogy korábban említettük, amikor a Visual Studio 2008 segítségével hozunk létre webalkalmazás-projekteket, nem kell manuálisan létrehoztunk a \bin alkönyvtárakat, és kézzel átmásolnunk a privát szerelvényeket. A példa kedvéért válasszuk ki az Add Reference párbeszédpanelt a Website menüpont segítségével, és hivatkozzunk az `AutoLotDAL.dll` szerelvényre. Ekkor a 31.14. ábrán látható módon a Solution Explorerben megjelenik az új \bin mappa.



31.14. ábra: *A Visual Studio rendszerint karbantart „speciális" ASP.NET-mappákat*

A kódfájl módosítása

Kezeljük a gomb kattintási eseményét úgy, hogy kétszer kattintunk a tervezőben elhelyezett `Button` objektumra. Mint eddig is, a `Button` definíciója most is `OnClick` attribútummal frissült. A kiszolgálóoldali eseménykezelő azonban már nincs az `*.aspx` fájl `<script>` hatókörében, hanem a `_Default` osztály metódusa lett.

A példa befejezéseként adjunk a mögöttes kódfájlhoz egy `using` utasítást az `AutoLotConnectedLayerre`, majd implementáljuk a gomb eseménykezelőjét az előző logika alkalmazásával:

```
using AutoLotConnectedLayer;

public partial class _Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
  }
```

```
    protected void btnFillData_Click(object sender, EventArgs e)
    {
      InventoryDAL dal = new InventoryDAL();
      dal.OpenConnection(@"Data Source=(local)\SQLEXPRESS;" +
          "Initial Catalog=AutoLot;Integrated Security=True");
      carsGridView.DataSource = dal.GetAllInventory();
      carsGridView.DataBind();
      dal.CloseConnection();
    }
}
```

Ha a `File System` opciót választottuk, a `WebDev.WebServer.exe` automatikusan elindul a webalkalmazás futtatásakor (ha az IIS opciót választottuk, ez nyilván nem történik meg). Az alapértelmezett böngésző most mindkét esetben megjeleníti az oldal tartalmát.

Az ASP.NET-oldalak hibakeresése és nyomkövetése

ASP.NET-projektek létrehozásakor nagyjából ugyanazokat a hibakeresési módszereket használhatjuk, mint bármilyen más Visual Studio 2008 projekttípus esetében. Megadhatunk töréspontokat a mögöttes kódfájlunkban (csakúgy, mint beágyazott szkriptblokkokat egy `*.aspx` fájlban), elindíthatunk hibakeresési munkameneteket (alapértelmezés szerint az F5 billentyű megnyomásával), és végiglépkedhetünk a kódon.

Az ASP.NET-webalkalmazás hibakereséséhez azonban a webhelynek megfelelően konfigurált `Web.config` fájlt kell tartalmaznia. A fejezet utolsó része bemutatja a `Web.config` fájlokat, de addig is röviden annyit, hogy ezen XML-fájlok általános rendeltetése megegyezik a végrehajtható szerelvények `App.config` fájljának céljával. Alapértelmezés szerint az összes Visual Studio 2008 webprojekt automatikusan rendelkezik egy `Web.config` fájllal. A hibakeresési támogatás kezdetben le van tiltva (mivel csökkenti a teljesítményt). Hibakeresési munkamenet indításakor az IDE megkéri, hogy engedélyezzük a hibakeresést. Ha ezt megtettük, a `Web.config` fájl `<compilation>` eleme a következőképpen módosul:

```
<compilation debug="true"/>
```

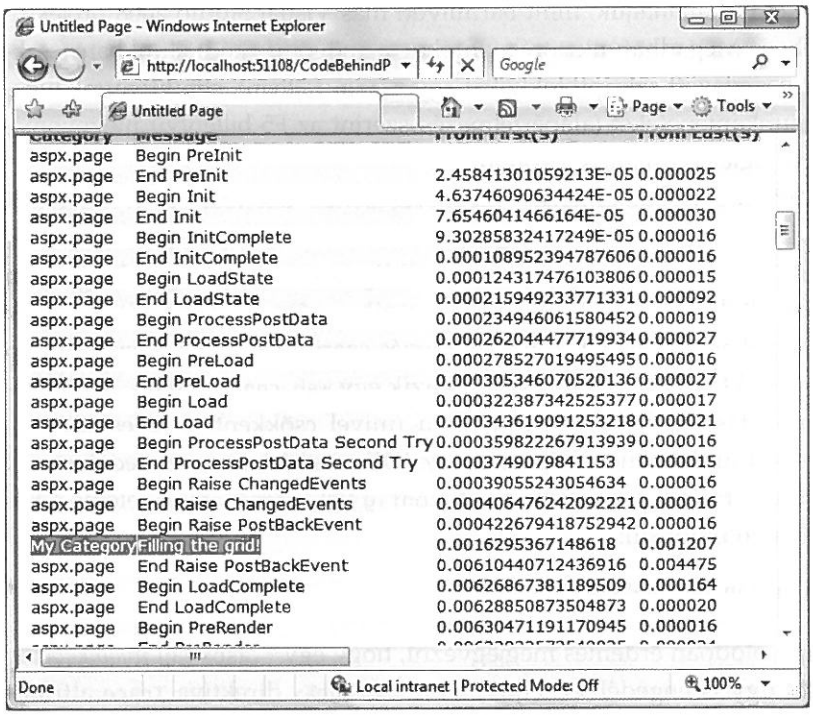
Ehhez kapcsolódóan érdemes megjegyezni, hogy egy `*.aspx` fájl *nyomkövetési támogatását* úgy is engedélyezhetjük, ha a `<%@Page%>` direktíva `Trace` attribútumának `true` értéket adjuk (a teljes webhely nyomkövetésének engedélyezéséhez módosítsuk a `Web.config` fájlt):

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="Default.aspx.cs" Inherits="_Default" Trace="true" %>
```

Ha ezzel készen vagyunk, a generált HTML számos, az előző HTTP-kérésre/válaszra vonatkozó részletet tartalmaz (kiszolgáló oldali változók, munkamenet- és alkalmazásváltozók, kérés/válasz stb.). A saját nyomkövetési üzeneteink beszúrásához használjuk a `System.Web.UI.Page` típus `Trace` tulajdonságát. Ha bármikor naplózni szeretnénk az egyedi üzeneteket (szkriptblokkból vagy C#-forráskódfájlból), csak hívjuk meg a `Write()` metódust:

```
protected void btnFillData_Click(object sender, EventArgs e)
{
  Trace.Write("My Category", "Filling the grid!");
  ...
}
```

Ha újra futtatjuk a projektet, és visszaküldünk adatokat a webkiszolgálóra, láthatjuk, hogy az egyedi kategóriánk és az egyedi üzenetünk jelen vannak. Figyeljük meg a 31.15. ábrán látható kiemelt üzenetet, amely nyomkövetési adatokat jeleníti meg.



31.15. ábra: *Egyedi nyomkövetési üzenetek naplózása*

Forráskód A CodeBehindPageModel kódfájlt a forráskódkönyvtár 31. fejezetének alkönyvtára tartalmazza. A forráskódkönyvtárról lásd a Bevezetés xlv. oldalát.

ASP.NET-webhely könyvtárszerkezetének részletei

Az új Visual Studio 2008 webalkalmazások alapértelmezés szerint tartalmaznak egy kezdeti weboldalt, egy Web.config fájlt és egy `App_Data` nevű mappát (amely alapértelmezés szerint üres). Az ASP.NET-webhelyek bármennyi speciálisan elnevezett alkönyvtárat tartalmazhatnak, amelyek mindegyike egyedi jelentéssel bír az ASP.NET-futtatókörnyezet számára. A 31.3. táblázat ezeket a „speciális alkönyvtárakat" ismerteti.

Alkönyvtár	Jelentés
App_Browsers	Ez a mappa böngésződefiníciós fájlokat tartalmaz, amelyekkel azonosíthatjuk a böngészőket és meghatározhatjuk a képességeiket.
App_Code	Ez a mappa olyan összetevők vagy osztályok forráskódjait tartalmazza, amelyeket az alkalmazás részeként szeretnénk lefordítani. Az ASP.NET az ebben a mappában található kódot akkor fordítja le, mikor elkérjük az oldalt. Az alkalmazásunk automatikusan hozzáfér az App_Code mappában lévő kódhoz.
App_Data	Access `*.mdb` fájlokat, SQL Express `*.mdf` fájlokat, XML-fájlokat vagy egyéb adattárakat tartalmazó mappa.
App_GlobalResources	Az alkalmazáskódból programozott módon hozzáférhető `*.resx` fájlok mappája.
App_LocalResources	Meghatározott oldalhoz tartozó `*.resx` fájlok mappája.
App_Themes	Fájlok gyűjteményét tartalmazó mappa, amelyek az ASP.NET-weboldalak és -vezérlőelemek megjelenését definiálják.
App_WebReferences	Proxyosztályokat, sémákat, illetve egyéb olyan fájlokat tartalmazó mappa, amelyek az alkalmazásban használt webszolgáltatással kapcsolatosak.
Bin	Lefordított privát szerelvényeket (`*.dll` fájlokat) tartalmazó mappa. Az alkalmazásunk a Bin mappában található szerelvényekre automatikusan hivatkozik.

31.3. táblázat: *Speciális ASP.NET-alkönyvtárak*

Ha a fenti alkönyvtárak bármelyikét szeretnénk hozzáadni az aktuális webalkalmazásunkhoz, azt a Web Site > Add Folder menüpont segítségével megtehetjük. Sok esetben ezt az IDE azonban automatikusan megteszi, mikor kapcsolódó fájlokat szúrunk be a webhelyre (például ha új osztályfájlt illesztünk a projektbe, az IDE automatikusan hozzáad egy App_Code mappát a könyvtárszerkezethez, ha az még nem létezik).

Hivatkozás szerelvényekre

Ahogy azt néhány oldallal korábban említettük, az ASP.NET-weboldalakat végül .NET-szerelvényekké fordítjuk. Így nem meglepő, hogy a webhelyek tetszőleges számú privát vagy megosztott szerelvényre hivatkozhatnak. Az ASP.NET alatt az ASP.NET 1.x-hez képest meglehetősen eltérő módon rögzítjük a webhelyek külső szerelvényeit. Ennek az alapvető eltérésnek az az oka, hogy a Visual Studio 2008 *projekfájl nélkül* kezeli a webhelyeket.

Habár a Web Site sablon generál egy `*.sln` fájlt, amely betölti az `*.aspx` fájlokat az IDE-be, a kapcsolódó `*.csproj` fájl már nem létezik. Ahogy azt bizonyára tudjuk, az ASP.NET 1.x Web Application projektek az összes külső szerelvényt egy `*.csproj` fájlban rögzítették. Ez felveti a nyilvánvaló kérdést: az ASP.NET hol rögzíti a külső szerelvényeket?

Láthattuk, hogy amikor egy privát szerelvényre hivatkozunk, a Visual Studio 2008 a könyvtárstruktúránkban automatikusan létrehoz egy \bin könyvtárat a bináris fájlok helyi másolatainak tárolására. Amikor a kód ezekben a kódkönyvtárakban tárolt típusokat használja, azokat a rendszer igény szerint automatikusan betölti.

Ha megosztott szerelvényre hivatkozunk, a Visual Studio 2008 automatikusan beilleszt egy `Web.config` fájlt az aktuális solutionbe (ha még nincs benne), és az `<assemblies>` elemben rögzíti a külső referenciát. Például, ha újra kiválasztjuk a Web Site > Add Reference menüpontot, és most megosztott szerelvényt választunk (mint például a `System.Data.OracleClient.dll`) láthatjuk, hogy a `Web.config` fájl a következőképpen módosul:

```
<assemblies>
  <add assembly="System.Data.OracleClient, Version=2.0.0.0,
                Culture=neutral, PublicKeyToken=B77A5C561934E089"/>
</assemblies>
```

Láthatjuk, hogy minden szerelvényt ugyanazzal a dinamikus betöltéshez szükséges adattal írtunk le az `Assembly.Load()` metódusban (lásd a 16. fejezetet).

Az App_Code mappa szerepe

Az App_Code mappában olyan forráskódfájlokat tárolunk, amelyek nem tartoznak közvetlenül egy adott weboldalhoz (mint egy mögöttes kódfájl), de a webhely használatához a rendszer lefordítja azokat. Az App_Code mappában található forráskódokat a rendszer automatikusan, menet közben lefordítja, ha szükséges. A szerelvény ezután a webhely bármely más forráskódja számára elérhető. Ezért az App_Code mappa nagyon hasonlít a Bin mappára, kivéve, hogy a lefordított kódok helyett forráskódokat tárolhatunk benne. Ennek a megközelítésnek az a legnagyobb előnye, hogy anélkül definiálhatunk egyedi típusokat a webalkalmazás számára, hogy le kellene őket fordítanunk.

Egyetlen App_Code mappa több, különböző nyelven készült forráskódfájlt is tartalmazhat. Futásidőben a megfelelő fordító generálja a kérdéses szerelvényt. Ha azonban inkább szétválogatnánk a forráskódokat, több alkönyvtárat definiálhatunk, amelyekben tetszőleges számú felügyelt kódfájlt (`*.vb`, `*.cs` stb.) tárolhatunk.

Tételezzük fel például, hogy hozzáadtunk egy App_Code mappát egy webhely gyökérkönyvtárához, amely két almappát (MyCSharpCode és MyVbNetCode) tartalmaz, és az almappák nyelvspecifikus fájlokat tárolnak. Ha ezt megtettük, módosítsuk a `web.config` fájlt, és meghatározhatjuk az alkönyvtárakat a `<configuration>` elembe ágyazott `<codeSubDirectories>` elem segítségével:

```
<compilation debug="true" strict="false" explicit="true">
  <codeSubDirectories>
    <add directoryName="MyCSharpCode" />
    <add directoryName="MyVbNetCode" />
  </codeSubDirectories>
</compilation>
```

Megjegyzés Az App_Code könyvtárban olyan fájlokat is tárolunk, amelyek nem nyelvfájlok, ennek ellenére mégis hasznosak (`*.xsd` fájlok, `*.wsdl` fájlok stb.).

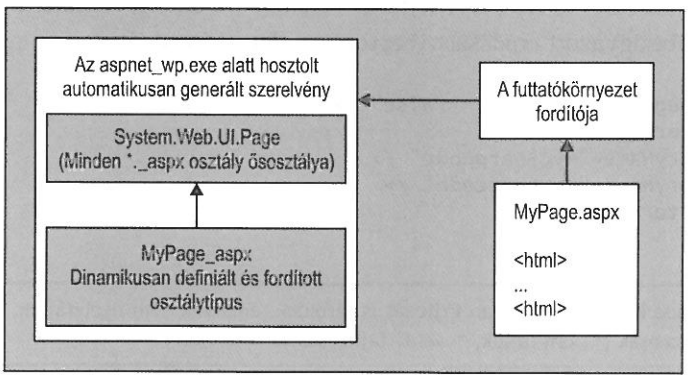
A Bin, az App_Code, az App_Data és az App_Themes mappákon kívül még létezik további két „speciális alkönyvtár", amellyel meg kell ismerkednünk, mindkettőt megvizsgáljuk a következő néhány fejezetben. Mint mindig, a fennmaradó ASP.NET-alkönyvtárakkal kapcsolatos további információkért forduljunk a .NET Framework 3.5 SDK dokumentációjához.

Az ASP.NET-oldal fordítási ciklusa

Függetlenül attól, hogy melyik oldalmodellt használjuk (egyfájlos vagy mögöttes kód alapú), az `*.aspx` fájlokat (és a kapcsolódó mögöttes kódfájlokat) menet közben fordítjuk érvényes .NET-szerelvényekké. Ezt a szerelvényt aztán az ASP.NET munkavégző folyamat (`aspnet_wp.exe`) hosztolja a saját alkalmazástartományában (az alkalmazástartományokkal kapcsolatos további információkért lásd a 17. fejezetet). Az ASP.NET alatt azonban meglehetősen eltérő módon fordítjuk a webhelyek szerelvényeit.

Egyfájlos oldalak fordítási ciklusa

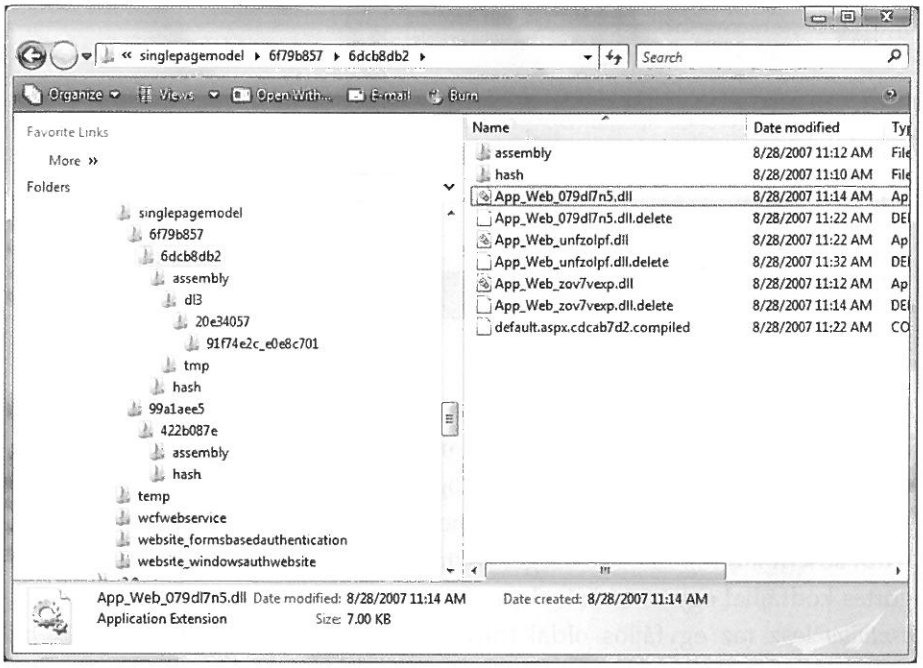
Ha az egyfájlosoldal-modellt alkalmazzuk, a HTML-markup, a kiszolgálóoldali `<script>` blokkok és a webes vezérlőelemek definícióját dinamikusan fordítjuk egy `System.Web.UI.Page` osztályból származó osztálytípusba. Az osztály neve az `*.aspx` fájl nevén alapul, és egy `_aspx` utótagot kap (például a `MyPage.aspx` oldalból `MyPage_aspx` osztálytípus lesz). A 31.16. ábra a ezt a folyamatot mutatja be.



31.16. ábra: *Egyfájlos oldalak fordítási modellje*

Ezt a dinamikusan fordított szerelvényt a C:\WINDOWS\Microsoft.NET\Framework\v2.0.50727\Temporary ASP.NET Files gyökérkönyvtár olyan alkönyvtárába települ, amelyet a futtatókörnyezet definiált. A gyökér alatti útvonal több tényezőnek köszönhetően (hash kódok stb.) eltérő lehet, de kitartással végül megtaláljuk a keresett `*.dll`-t (és a támogatófájlokat). A 31.17. ábra a fejezet korábbi részében bemutatott `SinglePageModel` példa generált szerelvényét ábrázolja.

Megjegyzés Mivel ezek az automatikusan generált szerelvények igazi .NET bináris fájlok, ha megnyitnánk a webalkalmazásunkhoz kapcsolódó `*.dll`-t az `ildasm.exe` vagy a `reflector.exe` segítségével, csakugyan köztes nyelvi kódot, metaadatokat és egy szerelvényszintű manifesztumokat találnánk ott.

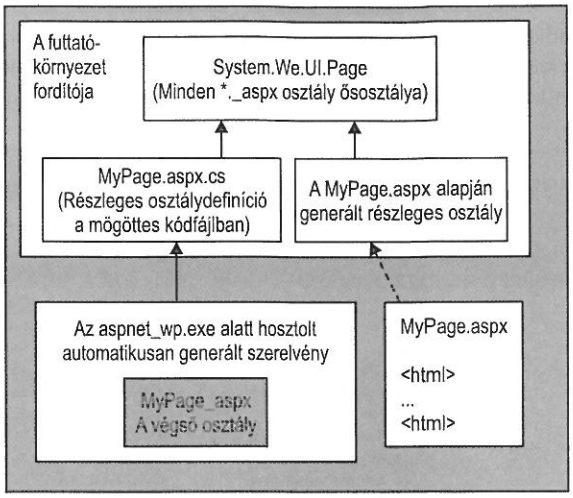


31.17. ábra: *Automatikusan generált ASP.NET-szerelvény*

Többfájlos oldalak fordítási ciklusa

A mögöttes kód alapú modellt alkalmazó oldalak fordítási folyamata hasonló az egyfájlos oldaléhoz. Azonban a `System.Web.UI.Page` osztályból származó típus három fájlból áll (igen, *háromból*), és nem kettőből.

Emlékezzünk vissza, hogy az előző `CodeBehindPageModel` példában a `Default.aspx` fájlt a mögöttes kódfájlban található `_Default` nevű részleges osztályhoz kapcsoltuk. ASP.NET 1.x tapasztalatok birtokában meglepődhetünk, hogy mi történt a különféle webes vezérlőelemek tagváltozóinak, illetve az `InitializeComponent()` kódjának deklarációjával, például az eseménykezelő logikával. ASP.NET alatt ezekről a részletekről a memóriában generált részleges osztály egy harmadik jellemzője gondoskodik. A valóságban ez szó szerint nem fájl, hanem a részleges osztály egy memórián belüli megjelenítése. Nézzük meg a 31.18. ábrát.



31.18. ábra: *Többfájlos oldalak fordítási modellje*

Ebben a modellben az `*.aspx` fájlban deklarált webes vezérlőelemekkel építjük fel a kiegészítő részleges osztályt, amely a felhasználói felület összes tagváltozóját, valamint a konfigurációs logikát definiálja, amelyet korábban az ASP.NET 1.x `InitializeComponent()` metódusában találhattunk (csak közvetlenül sosem látjuk). Ezt a részleges osztályt fordítási időben a rendszer a mögöttes kódfájllal egyesíti, aminek eredménye a generált `_aspx` osztálytípus *ősosztálya* lesz (az egyfájlos oldal fordítási modelljében a generált `_aspx` fájl közvetlenül a `System.Web.UI.Page` osztályból származik).

Mindkét esetben, ha a szerelvényt a kezdeti HTTP-kérés során hoztuk létre, a rendszer az elkövetkező kérések során is használja, így nem kell a szerelvényt újra lefordítani. Ennek ismeretében könnyebb megérteni, hogy egy `*.aspx` oldal első lekérése miért vesz jóval több időt igénybe, és ugyanazon oldal későbbi találatai miért rendkívül hatékonyak.

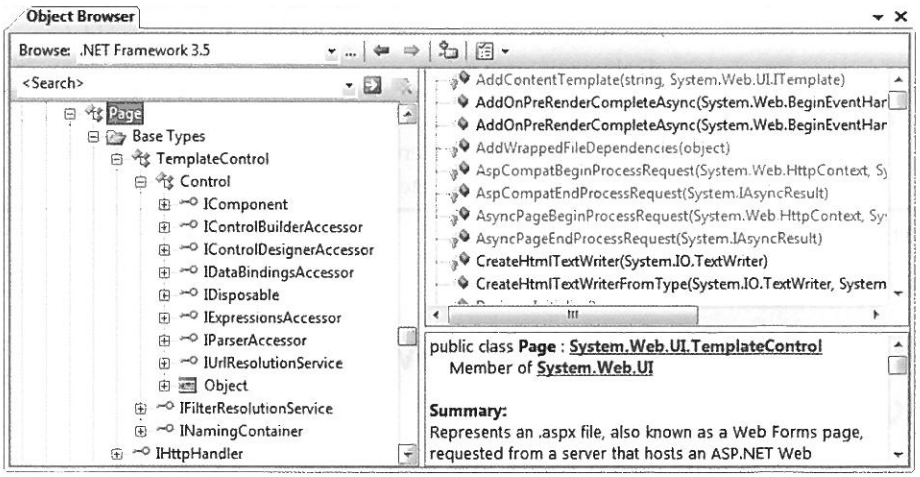
Megjegyzés Az ASP.NET alatt lehetőségünk nyílik, hogy a webhelyek összes oldalát (vagy azok közül néhányat) előfordítsunk az `aspnet_compiler.exe` parancssori eszköz segítségével. További részletekért forduljunk a .NET Framework 3.5 SDK dokumentációjához.

A Page típus származtatási lánca

Ahogy az imént láthattuk, az utolsó generált osztály, amely az `*.aspx` fájlt képviseli a `System.Web.UI.Page` osztályból származik. Mint minden ősosztály, ez a típus is polimorf interfészt biztosít az összes származtatott típus számára. A `Page` típus azonban nem az egyetlen tagja a származtatási hierarchiának. Ha megkeresnénk a `Page` típust (a `System.Web.dll` szerelvényben) a Visual Studio 2008 objektumböngészője segítségével, azt látnánk, hogy a `Page` „az-egy" `TemplateControl`, „az-egy" `Control` és „az-egy" `Object` (lásd a 31.19. ábrát).

Ahogy már rájöhettünk, ezek az ősosztályok rengeteg funkcionalitással ruházzák fel az `*.aspx` fájlokat. A projektjeink többségében a `Page` és a `Control` szülőosztályokban definiált tagokat alkalmazzuk majd. A `System.Web.UI.TemplateControl` osztályból nyert funkcionalitással lényegében csak az egyedi Web Form vezérlőelemek létrehozásakor vagy a renderelési folyamattal folytatott együttműködés közben foglalkozunk.

Az első lényeges szülőosztály maga a `Page`. Ebben több olyan tulajdonsággal találkozhatunk, amelyek révén különféle webprimitívekkel dolgozhatunk, mint például az alkalmazás- és munkamenet-változókkal, a HTTP kérésekkel/válaszokkal, a tématámogatással és így tovább. A 31.4. táblázat néhány (de közel sem az összes) alapvető tulajdonságot mutat be.



31.19. ábra: *Egy ASP.NET-oldal származtatása*

Tulajdonság	Jelentés
`Application`	Ez a tulajdonság az aktuális webhely alkalmazásváltozóival folytatott kommunikációt teszi lehetővé.
`Cache`	Ez a tulajdonság az aktuális webhely gyorsítótár-objektumaival folytatott kommunikációt teszi lehetővé.
`ClientTarget`	Ennek a tulajdonságnak a segítségével megadhatjuk, hogy a kérelmező böngészőtől függően az oldal hogyan jelenjen meg.
`IsPostBack`	A tulajdonság értéke azt jelzi, hogy az oldal az ügyfél visszaküldésre adott válasz, vagy első ízben kerül letöltésre és hozzáférésre.
`MasterPageFile`	Ez a tulajdonság az aktuális oldal mesteroldalát adja meg.
`Request`	Ez a tulajdonság az aktuális HTTP-kéréshez biztosít hozzáférést.
`Response`	Ez a tulajdonság a kimenő HTTP-válasszal folytatott kommunikációt teszi lehetővé.
`Server`	Ez a tulajdonság a `HttpServerUtility` objektumhoz biztosít hozzáférést, amely különböző kiszolgálóoldali segédfüggvényeket tartalmaz.
`Session`	Ez a tulajdonság az aktuális hívó munkamenetadataival folytatott kommunikációt teszi lehetővé.
`Theme`	Ez a tulajdonság lekérdezi vagy beállítja az aktuális oldalon alkalmazott téma nevét.
`Trace`	Ez a tulajdonság a `TraceContext` objektumhoz biztosít hozzáférést, amely egyedi üzenetek naplózását teszi lehetővé a hibakeresési munkamenetek alatt.

31.4. táblázat: *A Page típus néhány tulajdonsága*

Együttműködés a bejövő HTTP-kérésekkel

Ahogy a fejezet korábbi részében láthattuk, a webes munkamenetek alapvető folyamata úgy kezdődik, hogy az ügyfél belép a webhelyre, megadja a felhasználói adatokat, majd egy Submit gombra kattint, és feldolgozásra visszaküldi a HTML-űrlapadatokat az adott weboldalnak. A legtöbb esetben a `form`

utasítás nyitó címkéje megad egy `action` és egy `method` attribútumot, amelyek arra a fájlra mutatnak a webkiszolgálón, amely megkapja a különböző HTML-vezérlők adatait, valamint az adatok küldésének módját (GET vagy POST):

```
<form name="defaultPage" id="defaultPage"
      action="http://localhost/Cars/ClassicAspPage.asp"
      method = "GET">
  ...
</form>
```

Az ASP-től eltérően az ASP.NET nem támogatja a `Request` nevű objektumot. Azonban az összes ASP.NET-oldal örökli a `System.Web.UI.Page.Request` *tulajdonságot*, amely a `HttpRequest` osztálytípus egy példányához biztosít hozzáférést. A 31.5. táblázat néhány alapvető tagot sorol fel, amelyek nem meglepő módon a klasszikus ASP `Request` objektum azonos tagjaira hasonlítanak.

Tag	Jelentés
`ApplicationPath`	Lekérdezi az ASP.NET-alkalmazás virtuális alkalmazásgyökerének elérési útját a kiszolgálón.
`Browser`	A kliensböngésző lehetőségeiről biztosít információkat.
`Cookies`	Lekérdezi a kliensböngésző által küldött sütik gyűjteményét.
`FilePath`	Az aktuális kérés virtuális útvonalát jelzi.
`Form`	Lekérdezi a HTTP-űrlapváltozók gyűjteményét.
`Headers`	Lekérdezi a HTTP-fejlécek gyűjteményét.
`HttpMethod`	A kliens által alkalmazott HTTP adatátviteli módot (`GET`, `POST`) jelzi.
`IsSecureConnection`	A HTTP-kapcsolat biztonságosságát jelzi (például HTTPS).
`QueryString`	Lekérdezi a HTTP-lekérdezéssztring változók gyűjteményét.
`RawUrl`	Lekérdezi az aktuális kérés nyers URL-jét.
`RequestType`	A kliens által alkalmazott HTTP adatátviteli módot (`GET`, `POST`) jelzi.

Tag	Jelentés
`ServerVariables`	Lekérdezi a webkiszolgáló-változók gyűjteményét.
`UserHostAddress`	Lekérdezi a távoli kliens IP-címét.
`UserHostName`	Lekérdezi a távoli kliens DNS-nevét.

31.5. táblázat: *A HttpRequest típus tagjai*

Ezeken a tulajdonságokon kívül a `HttpRequest` típus számos hasznos metódussal rendelkezik, többek között a következőkkel:

- `MapPath()`: A kért URL virtuális útvonalát fizikai útvonalra képezi le a kiszolgálón az aktuális kérés számára.
- `SaveAs()`: Az aktuális HTTP-kérés részleteit fájlba menti a webkiszolgálón (ez hibakereséskor lehet hasznos).
- `ValidateInput()`: Ha az oldaldirektíva `Validate` attribútuma révén engedélyeztük az ellenőrzési szolgáltatást, a metódus meghívásával az összes felhasználói bemenetet (köztük a sütiket) egy adott lista alapján ellenőrizhetjük, amely a veszélyesnek tartott bemeneti adatokat sorolja fel.

Böngészőstatisztikák

A `HttpRequest` típus első említésre méltó jellegzetessége a Browser tulajdonság, amely az alapul szolgáló `HttpBrowserCapabilities` objektumhoz biztosít hozzáférést. A `HttpBrowserCapabilities` viszont több olyan tagot bocsát a rendelkezésünkre, amelyek révén programozott módon megvizsgálhatjuk a bejövő HTTP-kérést küldő böngészőre vonatkozó statisztikákat.

Hozzuk létre a `FunWithPageMembers` nevű új ASP.NET-webhelyet (ismét a File System opció segítségével). Az első feladatunk, hogy olyan felhasználói felületet készítsünk, amelyen a felhasználó a `btnGetBrowserStats` nevű gombra kattintva megtekintheti a hívó böngésző különféle statisztikáit. Ezeket a statisztikákat dinamikusan generáljuk, és csatoljuk a `lblOutput` nevű címkéhez. A kattintási esemény kezelője a következő:

```
protected void btnGetBrowserStats_Click(object sender, EventArgs e)
{
  string theInfo = "";
  theInfo += string.Format("<li>Is the client AOL? {0}</li>",
             Request.Browser.AOL);
```

```
    theInfo += string.Format("<li>Does the client support ActiveX?
               {0}</li>", Request.Browser.ActiveXControls);
    theInfo += string.Format("<li>Is the client a Beta? {0}</li>",
               Request.Browser.Beta);
    theInfo += string.Format("<li>Does the client support Java
               Applets? {0}</li>", Request.Browser.JavaApplets);
    theInfo += string.Format("<li>Does the client support Cookies?
               {0}</li>", Request.Browser.Cookies);
    theInfo += string.Format("<li>Does the client support VBScript?
               {0}</li>", Request.Browser.VBScript);
    lblOutput.Text = theInfo;
}
```

Ezzel több böngészőlehetőséget tesztelünk. Ahogy kitalálhattuk, rendkívül hasznos, ha felfedezzük, hogy a böngésző támogatja-e az ActiveX vezérlőelemeket, a Java-kisalkalmazásokat és az ügyféloldali VBScript kódot. Ha a hívó böngésző nem támogat egy adott webtechnológiát, az `*.aspx` oldalunk más lehetőséget választhat a működés biztosítására.

Hozzáférés a bemeneti űrlapadatokhoz

A `HttpResponse` típus további jellegzetességei a `Form` és a `QueryString` tulajdonságok. Ezen két tulajdonság lehetővé teszi, hogy megvizsgáljuk a bemeneti űrlapadatokat név/érték párok segítségével. A tulajdonságok a klasszikus ASP-hez hasonlóan működnek. Emlékezzünk vissza arra a klasszikus ASP korábbi ismertetéséből, hogy ha az adatokat HTTP `GET`-tel továbbítjuk, az űrlapadatokat a `QueryString` tulajdonság révén érhetjük el, míg a HTTP `POST`-tal továbbított adatokhoz a `Form` tulajdonságon keresztül lehet hozzáférni.

Noha biztosan hozzáférhetnénk az ügyfél által küldött űrlapadatokhoz a webkiszolgálón a `HttpRequest.Form` és a `HttpRequest.QueryString` tulajdonságok segítségével is, ezek az elavult módszerek (nagyrészt) feleslegesek. Mivel az ASP.NET biztosítja számunkra a kiszolgálóoldali webes vezérlőelemeket, a HTML felhasználóifelület-elemeket objektumokként kezelhetjük. Ezért ahelyett, hogy az alábbihoz hasonló szövegdobozzal megszereznénk az adatokat:

```
protected void btnGetFormData_Click(object sender,
                                    System.EventArgs e)
{
  // Szerezzük meg a céleszköz értékét a txtFirstName azonosítóval.
  string firstName = Request.Form("txtFirstName");

  // Használjuk ezt az értéket az oldalon...
}
```

egyszerűen a kiszolgálóoldali vezérlőtől közvetlenül kérhetjük el az adatokat a Text tulajdonságon keresztül, hogy használhassuk azt a programunkban:

```
protected void btnGetFormData_Click(object sender,
                                    System.EventArgs e)
{
  // Szerezzük meg a céleszköz értékét a txtFirstName azonosítóval.
  string firstName = txtFirstName.Text;

  // Használjuk ezt az értéket az oldalon...
}
```

Ez a módszer nemcsak szilárd OO elvekre épül, de azzal sem kell törődnünk, hogy az űrlapadatokat milyen módon (`GET` vagy `POST`) továbbítottuk, mielőtt az értékeiket megkaptuk. Továbbá, a vezérlőelemek közvetlen használata jóval típusbiztosabb, mivel a gépelési hibák már fordítási időben kiderülnek, nem csak futási időben. Ez természetesen nem azt jelenti, hogy *soha* nem lesz szükségünk a `Form` vagy a `QueryString` tulajdonságra az ASP.NET alatt, de egyre inkább nélkülözhetőek, és általában tetszés szerint használhatók.

Az IsPostBack tulajdonság

A `HttpRequest` másik nagyon fontos tagja az `IsPostBack` tulajdonság. Emlékezzünk vissza, hogy a „visszaküldés" arra utal, hogy visszatérünk egy adott weboldalra, miközben munkafolyamatunk még él a kiszolgálón. A definíció alapján az `IsPostBack` tulajdonság értéke `true`, ha az aktuális HTTP-kérést a pillanatnyilag bejelentkezett felhasználó küldi, és `false`, ha a felhasználó most először lép kapcsolatba az oldallal.

Annak a meghatározása, hogy a HTTP-kérés valóban visszaküldés-e, akkor a leghasznosabb, ha egy kódblokkot csak abban az esetben szeretnénk végrehajtani, mikor a felhasználó első ízben fér hozzá az adott oldalhoz. Például előfordulhat, hogy egy ADO.NET `DataSet` típust szeretnénk adatokkal feltölteni, mikor a felhasználó először fér hozzá egy `*.aspx` fájlhoz, és az objektumot szeretnénk későbbi használatra gyorsítótárazni. Az `IsPostBack` tulajdonság vizsgálatával elkerülhetjük a felesleges adatbázishoz fordulást, amikor az oldal visszaküldés miatt kerül betöltésre (persze néhány oldal megkövetelheti, hogy a `DataSet` típust minden egyes kéréskor frissítsük, de ez más kérdés). Tételezzük fel, hogy az `*.aspx` fájlunk kezelte az oldal `Load` eseményét (erre a fejezet későbbi részében részletesen visszatérünk), ekkor a visszaküldés feltételeit az alábbiak szerint programozott módon tesztelhetjük:

```
protected void Page_Load(object sender, System.EventArgs e)
{
  // A DataSet típust csak akkor töltsük fel,
  // amikor a felhasználó legelőször fér hozzá az oldalhoz.
  if (!IsPostBack)
  {
    // Töltsük fel a DataSet típust, és gyorsítótárazzuk!
  }
  // Használjuk a gyorsítótárazott DataSet típust.
}
```

Együttműködés a kimenő HTTP-válaszokkal

Most, hogy már jobban átlátjuk, a `Page` típus hogyan teszi lehetővé a bejövő HTTP-kérésekkel folytatott együttműködést, a következő lépés a kimenő HTTP-válaszok együttműködésének vizsgálata. Az ASP.NET alatt a `Page` osztály `Response` tulajdonsága biztosít hozzáférést a `HttpResponse` típus példányához. Ez a típus több olyan tulajdonságot definiál, amelyek révén formázhatjuk a kliensböngészőnek visszaküldött HTTP-válaszokat. A 31.6. táblázat néhány alapvető tulajdonságot sorol fel.

Tulajdonság	Jelentés
`Cache`	A weboldal gyorsítótárazási szemantikáját adja vissza (például lejárati idő, adatvédelem, kifejezések módosítása).
`ContentEncoding`	Lekérdezi vagy beállítja a kimeneti adatfolyam HTTP-karakterkészletét.
`ContentType`	Lekérdezi vagy beállítja a kimeneti adatfolyam HTTP MIME-típusát.
`Cookies`	Lekérdezi az aktuális kérésben küldött `HttpCookie` gyűjtemény értékét.
`IsClientConnected`	Lekérdezi, hogy a kliens kapcsolódik-e még a kiszolgálóhoz.
`Output`	Egyedi kimenetet engedélyez a kimenő HTTP-tartalomtörzs számára.
`OutputStream`	Bináris kimenetet engedélyez a kimenő HTTP-tartalomtörzs számára.

Tulajdonság	Jelentés
`StatusCode`	Lekérdezi vagy beállítja a kliensnek visszaküldött kimenet HTTP-állapotkódját.
`StatusDescription`	Lekérdezi vagy beállítja a kliensnek visszaküldött kimenet HTTP-állapotsztringjét.
`SuppressContent`	Lekérdezi vagy beállítja, hogy a HTTP-tartalmat a rendszer elküldje-e a kliensnek.

31.6. táblázat: *A HttpResponse típus tulajdonságai*

A 31.7. táblázatban tekintsünk meg a `HttpResponse` típus által támogatott metódusok közül néhányat.

Metódus	Jelentés
`AddCacheDependency()`	Ez a metódus egy objektumot ad hozzá az alkalmazás-gyorsítótárhoz (lásd a 33. fejezetet).
`Clear()`	Törli az összes fejlécet és tartalomkimenetet a pufferfolyamból.
`End()`	Ez a metódus az összes aktuálisan pufferelt kimenetet elküldi a kliensnek, majd lezárja a csatornát.
`Flush()`	Ez a metódus az összes aktuálisan pufferelt kimenetet elküldi a kliensnek.
`Redirect()`	Elirányítja a klienst az új URL-re.
`Write()`	Egy megadott értékeket ír a kimenő HTTP-folyamba.
`WriteFile()`	Ez a metódus közvetlenül a kimenő HTTP-tartalom-folyamba ír egy fájlt.

31.7. táblázat: *A HttpResponse típus metódusai*

HTML-tartalom kibocsátása

A `HttpResponse` típusnak talán a legismertebb jellegzetessége, hogy képes a tartalmat közvetlenül a HTTP-kimenetfolyamba írni. A `HttpResponse.Write()` metódus révén bármilyen HTML-címkét és/vagy -szöveget átadhatunk.

A `HttpResponse.WriteFile()` metódus annyiban viszi ezt tovább, hogy megadhatjuk annak a fizikai fájlnak a nevét a webkiszolgálón, amelynek tartalmát szeretnénk megjeleníteni a kimeneti folyamban (ez elég hasznos egy meglévő `*.htm` fájl tartalmának gyors kibocsátásakor).

Ennek bemutatásához tételezzük fel, hogy újabb gombot adtunk az aktuális `*.aspx` fájlhoz, amely a következőképpen valósítja meg a kiszolgálóoldali kattintási esemény kezelőjét:

```
protected void btnHttpResponse_Click(object sender, EventArgs e)
{
  Response.Write("<b>My name is:</b><br>");
  Response.Write(this.ToString());
  Response.Write("<br><br><b>Here was your last request:</b><br>");
  Response.WriteFile("MyHTMLPage.htm");
}
```

A segédfüggvény (amelyet feltehetően valamilyen kiszolgálóoldali eseménykezelő hív meg) szerepe elég egyszerű. Az egyetlen érdekessége, hogy a `HttpResponse.WriteFile()` metódus a webhely gyökérkönyvtárában bocsátja ki a kiszolgálóoldali `*.htm` fájl tartalmát.

Ismétlésképpen, annak ellenére, hogy bármikor alkalmazhatjuk ezt az elavult módszert, és a HTML-címkéket és -tartalmat megjeleníthetjük a `Write()` metódus segítségével, ez az eljárás már jóval kevésbé elterjedt az ASP.NET alatt, mint a klasszikus ASP esetében volt. Ennek oka a kiszolgálóoldali webes vezérlőelemek megjelenése. Ezért, ha egy szöveges adatblokkot szeretnénk a böngészőben megjeleníteni, csak rendeljünk hozzá egy sztringet a Label vezérlő Text tulajdonságához.

Felhasználók átirányítása

A `HttpResponse` típus másik jellegzetessége, hogy képes a felhasználót átirányítani egy új URL-re:

```
protected void btnSomeTraining_Click(object sender, EventArgs e)
{
  Response.Redirect("http://www.intertech.com");
}
```

Ha az eseménykezelőt egy ügyféloldali visszaküldés hívja meg, a rendszer a felhasználót automatikusan átirányítja a megadott URL-re.

Megjegyzés A `HttpResponse.Redirect()` metódus mindig igényel egy újabb kört a kliens-böngésző felé. Ha egyszerűen csak szeretnénk átadni a vezérlést ugyanazon virtuális könyvtár egy másik `*.aspx` fájljának, az a `HttpServerUtility.Transfer()` metódus (az öröklött `Server` tulajdonság révén érhető el) használata sokkal hatékonyabb.

Ennyit a `System.Web.UI.Page` működésének vizsgálatáról. A `System.Web.UI.Control` ősosztály szerepét a következő fejezetben tanulmányozzuk át. Ezek után vizsgáljuk meg egy `Page`-leszármazott objektum életét.

Forráskód A FunWithPageMembers kódfájlokat a forráskódkönyvtár 31. fejezetének alkönyvtára tartalmazza. A forráskódkönyvtárról lásd a Bevezetés xlv. oldalát.

Az ASP.NET-weboldalak életciklusa

Minden ASP.NET-weboldal életciklusa rögzített. Amikor az ASP.NET-futtatókörnyezethez bemeneti kérés érkezik egy adott `*.aspx` fájlra vonatkozóan, a kapcsolódó `Page`-leszármazott `System.Web.UI.` típus a memóriába kerül a típus alapértelmezett konstruktorának révén. A keretrendszer ezután egy sor eseményt indít el automatikusan. A `Load` eseménykezelő alapértelmezés szerint rendelkezésünkre áll, amelyben megadhatjuk az egyedi kódunkat:

```
public partial class Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    Response.Write("Load event fired!");
  }
}
```

A `Page` típusok a `Load` eseményen kívül képesek elkapni bármely, a 31.8. táblázatban kiváltódási sorrendben felsorolt eseményt (az oldalak életciklusa alatt felmerülő eseményekkel kapcsolatos további információkért lapozzuk fel a .NET Framework 3.5 SDK dokumentációját).

Esemény	Jelentés
`PreInit`	A keretrendszer a webes vezérlőelemek allokálására, témák alkalmazására, a mesteroldal és a felhasználói profilok beállítására használja ezt az eseményt. Az esemény kezelésével testre szabhatjuk a folyamatot.

Esemény	Jelentés
`Init`	A keretrendszer a webes vezérlőelemek korábbi értékeinek visszaállítására használja ezt az eseményt visszaküldés vagy viewstate révén.
`Load`	Amikor ez az esemény bekövetkezik, az oldalt és a vezérlőelemeit a rendszer teljesen inicializálja, és visszaállítja a korábbi értékeiket. Ekkor biztonságosan együttműködhetünk az összes webes vezérlőelemmel.
„A visszaküldést kiváltó esemény"	Ilyen nevű esemény természetesen nincs. Ez az „esemény" arra utal, hogy melyik esemény késztette a böngészőt a webkiszolgálóra való visszaküldés végrehajtására (mint például egy Button kattintás).
`PreRender`	Az összes vezérlőelem-adatkötés és felhasználóifelület-konfiguráció megtörtént, és a vezérlőelemek készen állnak az adataik továbbítására a kimenő HTTP-válaszba.
`Unload`	Az oldal és vezérlőelemei befejezték a renderelési folyamatot, és az oldalobjektum megsemmisítés előtt áll. Ha ekkor kommunikálunk a kimenő HTTP-válasszal, futási idejű hiba lép fel. Azonban kezelhetjük ezt az eseményt bármilyen oldalszintű takarítás végrehajtása érdekében (fájl- vagy adatbázis-kapcsolatok lezárása, naplózási tevékenység végrehajtása, objektumok eldobása stb.) is.

31.8. táblázat: *A Page típus néhány eseménye*

Amikor egy C#-programozónak a `Load` eseményen kívül más eseményt kell kezelnie, meglepődhet, hogy azt az IDE nem támogatja! Ehelyett manuálisan kell megírnunk egy Page_*EseményNeve* nevű metódust a kódfájlban. Az `Unload` eseményt például így kezelhetjük:

```
public partial class Default : System.Web.UI.Page
{
  protected void Page_Load(object sender, EventArgs e)
  {
    Response.Write("Load event fired!");
  }
  protected void Page_Unload(object sender, EventArgs e)
  {
    // Már nem lehet adatokat továbbítani a HTTP-válaszba,
    // így egy helyi fájlba írunk.
    System.IO.File.WriteAllText(@"C:\MyLog.txt", "Page unloading!");
  }
}
```

Megjegyzés A `Page` típus minden eseménye együttműködik a `System.EventHandler` metódusreferenciával, ezért az eseményeket kezelő szubrutinok első paramétere mindig egy `Object` típus, a második pedig egy `EventArgs`.

Az AutoEventWireup attribútum szerepe

Amikor eseményeket szeretnénk kezelni az oldalunkon, ki kell egészítenünk a `<script>` blokkunkat vagy a mögöttes kódfájlunkat a megfelelő eseménykezelővel. Azonban, ha megvizsgáljuk a `<%@Page%>` direktívát, egy `AutoEventWireUp` nevű attribútumot figyelhetünk meg, amelynek alapértelmezett értéke true:

```
<%@ Page Language="C#" AutoEventWireup="true"
    CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

Ezzel az alapértelmezés szerinti viselkedéssel minden egyes oldalszintű eseménykezelőt automatikusan kezelünk, ha megadjuk a megfelelően elnevezett metódust. Azonban, ha kikapcsoljuk az `AutoPageWireUp` attribútumot azzal, hogy `false` értékre állítjuk:

```
<%@ Page Language="C#" AutoEventWireup="false"
    CodeFile="Default.aspx.cs" Inherits="_Default" %>
```

többé nem kapjuk el az oldalszintű eseményeket. Ahogyan a neve is sugallja, ez az attribútum (ha engedélyezett) generálja a szükséges eseményeket a fejezet korábbi részében említett, automatikusan generált részleges osztályban. Az oldalszintű eseményeket akkor is feldolgozhatjuk, ha kikapcsoltuk az `AutoEventWireup` attribútumot, méghozzá a C# eseménykezelő logikája segítségével, például a következőképpen:

```
public _Default()
{
  // Explicit módon kapcsolódjunk a Load és az Unload eseményekhez.
  this.Load += new EventHandler(Page_Load);
  this.Unload += new EventHandler(Page_Unload);
}
```

Ahogy gyaníthattuk, általában engedélyezzük az `AutoEventWireup` attribútumot.

Az Error esemény

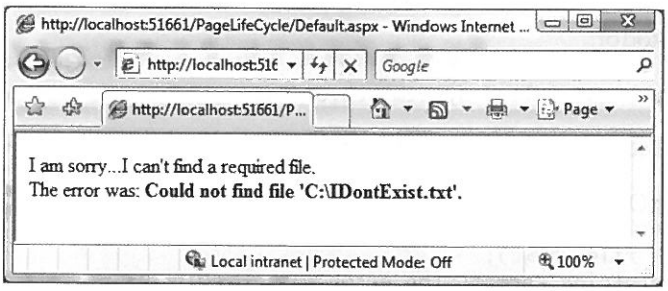
Egy másik esemény, amely bekövetkezhet az oldal életciklusa alatt, az `Error`. Ez az esemény akkor következik be, ha a `Page`-leszármazott típus metódusa olyan kivételt jelez, amelyet nem kezeltünk explicit módon. Tételezzük fel, hogy kezeltük az oldalon egy adott gomb a kattintási eseményét, és az eseménykezelőben (amelyet `btnGetFile_Click` névre kereszteltünk) megpróbáljuk kiírni egy helyi fájl tartalmát a HTTP-válaszba.

Tételezzük fel azt is, hogy *nem sikerült* tesztelnünk a fájl meglétét szabványos strukturált kivételkezeléssel. Ha az alapértelmezett konstruktorban létrehoztuk az oldal `Error` eseményét, még van rá esélyünk, hogy megoldjuk a problémát, mielőtt a végfelhasználó egy csúnya hibával találkozna. Tekintsük meg a következő kódot:

```
public partial class _Default : System.Web.UI.Page
{
  void Page_Error(object sender, EventArgs e)
  {
    Response.Clear();
    Response.Write("I am sorry...I can't find a required
                   file.<br>");
    Response.Write(string.Format("The error was: <b>{0}</b>",
                   Server.GetLastError().Message));
    Server.ClearError();
  }

  protected void Page_Load(object sender, EventArgs e)
  {
    Response.Write("Load event fired!");
  }

  protected void Page_Unload(object sender, EventArgs e)
  {
    // Már nem továbbíthatunk adatokat a HTTP-válaszba,
    // ezért helyi fájlba fogjuk írni.
    System.IO.File.WriteAllText(@"C:\MyLog.txt", "Page unloading!");
  }

  protected void btnPostback_Click(object sender, EventArgs e)
  {
    // Semmi nem történik, ezzel csak visszaküldést
    // biztosítunk az oldalra.
  }
  protected void btnTriggerError_Click(object sender, EventArgs e)
  {
    System.IO.File.ReadAllText(@"C:\IDontExist.txt");
  }
}
```

Figyeljük meg, hogy az `Error` eseménykezelő azzal kezdi, hogy a HTTP-válasz aktuális tartalmát törli, és generikus hibaüzenetet küld. Ha szeretnénk hozzáférni a `System.Exception` objektumhoz, azt az öröklött Server tulajdonság által kibocsátott `HttpServerUtility.GetLastError()` metódus segítségével tehetjük meg.

Végezetül figyeljük meg, hogy mielőtt kilépünk ebből a generikus hibakezelőből, explicit módon hívjuk a `HttpServerUtility.ClearError()` metódust a Server tulajdonságon keresztül. Ez azért szükséges, mert így tudatjuk a futtatókörnyezettel, hogy foglalkoztunk a problémával, és nem igényel további intézkedést. Ha ezt elfelejtjük megtenni, a végfelhasználó számára a futtatókörnyezet hibalapja jelenik meg. A 31.20. ábra a hibaelkapó logika eredményét mutatja be.



31.20. ábra: *Oldalszintű hibakezelés*

Ezek után már magabiztosak lehetünk az ASP.NET `Page` típusának felépítését illetően. Ilyen alapok birtokában figyelmünket az ASP.NET webes vezérlőelemeinek, témáinak és mesteroldalainak szerepére fordíthatjuk, amelyekkel a következő fejezet témakörei foglalkoznak. A fejezet lezárásaként azonban vizsgáljuk meg a `Web.config` fájl szerepét.

Forráskód A PageLifeCycle kódfájlokat a forráskódkönyvtár 31. fejezetének alkönyvtára tartalmazza. A forráskódkönyvtárról lásd a Bevezetés xlv. oldalát.

A Web.config fájl szerepe

Alapértelmezés szerint az összes Visual Studio 2008 használatával létrehozott C# ASP.NET webalkalmazás automatikusan tartalmaz egy `Web.config` fájlt. Azonban, ha bármikor szeretnénk manuálisan beszúrni egy `Web.config` fájlt a webhelyre (például amikor egyoldalas modellel dolgozunk, és nem hoztunk létre

solutiont), azt a Web Site > Add New Item menüpont segítségével tehetjük meg. Mindkét esetben hozzáadhatunk beállításokat a `Web.config` fájl hatókörén belül, amelyekkel a webalkalmazás futási idejű működését szabályozhatjuk.

Megjegyzés A webalkalmazásnak nem kell feltétlenül tartalmaznia `Web.config` fájlt. Ha nincs ilyen fájlunk, a webhelyünk a .NET könyvtárában található `machine.config` fájljában rögzített, alapértelmezett webközpontú beállításokkal rendelkezik.

Emlékezzünk vissza a .NET-szerelvényekkel kapcsolatos tanulmányainkra (a 15. fejezetben), ahol azt mondtuk, hogy a kliensalkalmazások XML-alapú konfigurációs fájl révén utasítják a CLR-t a kötési kérések, szerelvény-próbavételezések és egyéb futás közbeni események kezelésének módjára. Ugyanez érvényes az ASP.NET-webalkalmazásokra is, azzal a jelentős különbséggel, hogy a webközpontú konfigurációs fájlok neve mindig `Web.config` (az `*.exe` konfigurációs fájloktól eltérően, amelyek neve a kapcsolódó kliens végrehajtható fájlon alapul).

A `Web.config` fájlok alapértelmezett szerkezete a .NET 3.5 verziójában meglehetősen részletes, de a lényeges beállításokat a következőkben bemutatjuk. A 31.9. táblázat a `Web.config` fájlok legérdekesebb elemei közül sorol fel néhányat.

Elem	Jelentés
`<appSettings>`	Ez az elem olyan név/érték párokat hoz létre, amelyeket programozott módon beolvashatunk a memóriába a `ConfigurationManager` típus segítségével.
`<authentication>`	Ezt a biztonsággal kapcsolatos elemet a webalkalmazás hitelesítési üzemmódjának definiálására használjuk.
`<authorization>`	Ez egy másik biztonságközpontú elem, amellyel azt definiáljuk, hogy mely felhasználók mely erőforrásokhoz férhetnek hozzá a webkiszolgálón.
`<connectionStrings>`	Ebben az elemben a webhelyen használt külső kapcsolatsztringeket tároljuk.
`<customErrors>`	Ez az elem tudatja a futtatórendszerrel, hogy pontosan hogyan jelenítse meg azokat a hibákat, amelyek a webalkalmazás működése közben lépnek fel.
`<globalization>`	Ezzel az elemmel a webalkalmazás globalizációs beállításait konfiguráljuk.

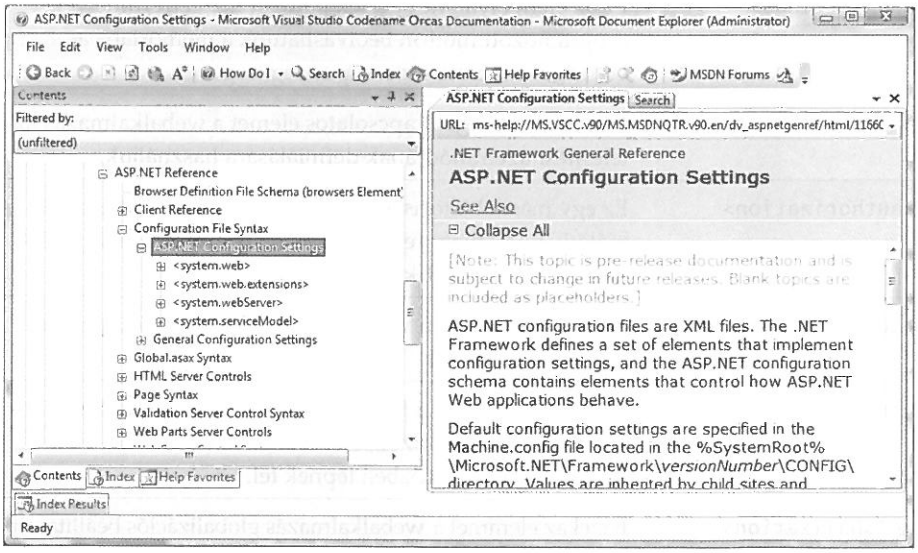
Elem	Jelentés
`<namespaces>`	Ez az elem az alkalmazni kívánt névtereket ismerteti, amelyeket importálnunk kell, ha a webalkalmazásunkat előfordítottuk az új `aspnet_compiler.exe` parancssori eszközzel.
`<sessionState>`	Ezzel az elemmel szabályozhatjuk, hogy a .NET-futtatórendszer hogyan és hol tárolja a munkamenet-állapot adatait.
`<trace>`	Ezzel az elemmel engedélyezzük (vagy letiltjuk) a webalkalmazás nyomkövetési támogatását.

31.9. táblázat: *A Web.config fájlok néhány eleme*

A `Web.config` fájlok, a 31.9. táblázatban felsoroltakon kívül további elemeket is tartalmazhatnak. Ezeknek az elemeknek a nagy része biztonságközpontú, míg a fennmaradóak csak speciális ASP.NET-helyzetekben hasznosak, mint például egyedi HTTP-fejlécek vagy egyedi HTTP-modulok létrehozása során (olyan témákban, amelyeket nem érintünk).

Ha a `Web.config` fájlokban megtalálható összes elemet (és a kapcsolódó attribútumokat) szeretnénk megtekinteni, lapozzuk fel a .NET Framework 3.5 SDK dokumentációját. Keressük az „ASP.NET Configuration Settings" témát, ahogy a 31.21. ábra mutatja, és tanulmányozzuk részletesen.

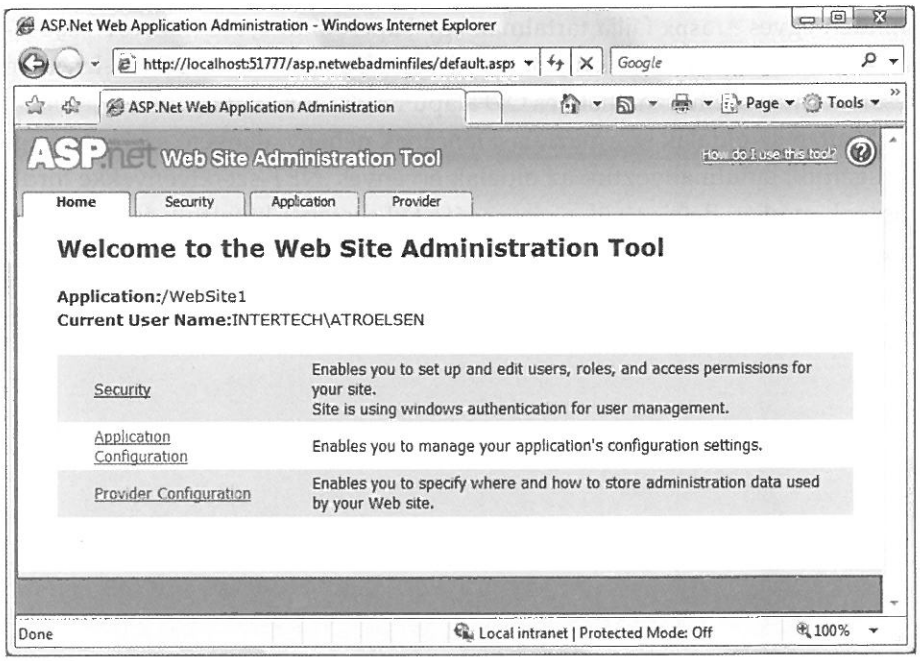
A Web.config fájl különféle jellegzetességeivel a könyv további részeiben fogunk megismerkedni.



31.21. ábra: *A Web.config fájlok dokumentációjának részletei*

Az ASP.NET Website Administration segédprogramja

Habár a Web.config fájlok tartalmát bármikor közvetlenül módosíthatjuk a Visual Studio 2008 segítségével, az ASP.NET-webprojektek hasznos web-alapú szerkesztőt használhatnak, amely révén grafikus módon szerkeszthetjük a projekt `Web.config` fájljának több elemét és attribútumát. Az eszköz elindításához a 31.22. ábrán látható módon aktiváljuk a Web Site > ASP.NET Configuration menüpontot.



31.22. ábra: *Az ASP.NET Web Site Administration eszköze*

Ha az oldal felső részén található fülekre kattintunk, hamar észrevehetjük, hogy az eszköz funkcionalitása nagyrészt a webhely biztonsági beállításainak létrehozására alkalmas. Ez az eszköz azonban azt is lehetővé teszi, hogy beállításokat adjunk hozzá az `<appSettings>` elemhez, hibakeresési és nyomkövetési beállításokat definiáljunk, és létrehozzunk egy alapértelmezett hibalapot.

Az eszköz működését majd akkor láthatjuk részletesebben, ha szükséges. Azonban jó, ha tudjuk, hogy ez a segédprogram *nem teszi lehetővé* az összes lehetséges beállítás hozzáadását a `Web.config` fájlhoz. Biztosan kerülünk olyan helyzetbe, amikor egy választott szövegszerkesztő segítségével kézzel kell módosítanunk ezt a fájlt.

Összefoglalás

A webalkalmazások létrehozásához más gondolkodásmódra van szükség, mint a hagyományos asztali alkalmazások összeállításakor. Ezt a fejezetet néhány alapvető webes téma - köztük a HTML, a HTTP, az ügyféloldali szkriptírás szerepe és a kiszolgálóoldali szkriptírás a klasszikus ASP segítségével - gyors, áttekinthető bemutatásával kezdtük. A fejezet nagy része az ASP.NET-oldalak architektúráját vizsgálta. Ahogy azt láthattuk, a projektünk minden egyes `*.aspx` fájlja tartalmaz egy kapcsolódó `System.Web.UI.Page`-leszármazott osztályt. Ennek az OO módszernek a révén az ASP.NET lehetővé teszi, hogy újrafelhasználható és OO alapú rendszereket építsünk fel.

Miután az oldalak származtatási láncának néhány alapvető szerepét megvizsgáltuk, tanulmányoztuk az oldalak érvényes .NET-szerelvényekké fordításának módját. Befejezésül a `Web.config` fájl szerepét kutattuk, és áttekintettük az ASP.NET Web Site Administration eszközét.