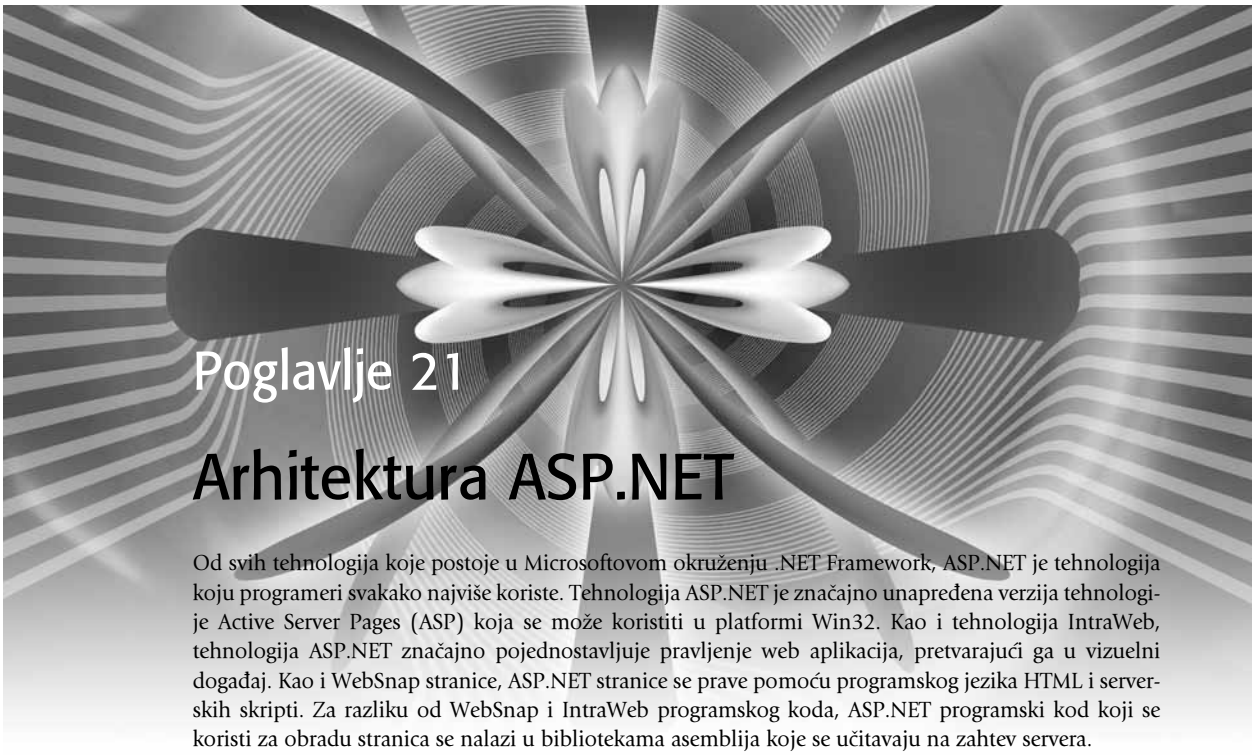


Poglavlje 21

Arhitektura ASP.NET

Od svih tehnologija koje postoje u Microsoftovom okruženju .NET Framework, ASP.NET je tehnologija koju programeri svakako najviše koriste. Tehnologija ASP.NET je značajno unapređena verzija tehnologije Active Server Pages (ASP) koja se može koristiti u platformi Win32. Kao i tehnologija IntraWeb, tehnologija ASP.NET značajno pojednostavljuje pravljenje web aplikacija, pretvarajući ga u vizuelni događaj. Kao i WebSnap stranice, ASP.NET stranice se prave pomoću programskog jezika HTML i serverskih skripti. Za razliku od WebSnap i IntraWeb programskog koda, ASP.NET programski kod koji se koristi za obradu stranica se nalazi u bibliotekama asemblija koje se učitavaju na zahtev servera.

U ovom poglavlju ću objasniti osnove arhitekture ASP.NET, kao što sam to u prethodnom poglavlju uradio za druge web tehnologije koje možete koristiti u Delphiju. Iako sam ja za arhitekturu ASP.NET odvoji posebno poglavlje, i opisao Borlandove DBWeb komponente, ovo poglavlje nije detaljan opis arhitekture ASP.NET.

Podrška koja u Delphiju postoji za arhitekturu ASP.NET je ogromna. Za razliku od biblioteke WebForms, podrška za korisnički interfejs web aplikacije obuhvata HTML/ASP.NET dizajnere, specifične editore i dodatne komponente. Programiranje ASP.NET aplikacija u Delphiju 2005 (pomoću programskih jezika Delphi i C#) je veći vizuelan doživljaj nego u bilo kom drugom IDE-u za .NET.

U ovom poglavlju ćete naučiti osnove arhitekture ASP.NET, kako se prave ASP.NET stranice i kompletne aplikacije, naučićete kako se upravlja sesijama i korisnicima, kako da koristite Borlandove kontrole, kako da koristite Delphijevog čarobnjaka Deploymnet Wizard, i još mnogo toga.

Osnove arhitekture ASP.NET

ASP.NET je naslednik Microsoftove tehnologije ASP, rešenja za serverske skripte koje se mogu unaprediti korišćenjem COM tehnologije za obradu podataka na serveru.

Dok se u tehnologiji ASP često istovremeno koristi HTML programski kod i serverske skripte, pri čemu se često koriste SQL iskazi za rad sa bazom podataka, u tehnologiji ASP.NET je jasnije razdvojena struktura stranice od obrade podataka na serveru. Oba elementa se mogu zapisati u jednu .aspx datoteku, ali se savetuje da strukturu stranice zapišete u aspx datoteku, a da se programski kod zapiše u standardnu datoteku sa izvornim programskim kodom koja se kompajlira u .NET biblioteku koju prosleđujete. Takvo razdvajanje programskog koda od strukture stranice se naziva code behind. (Ukoliko koristite jednu .aspx datoteku, u kojoj se nalazi i programski kod, onda se asembli generiše u pozadini.) Upamtite da u Delphiju 2005 i Microsoftovom Visual Studio .NET-u možete da koristite samo ovakav model.

U tehnologiji ASP.NET se koristi biblioteka WebForms, što je još veća razlika u odnosu na tehnologiju ASP. Kao što pogađate, biblioteka WebForms (koja je definisana u imenovanom prostoru System.Web.UI) je parnjak biblioteke WinForms, i obezbeđuje sličan RAD pristup programiranju. Kao što ste naučili u prethodnom poglavlju, i u tehnologiji IntraWeb se koriste formulari, ali se IntraWeb udaljava od standardnih HTML datoteka (osim kada koristite šablone), ASP.NET WebForms je blisko povezan sa web stranicom.

Arhitektura ASP.NET i web serveri

Za sada postoji samo jedan server koji podržava arhitekturu ASP.NET. To je Microsoftov server Internet Informational Services (IIS). Možete koristiti server Cassini, server koji je Microsoft napravio pomoću programskog jezika C# u platformi .NET, ali samo dok pravite aplikacije i za jednostavna debagiranja. Server Cassini se ne može koristiti za prosleđivanje datoteka, jer se može izvršavati samo na lokalnom računaru.

NAPOMENA

Druga mogućnost koja je u razvoju je korišćenje servera Apache na kojem se nalaze ASP.NET aplikacije koje su prosleđene za Mono (`www.mono-project.com/ASP.NET`), kako za operativni sistem Linux tako i za operativni sistem Windows. To je veoma interesantan napredak, jer je moj utisak da je prosleđivanje datoteka pomoću servera IIS u operativnom sistemu Windows najveći nedostatak tehnologije ASP.NET.■

Server IIS ne možete automatski koristiti za ASP.NET u operativnom sistemu Windows 2000 ili Windows XP, već kada u računar instalirate okruženje .NET Framework onda se konfiguracija servera IIS pravilno ažurira. Ukoliko ste isključili server IIS, pročitajte Delphijevu datoteku `readme.txt`, i uradite ono što piše u odeljku "Internet and HTML notes" ("Napomene za Internet i HTML").

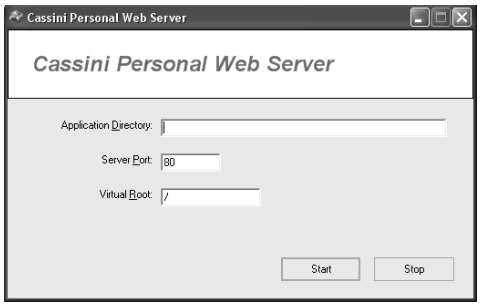
Ukoliko želite da isprobate server Cassini, ne morate ga preuzimati sa Weba, jer se prilikom instaliranja Delphija njegov programski kod instalira u poddirektorijum Cassini direktorijuma Demos. U tom direktorijumu ćete pronaći datoteku build.bat. Datoteka se lako koristi sa komandne linije, ukoliko se u putanji nalaze direktorijumi sa izvršnim datotekama okruženja .NET Framework (koje se nalaze u direktorijumu `Windows\Microsoft.NET\Framework\v1.1.4322`), kao i .NET SDK alatki (koje se nalaze u direktorijumu Program `Files\Microsoft.NET\SDK\v1.1\Bin`). (Pročitajte u odeljku "Interop biblioteke tipova i omotači biblioteke COM koji se mogu pozivati (CCW)", u Poglavlju 12, kako da sa komandne linije zadate ove direktorijume pomoću SDK .bat datotke.)

Ja sam za primer u knjizi koristio server Cassini za debagiranje programskog koda, jer se taj server može koristiti u Delphijevom IDE-u, a konfigurisanje i instaliranje je jednostavno. I vama bi trebalo da bude lakše da server Cassini koristite za primere, jer nećete morati da promenite konfiguraciju servera ISS kako biste mogli da ih vidite. Da biste proverili da li je server Cassini pravilno kompajliran, treba da pokrenete datoteku `CassiniWebServer.exe` koja se dobija kao rezultat kompajliranja. Ukoliko za direktorijum aplikacije, port servera i virtuelni koreni direktorijum ne zadate odgovarajuće parametre, prikazaće se prozor u kojem to možete da uradite (pogledajte sliku 21.1).

Pošto je server Cassini kompajliran i spreman za upotrebu, možete konfigurisati Delphijev IDE kako biste ga koristili tako što ćete zadati direktorijum u kojem je server kompajliran i TCP/IP port koji će server koristiti. Podešavanja ćete pronaći na stanici ASP.NET u odeljku HTML/ASP.NET okvira za dijalog Tools➥Options.

NAPOMENA

Na istoj stranici ASP.NET okvira za dijalog Options možete za debagiranje ASP.NET aplikacija konfigurisati Delphi 2005, tako da koristi neki drugi pretraživač, a ne Internet Explorer.■



SLIKA 21.1
Cassini Web Server koji se izvršava će zatražiti parametre koje niste zadali na komandnoj liniji. Ukoliko ste parametre zadali ove opcije će biti isključene.

Web server koji odaberete za debagiranje aplikacija koje pravite se može razlikovati od ciljnog servera na koji ćete proslediti aplikaciju, jer opcije ne utiču na izvorni programski kod koji Delphi pravi za novu ASP.NET aplikaciju, već samo na podešavanja. Bez obzira na web server koji koristite prilikom pravljenja aplikacije, na kraju ovog poglavlja ću objasniti kako se pomoću Delphija prosleđuje ASP.NET aplikacija, mada sam ja to već pokazao u prethodnom poglavlju kada je trebalo proslediti IntraWeb aplikaciju. Pomoću čarobnjaka Deploymnet Wizard ASP.NET aplikaciju možete proslediti na server koji nije server koji se koristio tokom pravljenja aplikacije, što je važno samo za debagiranje.

Prazna stranica u Delphiju

Da bismo proverili da li je server dobro konfigurisan i da bismo napravili polaznu tačku za dalje objašnjenje arhitekture ASP.NET, počećemo od jednostavnog primera. Ukoliko odaberete File➥New➥ASP.NET Web Application – Delphi for .NET, u narednom okviru za dijalog možete da odaberete server koji nameravate da koristite za programiranje i direktorijum u kojem će se nalaziti datoteke:

Pošto u tom okviru za dijalog zadate informacije, Delphi će generisati potpuno nov ASP.NET projekat za koji će napraviti datoteku sa izvornim programskim kodom, WebForms1.aspx stranicu i odgovarajuću WebForms1.pas datoteku, kao i nekoliko konfiguraciskih datoteka (`web.config`, `global.asax` i odgovarajuću datoteku global.pas). Pošto se datoteke naprave, Delphi će otovoriti WebForm1 i u dizajneru ćete videti praznu HTML stranicu.

NAPOMENA

Datoteka web.config služi za konfigurisanje aplikacije. Konfiguracijske datoteke za standardne aplikacije imaju ime programa, ali im je ekstenzija .config (na primer, `myprogram.exe.config`). Pošto su ASP.NET projekti biblioteke asemblija, koriste različite šeme: konfiguracijska datoteka se zove `web.config` i nalazi se u direktorijumu u kojem se nalazi ASP.NET aplikacija.■

Možete uneti bilo šta i upotrebiti paletu alatki kako biste zadali font i druge vizuelne elemente. Rezultujuća stranica se može prikazati u pretraživaču tako što će se kompajlirani program pokrenuti u debageru (Run➥Run ili F9), i kao samostalan program (Run➥Run Without Debugging ili Shift+Ctrl+F9). U oba slučaja Delphi će pokrenuti server Cassini (i videćete korisničke interfejse, kao na slici 21.1), i pokrenuće Internet Explorer koji usmerava na zadatu stranicu, što je u ovom primeru stranica (za zadatu konfiguraciju):

```
http://localhost:8888/AspNetOne/WebForm1.aspx
```

Za početak sam u Delphijevom HTML editoru/dizajneru napisao tekst, koji je doveo do sledeće .aspx datoteke:

```
<!DOCTYPE HTML PUBLIC »-//W3C//DTD HTML 4.01 Transitional//EN«>
<html>
  <head>
    <title>AspNetOne</title>
  </head>
  <body ms_positioning=«GridLayout«>
    <form runat=«server«><font face=«Verdana« size=«5«>This
      is the AspNetOne example.</font>
    </form>
  </body>
</html>
```

U prethodnom HTML programskom kodu vidite da je stranica zaista web formular, jer se sav sadržaj koji je definisan u dizajneru nalazi u markeru `form`. Atribut `runat` je HTML proširenje pomoću kojeg se zahteva obrada na serveru. To znači da će ASP.NET mehanizam obraditi HTML, na serveru napraviti kontrolu koja predstavlja marker, i zatražiti da se kontrola renderuje tako što će generisati odgovarajuće HTML markere. Na primer, proces uklanja atribut runat, jer ničemu ne služi u HTML-u, ali se dodaju drugi atributi, kao i marker p. Evo kako izgleda HTML programski kod, na osnovu prethodnog programskog koda ASP.NET stranice, koji će dobiti pretraživač:

```
<!DOCTYPE HTML PUBLIC »-//W3C//DTD HTML 4.01 Transitional//EN«>
<html>
  <head>
    <title>AspNetOne</title>
  </head>
  <body ms_positioning=«GridLayout«>
    <form name=«_ctl0« method=«post«
        action=«WebForm1.aspx« id=«_ctl0«>
      <input type=«hidden« name=«__VIEWSTATE«
        value=«dDwtMTI3OTMzNDM4NDs7PtGYtm0s9fPMySQ++m1o7wb0xUxA« />
      <p><font face=«Verdana« size=«5«>This is
        the AspNetOne example.</font>
      </p>
    </form>
  </body>
</html>
```

Marker `_VIEWSTATE` ću objasniti u odeljku "Uvod u upravljanje stanjima". Ovo je jednostavan rezultat (primer AspNetOne ima još nekoliko osobina):

Tehnologija pisanja skriptova

Naučili ste da je ASP.NET web formular (koji se još uvek naziva stranica) definsan u .aspx datoteci. Ta datoteka je tekstualna datoteka u kojoj se koriste HTML markeri, uključujući kontrolne makere kao što su direktive i blokovi skripte, na način koji je sličan načinu koji se koristi u starijoj ASP tehnologiji.

NAPOMENA

Nema ničeg magičnog u ekstenziji .aspx. Možete zadati bilo koju ekstenziju i podesiti server IIS da je koristi.■

Na samom početku ASP.NET datoteke se nalazi deklaracija sa direktivom Page (direktive u ASP.NET tehnologiji se označavaju simbolima `<%@`), kao u sledećem programskom redu koji je deo primera AspNetOne:

```
<%@ Page Language="c#" Debug="true" Codebehind="WebForm1.pas"
AutoEventWireup="false" Inherits="WebForm1.TWebForm1"%>
```

Pomoću direktive Page se, između ostalog, zadaje jezik koji se koristi za automatsko generisanje programskog koda za izvedene stranice koji pravi sve kontrole serverske aplikacije. Pomoću direktive Page se obavljaju još neki poslovi "održavanja" (atribut Language), kao što je označavanje pridruženog kompajliranog modula i klase iz koje će se izvoditi generisane stranice (atributi `CodeBehind` i `Inherits`, koje ću objasniti u narednom odeljku).

Osim velikog broja direktiva postoje i mnogi drugi specijalni blokovi koje možete koristiti, uključujući deklaracije blokova ASP programskog koda (`<script runat="server">`), blokovi renderovanja programskog koda (koji su označeni simbolima <% i %>), proste ekstenzije za povezivanje podataka (koje su označene simbolima <%# i %>), i proširena kontrola HTML sintaksa (`<asp:button>`).

Serverske skripte se mogu direktno umetnuti u stranicu pomoću nekog od programskih jezika, kao što su programski jezici C# i VB.NET, koji se zadaju atributom Language u direktivi Page. Programski kod koji pišete se ne razlikuje mnogo od WebSnap serverskih skripti (za koje se obično koristi programski jezik JavaScript). Na primer, sve što treba da uradite kako biste prilagodili programski kod skripta primera WSnap2 (iz prethodnog poglavlja) jeste da dodate tip promenljive (int, za cele brojeve) u petlje for:

```
<table border="1">
  <tbody>
    <tr>
      <th><br></th>
      <% for (int j=1;j<=5;j++) { %>
      <th>
        Column <%=j %>
      </th>
```

```
      <% } %>
    </tr>
    <% for (int i=1;i<=5;i++) { %>
    <tr>
      <td>Line <%=i %></td>
      <% for (int j=1;j<=5;j++) { %>
      <td>Value= <%=i*j %></td>
      <% } %>
    </tr>
    <% } %>
  </tbody>
</table>
```

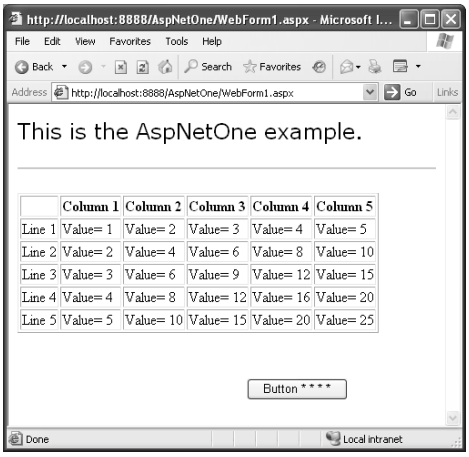
Rezultat prethodnog programskog koda vidite na slici 21.2. U rezultatu postoji komandno dugme, za koje je izmenjen naslov, što ću objasniti u narednom odeljku.

NAPOMENA

Teorijski je moguće da programski jezik Delphi koristite kao jezik skripta u ASP.NET tehnologiji, pošto instalirate Delphi provajder (što Borlandova licenca ne dozvoljava), ali je to praktično teško ostvariti i programski jezik se ne može direktno koristiti iz IDE-a. U Delphiju se programski kod izvršva u pozadini (što ćete videti u narednom odeljku) što je osnovno i preporučeno rešenje za ASP.NET programranje (koriste ga i programeri koji koriste Microsoftove alate).■

Razdvajanje programskog koda

Kao što možete videti, kada u Delphiju napravite novu ASP.NET aplikaciju, svaki formular ima svoju .aspx datoteku, čija struktura nalikuje HTML datoteci, i Delphijevu datoteku koja sadrži deklaraciju odgovarajuće klase stranice. Kao i obično, iz Delphijeve datoteke u .aspx datoteku možete preći pomoću tastera F12 (ne zaboravite, u Poglavlju 1 sam rekao da možete da koristite i kombinacije tastera Alt+PgUp i Alt+PgDn za kretanje kroz kartice koje se prikazuju u dnu Delphijevog editora).



STRANICA 21.2
Rezultat primera AspNetOne, koji se sastoji od teksta, tabele koja se zasniva na serverskom skriptu i ASP komandnog dugmeta

Kao i kod GUI formulara, klasa stranice se izvodi iz generičke klase (klase `Page`) koja postoji u biblioteci. Dok se u biblioteci WinForms koristi inicijalizcaioni programski kod za pravljenje komponenenata i zadavanje njihovih svojstava, a u biblioteci VCL koriste DFM datoteke za učitavanje podataka formulara, u tehnologiji ASP.NET se generiše izvedena klasa. Kada se pošalje zahtev za stranicom, sistem čita odgovarajuću .aspx datoteku i dinamički pravi klasu koja se izvodi iz odgovarajuće klase stranice i dodaje joj inicijalizacioni programski kod.

To se obavlja pomoću direktive Page koju sam objasnio u prethodnom odeljku, i pomoću sledeća dva atributa:

```
Codebehind=«WebForm1.pas«
Inherits=«WebForm1.TWebForm1«
```

Delphijev programski kod kompajlirate u biblioteku asemblija, koja se smešta u direktoijum `bin` koji se nalazi u virtuelnom direktorijumu web aplikacije (to je direktorijum u kojem se nalaze .aspx datoteke). Kada se zatraži ASP stranica, web server obrađuje .aspx datoteku i učitava odgovarajuću biblioteku asemblija. Za ASP kontrole .aspx datoteka sadrži njihove deklaracije, kao što je sledeća deklaracija (koja je izdvojena iz primera AspNetOne):

```
<asp:Button
  id=«Button1«
  style=«Z-INDEX: 1; LEFT: 262px; POSITION: absolute; TOP: 302px«
  runat=«server«
  text=«Button«>
</asp:Button>
```

Upamtite da se za komandno dugme koristi lokalni stil sa apsolutim pozicioniranjem (što je osnovno ponašanje koje možete izmeniti), i ima atribut `runat` kojem je zadata vrednost `server`, što znači da se komponenta obrađuje i menja u serverskoj aplikaciji kako bi postala standardna HTML kontrola. Izvršavanjem stranice komandno dugme generiše sledeći HTML programski kod:

```
<input
  type=«submit«
  name=«Button1«
  value=«Button«
  id=«Button1«
  style=«Z-INDEX: 1; LEFT: 262px; POSITION: absolute; TOP: 302px« />
```

Kao što vidite, stil se ne menja (mada to nije pravilo), dok je gotovo sve ostalo pretvoreno u standardni HTML programski kod, uključujući marker input i atribute `type`, `value` i `name`. Programski kod događaja Click komnandnog dugmeta se nalazi u Delphijevoj datoteci, zajedno sa ostalim programskim kodom i ne generiše direktno HTML programski kod. Umesto toga vi ćete pomoću programskog koda izmeniti svojstva kontrole, što se reflektuje na generisani HTML programski kod.

Ukoliko pogledate izvorni programski kod primera AspNetOne (ili bilo kojeg jednostavnog web formulara), videćete deklaraciju klase kao što je sledeća deklaracija:

```
type
  TWebForm1 = class(System.Web.UI.Page)
  strict private
    procedure InitializeComponent;
    procedure Button1_Click(sender: System.Object;e: System.EventArgs);
```

```
  strict private
    procedure Page_Load(sender: System.Object; e: System.EventArgs);
  strict protected
    Button1: System.Web.UI.WebControls.Button;
    procedure OnInit(e: EventArgs); override;
  end;
```

Za razliku od WinForms aplikacije, metod `InitializeComponent` se poziva iz obrade događaja OnInit, dok je obrada događaja Page_Load jedina prazna obrada događaja. Obrada događaja `Page_Load` se zadaje zajedno sa obradom događaja `Click` komandnog dugmeta u metodu `InitializeComponent`:

```
procedure TWebForm1.InitializeComponent;
begin
  Include(Self.Button1.Click, Self.Button1_Click);
  Include(Self.Load, Self.Page_Load);
end;
```

Ovo se značajno razlikuje od WinForms aplikacija. Pomoću inicijalizacionog programskog koda se ne prave komponente i ne zadaju se njihova svojstva. Te informacije se već nalaze u .aspx datoteci, koja je kompajlirana u izvedenu klasu lokalne klase (u ovom primeru je to klasa `TWebForm1`). U klasi postoje obrade događaja, uključujući i one koje ste vi napisali, kao što je obrada događaja `Click` koju sam ja napisao za komandno dugme (rezultat tog programskog koda vidite na slici 21.2):

```
procedure TWebForm1.Button1_Click(
  sender: System.Object; e: System.EventArgs);
begin
  Button1.Text := Button1.Text + ' *';
end;
```

Postojanje Delphijeve datoteke sa izvornim programskim kodom (koja sadrži način za rad sa bazom podataka) i posebne .aspx datoteke sa definicijom strukture web formulara svakako doprinosi boljem razdvajanju logičkog toka aplikacije i korisničkog interfejsa. Drugačije rečeno, prednost je to što se .aspx data može izmeniti, a da se pri tom ne mora menjati ili kompajlirati programski kod asemblija. Istovremeno, programski kod aplikacije se proverava prilikom kompajliranja i nema kasnog povezivanja (a provera tipova podataka se obavlja tek prilikom izvršavanja aplikacije). Ipak, najveći deo HTML programskog koda će se automatski generisati, što je naročito tačno za složene komponente, kao što su tabele, tako da postoje mnoga prilagođavanja .aspx datoteke koja dizajner ne može obaviti bez pomoći programera.

Ova arhitektura je svakako mnogo bolja od stare ASP tehnologije, u kojoj postoji samo jedan tok skripti u stranici koja sve sadrži, od korisničkog interfejsa do programskog koda za rad sa bazama podataka. ASP.NET pristup bi trebalo da bude mnogo bliži RAD programerima, uključujući programere koji koriste Visual Basic ili Delphi.

Uvod u upravljanje stanjima

Poslednji tehnički element koji želim da vam objasnim jeste upravljanje stanjima. Pošto protokol HTTP ne poznaje stanja, kako onda program AspNetOne zna koliko puta je njegovo komandno dugme upotrebljeno (to jest, koliko je zvezdica dodato tekstu komandnog dugmeta)? Ta informacija se ne nalazi na serveru već se razmenjuje između serverske aplikacije i klijentskog pretraživača pomoću sakrivenog polja `_VIEWSTATE` koje sam ranije pomenuo. Ovo je vrednost polja kada se primer AspNetOne prvi put renderuje:

```
<input
  type=«hidden«
  name=«__VIEWSTATE«
  value=«dDwtMTU3NTI5MDMzMzs7PiHiogZzBA15Uody+/El99QFnWs5« />
```

Pošto nekoliko puta kliknete komandno dugme, informacija se menja i povećava se veličina podataka (na otprilike 140 znakova pošto komandno dugme nekoliko puta kliknete, a na otprilike 200 znakova kada ga kliknete 20 puta). Pomoću skrivenog polja `_VIEWSTATE` se prati status ASP.NET formulara i njegovih kontrola. Vrednost skrivenog polja `_VIEWSTATE` predstavlja vrednosti kontrola stranice: sve kontrole šalju svoje stanje koje je kopirano u polju `_VIEWSTATE` sa osnovom 64. Iako se čini da je vrednost nečitljiva, veoma se lako dekodira i otvara. Ukoliko želite da klijentskom pretraživaču informaciju pošaljete tako da bude bezbednija, možete podesiti atribut EnableviewStateMac dirketive Page tako da koristi kodiranu vrednost `_VIEWSTATE` ili pomoću HTTP protokola.

Nepostojanje informacija na serveru dovodi do povećane skalabilnosti i fleksibilnosti, jer možete imati nekoliko servera koji odgovaraju na zahteve koji su usmereni na popularan web sajt. Korišćenje sakrivenog polja `_VIEWSTATE` je jedna od mogućih opcija za ASP.NET aplikacije, što ćete videti u odeljku "Upravljanje sesijama". U programskom kodu se ova informacija kopira u odgovarajuće kontrole, pa ukoliko pogledate svojstvo Text komandnog dugmeta, videćete tekuću vrednost za odgovarajućeg korisnika, a ne početnu vrednost. Zbog toga jednostavna obrada događaja komandnog dugmeta (koja tekućem tekstu dodaje zvezdicu) radi pravilno u jednokorisničkim i višekorisničkim okruženjima.

Pravljenje ASP.NET stranica

Pošto sam vam objasnio osnove ASP.NET aplikacija, možemo se posvetiti tehnologijama koje se koriste. Počećemo od klasa koje se koriste za predstavljanje kontrola i stranica, a kasnije ćemo preći na aplikacije koje imaju više stranica. Pošto je klasa `Control` osnovna klasa za klasu Page, kao što je to u bibliotekama WinForms i VCL, ja ću početi od biblioteke WinForms i napraviću uvod u kontrole koje možete koristiti pre nego što se pozabavim osobinama stranica (to jest, web formulara).

Klasa Control

Biblioteka WebForms, kao i biblioteka WinForms, sadrži klasu `Control`. Klasa Control je osnovna klasa za sve vizuelne kontrole i njihove kontejnere. Sve kontrole imaju veliki broj zajedničkih svojstava koja su definisana u klasi `Control`. Svaka kontrola ima identifikator (svojstvo ID) koje identifikuje kontrolu u serverskom programskom kodu (kao što svojstvo Name identifikuje kontrolu u biblioteci WinForms ili biblioteci VCL), i svojstvo `ClientID` koje se koristi u klijentskim skriptama. Svaka kontrola ima i svojstvo `Controls` koje sadrži spisak dete-kontrola, što ćete videti u primeru AspNetViewState koji ću opisati u ovom odeljku.

Važna osobina kontrola je način na koji upravljaju svojim stanjem. Pomoću svojstva `EnableViewState` tipa Boolean se određuje da li će kontrola održavati svoje stanje kada se sadržaj stranice osveži kao odgovor na operacije koje korisnik obavlja nad kontrolom (termin koji se koristi je postback). Svojstvo `EnableViewState` utiče i na dete-kontrole. Pomoću svojstva `ViewState` se definišu parovi ime/vrednost u kojima se čuvaju informacije o statusu specifične kontrole. Svojstvo ViewState je instanca klase StateBag koja sadrži brojne objekte StateItems. Vrednost svojstva ViewState se šalje i kodira (za sve kontrole) pomoću sakrivenog polja `_VIEWSTATE` stranice.

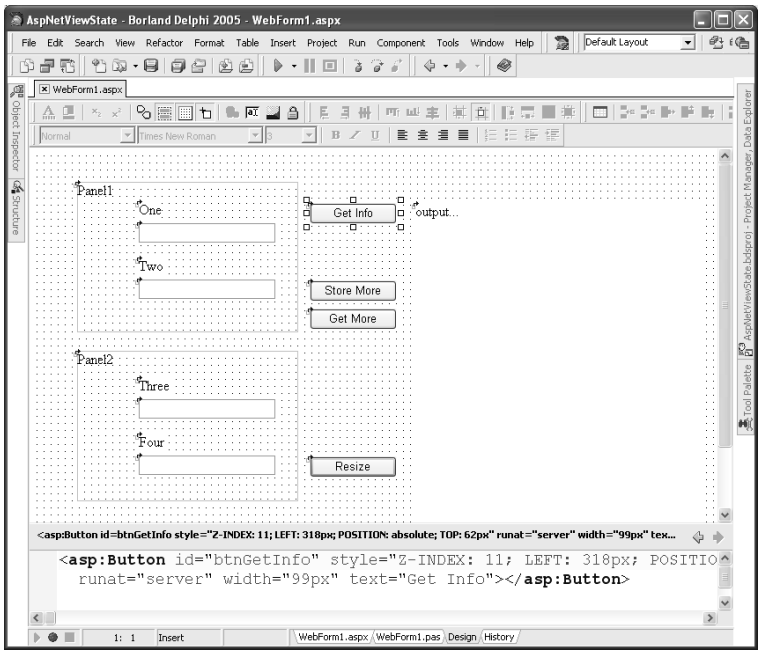
Pošto je svojstvo `ViewState` za kontrole važna osobina i teško se razume njegovo korišćenje, ja sam napravio primer AspNetViewState. U primeru AspNetViewState ću pokazati i zavisnosti roditelj/dete koje postoje između kontrola. Formular primera AspNetViewState sadrži dva panoa u kojima se nalaze tekstualna polja i po dve labele, kao i nekoliko komandnih dugmadi i izlaznih labela. Web formular prilikom njegovog dizajniranja vidite na slici 21.3.

U primeru AspNetViewState postoji komandno dugme Get Info pomoću kojeg se obavljaju dve stvari: prikazuje se spisak klasa i imena dete-kontrola prvog panoa, i prikazuje elemente svojstva `ViewState` prvog tekstualnog polja. Rezultat, za čije se formatiranje koriste markeri `<br>`, se zapisuje u objekat `StringBuilder`, a zatim se kopira u labelu, što vidite u narednom programskom kodu:

```
procedure TWebForm1.btnGetInfo_Click(
  sender: System.Object; e: System.EventArgs);
var
  strOutput: StringBuilder;
  Ctrl: Control;
  key: string;
begin
  strOutput := StringBuilder.Create;
  strOutput.Append('Panel Child Controls Count: ');
  strOutput.Append(Panel1.Controls.Count);
  strOutput.Append('<br>');
  for Ctrl in Panel1.Controls do
  begin
    strOutput.Append(Ctrl.ToString); // long class name
    strOutput.Append(': ');
    strOutput.Append(Ctrl.ID);
    strOutput.Append('<br>');
  end;

  strOutput.Append('<br>');
  strOutput.Append('TextBoxOne.ViewState Count: ');
  strOutput.Append(TextBoxOne.ViewState.Count);
  strOutput.Append('<br>');
  for key in TextBoxOne.ViewState.Keys do
  begin
    strOutput.Append(key + ': ');
    strOutput.Append(TextBoxOne.ViewState.Item[key]);
    strOutput.Append('<br>');
  end;

  LabelOutput.Text := strOutput.ToString;
end;
```



SLIKA 21.3
Web formulari primera AspNetViewState tokom njihovog dizajniranja u Delphijevom IDE-u

Pomoću programskog koda se prolazi kroz dete-kontrole panoa i ključeve svojstva ViewState prvog tekstualnog polja. Ukoliko odmah nakon pokretanja programa kliknete komandno dugme, dobićete ovakav rezultat:

```
Panel Child Controls Count: 9
System.Web.UI.LiteralControl:
System.Web.UI.WebControls.TextBox: TextBoxOne
System.Web.UI.LiteralControl:
System.Web.UI.WebControls.TextBox: TextBoxTwo
System.Web.UI.LiteralControl:
System.Web.UI.WebControls.Label: Label1
System.Web.UI.LiteralControl:
System.Web.UI.WebControls.Label: Label2
System.Web.UI.LiteralControl:

TextBoxOne.ViewState Count: 0
```

Primetili ste da sadržaj panoa obuhavata tekst (Panle1 tekst i nove redove i razmake između kontrola u .aspx datoteci). Ukoliko unesete neki tekst u prvo tekstualno polje, poslednji deo rezultata će biti:

```
TextBoxOne.ViewState Count: 1
Text: some text
```

Osim čitanja vrednosti svojstva ViewState za kontrolu, možete zapisati svoje vrednosti. Pomoću komandnog dugmeta Store More tekst drugog tekstualnog polja se zapisuje kao deo stanja prvog tekstualnog polja (bilo koji tekst):

```
procedure TWebForm1.btnStoreMore_Click(
  sender: System.Object; e: System.EventArgs);
begin
  TextBoxOne.ViewState['More'] := TextBoxTwo.Text;
end;

procedure TWebForm1.btnGetMore_Click(
  sender: System.Object; e: System.EventArgs);
begin
  LabelOutput.Text := TextBoxOne.ViewState['More'].ToString;
end;
```

Naravno, dodvanje informacija u svojstvo `ViewState` utiče na rezultat koji se dobija pomoću prvog komandnog dugmeta i još više povećava količinu podataka koja se šalje pomoću sakrivenog polja `_VIEWSTATE` formulara.

Primer AspNetViewState ima još dve osobine. Prva osobina je da drugo tekstualno polje ima svojstvo AutoPostBack kojem je zadata vrednost True. To znači da se, čim korisnik završi sa izmenama sadržaja tekstualnog polja (to jest, kada se fokus pomeri na neku drugu kontrolu), podaci šalju serveru i da se sadržaj stanice osvežava. U nekim slučajevima je to potrebno, ali će to u opštem slučaju zbuniti korisnika, a server će imati više posla. Najzad, tekstualne kontrole u drugom panou imaju svojstvo `EnableViewState` za koje je zadata vrednost False, tako da se njihovo stanje ne čuva kada server osveži sadržaj stranice. To ne znači da će se tekst koji se nalazi u tim kontrolama promeniti kada pošaljete podatke. Tekst je, zapravo, deo podataka formulara iako se ne uključuje u svojstvo `ViewState`. Ukoliko za neke kontrole isključite svojstvo `EnableViewState`, nećete moći da pratite stanje ostalih svojstava, na primer, njihovu veličinu. To sam pokazao korišćenjem komandnog dumeta Resize na formularu koje menja veličinu četiri tekstualna polja, od kojih se dva koriste za zapisivanje novih podešavanja, a ostala se vraćaju na početna podešavanja (kada je svojstvo `ViewState` isključeno).

NAPOMENA

Među metodima klase `Controls`, postoji metod Render koji vam omogućava da generišete specifian HTML programski kod za kontrole koje pravite pomoću parametra tipa `HtmlTextWriter`.■

HTML kontrole

Ukoliko pogledate paletu Tool Palette dok je dizajner za ASP.NET web formular aktivan, primetićete dva odeljka u kojima se nude komponente čija se funkcija preklapa, odeljke Webcontrols i HTML Elements. Na pravoj stranici se nalaze ASP.NET kontrole koje ću objasniti u narednom odeljku. Na drugoj stranici se nalaze standardni HTML markeri, koji se u arhtitekturi ASP.NET nazivaju HtmlControls. To su serverski objekti koji odgovaraju HTML kontrolama.

Za markere HtmlControls se koristi standardna sintaksa programskog jezika HTML, a to je obavezni identifikator i atribut runat koji se uklanja tokom obrade. Markeri HtmlControls se koriste za definisanje izgleda stranice i prosleđuju se kao da su pretraživači. Razlikuju se od standardnih ASP.NET kontrola koje tokom izvršavanja aplikacije generišu ekvivalentan HTML programski kod. Iako markeri HtmlControls ne traže mnogo resursa, uglavnom se programiraju pomoću klijentskog skripta u programskom jeziku JavaScript, a ne Delphijevog programskog koda na strani servera.

Web kontrole

S druge strane, familija WebControls obuhvata standarde ASP.NET kontrole. Neke od tih kontrola (na primer, kontrola TextBox ili kontrola Label) su slične odgovarajućim HTML kontrolama, ali se njima u potpunosti upravlja pomoću tehnologije ASP.NET i imaju osobine kao što su upravljanje stanjem i serverski događaji. Osim toga, među ASP.NET kontrolama ćete pronaći kalendare i tabele, koje su napravljene na osnovu složenih kolekcija HTML markera.

Familiju WebControls ćete lako uočiti na stranici ASPX, jer se za nju koristi sintaksa kao što je ova koja je izdvojena iz prethodnog primera:

```
<asp:Button
  id=«btnGetInfo«
  style=«Z-INDEX: 11; LEFT: 318px; POSITION: absolute; TOP: 62px«
  runat=«server«
  width=«99px«
  text=«Get Info«>
</asp:Button>
```

Postoji mnogo web kontrola koje možete koristiti u vašim aplikacijama, uključujući jednostavne kontrole, kao što je kontrola textBox koja se koristi za editovanje jednog ili više redova teksta, kontrole RadioButton i RadioButtonList, kontrole CheckBox i CheckBoxList, i kontrole DropDownList i ListBox.

Među složenijim kontrolama su kontrola AdRotator, kontrola calendar i kontrola Table. Kontrola Literal je interesantna jer se pomoću nje u stranicu ugrađuje deo standardnog HTML programskog koda. Za rad sa tabelama postoje tri specifične kontrole, kontrole DataGrid, DataList i Repeater (koje ću objasniti u odeljku "Korišćenje baza podataka" i primeru AspNetDataOne).

Klasa Page

Kao što sam već pomenuo, klasa `Page` je izvedena iz klase `Control`, kao što je klasa form izvedena iz klase Control u biblioteci WinForms (odnosno, klasa `TForm` je izvedena iz klase `TControl` u biblioteci VCL). Pored toga što svojstva osnovne klase postoje u svakoj kontroli, klasa Page ima nekoliko sopstvenih svojstava, uključujući svojstvo `ErrorPage`, pomoću kojeg se naznačava stranica na koju se preusmerava pretraživač kada se dogodi greška, svojstvo `IsPostBack`, pomoću kojeg se naznačava da li se stranici pristupa prvi put i kolekcija `Validators` koja se koristi za proveravanje serverskih kontrola koje se nalaze na stranici.

Stranica direktno pristupa tekućem HTTP zahtevu i odgovor šalje pomoću skupa objekata koji je poznat ASP programerima, na primer, pomoću objekata `Application`, `server`, `sessin`, `Request` i `Response`, kao i pomoću novih objekata `Cache`, `User` i `Trace`.

U primeru AspNetReqResp ćete videti kako možete direktno da pristupite obektima `Request` i `Response`. Aplikacija AspNetReqResp je oblik ekstremnog slučaja, jer se u njoj ne koristi ni jedna ASP kontrola. Stranica je prazna, a jedina obrada događaja (za događaj Load) se piše direktno u HTML rezultat:

```
procedure TWebForm1.Page_Load(
  sender: System.Object; e: System.EventArgs);
var
  str: string;
begin
  Response.Write('<h2>AspNetReqResp</h2>');
```

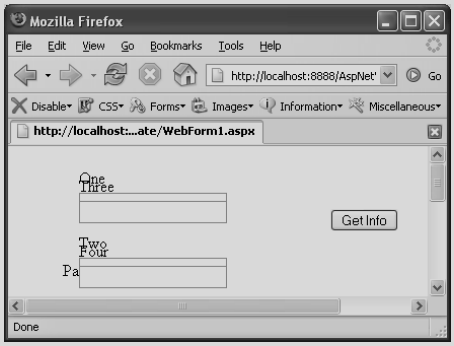
```
  // description omitted...

  for str in Request.QueryString do
  begin
    Response.Write('<p>');
    Response.Write(str); // name
    Response.Write(' = ');
    Response.Write(Request.QueryString[str]); // value
    Response.Write('</p>');
  end;
end;
```

TRIKOVI U PRETRAŽIVAČU KOJI SE OSTVARUJU POMOĆU ASP.NET KONTROLA

Serverske kontrole ne zavise od pretraživača koji se koristi, bar se tako u teoriji tvrdi. Međutim, mnoge od tih kontrola koriste svojstvo Browser objekta Request kako bi saznale koji pretraživač koristi korisnik, i kako bi generisale HTML koji odgovara sposobnostima pretraživača. Problem je to što izgleda da ASP.NET malo zna o (ili ne želi da pruži podršku) novim pretraživača koji postoje van Microsoftovog carstva. Nije problem to što pretraživač ne podržava neku osobinu, što ne bi bilo iznenađenje, već to što kada se koristi takav pretraživač onda u programskom kodu HTML nedostaju neke njegove standardne osobine.

Da biste videli primer pokrenite poslednji program (AspNetViewState) u Mozillinom pretraživaču Firefox i videćete da su sva tekstualna polja panoa ispremeštana:



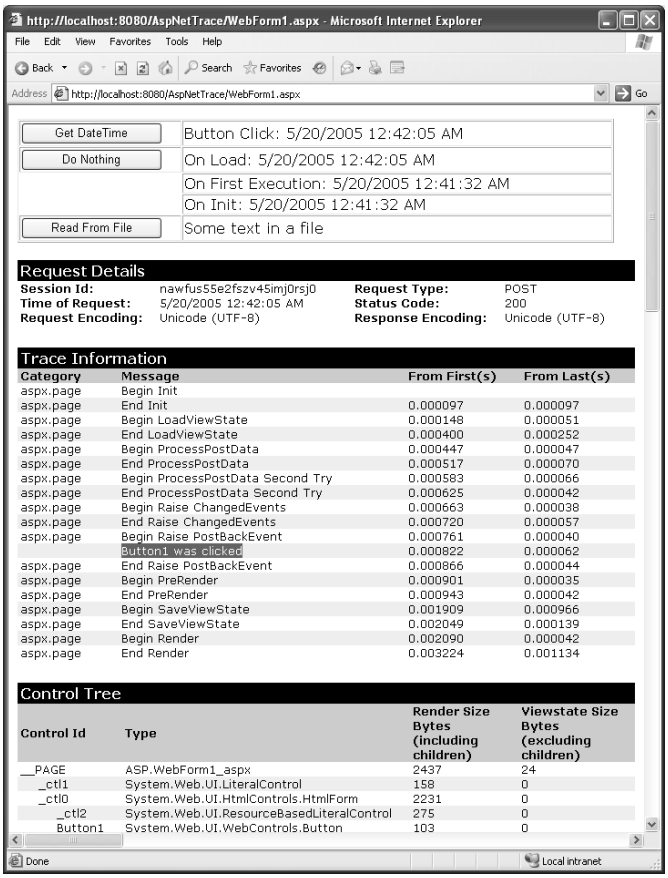
Isprva ćete pomisliti da je problem u pretraživaču Firefox, ali on zavisi od programskog koda HTML koji je generisao ASP.NET, koji pak zavisi od korisničkog agenta koji je naznačen u HTTP zahtevu. Ukoliko izmenite agenta koji Firefox šalje serveru kako bi se predstavio kao Internet Explorer, stranica će se pravilno prikazati i sve ostalo u ASP.NET aplikacijama će funkcionisati pravilno. (Promena agenta nije jedna od osnovnih osobina pretraživača Firefox već je dodatak, kao što je Agent Switcher Extension Krisa Pederika (Chris Pederick)).

Ja lično smatram da je to veliki problem tehnologije ASP.NET. Tačno je da ukoliko pažljivo napravite stranice, koje su međusobno slične, i koristite dodatne ASP.NET kontrole sa dobrom podrškom za razne pretraživače, onda možete da generišete stranice koje se pravilno prikazuju u raznim pretraživačima. Međutim, trud koji ćete uložiti ne bi bio potreban da je ASP.NET napravio standardan HTML programski kod za svaki pretraživač.

Druga interesantna (za programere) osobina jeste mogućnost prikazivanja dodatnih informacija. Da biste to ostvarali možete u deklaraciju `<%@ Page` u .aspx datoteci dodati atribut `Trace="true"`. Na taj način će stranica biti ukrašena velikim brojem informacija o izvršavanju stranice (uključujući detaljne informacije o vremenu, što vidite na slici 21.4), HTTP parametrima, i tako dalje. Osim toga, svojstvo `Trace` možete koristiti kako biste dodali vaše informacije. To je ono što sam ja uradio u primeru AspNetTrace, a rezultat vidite na slici.

Isti rezultat možete ostvariti tako što ćete vrednost True zadati atributu `pageOutput` čvora trace u datoteci `web.config`:

```
<trace enabled=«true« pageOutput=«true«/>
```



SLIKA 21.4 *U rezultatu AspNetTrace se prikazuju detaljne informacije (a posebne informacije su naročito istaknute).*

Drugo rešenje je da dodate čvor, ali da atribut i dalje ima vrednost False, tako da se podaci ne prikazuju u stranici već da se mogu pogledati kada se pretraživač usmeri na secijalan URL `trace.axd` koji se nalazi u istoj putanji kao i ASP.NET aplikacija.

U primeru se prikazivanje dodatnih informacija ostvaraje kada se klikne prvo komandno dugme:

```
procedure TWebForm1.Button1_Click(
  sender: System.Object; e: System.EventArgs);
begin
  Label1.Text := 'Button Click: ' + DateTime.Now.ToString;
  Trace.Write('Button1 was clicked');
end;
```

Pomoću drugog komandnog dugmeta programa se ništa ne obavlja (ne postoji obrada događaja), ali ipak uzrokuje osvežavanje sadržaja stranice kada se klikne. To je korisno jer se u labele upisuju različita vremena pomoću obrade događaja Load, u kojoj se pomoću svojstva `IsPostBack` menjaju podaci samo prilikom prvog izvršavanja:

```
procedure TWebForm1.Page_Load(
  sender: System.Object; e: System.EventArgs);
begin
  Label2.Text := 'On Load: ' + DateTime.Now.ToString;
  if not IsPostback then
    Label3.Text := 'On First Execution: ' + DateTime.Now.ToString;
end;
```

Sadržaj četvrte labele se menja pomoću metoda OnInit, koji se izvršava samo prvi put, a ne za svaku operaciju postback (što je alternativa za korišćenje svojstva `IsPostBack`). Drugim rečima, metod `OnInit` se izvršava kada se stranica i kontrole prave (slično metodu `Load` biblioteke VCL), dok se metod Load izvršava za svaki zahtev, pa čak i u slučaju postbacka (pa možete testirati svojstvo `IsPostBack` kako biste izbegli nepotrebnu inicijalizaciju). Najzad, možda ćete metod Unload koristiti za oslobađanje resursa koje ste rezervisali prilikom inicijalizovanja.

U primeru postoji četvrto komandno dugme za prikazivanje metoda klase `Page`, metoda `MapPath`, koji se koristi za izračunavanje putanje koja je relativna u odnosu na direktorijum u kojem se nalazi stranica. Dakle, možete koristiti datoteku koja se nalazi u istom direktorijumu kao i .aspx datoteka pomoću sledećeg programskog koda:

```
procedure TWebForm1.Button3_Click(
  sender: System.Object; e: System.EventArgs);
var
  strReader: StreamReader;
begin
  strReader := StreamReader.Create(MapPath('.') + '/demo.txt');
  try
    Label5.Text := strReader.ReadToEnd;
  finally
    strReader.Free; // dispose
  end;
end;
```

Pošto ne postoji referenca na direktorijum već samo relativna putanja, to vam pomaže da prosledite aplikaciju.

U ovom primeru treba razmotriti još jedan element, a to je način na koji se kontrole raspoređuju u web stranici. Ja sam na stranicu dodao tabelu i njenim ćelijama sam dodao razne kontrole. To znači da iako se na stranici prikazuje standardna tabela (atributu `ms_positionig` markera body je zadata vrednost GridLayout), stranica koristi strukturu koja se zasniva na tabeli. Od dizajnera dobijate značajnu pomoć jer dizajner poravnava kontrole koje se nalaze u ćelijama kada ih prevlačite iznad ćelija. Postoji mnogo drugih dizajnera pomagača, među kojima je alatka za pravljenje tabela, koja vam omogućava da koristite tabele i da napravite privlačan korisnički interfejs za web stranice.

Za strukturu tabele se za svaki element stranice koristi apsolutna pozicija, što je lako koristiti, ali može stvoriti probleme jer se ne prilagođava pravilno za različite veličine prozora pretraživača. Taj problem je još veći kada se koriste mali uređaji. U tehnologiji ASP.NET postoji alternativan način za raspoređivanje kontrola. To je standardni HTML tok. Raspored možete izmeniti u Object Inspectoru pošto odaberete objekat `DOCUMENT`, a atribut `ms_positioning` markera body će imati vrednost FlowLayout. Pošto se primer AspNetTrace zasniva na tabeli, ja sam mogao bez problema da izmenim izgled stranice.

SAVET

Korišćenje tabela u koje se smeštaju kontrole pomaže prilikom dizajniranja stranica koje se bolje prilagođavaju različitim pretraživačima i različitim veličinama prozora pretraživača. Druga važna tehnika koju treba da razmotrite jeste korišćenje spoljašnjih stilova (Cascading Style Sheet (CSS)) koji se pridružuju svakoj stranici ASP.NET aplikacije.■

Nostrifikovanje sadržaja kontrola

U arhitekturi ASP.NET postoji podrška za nostrifikovanje unosa korisnika. To je važno, jer je nostrifikacija u web aplikacijama bez stanja daleko složenija od CGI programa. Podrška za nostrifikovanje se u arhitekturi ASP.NET zasniva na skupu kontrola koje obavljaju različite vrste proveravanja (na primer, provera opsega ili provera regularnog izraza). To su kontrole koje prikazuju poruke o greškama (ili ne prikazuju ništa ukoliko je sve u redu), i koje su povezane sa ciljnom kontrolom pomoću svojstva `ControlToValidate`.

Nostrifikacija se obavlja nakon izmene sadržaja svakog polja tipa TextBox i prelaska na sledeće polje, jer svojstvo `EnableClientScript` ima vrednost True. Postupak nostrifikovanja možete direktno pokrenuti tako što ćete pozvati metod Validate klase Page. Da bih pokazao kako se u praktičnom primeru koristi nostrifikovanje kontrola, ja sam napravio formular u kojem postoji pet tekstualnih polja i pet različitih kontrola za nostrifikovanje. Formular tokom dizajniranja vidite na slici 21.5.

UPOZORENJE

Nostrifikovanje pomoću JavaScript programskog koda (kao što je ono u primeru AspNetValidate) koristi specifične dodatke koji postoje u Internet Exploreru i ne može se koristiti u drugim pretraživačima koje sam isprobao. Nije sve tako loše: u drugim pretraživačima će se ipak obaviti nostrifikovanje, ali na serveru – na primer, kada se klikne komandno dugme.■

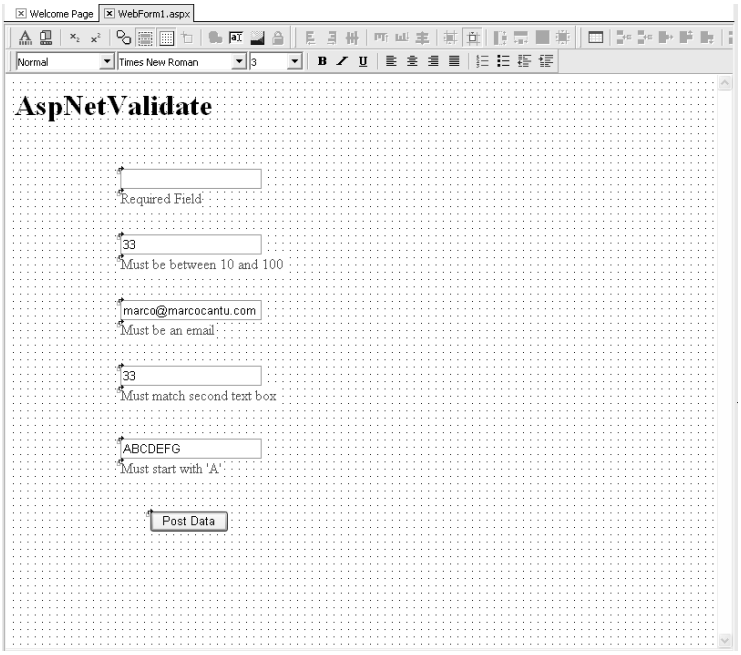
Najjednostavnija kontrola za nostrifikovanje je kontrola `RequiredFieldValidator`, koja ima svojstvo InitialValue pomoću kojeg se naznačava tekst koji zadajete na početku teksta i koji korisnik treba da izmeni. Možete uneti tekst `[ime]`, sa srednjim zagradama kako biste korsniku naznačili da mora da unese ime. Ukoliko isti string zadate svojstvu `InitialValue` onda će korisnik morati da unese

nekakav tekst, ali korisnik može da ukloni tekst i polje ostavi prazno, što je prilično čudno. Ukoliko napustite polje i svojstvo Initialvalu nema vrednost, sve funkcioniše kako očekujete. Evo definicije za nostrifikovanje u ASP.NET programskom kodu primera AspNetValidation:

```
<asp:RequiredFieldValidator
  id=«RequiredFieldValidator1«
  controltovalidate=«TextBox1«
  errormessage=«Required Field«>
</asp:RequiredFieldValidator>
```

U kontroli RangeValidator morate zadati minimalnu i maksimalnu vrednost, kao i tip podataka koji se koristi za poređenje (svojstvo koje je lako zaboraviti):

```
<asp:RangeValidator
  id=«RangeValidator1«
  controltovalidate=«TextBox2«
  errormessage=«Must be between 10 and 100«
  minimumvalue=«10«
  maximumvalue=«100«
  type=«Integer«>
</asp:RangeValidator>
```



SLIKA 21.5
Web formular primera AspNetValidate tokom dizajniranja

Za kontrolu RangeExpressionValidator, s druge strane, se u svojstvu `ValidationExpression` mora koristiti regularni izraz (unapred definisani editor već sadrži nekoliko regularnih izraza):

```
<asp:RegularExpressionValidator
   id=«RegularExpressionValidator1«
   controltovalidate=«TextBox3«
   errormessage=«Must be an email«
   validationexpression=«\w+([-+.]\w+)*@\w+([-.]\w+)*\.\w+([-.]\w+)*«>
</asp:RegularExpressionValidator>
```

Kontrola CompareValidator može imati dva tekstualna polja sa istom vrednošću (što je korisno kada od korisnika želite da zatražite da lozinku ili adresu elektronske pošte dva puta unese), ili malo i veliko tekstualno polje, ili bilo koju kombinaciju. U primeru treba u oba tekstualna polja uneti istu vrednost:

```
<asp:CompareValidator
  id=«CompareValidator1«
  controltovalidate=«TextBox4«
  errormessage=«Must match second text box«
  controltocompare=«TextBox2«
  type=«Integer«>
</asp:CompareValidator>
```

Postoji kontrola CustomValidator koja vam omogućava da zadate funkciju koja se koristi za nostrifikovanje u klijentskom Javascript programskom kodu ili događaj na strani servera. Slede te kontrole koje se nalaze u .aspx datoteci i odgovarajuća obrada događaja, pomoću koje se proverava da li string počinje velikim slovo A:

```
<asp:CustomValidator
  id=«CustomValidator1«
  controltovalidate=«TextBox5«
  errormessage=«Must start with 'A'«>
</asp:CustomValidator>

procedure TWebForm1.CustomValidator1_ServerValidate(
  source: System.Object;
  args: System.Web.UI.WebControls.ServerValidateEventArgs);
begin
  args.IsValid := args.Value[1] = 'A';
end;
```

Rad sa podacima u tehnologiji ASP.NET

Sve jednostavne aplikacije koje smo do sada napravili nisu imale jednu osnovnu osobinu, jer ne zapisuju svoje podatke. U mnogim slučajevima ćete praviti baze podataka koje uspostavljaju vezu sa bazom podataka. ASP.NET kontrole mogu direktno uspostaviti vezu sa ADO.NET skupom podataka, ali morate da napišete programski kod pomoću kojeg ćete napraviti skup podataka i uspostaviti vezu između podataka i kontrola tako što ćete pozvati njihov metod DataBind. Prođimo kroz sve korake tako što ćemo početi od jednostavnije tehnike.

NAPOMENA

Korišćenje provajdera Borland Data Provider (BDP) za ADO.NET će vam pomoći da dobijete podatke tokom dizajniranja ASP.NET web stranica.■

Pre nego što objasnim neke od ASP.NET kontrola koje možete upotrebiti za prikazivanje podataka, postoji osobina ASPX skripta koju sam pomenuo, ali nisam objasnio: to su jednostavni izrazi za uspostavljanje veze sa podacima. Ti izrazi se zadaju u notaciji <%#...%> kako bi uspostavili vezu sa podacima iz definicije stranice. Na primer, da bi se prikazala vrednost svojstva `MyData` stranice u HTML pasusu ili ASP.NET oznaci, možete upotrebiti sledeća dva reda programskog koda, i dodati izvorni programski kod iz kojeg se poziva metod DataBind kontrole (ili cele stranice):

```
<p><%# MyData %></p>
<asp:Label text='<%# MyData %>' runat=server/>
```

Sličnu tehniku možete upotrebiti za rad sa podacima baze podataka kada koristite izraz `DataBinder.Eval`, kao u narednom iskečku programskog koda koji je deo sledećeg primera:

```
<span><%# DataBinder.Eval (Container.DataItem, »Company«) %></span>
```

Rad sa bazama podataka

Kao i u biblioteci WinForms, tako se i u biblioteci WebForms za sve kontrole može koristiti uspostavljanje veze sa podacima. Postoje neke kontrole koje se koriste samo za uspostavljanje veze sa podacima. Kontrole DataGrid, DataList i Repeater se povezuju sa celom tabelom podataka, a ne jednim poljem kao što je slučaj sa jednostavnim kontrolama. Sledi kratak rezime njihovih osobina, koje ćete bolje razumeti ukoliko pogledate sliku 21.6, na kojoj se nalazi primer rezultata sve tri kontrole:

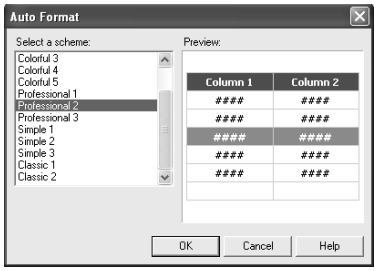
Kontrola DataGrid Kontrola DataGrid odgovara kontroli DataGrid (odnosno, kontroli DBGrid) biblioteke WinForms (odnosno, biblioteke VCL). U kontroli DataGrid se prikazuju sve kolone tabele ili podskup kolona tabele. Postoji mnogo načina na koje možete prilagoditi izgled svake kolone, a jedino ograničenje je da mora biti na osnovu strukture tabele. Svaki element tabele prikazuje tekstualne podatke koji se mogu prilagoditi tako da prikazuju link, komandno dugme ili HTML na osnovu šablona.

Kontrola DataList Kontrola DataList je, slično komponenti DBCtrlGrid biblioteke VCL, kontejner za druge kontrola, koja se ponavlja u svakom redu tabele skupa podataka. Kontrola DataList koristi tabelu za svoju strukturu sa po jednom ćelijom za svaki red. To znači da možete da imate jednu ćeliju za svaki red ili više kolona u kojima se prikazuju različiti podaci. To se postiže izmenom svojstva RepeatColumns. Za svaki element definišete šablon koji sadrži HTML rezultat ili ASP.NET kontrole.

Kontrola Repeater Kontrola Repeater je fleksibilnija, ali ima manje mogućnosti. Kontrola Repeater se zasniva na HTML šablonu koji se ponavlja za svaki element, ali ne postoji stalna struktura (recimo, tabela) u kojoj se nalaze elementi. To znači da u najjednostavnijem obliku definišete pasus teksta pomoću informacija koje se izdvajaju iz redova tabele.

SLIKA 21.6 *Rezultat primera AspNetDataOne, u kojem se koristi više kontrola u kojima se prikazuju podaci*

Pomoću mnogih svojstava možete prilagoditi stilove raznih elemenata kontrola DataGrid i DataList. Postoje neke unapred definisane šeme boja koje možete upotrebiti pomoću komande Auto Format koja se nalazi u dnu Object Inspectora, koja kada je zadate prikazuje sledeći okvir za dijalog:



Ja sam taj okvir za dijalog koristio za prilagođavanje rezultata kontrola primera AspNetDataOne (u kojem se koriste tri kontrole). Primer se zasniva na HTML tabeli, a u prvoj ćeliji se nalaze kontrole Repeater za koje je svojstvu `colspan` zadata vrednost 2 kako bi se odnosile na ceo red, a u drugom redu se nalaze druge dve kontrole u dve ćelije. U programu se koristi komponenta OleDbConnection i odgovarajuća komponenta OleDbAdapter pomoću kojih se čitaju podaci. Ja sam tokom dizajniranja programa napravio i objekat `DataTable` koji sam nazvao `CustomerTable`. Evo najvažnijih svojstava tih komponenata, koja se zadaju u metodima `InitializeComponent` web formulara:

```
Self.OleDbConnection1.ConnectionString :=
  'Provider=«Microsoft.Jet.OLEDB.4.0«;' +
  'Data Source=«C:\..\Borland Shared\Data\dbdemos.mdb«';
Self.oleDbSelectCommand1.Connection := Self.OleDbConnection1;
Self.oleDbSelectCommand1.CommandText :=
  'select Company, City, State from customer where Country=«US«';
Self.OleDbDataAdapter1.SelectCommand := Self.oleDbSelectCommand1;
Self.DataSet1.Tables.AddRange(
  TArrayOfSystem_Data_DataTable.Create(Self.CustomerTable));
Self.CustomerTable.TableName := 'CustomerTable';
```

Sa bazom podataka smo uspostavili vezu u obradi događaja `Load` stranice tako što se veza aktivira, podaci premeštaju u tabelu i tako što se vizuelne kontrole povezuju sa skupom podataka:

```
procedure TWebForm1.Page_Load(
  sender: System.Object; e: System.EventArgs);
begin
  if not IsPostBack then
  begin
    OleDbConnection1.Open;
    OleDbDataAdapter1.Fill(CustomerTable);
    DataGrid1.DataBind;
    Repeater1.DataBind;
    DataList1.DataBind;
  end;
end;
```

SAVET

Alternativa za pozivanje metoda `DataBind` za svaku kontrolu koja je uspostavila vezu sa skupom podataka je pozivanje metoda DataBind stranice. Pozivanjem metoda `DataBind` stranice će uzrokovati pozivanje metoda `DataBind` kontrola stranice i zamenu jednostavnih izraza za povezivanje podataka. U opštem slučaju, pozivanje jednog metoda `DataBind` stanice je bolje nego pozivanje metoda za svaku kontrolu. U ovom primeru sam ostavio pozivanje metoda za svaku kontrolu kako bih istakao ponašanje tih kontrola u arhitekturi ASP.NET.■

Sve tri vizuelne kontrole imaju svojstvo DataSource za koje je zadat objekat (vrednost DataSet1), i svojstvo DataMember za koje je zadat objekat CustomerTable. Za tabelu ne postoje specijalna podešavanja, osim zadavanja standardne šeme boja. Vredi pogledati ASPX programski kod. To bi trebalo da vam razjasni zašto takav način pisanja programskog koda pomaže razdvajanju programskog koda korisničkog interfejsa i aplikacije, ali imate ograničenu kontrolu nad HTML programskim kodom koji se generiše kada koristite složene kontrole ASP.NET:

```
<asp:DataGrid id=«DataGrid1«
  style=«Z-INDEX: 2«
  datamember=«CustomerTable«
  datasource=«<%# DataSet1 %>«
  cellpadding=«3«
  backcolor=«White«
  bordercolor=«#999999«
  borderwidth=«1px«
  gridlines=«Vertical«
  borderstyle=«None«>
  <FooterStyle
    forecolor=«Black«
    backcolor=«#CCCCCC«>
  </FooterStyle>
  <HeaderStyle
    font-bold=«True«
    forecolor=«White«
    backcolor=«#000084«>
  </HeaderStyle>
  <PagerStyle
    horizontalalign=«Center«
    forecolor=«Black«
    backcolor=«#999999«
    mode=«NumericPages«>
  </PagerStyle>
  <SelectedItemStyle
    font-bold=«True«
    forecolor=«White«
    backcolor=«#008A8C«>
  </SelectedItemStyle>
  <AlternatingItemStyle
    backcolor=«#DCDCDC«>
  </AlternatingItemStyle>
  <ItemStyle
    forecolor=«Black«
    backcolor=«#EEEEEE«>
  </ItemStyle>
</asp:DataGrid>
```

HTML programski kod koji generiše kontrola je sasvim drugačiji (sledi isečak tog programskog koda), mada su mnogi HTML atributi generisani iz atributa ASP.NET kontrola:

```
<table id=«DataGrid1« style=«z-index: 2;« bgcolor=«White« border=«1«
  bordercolor=«#999999« cellpadding=«3« cellspacing=«0« rules=«cols«>
```

```
<tbody>
  <tr bgcolor=«#000084«>
    <td><font color=«White«><b>Company</b></font></td>
    <!— more headers —>
 </tr>
 <tr bgcolor=«#eeeeee«>
    <td><font color=«Black«>Kauai Dive Shoppe</font></td>
    <td><font color=«Black«>Kapaa Kauai</font></td>
    <td><font color=«Black«>HI</font></td>
 </tr>
 <!— more lines —>
```

Prilagođavanje kontrole DataList nije jednako intuitivno. Kada iz Delphijevog IDE-a kontrolu DataList smestite na web formular dobijate mesto sa opisom operacija koje treba da obavite:

Zapravo, kontekstni meni ove komponente sadrži dodatne komande (upamtite da ih ne zadajete iz Object Inspectora kao inače). Ukoliko odaberete Item Templates, kontrola će obojiti editor šablona u koji možete da unosite informacije:

U maloj oblasti šablona možete uneti rezultujući tekst ili smestiti HTML ili ASP.NET kontrole tako što ćete ih prevući. Na primer, ja sam šablonu dodao ASP.NET kontrolu Label i za svojstvo `dataBinding` sam zadao sledeću vrednost (u odgovarajućem specijalnom editoru):

```
DataBinder.Eval (Container.DataItem, »Company«)
```

Kontrola DataList je u ovom slučaju kontejner. Ovaj i druge šablone možete direktno prilagoditi, što je neophodno za povezivanje standardnih HTML elemenata sa podacima iz baze podataka. Sledi skraćena verzija ASPX programskog koda za kontrolu DataList primera AspNetDataOne:

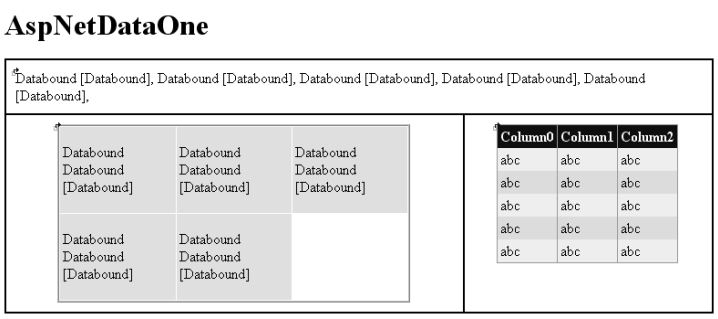
```
<asp:DataList
  id=«DataList1«
  datamember=«CustomerTable«
  datasource=«<%# DataSet1 %>«
  repeatdirection=«Horizontal«
  repeatcolumns=«3«>
  <ItemStyle
    forecolor=«Black«
    backcolor=«#DEDFDE«>
```

```
    </ItemStyle>
    <ItemTemplate>
      <asp:Label
        id=«Label1«
        text='<%# DataBinder.Eval (Container.DataItem, »Company«) %>'>
      </asp:Label>
      <div>
        <%# DataBinder.Eval (Container.DataItem, »City«) %>
        [<%# DataBinder.Eval (Container.DataItem, »State«) %>]
      </div>
    </ItemTemplate>
  </asp:DataList>
```

Kao što u šablonu čvora vidite, postoji `asp:Label` za kojiom sledi marker `div` koji sadrži direktan HTML programski kod sa dva izraza za povezivanje podataka. Kontrola DataList ima tri kolone za svaki red (atribut repeatcolumns), i koristi horizontalni tok (atribut `repeatdirection`); te atribute podešavate u Object Inspectoru tako što menjate vrednosti odgovarajućih svojstava.

Direktno editovanje HTML programskog koda je nešto što morate uraditi kada koristite kontrolu repeater, za koju tokom dizajniranja gotovo i da nema podrške, što vidite na slici 21.7. Slično kao i za kontrolu DataList, kada kontrolu Repeater postavite na web formular dobijate poruku koja opisuje šta morate da uradite, što je u osnovi otvaranje .aspx datoteke i unošenje HTML programskog koda. Programski kod se ne razlikuje mnogo od programskog koda koji ste već videli:

```
<asp:Repeater
  id=«Repeater1«
  datamember=«CustomerTable«
  datasource=«<%# DataSet1 %>«>
  <ItemTemplate>
    <span>
      <%# DataBinder.Eval (Container.DataItem, »Company«) %>
      [<%# DataBinder.Eval (Container.DataItem, »State«) %>],
    </span>
  </ItemTemplate>
</asp:Repeater>
```



SLIKA 21.7 *Konačan oblik primera AspNetDataOne tokom njegovog dizajniranja*

HTML programski kod koji u ovom slučaju generišu kontrole ne sadrži spoljašnju strukturu (nema tabele ni markera `div`), te postaje ponavljanje podataka sa tekućim podacima:

```
<span>Kauai Dive Shoppe [HI], </span>
<span>Blue Jack Aqua Center [HI], </span>
<span>Ocean Paradise [HI], </span>
```

Dobra vest je da možete fino podesiti HTML programski kod. Loše je to što morate uložiti više truda da biste napravili privlačan korisnički interfejs, za razliku od kontrola DataGrid i DataList koje vam u tome pomažu.

Korišćenje Borlandovih DBWeb komponenata

Ukoliko želite da još više pojednostavite i ubrzate pravljenje ASP.NET aplikacija za baze podataka, onda možete koristiti skup kontrola koje je napravio Borland. To su kontrole DBWeb. Struktura tih komponenata podseća na Delphijeve kontrole biblioteke VCL koje prepoznaju podatke, pri čemu se objekat izvora podataka ponaša kao broker između skupa podataka i vizuelnih kontrola.

Postoje neke prednosti korišćenja kontrola DBWeb u odnosu na standardne ASP.NET kontrole, uključujući mogućnost (kada se koriste zajedno sa BDP ADO.NET komponentama) prikazivanja podataka tokom dizajniranja programa i rada na stranici koja je slična stranici koju će korisnik videti. Pomoću DBWeb kontrola možete izbeći da iz programskog koda pozivate metod DataBind. Jedna od prednosti je i automatsko slanje izmena skupu podataka.

Osnovna osobina DBWeb kontrola je da korišćenjem komponente DBWebDataSource web formular održava tekuću poziciju reda na bazi sesija. Komponenta DBWebDataSource ima spisak izmena koje su načinjene u podacima, koje korisnik može poništiti. Komponenta DBWebGrid ima kompletnu podršku za straničenje (slično kontroli DataGrid), za lokalno editovanje i obezbeđuje mnoge osobine za koje ne morate pisati programski kod.

KOMPONENTE

Borlandove komponente DBWeb su veliki skup kontrola i nevizuelne komponente, komponente DBWebDataSource, koja se ponaša kao spona između kontrola koje prepoznaju podatke i skupa podataka. Pomoću komponente DBWebDataSource se uspostavlja veza sa kontrolom DataTable ili kontrolom DataView i ima za zadatak da prati tekući slog za svaku sesiju/korisnika koja je uspostavljena sa aplikacijom. Komponenta DBWebDataSource ima specifičnu podršku za XML, što ćete videti u narednom poglavlju.

NAPOMENA

Iako je osnovna DBWeb tehnologija postojala u Delphiju 8 za .NET, neke kontrole su nove u Delphiju 2005, uključujući audio i video kontrole, agregatne kontrole i proširenje navigacije.■

Na stranici DBWeb palete Tool Palette postoje različite grupe kontrola. Pomoću prve grupe kontrola možete raditi sa pojedinačnim kolonama osnovnih tipova podataka:

- Kontrola DBWebLabel je oznaka koja se može samo čitati.
- Kontrola DBWebTextBox je standardno tekstualno polje.
- Kontrola DBWebLabeledTextbox je tekstualno polje koje ima standardnu oznaku (ne kontrolu DBWebLabel), koja automatski prikazuje ime kolone (ili polja).
- Kontrola DBWebCheckBox je polje za potvrdu.

- Kontrola DBWebCalendar je kalendar koji koristite za promenu datuma.
- Kontrola DBWebAggregateControl je tekstualno polje u kojem se prikazuju agregatne vrednosti (suma, maksimum, minimum, prosek i broj elemenata) iz zadate kolone i ima zaglavlje.

Postoje kontrole koje mogu da prikažu sadržaj polja tipa BLOB baze podataka ili datoteke:

- Kontrola DBWebMemo je višelinijski editor koji možete povezati sa poljem tipa Memo.
- Kontrola DBWebImage se koristi za prikazivanje slike.
- Kontrola DBWebSound je audio kontrola koja aktivira plejer korisnikovog sistema.
- Kontrola DBWebVideo je video kontrola koja aktivira plejer.

Sledeća grupa kontrola prestavlja kontrole pomoću kojh se obavlja selektovanje, koje se mogu upotrebiti za implementiranje lookup polja (kao u primeru AspNetEmployee) ili da obezbede selektovanje u spisku elemenata:

- Kontrola DBWebRadioButtonList prikazuje grupu opcionih komandnih dugmadi.
- Kontrola DBWebListBox je polje tipa ListBox.
- Kontrola DBWebDropDownList je polje tipa combo.

Najzad, postoje kontrole pomoću kojih se tabela koristi kao celina, a ne koriste se pojedinačne kolone:

- Kontrola DBWebGrid je fleksibilna tabela podataka.
- Kontrola DBWebNavigator je paleta za kretanje kroz podatke.
- Kontrola DBWebNavigatorExtender je pomagač kako bi se standardne web kontrole ponašale kao komandna dugmad za kretanje kroz podatke.

PRVI PRIMER: KORIŠĆENJE KONTROLE DBWEBGRID

Da biste bolje razumeli mogućnosti i prednosti korišćenja kontrola DBWeb, ništa nije bolje od pravljenja jednostavnog primera korak po korak. Kao što ćete videti, sve se svodi na prevlačenje komponenata i podešavanje svojstava (uglavnom za povezivanje tih komponenti), pa ne morate pisati programski kod.

Pošto napravite novu ASP.NET aplikaciju, pravljenje aplikacije još više možete ubrzati tako što ćete iz Data Explorera prevući tabelu koju želite da prikažete u web formularu. U primeru sam prevukao tabelu Employee primera InterBase baze podataka koju Delphi instalira u direktorijum `Common Files/Borland Shared/Data`. Prevlačenjem tabele ćete napraviti komponente BdpConnection i BdpDataAdapter. Nemojte zaboraviti da za stringovno svojstvo za uspostavljanje veze komponente BdpConnection upotrebite prefiks localhost: ispred putanje (u suprotnom će ASP.NET program funkcionisati tokom dizajniranja, ali će biti problema kada ga prosledite). Sada dodajte komponentu DataSet i u svojstvu `DataSet` komponente BdpDataAdapter je odaberite kao ciljnu komponentu. Ukoliko je sve prvilno podešeno, onda možete aktivirati komponentu BdpDataAdapter (svojstvu Active zadajte vrednot True) i neće se prijaviti greška. Trebalo bi da vidite tabelu sa podacima o zaposlenima (ili podatke tabele koju ste odabrali) u kolekciji Tables skupa podataka.

Pošto smo završili sa delom programa koji se odnosi na bazu podataka, pređimo na korisnički interfejs. Dizajneru dodajte komponentu DBWebDataSource i njeno svojstvo `DataSource` povežite sa tabelom (u listi ćete videti skup podataka i tabele koje definiše). U web formular smestite komponentu DBWebGrid i podesite dva njena ključna svojstva: svojstvo `DBDataSource` treba da se odnosi na komponentu DBWebDataSource, a svojstvo `TableName` na tabelu. Najzad, upotrebite jedan od standardnih stilova formatiranja tabele sa podacima iz baze podataka (pogledajte sliku 21.8), u kojoj se prikazuje web formular primera AspNetDBTable koji sam na ovaj način napravio.

Aplikacija AspNetDBTable ima nekoliko interesantnih osobina. Prva je straničenje tabele sa podacima. Kontrola DBWebGrid ima svojstvo `AllowPaging` tipa Boolean, pomoću kojeg se uključuje ili isključuje straničenje podataka (unapred je definisano da je uključeno), i svojstvo `PageSize` pomoću kojeg određujete koliko slogova se prikazuje na svakoj stranici (unapred je zadata vrednost 10). U dnu tabele pomoću kontrola se prikazuje broj stranica na koje možete preći ili jednostavne kontrole za kretanje, u zavisnosti od vrednosti svojstva `PagerStyle` (tačnije, atributa `Mode`).

Druga osobina kontrole DBWebGrid je to što omogućava direktno editovanje podataka u bazi podataka pomoću tabele (osim kada je vrednost True zadata za svojstvo `ReadOnly`). Da bi to pravilno funkcionisalo u formular morate dodati još jednu komponentu, kontrolu DBWebNavigator. Podesite svojstva `DBDataSource` i `TableName`, i odaberite jednu od opcija `ButtonType` (ja sam u primeru upotrebio vrednost Button Text). Na taj način ćete web formular pretvoriti u kompletan editor podataka iz skupa podataka. Korisnik će moći da menja podatke tekućeg sloga, da promeni tekući slog, unese nove slogove, ukloni slogove, poništi izmene koje je napravio i osveži orginalne podatke tabele.

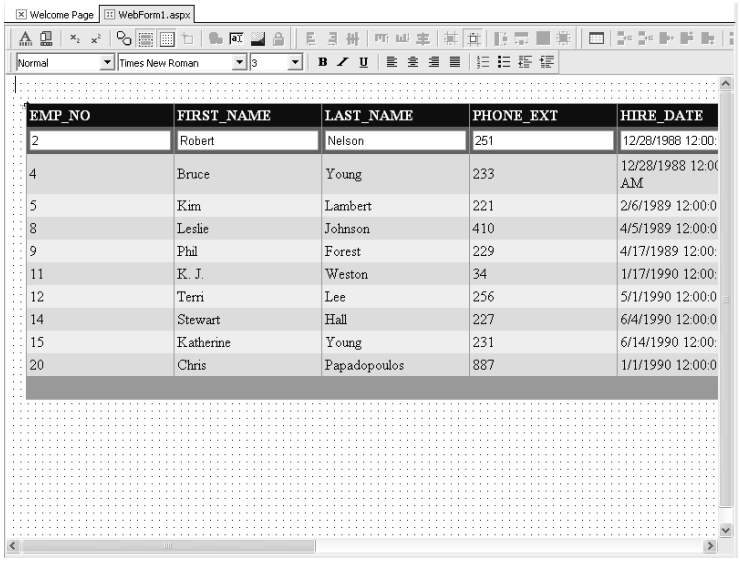
Jedina osobina koja nedostaje je mogućnost ažuriranja podataka u bazi podataka, jer će komandno dugme Apply To Server ostati nekativno sve dok ne napišete programski kod za obradu događaja `OnApplyUpdateChanges` (a ne događaja `OnApplyChanges`, kao što se to sugeriše u Helpu) komponente DBWebDataSource. Ukoliko koristite BDP, onda možete da napišete veoma jednostavan programski kod za obradu ovog događaja:

```
procedure TWebForm1.DBWebDataSource1_OnApplyChangesRequest(
  sender: System.Object; e: Borland.Data.Web.WebControlEventArgs);
begin
  BdpDataAdapter1.AutoUpdate;
end;
```

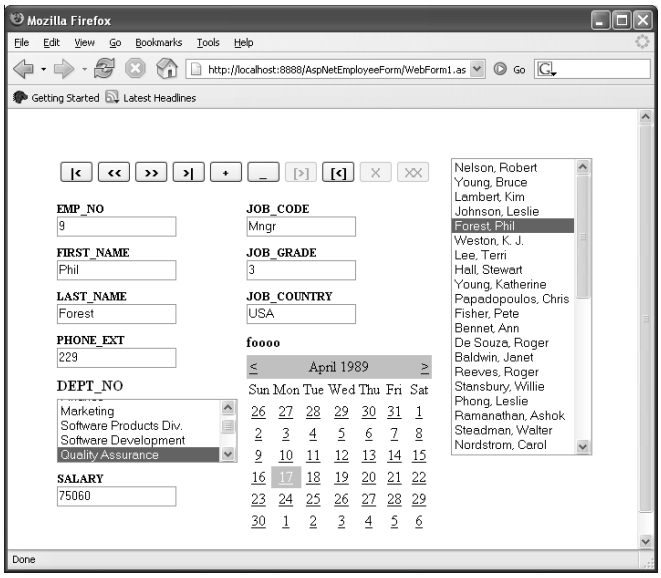
PRAVLJENJE FORMULARA

Za izmenu podataka je mnogo bolje da koristite strukturu formulara nego strukturu tabele. U primeru AspNetEmployeeForm ja sam upotrebio iste komponente za rad sa podacima, ali sam dodao drugu tabelu (tabelu Departments) koja služi kao lookup kontrola. Izmenio sam korisnički interfejs tako što sam za razna tekstualna polja upotrebio komponente DBWebLabeledTextBox, kao i komponentu DBWebCalendar za unošenje datuma zapošljavanja, komponentu DBWebListBox kao lookup polje, komponentu DBWebAggregateControl za prikazivanje ukupne plate, i komponentu DBWebNavigator za kretanje kroz slogove. Na slici 21.9 vidite formular tokom izvršavanja programa.

Za kontrole koje prepoznaju podatke, a u kojima se prikazuje jedna kolona, pored toga što sam podesio svojstva `DBDataSource` i `TableName`, morao sam da podesim svojstvo `ColumnName`. Za lookup kontrole morao sam da podesim svojstva `LookupTableName`, `DataTextField` i `DataValueField`.



SLIKA 21.8 *Web formular primera AspNetDBTable u kojem se tokom dizajniranja prikazuju podaci*



SLIKA 21.9 *Primer AspNetEmployee Form tokom izvršavanja*

Sledi mali isečak svojstava nekih kontrola različitih tipova (stilovi i pozicije kontrola su izostavljeni) iz .aspx datoteke:

```
<body ms_positioning="GridLayout">
  <form runat="server">
    <borland:DBWebNavigator
      id="DBWebNavigator1"
      dbdatasource="<%# DBWebDataSource1 %>"
      tablename="EMPLOYEE">
    </borland:DBWebNavigator>
    <borland:DBWebCalendar
      id="DBWebCalendar1"
      dbdatasource="<%# DBWebDataSource1 %>"
      tablename="EMPLOYEE"
      columnname="HIRE_DATE">
    </borland:DBWebCalendar>
    <borland:DBWebLabeledTextBox
      id="DBWebLabeledTextBox1"
      dbdatasource="<%# DBWebDataSource1 %>"
      tablename="EMPLOYEE"
      columnname="EMP_NO"
      caption="EMP_NO"
      captionposition="LabelAbove">
    </borland:DBWebLabeledTextBox>
    <borland:DBWebListBox
      id="DBWebListBox1"
      dbdatasource="<%# DBWebDataSource1 %>"
      tablename="EMPLOYEE"
      columnname="DEPT_NO"
      datavaluefield="DEPT_NO"
      datatextfield="DEPARTMENT"
      lookuptablename="DEPARTMENT">
    </borland:DBWebListBox>
    <borland:DBWebAggregateControl
      id="DBWebAggregateControl1"
      caption="TOTAL SALARY"
      captionposition="LabelAbove"
      aggregatetype="aggSum"
      dbdatasource="<%# DBWebDataSource1 %>"
      tablename="EMPLOYEE"
      columnname="SALARY"
      enableviewstate="False"
      ignorenullvalues="True">
    </borland:DBWebAggregateControl>
    <asp:ListBox
      id="ListBox1"
```

```
      datavaluefield=«EMP_NO«
      datatextfield=«FULL_NAME«
      rows=«20«
      datasource=«<%# DataTable1 %>«>
    </asp:ListBox>
  </form>
</body>
```

Kao što na kraju spiska vidite, u programu se koristi standardna kontrola ListBox, koja se ne koristi kao lookup polje već samo za prikazivanje slogova sa kojima može da se radi. Ta komponenta podatke dobija direktno iz tabele skupa podataka, i za nju je potrebno napisati standardan ASP.NET programski kod za uspostavljanje veze:

```
procedure TWebForm1.Page_Load(
  sender: System.Object; e: System.EventArgs);
begin
  if not IsPostBack then
  begin
    ListBox1.DataBind;
    ListBox1.SelectedIndex := 0;
  end;
end;
```

Na početku program odabira prvi element iz polja tipa ListBox. Kada se korisnik kreće kroz slogove, sadržaj komponente se ažurira odgovarajućim elementom. Za ovu operaciju morate upotrebiti jedan od brojnih interfejsa koji su podržani u komponenti DBWebDataSource i dokumentu. U ovom primeru možete upotrebiti interfejs `IDBDataSource` koji ima metod `GetCurrentRow` pomoću kojeg se dobija indeks tekućeg reda:

```
procedure TWebForm1.DBWebDataSource1_OnScroll(
  sender: System.Object; e: Borland.Data.Web.OnScrollEventArgs);
var
  iDataSource: Borland.Data.Web.IDBDataSource;
begin
  iDataSource := Borland.Data.Web.IDBDataSource (DBWebDataSource1);
  ListBox1.SelectedIndex := iDataSource.
    GetCurrentRow (Self, 'EMPLOYEE');
end;
```

Pomoću programa se postiže suprotan rezultat: ukoliko u polju tipa ListBox odaberete element, onda na taj slog možete preći pomoću komandnog dugmeta Goto, za koje se koristi još jedan nedokumentovan interfejs, interfejs `IDBPostStateManager`, kako bi se promenila pozicija tekućeg sloga. Pre nego što se to uradi, mora se ukloniti informacija o tekućem slogu koja je zapisana u podacima sesije, ili će tekući slog zaobići unapred zadato podešavanje:

```
procedure TWebForm1.Button1_Click(
  sender: System.Object; e: System.EventArgs);
var
  iStateManager: Borland.Data.Web.IDBPostStateManager;
begin
```

```
  DBWebDataSource1.ClearSessionChanges(Self);
  iStateManager := Borland.Data.Web.IDBPostStateManager
(DBWebDataSource1);
  iStateManager.SetCurrentRow (
    Self, 'EMPLOYEE', ListBox1.SelectedIndex);
end;
```

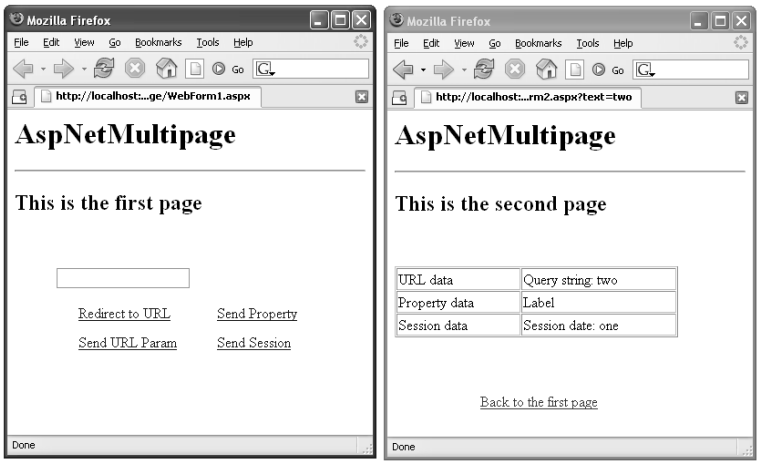
Pravljenje ASP.NET aplikacija

Do sada ste pravili ASP.NET aplikacije sa samo jednom stranicom. U opštem slučaju aplikacije mogu imati više stranica. Zbog toga ću ja u ovom odeljku prvo pokazati kako se podaci premeštaju sa jedne na drugu stranicu, pa ću zatim objasniti sesije podataka i upravljanje sesijama i prijavljivanjem korisnika. Sve te informacije će vam pomoći da sa pravljenja aplikacija sa jednom stranicom pređete na pravljenje kompletnih aplikacija.

Prebacivanje podataka između stranica

Videli ste da arhitektura ASP.NET pomoću kolekcije `ViewState` pojednostavljuje upravljanje podacima koje korisnik upotrebljava na specifičnoj stranici. Međutim, ta tehnika vam ne pomaže kada korisnik sa jedne web stranice pređe na drugu web stranicu iste aplikacije (ili virtuelnog direktorijuma).

Ukoliko je potrebno da podatke sa jedne stanice prebacite na drugu stranicu, onda možete upotrebiti razne tehnike, a svaka od njih ima prednosti i nedostatke. U primeru AspNetMultipage ja sam pokazao tri pristupa: korišćenje URL parametara (parametara upita), korišćenje pomoćne stranice i korišćenje parametara sesije. Na prvoj stranici programa AspNetMultipage se nalazi kontrola TextBox i četiri kontrole LinkButton, dok se na drugoj stranici nalazi tabela sa nekoliko kontrola Label u koje program kopira (pomoću različitih tehnika, koje se ne upotrebljavaju istovremeno) tekst iz kontrole TextBox koja se nalazi na prvoj stranici. Formulare vidite na slici 21.10.



SLIKA 21.10
Dve stranice primera AspNetMultipage nakon prebacivanja podataka u sesiji i redirekcije URL adrese

Najjednostavniji način za prebacivanje podataka je korišćenje URL adrese za prosleđivanje parametra u zahtevu koji se zadaje aplikaciji (redirekcija funkcioniše tako što se dobija specijalan HTTP zahtev koji će od pretraživača zatražiti da zahteva drugu stranicu, što je naznačeno kao redirection URL). Link Redirect to URL prvog formulara sadrži sledeći programski kod:

```
Response.Redirect('WebForm2.aspx?text=' + TextBox1.Text);
```

URL parametar se prosleđuje drugoj stranici u okviru parametara upita novog HTTP zahteva. Stranica može pročitati parametar pomoću sledećeg programskog koda koji se nalazi u obradi događaja `Page_Load`:

```
var
  strText: string;
begin
  strText := Request.Params.Item['text'];
  if strText <> '' then
    Label1.Text := 'Query string: ' + strText;
```

Slična tehnika se koristi za drugu kontrolu LinkButton, link Send URL Param, koja direktno aktivira drugu stranicu, bez daljeg zahteva pretraživača:

```
Server.Transfer('WebForm2.aspx?text=' + TextBox1.Text);
```

Razlika je u tome što u drugom slučaju neće videti odredišni URL, ali ćete i dalje videti URL prve stranice, bez parametara. Pomoću ovog linka se aktivira mehanizam za prosleđivanje drugog parametra. Treći link ima sličan programski kod:

```
Server.Transfer('WebForm2.aspx');
```

U oba slučaja drugi formular se direktno poziva pomoću programskog koda prvog linka (bez daljih HTTP zahteva kao u `Response.Redirect` koji sam ranije opisao). To znači da se formular koji se poziva može obratiti objektu koji šalje poziv pomoću svojstva `Context.Handler`:

```
var
  wf1: TWebForm1;
begin
  wf1 := TWebForm1 (Context.Handler);
  if Assigned (wf1) then
    Label2.Text := 'Form data: ' + wf1.Text;
```

Četvrti link kopira sadržaj polja TextBox u parametar sesije, koji je zbog toga na raspolaganju svakoj stranici ili zahtevu koji se pravi u istoj sesiji. Sledi programski kod koji se izvršava za link (ja sam upotrebio `response.Redirect` samo kako bih izbegao aktiviranje prethodnih tehnika):

```
Session.Add('Text', TextBox1.Text);
Response.Redirect('WebForm2.aspx');
```

Objekat Session je spisak kojem se može pristupiti pomoću indeksa drugog formulara:

```
var
  strText: string;
  obj: System.Object;
```

```
begin
  obj := Session.Item ['Text'];
  if Assigned (obj) then
  begin
    strText := obj.ToString;
    if strText <> '' then
      Label3.Text := 'Session date: ' + strText;
  end;
```

Poslednja tehnika je verovatno najfleksibilnija i najrobusnija, iako je prosleđivanje parametara pomoću URL adrese ili dobijanje podataka iz drugog formulara mnogo jednostavnije, ja bih ipak izbegao zapisivanje previše podataka u svakoj sesiji na web sajtu koji je veoma posećen.

Upravljanje sesijama

Iako je korišćenje objekta Session veoma jednostavno, kao što ste videli u prethodnom primeru, upravljanje sesijama u arhitekturi ASP.NET je fleksibilno. Bez menjanja izvornog programskog koda, možete konfigurisati ASP.NET tako da upravlja sesijama na potpuno drugačiji način. to je dobro jer izmena skalabilnosti aplikacije i njeno prosleđivanje može dovesti do menjanja datoteke `web.config`, ali neće uticati na programski kod koji pišete.

Pomoću elementa `sessionState` datoteke web.config se definiše gde se podaci sesije zapisuju (lokalno, na udaljenom serveru stanja ili SQL Serveru), da li želite da koristite cookie ili ne, i vremenski prozor sesije. Takođe, obezbeđuje konfiguraciju za opcioni spoljašnji server. Evo unapred definisanih podešavanja za ASP.NET aplikaciju:

```
<sessionState
  mode=«InProc«
  stateConnectionString=«tcpip=127.0.0.1:42424«
  sqlConnectionString=«data source=127.0.0.1;user id=sa;password=«
  cookieless=«false«
  timeout=«20«/>
```

Autentičnost formulara

Drugo podešavanje koje ćete pronaći u datoteci web.config se odnosi na proveru korisnika. Delphi IDE generiše sledeći programski kod:

```
<authentication mode=«Windows« />
```

To znači da se koristi prijava koja se koristi za operativni sistem Windows. Ukoliko prijavljivanjem želite da upravljate pomoću posebne stranice, korisite režim Forms. Ukoliko korisnicima koji nisu prijavljeni ne želite da omogućite da koriste aplikaciju, umesto da za svaku stranicu pišete programski kod, u datoteku `web.config` možete dodati odeljak authorization. Evo podešavanja koja sam zadao u primeru AspNetLogin:

```
<authentication mode=«Forms«>
  <forms loginUrl=«WebForm2.aspx« name=«.authToken«/>
</authentication>
<authorization>
  <deny users=«?«/>
</authorization>
```

U programu postoji samo jedan formular (sa beskorisnim kalendarom), ali formular ne možete koristiti sve dok se ne prijavite. Kada pokušate da upotrebite formular, automatski ćete biti preusmereni na formular za prijavljivanje koji je zadat prethodnim podešavanjem. Formular za prijavljivanje sadrži standardna tekstualna polja za korisnižko ime i lozinku, i komandno dugme za koje se izvršava sledeći programski kod:

```
procedure TWebForm2.btnLogin_Click(
  sender: System.Object; e: System.EventArgs);
begin
  if textName.Text = textPassword.Text then
    FormsAuthentication.RedirectFromLoginPage(
      textName.Text, false);
end;
```

Iako je test u primeru trivijalan (u praksi ćete korisničko ime i lozinku proveravati u bazi podataka), pomoću sledećeg iskaza se poziva metod `RedirectFromLoginPage` ima značajnu ulogu. Metod RedirectFromLoginPage kontrolu vraća stranici koju je korisnik pokušao da vidi pre nego što se prikazala stranica za prijavljivanje i sistemu prosleđuje ime korisnika. To ime će se koristiti u aplikaciji pomoću objekta `Context.User`, kao u narednom isečku programskog koda koji je deo glavnog formulara programa:

```
procedure TWebForm1.Page_Load(
  sender: System.Object; e: System.EventArgs);
begin
  labelHello.Text := 'Hello, ' + Context.User.Identity.Name;
end;
```

Keširanje u arhitekturi ASP.NET

Druga važna osobina ASP.NET aplikacija koju želim da istaknem je keširanje. Keširanje postoji u dva oblika. Prvi oblik, koji se naziva izlazno keširanje (engl. output caching), je keširanje HTML teksta koji za stranice (ili za delove stanica) generiše sistem. Ukoliko znate da se podaci stranice neće promeniti tokom nekog vremena, bez obzira na to koji korisnik vidi stranicu, onda u stranicu možete da dodate sledeću deklaraciju:

```
<%@ OutputCache duration=«300« varbyparam=«none« %>
```

Pomoću prethodne direktive se podaci nalaze u kešu pet minuta, a zatim se vraćaju bez obzira na parametre. Ponekada će vam za svaku vrednost URL parametara biti potrebna kopija podataka koji su keširani.

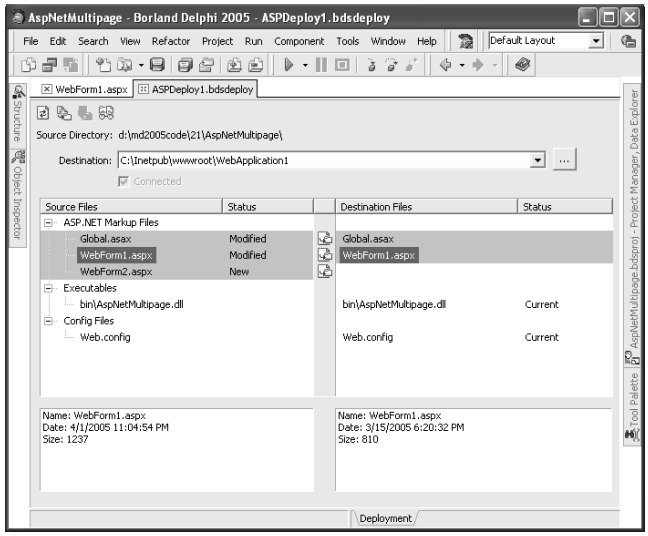
Potpuno drugačija tehnika je keširanje podataka aplikacije (engl. application data caching), to jest, keširanje podataka u memoriju za koje je potrebno neko vreme da se ponovo učitaju ili naprave. Na primer, možete keširati tabelu baze podataka ili XML dokument koji ste dobili sa udaljenog servera. U takvim situacijama možete koristiti kolekciju `Cache` aplikacije, to jest, niz elemenata u kojima se nalaze podaci sa prioritetom, trajanjem i zavisnostima sa ostalim keširanim elementima ili datotekama.

Prosleđivanje ASP.NET aplikacija

Kao što sam opisao u prethodnom poglavlju u kojem sam objasnio prosleđivanje IntraWeb aplikacija, u Delphiju 2005 postoji čarobnjak Deployment Manager koji je napravljen za prosleđivanje ASP.NET aplikacija.

Ukoliko odaberete bilo koji ASP.NET projekat, videćete direktorijum sa strelicom i ime Deplyment. Desnim tasterom miša kliknite direktorijum kako biste ga označili i iz kontekstnog menija odaberite komandu New ASP.NET Deployment, pa će se generisati nova .dbsdeploy datoteka koja otvara odgovarajući prozor. U njemu možete da odaberete odredišni direktorijum (lokalni, mrežni ili FTP lokaciju), a čarobnjak će prikazati datotke koje morate da prosledite i proveriće verziju na serveru, pa će vas obavestiti da kopirate samo novije datoteke. Primer vidite na slici 21.11 (primer AspNetMultipage).

SLIKA 21.11
Deployment Manager Delphija 2005 za ASP.NET aplikaciju



Šta je sledeće?

U ovom poglavlju sam napravio uvod u ASP.NET tehnologiju. Iako nisam detaljno objašnjavao tehnologiju ASP.NET, ovo poglavlje bi trebalo da vam pruži osećaj programiranja pomoću ove arhitekture i da vam posluži kao polazna tačka za dalje primere i literaturu. Nadam se da vam je ovo poglavlje, zajedno sa prethodna dva, pružilo sasvim dobar pregled alternativnih web tehnologija koje možete da koristite u Delphiju 2005. Više o tehnologiji ASP.NET ćete pronaći u narednom poglavlju koje se bavi XML tehnologijama. Saznaćete kako da u ASP.NET stranici koristite XML i XSLT, i kako da koristite XML datoteke zajedno sa DBWeb kontrolama. U poglavlju iza sledećeg, koje je posvećeno web uslugama, ćete nastaviti da koristite .aspx datoteke i ASP.NET tehnologije.