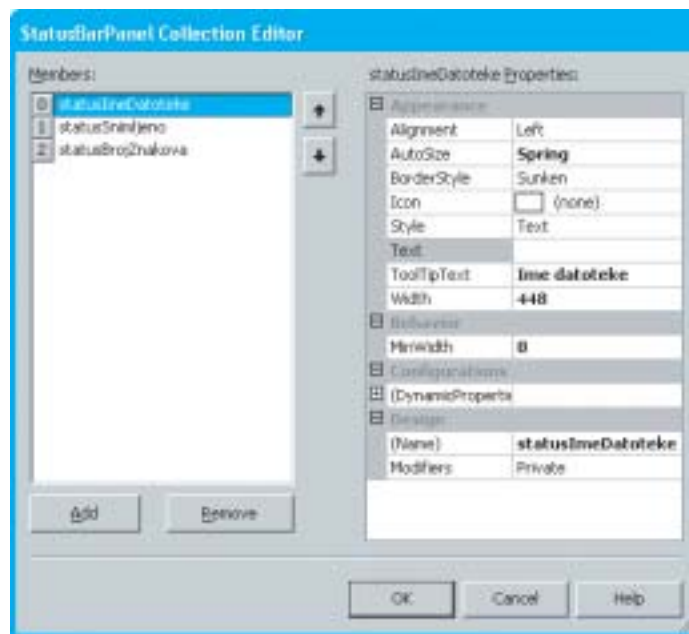


8. POGLAVLJE: WINDOWS FORMS

Slika 8-25:**Statusna traka s jednim podatkom i uključenim svojstvom SizingGrip**

Neki se vizualni i funkcionalni parametri panela definiraju na nivou cijele statusne trake, dok se ostali podešavaju na nivou pojedinog panela. Naravno, statusna traka koristi velik broj već kod drugih kontrola spomenutih svojstava (Font, Cursor, Enabled, TabIndex, TabStop, Context Menu...) pa nam spomenuti ostaje samo svojstvo SizingGrip. Ono određuje hoće li u donjem desnom uglu trake biti prikazan trokutić koji sam po sebi nema funkciju, no trebao bi korisnika podsjetiti na mogućnost razvlačenja forme (vidi sliku 8-25).

Ne uključujte svojstvo SizingGrip na formama koje se ne mogu razvlačiti jer ćete zbuniti korisnika.



Slika 8-26:
Pomoćni prozor za
dodavanje i uređivanje
panela

III. DIO: DIJELOVI .NET-A

Panelse definiramo pomoću svojstva `Panels` iza kojeg se krije pomoćni prozor vrlo sličan onome koji smo sreli kod trake s alatima, pa ćemo spomenuti samo razlike.

Svaki panel na statusnoj traci može biti određene veličine. Veličinu u pikselima određujemo svojstvom `Width`, no samo ako je svojstvo `AutoSize` postavljeno na vrijednost `None`. Osim te vrijednosti, svojstvo `AutoSize` može biti `Spring`, što znači da će se panel toliko povećati koliko treba da bi zauzeo sav slobodni prostor statusne trake. Ukoliko više panela imaju `AutoSize` postavljen na `Spring`, onda će taj slobodni prostor međusobno ravnopravno podijeliti. Treća dostupna vrijednost svojstva `AutoSize` je `Contents` koja panelu nalaže da svoju širinu prilagođava sadržaju koji se u njemu nalazi. Međutim, kako širina pojedinog panela ne bi pala ispod određene vrijednosti, možemo podesiti svojstvo `MinWidth` kojim u pikselima određujemo najmanju dozvoljenu širinu panela.

Mi smo u našem primjeru prvi panel postavili na `Spring`, ostala dva na vrijednost `Contents`, a kozmetike radi podesili smo i minimalne širine. Također, dali smo im logična imena – `statusImeDato` teke, `statusSnimljeno` i `statusBrojZnakova`.

Osim širine, panelu možemo određivati i izbočenost (svojstvo `BorderStyle`). Najčešći su uvučeni (`Sunken`) paneli, nešto češće se koriste ravni (`None`), a vrlo rijetko susrećemo izbočene panele (`Raised`). Primjere svih triju možete vidjeti na slici 8-27.

Slika 8-27:
Statusna traka s tri panela



U panele najčešće smještamo tekst (pogađate, svojstvo `Text`), a možemo postaviti i ikonu (svojstvo `Icon`). Hoće li tekst biti poravnan uz lijevi ili desni rub ili će biti centriran definiramo svojstvom `Alignment`.

Programiranje statusne trake

Iako statusna traka ima vlastite događaje, oni se rijetko koriste. Naime, kao što smo već spomenuli, ona se najčešće koristi za prikaz informacija, što znači da te informacije odnosno njihovu izmjenu moramo programirati na drugim mjestima u aplikaciji.

8. POGLAVLJE: WINDOWS FORMS

Kako bismo dovršili primjer Tekstualni editor, moramo popuniti statusnu traku trima informacijama. Prva je ime datoteke koju uređujemo, druga je status njezine snimljenosti i treća je broj znakova.

Želimo li u prvom panelu statusne trake imati uvijek aktualno ime datoteke, njegov sadržaj moramo osvježavati svaki put kada bi ime datoteke moglo biti promijenjeno. Promjena imena se može dogoditi prilikom stvaranja novog dokumenta, učitavanja i snimanja datoteke. Srećom, taj smo slučaj već imali prilikom spremanja imena datoteke u svojstvo `textBox1.Tag` tako da je dovoljno pronaći gdje sve pridružujemo vrijednom tom svojstvu i nakon njega dodati sljedeću liniju:

```
statusImeDatoteke.Text = textBox1.Tag;
```

Nemojte traženje dijelova kôda raditi ručno jer je jako vjerojatno da ćete nešto zaboraviti. Koristite vrlo bogate i moćne mogućnosti pretraživanja kôda i datoteka (u izborniku Edit, stavka Find and Replace).



Dosjetljiviji među vama uočiti će da smo nakon ove intervencije dobili dva svojstva koja uvijek imaju istu vrijednost pa možemo zaključiti da jedno od njih zapravo i nije potrebno. Drugim riječima, možemo izbaciti korištenje svojstva `textBox1.Tag` za pamćenje imena datoteke i sve što smo radili s njim možemo raditi pomoću vrijednosti svojstva `statusImeDatoteke.Text`. (Isto će vrijediti i za varijablu Mijenjano.)



Sljedeći panel treba držati informaciju o snimljenosti datoteke. I taj smo slučaj već imali (sjetite se varijable Mijenjano), pa će i u ovom slučaju biti dovoljno pronaći sva pridruživanja toj varijabli i nakon toga dodati liniju kôda:

```
if (Mijenjano) statusSnimljeno.Text = "nije snimljeno";
else statusSnimljeno.Text = "snimljeno";
```

Preostaje nam još zadnji panel, u kojem valja ispisati broj znakova. Ovo je najjednostavniji posao jer se broj znakova može promijeniti isključivo na jednom mjestu – prilikom promjene svojstva `textBox1.Text`. Kako postoji događaj vezan uz promjenu tog svojstva, naš jedini zadatak je u funkciju vezanu uz njega dodati podebljanu liniju kôda:

III. DIO: DIJELOVI .NET-A

```
private void textBox1_TextChanged(...)
{
    Mijenjano = true;
    if (Mijenjano) statusSnimljeno.Text = "nije snimljeno";
        else statusSnimljeno.Text = "snimljeno";
    statusBrojZnakova.Text = textBox1.TextLength.ToString() + " zn";
}
```

Broj znakova, kao što vidite iz primjera, čitamo iz svojstva `textBox1.TextLength` koje zbog spa-
janja sa *stringom* " zn" i pridruživanja svojstvu `statusBrojZnakova.Text` (također tipa *string*) kon-
vertiramo metodom `ToString()`.

I to je to! Tekstualni editor je gotov.

Primjer: Manipulator slikom

Treći veliki primjer ovog poglavlja bit će aplikacija nazvana Manipulator slikom. Cilj aplikacije je
pokazati još jednu skupinu kontrola koja se često koristi u prozorskim aplikacijama.

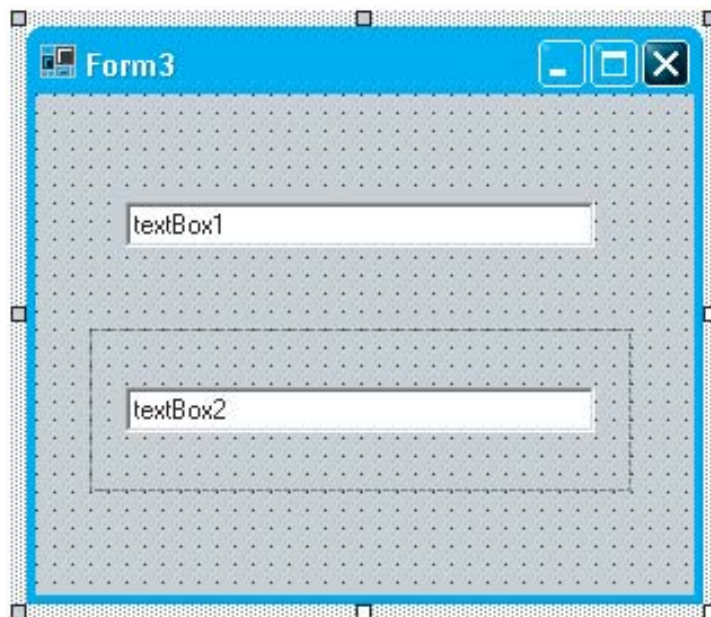
Slika 8-28:
Osnovno sučelje
aplikacije Manipulator
slikom



Za početak valja ukratko objasniti funkcionalnost aplikacije. Sučelje će biti podijeljeno u dva
dijela – s desne ćemo strane prikazati sliku veličine 200x300 piksela, dok ćemo s lijeve strane kre-
irati niz opcija koje će utjecati na prezentaciju slike.

Paneli

Prvo ćemo na formu dovući kontrolu Panel. To je najjednostavnija kontrola, koja praktički nema vlastitu funkcionalnost, nego služi isključivo ugošćavanju ostalih kontrola (služi kao spremnik, engl. *container*). Najbolje je cijelu stvar objasniti primjerom. Pogledajte sliku 8-29.



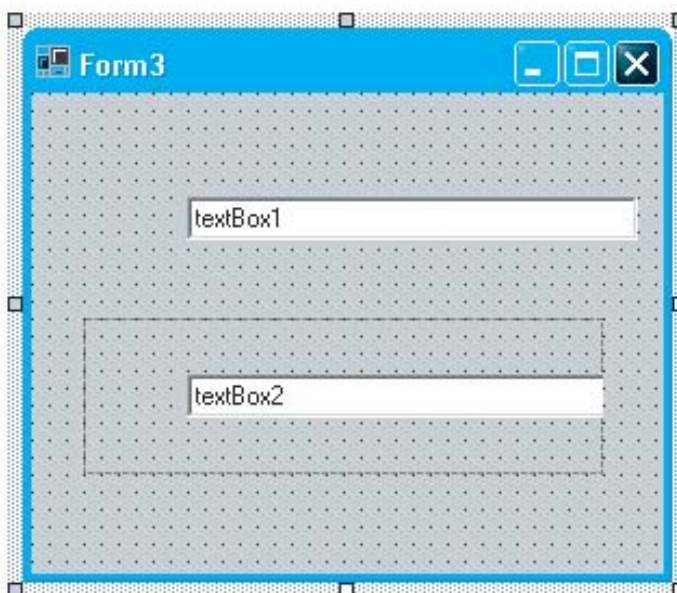
Slika 8-29:
*Demonstracija
uloge panela na
formi*

Kada bismo pokrenuli ovu aplikaciju, sve što bi korisnik vidio jesu dva polja za upis jednake veličine, jednako udaljena od lijevog ruba prozora u kojem se nalaze. Međutim, u dizajnerskom načinu vidimo i pravokutnik iscrtanog ruba koji nam ukazuje na postojanje panela odnosno činjenicu da se drugo polje za upis ne nalazi direktno na formi, već unutar tog panela.

Ako znamo da je između točkica na formi razmak od 8 piksela, jednostavno možemo izračunati poziciju kontrole `textBox1` – ona se nalazi 56 piksela od lijevog ruba (svojstvo `Left`) i 40 piksela od gornjeg ruba forme (svojstvo `Top`). Na isti način bismo mogli izračunati i lokaciju kontrole `textBox2`, kada se ona ne bi nalazila u panelu. Međutim, kako se spomenuta kontrola nalazi u panelu, njezina pozicija se računa u odnosu na panel, a ne na formu. Zato je vrijednost njezine lokacije 8; 34.

III. DIO: DIJELOVI .NET-A

Slika 8-30:
Pomicanje kontrola
unutar panela



Pogledajmo sada sliku 8-30. S obje kontrole smo napravili istu stvar – pomaknuli smo ih za 40 piksela udesno, no samo je prva kontrola u cijelosti vidljiva, dok je dio kontrole koji izlazi iz granica panela nevidljiv. Proširimo li i panel udesno, i druga kontrola će se vidjeti u cijelosti.

Pomaknemo li cijeli panel, u odnosu na formu ćemo pomaknuti i sve kontrole koje se u njemu nalaze. Istina, njihove će brojčane vrijednosti lokacije ostati neizmijenjene, no kako su one relativne u odnosu na panel, promjena lokacije panela promijenit će tek njihovu apsolutnu poziciju (poziciju u odnosu na formu).

Iako vam se cijela priča oko panela vjerojatno čini banalnom i nepotrebnom, postoji velik broj situacija u kojima ove karakteristike značajno doprinose jednostavnosti i brzini razvoja aplikacija. U nastavku razrade primjera primijetiti ćete nekoliko slučajeva, a u svojoj programerskoj karijeri naići ćete na barem još nekoliko.

Slike

Želite li na formi prikazati sliku, koristit ćete kontrolu PictureBox. Ona je znatno jednostavnija od kontrole ImageList i služi isključivo za prikaz slike na formi.

Jednu takvu kontrolu trebamo za naš primjer, no nećemo je direktno dovući na formu već ćemo je smjestiti unutar panela koji će zauzimati cijelu desnu polovicu forme (vidi sliku 8-28 – sivo područje).

8. POGLAVLJE: WINDOWS FORMS

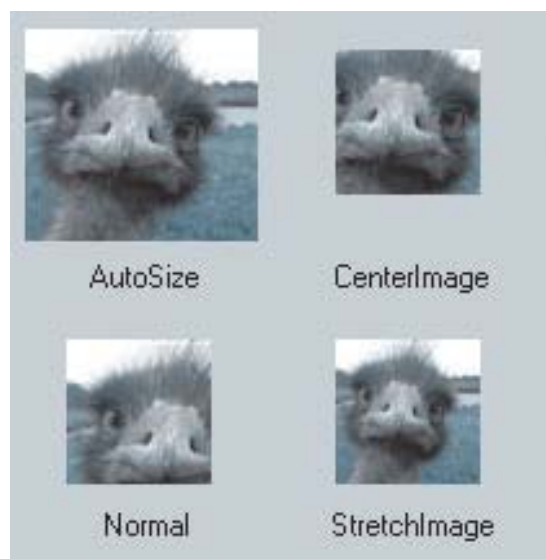
je oko slike je panel). Zašto smo sliku smjestili u panel i sve tako posložili, bit će vam jasno nešto kasnije.

Dovlačenje kontrola na panel jednostavno je kao i dovlačenje na formu. Željenu kontrolu izaberete na Toolboxu i odvučete je na panel.



Za panel nije bitno kako se zove (ostavili smo automatski dano ime), jer mu nećemo u primjeru direktno pristupati, no sa slikom ćemo puno toga raditi, pa smo je odmah preimenovali u "slika" (bez navodnika).

Prazna kontrola tipa PictureBox nema smisla ako se u njoj ne nalazi slika, pa je preko svojstva Image valja ubaciti. Slika će gotovo sigurno biti drugačijih dimenzija od kontrole pa ih valja međusobno prilagoditi što je najlakše učiniti postavljanjem svojstva SizeMode na vrijednost AutoSize. To će automatski povećati (ili smanjiti) kontrolu kako bi mogla prikazati cijelu sliku. Svojstvo SizeMode može poprimiti i druge vrijednosti – Normal će ignorirati originalnu veličinu slike i prikazati samo onaj dio slike koji stane u dimenzije kontrole, CenterImage će imati isti efekt (osim što neće prikazati gornji lijevi kut slike već njezin centralni dio) dok će vrijednost StretchImage tako razvući sliku da se prilagodi dimenzijama kontrole (vidi sliku 8-31).



Slika 8-31:
Usporedba ponašanja svojstva
SizeMode

III. DIO: DIJELOVI .NET-A

Svi za jednoga, jedan za sve

Sustavno organizirani tipovi podataka u .NET-u često rezultiraju vrlo zanimljivim mogućnostima. Naime, često ćete shvatiti da se neki tip svojstva podudara s nekim drugim, što znači da ih međusobno možete uspoređivati i, što je često korisnije, pridruživati.

Kako otkriti kojeg je tipa neko svojstvo? Osim čitanja dokumentacije, tip svojstva možete otkriti tako da napišete ime tog svojstva u kodu, te zatim nad njim zadržite pokazivač miša nekoliko trenutaka. Primjerice, ako pokazivač miša postavite nad riječi `Image` u sljedećem izrazu...

```
pictureBox1.Image
```

...otkriti dobit ćete sljedeću informaciju:

```
System.Drawing.Image PictureBox.Image
```

Iz nje možete iščitati tip svojstva (prvi dio) i tip kontrole kojem pripada (drugi dio).

Ako znamo da su i slike u kolekciji `imageList1.Images` istoga tipa, onda možemo napisati sljedeće:

```
pictureBox1.Image =  
imageList1.Images[0];
```

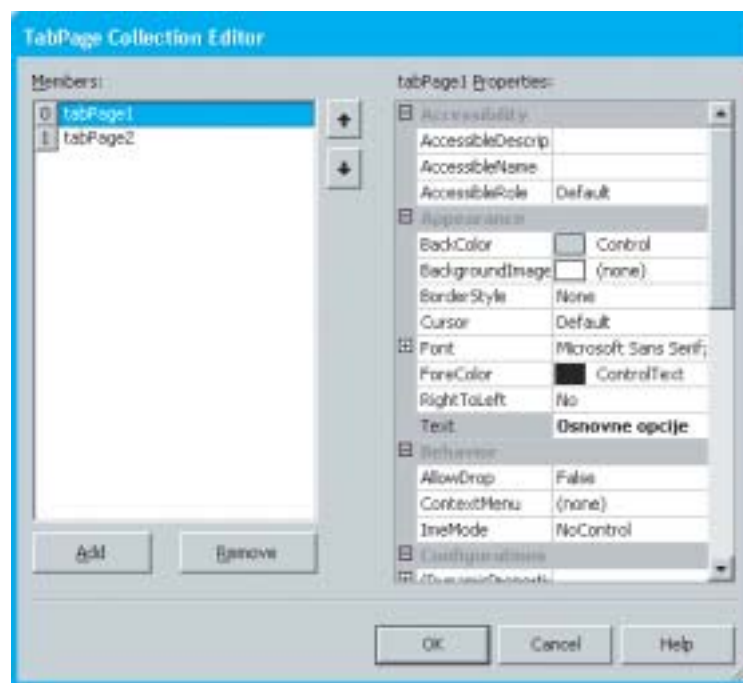
Taj izraz zamijenit će sliku u kontroli `pictureBox1` prvom slikom u kolekciji `imageList1.Images`.

Kartice

Na lijevoj strani našeg primjera (slika 8-28) primjećujete dvije tzv. kartice nazvane “Osnovne opcije” i “Veličina i pozicija” koje su kreirane pomoću kontrole `TabControl`. Iz prakse korištenja računala sigurno znate čemu te kartice služe i kako se ponašaju. Iz programerskog aspekta radi se o nekoliko panela, među kojima se prebacujemo klikom na naslov one koju želimo prikazati.

Nakon dovlačenja kontrole `TabControl` na formu primijetiti ćete da ne sadrži niti jednu karticu. Kartice dodajemo preko svojstva `TabPage` i pomoćnog prozora koji možete vidjeti na slici 8-32. Kako je svaka kartica zapravo panel (kontrola `TabPage` nasljeđuje velik broj svojih karakteristika od kontrole `Panel`), sve što smo u priči o panelima spomenuli vrijedi i ovdje.

Naravno, ima tu i nekih dodatnih mogućnosti. Primjerice, naslov kartice možemo odrediti kroz svojstvo `Text`, a moguće je i dodavati ikone (na već viđen način – roditeljsku kontrolu, koja je u ovom slučaju `TabControl`, povežemo s listom slika, a na nivou svake kartice upisujemo indeks ikone u toj listi).



Slika 8-32:
Pomoćni prozor za
upravljanje karticama
u kontroli TabControl

Vizualna svojstva (način prikaza, smještaj i slično) podešavamo većinom na roditeljskoj kontroli. Njima se ovaj puta nećemo baviti – ako vas zanima, poigrajte se samostalno.

Natpisi

Kontrola tipa Label vjerojatno je najjednostavnija kontrola koja postoji. Njezina glavna funkcija je ispisati neki tekst na ekranu iako bogatstvo svojstava i događaja koje sadrži omogućava da od nje napravite i mnogo više.

Mi ćemo spomenuti tek nekoliko osnovnih svojstava. U svojstvo Text unosite tekst koji želite da bude prikazan na ekranu. Svojstvom TextAlign određujete njegov smještaj u odnosu na površinu kontrole, a pomoću AutoSize uključujete automatsko prilagođavanje veličine kontrole sadržaju koji se u njoj nalazi. Naravno, tu je i cijela ergela već upoznatih svojstava: definiranje boja, dodavanje ikona, određivanje pokazivača miša, obrubi...

Na prvoj kartici kontrolu Label smo koristili dva puta kako bismo napisali natpise “Osnovne opcije” i “Tip ruba” (vidi sliku 8-28), a koristit ćemo je i na drugoj kartici (o tom potom).

Checkbox

Kontrolu CheckBox (u slobodnom prijevodu – kućicu s kvačicom) u primjeru koristimo za skrivanje i prikazivanje naše slike. Kao što smo već saznali, u tu se svrhu koristi svojstvo Visible pa

III. DIO: DIJELOVI .NET-A

ćemo u funkciju vezanu uz događaj `CheckedChanged` kontrole koju smo nazvali `checkPrikaz` upisati sljedeći kôd:

```
private void checkPrikaz_CheckedChanged(...)
{
    slika.Visible = checkPrikaz.Checked;
}
```

Uključenost odnosno isključenost kontrole `CheckBox` čita se iz svojstva `Checked`. Kad se vrijednost tog svojstva promijeni (dakle, kada korisnik klikne na kontrolu i promijeni joj stanje), poziva se gornja funkcija i prikaz slike (svojstvo `Visible`) se isključuje ukoliko je `CheckBox` isključen, i obrnuto.

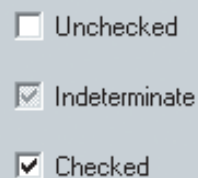
U našem je primjeru (i velikoj većini ostalih aplikacija) kućica s kvačicom smještena lijevo od teksta kontrole (pogađate, svojstvo `Text`). To je moguće promijeniti, i to pomoću svojstva `CheckAlign`, koje omogućava čak devet pozicija kućice u odnosu na tekst.

Slično kao kod gumba, i ovdje je moguće koristiti modernije (Flat i Popup) vrijednosti svojstva `FlatStyle`, a cijelu je priču moguće začiniti i slikom (svojstvo `Image`), cijelom listom slika (`ImageList` i `ImageIndex` – praktično je kada želite promijeniti sliku ovisno o uključenosti kontrole) te smještajem slike u odnosu na kontrolu (`ImageAlign`).

Kontrola `CheckBox` inicijalno je postavljena da klikom na nju automatski mijenja svoje stanje. Međutim, takvo je ponašanje moguće i isključiti preko svojstva `AutoCheck`. U tom ćete slučaju sami, pomoću događaja `Click`, morati isprogramirati što će se dogoditi kada korisnik klikne na kontrolu.



Slika 8-33:
Tri stanja kontrole `CheckBox`



Osim uključenosti i isključenosti, kontrola `CheckBox` može biti konfigurirana da poprimi i tzv. neodređeno stanje (vidi sliku 8-33). Da bismo to postigli, trebamo uključiti svojstvo `ThreeState`. Nakon toga stanje kontrole nećemo provjeravati preko svojstva `Checked`, već `CheckState` koje može poprimiti vrijednosti `Unchecked`, `Checked` i `Indeterminate` (u kodu ih referenciramo kao `CheckSta-`

te.Indeterminate). Također, događaj na koji se vežemo očekujući promjenu je `CheckStateChanged`.

Radiobutton

Svojevrсна blizanka kontrole `CheckBox` je `RadioButton`. Osim vizualne razlike (pravokutna prema okrugla), puno je važnija funkcionalna. Naime, dok *CheckBoxovi* rade samostalno, *radiobuttoni* surađuju s ostalim kontrolama svog tipa i dopuštaju da istovremeno izaberete samo jednu među njima. Izuzmemo li još i mogućnost trostrukog stanja, kontrola `RadioButton` je u svemu ostalom jednaka `CheckBoxu`.

Lijepo ime *radiobutton*...

Proučavanje imena stvari i tehnologija, posebno onih nastalih prije komercijalizacije naše industrijske grane, može biti vrlo zanimljivo jer nerijetko otkrijete neočekivane tokove misli onih koji su ih imenovali. Jednim od takvih imena može se podičiti i kontrola `RadioButton`.

Sjećate li se starih radioprijemnika za automobile koji su imali nekoliko gumba za prebacivanje između predefiniranih radijskih postaja? Kad ste kliknuli na jedan od njih, on bi ostao pritisnut, stanica bi se prebacila, a ostali bi gumbi iskočili u svoje izbočeno stanje. Taj

koncept i danas postoji na gotovo svim radioprijemnicima, jer je sasvim logično da ne možete odabrati više radiopostaja odjednom, samo što su izbočeno-uvučeni gumbi zamijenjeni LE-diodama i/ili prikazom broja stanice na zaslonu.

Zaključak okvira nazirete i sami – *radiobutoni* dobili su ime po tim radijskim gumbima. Ipak, kako je takvo imenovanje prilično neintuitivno, u literaturi namijenjenoj korisnicima aplikacija sve se češće koristi pojam *option button*.

Pitanje koje se samo po sebi nameće glasi: što ukoliko na nekoj formi trebamo više skupina *radiobuttona*? Tu u igru dolaze paneli (i svi drugi slični oblici spremnika poput kontrola `TabPage` ili `GroupBox`) koji ograničavaju međusobnu suradnju *radiobuttona*. Drugim riječima, grupiranje *radiobuttona* i njihovo odvajanje od ostalih skupina možete riješiti tako da ih stavite u zaseban spremnik.

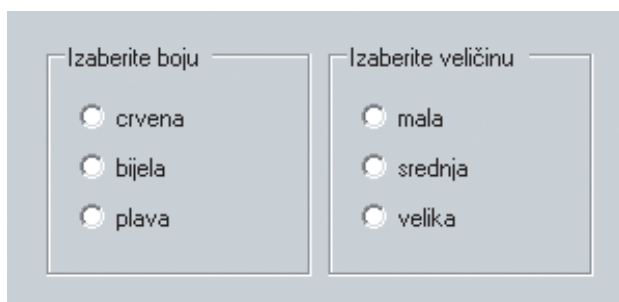
Obrada promjene odabranog radiobuttona

Kod promjene vrijednosti *radiobuttona*, za razliku od *checkboxa*, ne zanima nas status svakog pojedinog *radiobuttona* u skupini, već samo koji je od njih uključen. Kako je svaki *radiobutton*

III. DIO: DIJELOVI .NET-A

zaseban objekt, ne postoji događaj za slučaj promjene vrijednosti bilo kojeg *radiobuttona*, no možemo ga simulirati.

Slika 8-34:
Dvije skupine radiobuttona
na istoj formi, svaka u
svojoj kontroli tipa
GroupBox



Iskoristimo za to naš primjer. U prvu karticu (vidi sliku 8-28) dodajmo tri *radiobuttona* i imenujmo ih *radioBezRuba*, *radioTankiRub* i *radio3DRub*. Oni će služiti za mijenjanje obruba slike koja se nalazi s desne strane.

Označite prvu kontrolu i prebacite se u pomoćnom prozoru Properties na listu događaja te dvokliknite na događaj *CheckedChanged*. Otvorit će vam se funkcija koju preporučujemo da u kodu preimenujete jer trenutno u sebi sadrži ime kontrole kojoj pripada, a uskoro ćemo je povezati i s druge dvije. Mi smo je nazvali *radioTipRuba_CheckedChanged*. Nakon preimenovanja potrebno je vratiti se u dizajnerski način te i kod događaja *CheckedChanged* izmijeniti ime funkciji. To ne morate učiniti ručno, već pomoću padajućeg izbornika (vidi sliku 8-35).

Sada istu stvar treba napraviti i za ostale dvije kontrole, *radioTankiRub* i *radio3DRub*. Naravno, kod njih preskačete dvoklik za stvaranje nove funkcije; samo iz padajućeg izbornika izaberete istu funkciju koju ste izabrali i za kontrolu *radioBezRuba*. Rezultat – svaki put kada se promijeni svojstvo *CheckedChanged* neke od kontrola bit će pozvana ista funkcija, funkcija *radioTipRuba_CheckedChanged*.

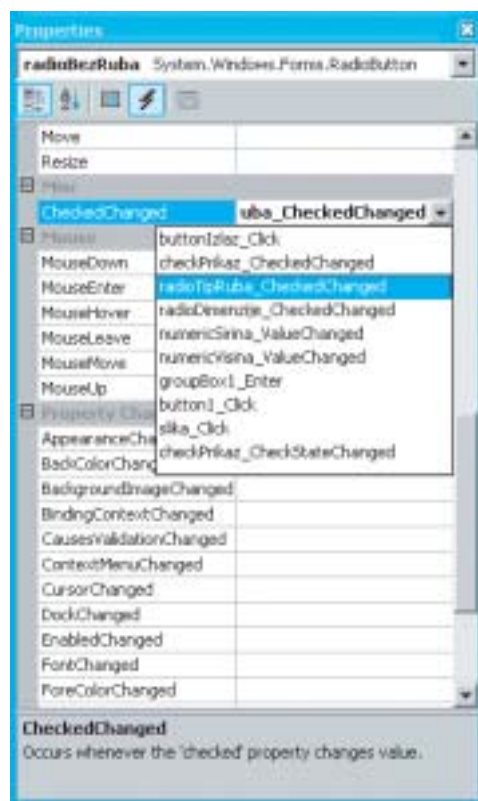
U toj funkciji moramo provjeriti koji je *radiobutton* nakon promjene označen i sukladno tome promijeniti tip obruba slike (svojstvo *slika.BorderStyle*). Evo kako to izgleda u praksi:



Spomenuti “trik” s pridruživanjem iste funkcije većem broju kontrola ne pali uvijek. Postoje slučajevi kada je bitno da svaki *radiobutton* pokreće drugu funkciju, bez obzira na to što su povezani.

8. POGLAVLJE: WINDOWS FORMS

```
private void radioTipRuba_CheckedChanged(...)
{
    if (radioBezRuba.Checked)
    {
        slika.BorderStyle = BorderStyle.None;
    }
    else if (radioTankiRub.Checked)
    {
        slika.BorderStyle = BorderStyle.FixedSingle;
    }
    else if (radio3DRub.Checked)
    {
        slika.BorderStyle = BorderStyle.Fixed3D;
    }
}
```



Slika 8-35:
Padajući izbornik koji omogućava povezivanje događaja s bilo kojom postojećom funkcijom koja ima iste parametre

III. DIO: DIJELOVI .NET-A

Gornji primjer nije dobar iako bez problema odrađuje ono što smo željeli. Zašto? Zato što se klikom na jedan *radiobutton* zapravo mijenjaju vrijednosti dvaju *radiobuttona* – onaj na koji kliknemo promijenit će svojstvo *Checked* na *true*, dok će onom koji je dotada bio izabran svojstvo *Checked* biti postavljeno na *false*. S obzirom na to da se promjena dogodila na dva mjesta, funkcija će biti pozvana dva puta. To u našem malom primjeru nema velikih negativnih posljedica jer je funkcionalnost vrlo jednostavna, no kod složenijih će primjera to biti uzaludno trošenje sistemskih resursa. Zato je sljedeći primjer puno primjereniji:

```
private void radioTipRuba_CheckedChanged (object sender, System.EventArgs e)
{
    RadioButton kliknut = (RadioButton) sender;
    if (kliknut.Checked)
    {
        if (kliknut == radioBezRuba)
        {
            slika.BorderStyle = BorderStyle.None;
        }
        else if (kliknut == radioTankiRub)
        {
            slika.BorderStyle = BorderStyle.FixedSingle;
        }
        else if (kliknut == radio3DRub)
        {
            slika.BorderStyle = BorderStyle.Fixed3D;
        }
    }
}
```

Primjećujete da po prvi put proučavamo funkciju prvog parametra funkcije vezane na događaj. Taj prvi parametar je tipa *object*, što znači da može sadržavati bilo kakvu vrijednost. U našem slučaju on sadrži onaj *radiobutton* čiji je događaj inicirao pozivanje funkcije. To vrijedi i općenito – parametar *sender* sadrži onu kontrolu kojoj pripada događaj koji je pozvao funkciju.

Međutim, kako je taj parametar tipa *object*, ne možemo ga direktno koristiti kao kontrolu (u našem slučaju *radiobutton*) već prije moramo izvršiti eksplicitnu konverziju (tzv. *castanje* o kojem smo govorili u petom poglavlju). To radimo u prvom redu funkcije – definiramo varijablu *kliknut*, tipa *RadioButton*, kojoj odmah pridružujemo vrijednost. Vrijednost pak dobivamo *castanjem* parametra *sender* u tip *RadioButton*. Nakon toga svako referenciranje varijable *kliknut* zapravo je referenciranje na *radiobutton* čiji je događaj pozvao funkciju.

Rekli smo da se funkcija poziva dvaput – jednom od strane označenog, drugi put od strane neoznačenog *radiobuttona*. Stoga, da bismo je izvršili samo jednom, dodali smo uvjet koji provjera-

8. POGLAVLJE: WINDOWS FORMS

va je li *radiobutton* koji je pozvao funkciju ovaj prvi (označen) te, ako jest, izvršavamo kôd unutar uvjeta. Ako nije, znači da je funkcija pozvana od neoznačenog *radiobuttona*, pa nema potrebe za (ponovnim) izvršavanjem koda.

Ako vas baš zanima koji će *radiobutton* prvi pozvati funkciju, točan odgovor je – onaj neoznačeni.



U drugom smo primjeru, u odnosu na prvi, izmijenili i ostale uvjete. Nije da “stari” nisu mogli ostati, već čisto da pokažemo kako je moguće uspoređivati dva objekta te i na taj način utvrditi koji je od potencijalnih kliknut).

Na kraju ove priče valja spomenuti da smo sličan efekt mogli postići i vezanjem na događaj Click. Štoviše, u tom slučaju funkcija ne bi bila pozvana dva puta jer će uvijek biti samo jedan *radio-button* na koji je kliknuto. Međutim, taj pristup ima dvije mane.

Prva je mogućnost da se vrijednost *radiobuttona* ne promijeni klikom već, primjerice, naredbom u kodu. U tom slučaju naša funkcija vezana uz Click ne bi bila pozvana i program ne bi radio kako očekujemo. Zato je izuzetno važno, ne samo u ovom primjeru nego i općenito, razmisliti na koji događaj vezati funkcionalnosti kako bi on obuhvatio sve promjene koje želimo obuhvatiti.

Druga mana je praktične prirode – da se nismo odlučili za ovakav pristup, ne bismo imali priliku pokazati *castanje* parametra *sender*, koje zna biti vrlo praktično u brojnim situacijama.

Brojčanik

Kako nam na ovoj kartici ne preostaje puno mjesta, prebacimo se na sljedeću (slika 8-36). Na njoj ćemo se igrati dimenzijama i pozicijom slikovne kontrole.

Prema slici 8-36 dovucite još četiri *radiobuttona* i dodijelite im, radi lakšeg snalaženja, imena *radioAutoVelicina*, *radioRazvucena*, *radioNormalna* i *radioCentrirana*. Osim toga, trebaju nam i dva brojčanika koja se skrivaju iza kontrole imena *NumericUpDown*. Njih nazovite *numericSirina* i *numericVisina*. (Mi smo dodali i pokoju kontrolu tipa *Label*, čisto radi jasnoće sučelja.)

Upravo će se dodatni *radiobuttoni* brinut za svojstvo *SizeMode* koje pripada kontroli slika. Funkcionalnost mijenjanja svojstva sada znate napraviti i sami (vidi priču o mijenjanju obruba) pa će funkcija (pridružena događajima *CheckedChanged* svih četiriju *radiobuttona*) izgledati ovako:

III. DIO: DIJELOVI .NET-A



Slika 8-36:
*Druga kartica primjera
Manipulator slikom*



```
private void radioDimenzije_CheckedChanged
    (object sender, System.EventArgs e)
{
    RadioButton kliknut = (RadioButton) sender;
    if (kliknut.Checked)
    {

        // tu ćemo dodati još kôda (A)

        if (radioAutoVelicina.Checked)
        {
            slika.SizeMode = PictureBoxSizeMode.AutoSize;
            // tu ćemo dodati još kôda (B)
        }
        else if (radioRazvucena.Checked)
        {
            slika.SizeMode = PictureBoxSizeMode.StretchImage;
        }
        else if (radioNormalna.Checked)
        {
            slika.SizeMode = PictureBoxSizeMode.Normal;
        }
    }
}
```


8. POGLAVLJE: WINDOWS FORMS

```

    }
    else if (radioCentrirana.Checked)
    {
        slika.SizeMode = PictureBoxSizeMode.CenterImage;
    }
}
}

```

Međutim, kada je odabrana vrijednost `AutoSize`, onda nam brojčanici za određivanje dimenzija kontrole nisu potrebni, pa ih treba zasiviti. Drugim riječima, omogućenost brojčanika (`numericSirina.Enabled`) treba biti inverzno vrijednosti od označenosti *radiobuttona* `radioAutoVelicina` (`radioAutoVelicina.Checked`). Dodajmo stoga u kôd na mjesto označeno komentarom (A) sljedeće linije:

```

    numericSirina.Enabled = !(radioAutoVelicina.Checked);
    numericVisina.Enabled = !(radioAutoVelicina.Checked);

```

Na mjesto (B) treba ubaciti kôd koji će sinkronizirati vrijednosti u brojčanicima sa stvarnom veličinom kontrole sa slikom nakon odabira automatskog određivanja veličine kontrole. Naime, već smo spomenuli da se veličina kontrole, ako je svojstvo `SizeMode` postavljeno na vrijednost `AutoSize`, automatski prilagođava veličini slike, što znači da će dimenzije kontrole biti izmijenjene. Kako naši brojčanici prikazuju te dimenzije, nakon što se dogodi automatsko prilagođavanje treba popraviti vrijednosti u njima. To radimo na sljedeći način:

```

    numericSirina.Value = Convert.ToDecimal(slika.Size.Width);
    numericVisina.Value = Convert.ToDecimal(slika.Size.Height);

```

Tu dolazimo do prvog nepoznatog svojstva kontrole `NumericUpDown`, svojstva `Value`. Kao što vidite iz primjera, radi se o vrijednosti tipa *decimal*, što znači da prije pridruživanja vrijednosti `slika.Size.Width` i `slika.Size.Height` (koje su tipa *int*) moramo napraviti konverziju pomoću metode u klasi `Convert` (vidi peto poglavlje).

Kontrole `NumericUpDown`, kao što možete vidjeti, služe za upis ili odabir neke brojčane vrijednosti. Vrijednosti koje može poprimiti definiramo pomoću nekoliko svojstava. Broj decimalnih mjesta upisujemo u svojstvo `DecimalPlaces`. Minimalnu i maksimalnu vrijednost u svojstva `Minimum` i `Maximum`, dok korak povećanja (koji će nastupiti kada kliknemo na gumbiće za povećanje odnosno smanjenje vrijednosti u kućici) podešavamo pomoću svojstva `Increment`.

Najkorišteniji događaj ove kontrole je `ValueChanged` koji, pogađate, nastupa kada se promijeni vrijednost svojstva `Value`, bez obzira je li do nje došlo pritiskom na gumbiće, ručnim unosom broja ili promjenom iz kôda.

III. DIO: DIJELOVI .NET-A

U primjeru uz svaki brojčanik vežemo zasebnu funkciju u kojima također moramo raditi konverziju, ovoga puta u drugom smjeru:

```
private void numericSirina_ValueChanged(...)
{
    slika.Width = Convert.ToInt16(numericSirina.Value);
}

private void numericVisina_ValueChanged(...)
{
    slika.Height = Convert.ToInt16(numericVisina.Value);
}
```

Ostale kontrole

Kontrola, dakako, ima još. Nažalost, zbog ograničenog prostora i brojnih tema kojih se još moramo dotaknuti, ne preostaje nam ništa drugo, nego ih samo ukratko spomenuti (tablica 8-2) i navesti čemu služe, a njihovo korištenje i proučavanje ostaviti vama. Podsjećamo da se reference svih kontrola nalaze u MSDN Library, na putanji .NET Development > .NET Framework SDK > NET Framework > Reference > Class Library > System.Windows.Forms.

Dodavanje novih kontrola

Iako kontrola koje dolaze unutar .NET Frameworka ima mnogo, kad-tad ćete poželjeti koristiti neku novu, koja se ne nalazi u originalnoj postavi. Osim što ih možete sami napraviti, možete ih i skinuti s Interneta i, ovisno o proizvođaču, besplatno ili uz naknadu koristiti u svojim aplikacijama.

Svaka kontrola dolazi s vlastitom dokumentacijom, prema kojoj možete shvatiti kako se koristi i kako radi. Međutim, dodavanje kontrola radi se uvijek na isti način.

Tu se vraćamo na priču o *assemblyjima*. (Sjeća li se tko drugog poglavlja?) Naime, svaka kontrola je zapravo *assembly* koji treba dodati na dohvat aplikacije u kojoj ga želimo koristiti. Za to imamo dva rješenja – smjestiti kontrolu u Global Assembly Cache ili u mapu *bin* aplikacije koju radimo.

Naravno, zahvaljujući Visual Studiju, mi se tim “sitnicama” ne moramo zamarati. Kroz sljedeći ćemo primjer pokazati kako s Interneta skinuti, instalirati i koristiti neku kontrolu.

Za primjer smo odabrali skupinu kontrola nazvanih SandBar, koje možete preuzeti s adrese <http://www.divil.co.uk/net/controls/sandbar/downloads.asp>. Radi se o kontrolama za izradu atraktivnih traka s alatima i izbornika u stilu Microsoft Officea 2003.

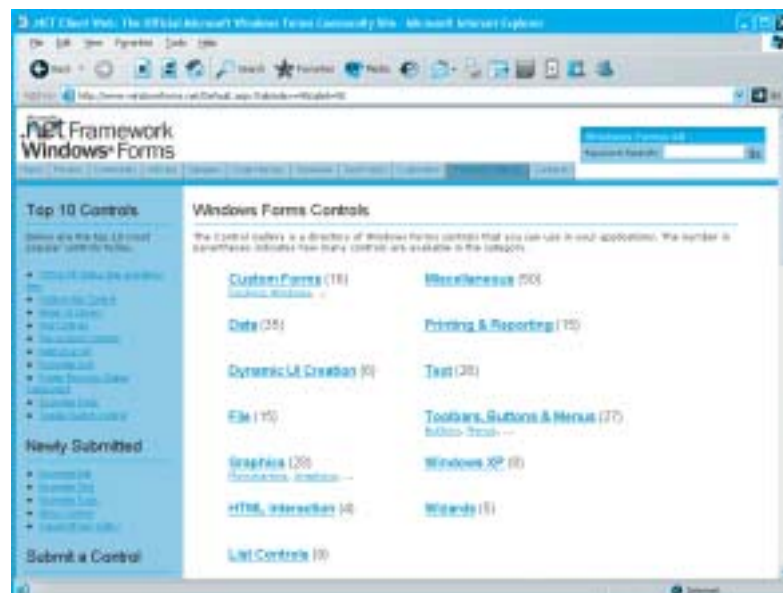
U arhivu koju ćete skinuti sa spomenutih stranica bitna je samo jedna datoteka – ona s nastavkom “.dll”. Ostale datoteke najčešće su dokumentacija i/ili konkretan primjer aplikacije u kojoj

Tablica 8-2:
Popis nekih kontrola koje nismo uspjeli detaljnije obraditi u poglavlju. Neke od njih bit će objašnjene u narednim poglavljima.

Ime kontrole	Kratki opis
CheckedListBox	isto kao i ListBox, samo što svaki zapis ima
CheckBox	ColorDialog dijaloški okvir za izbor boje
DataGrid	tablica za prikaz podataka iz baze (više o tome u sljedećem poglavlju)
DateTimePicker	kućica za unos datuma s kalendarom
DomainUpDown	mješavina ComboBoxa i NumericUpDown – izbor <i>stringa</i> na način kontrole NumericUpDown
FolderBrowserDialog	dijaloški okvir za izbor mape na disku
FontDialog	dijaloški okvir za izbor fonta
HScrollBar	vodoravni kliznik
LinkLabel	label s mogućnošću pretvaranja u hiperlink
ListView	kontrola za prikaz stvari (poput mapa i datoteka u Windows Exploreru)
MonthCalendar	kalendar
NotifyIcon	ikona programa u <i>trayu</i> (kraj sata)
PageSetupDialog	dijaloški okvir za prilagođavanje stranice
PrintDialog	dijaloški okvir za ispis
PrintPreviewControl	kontrola za pregled dokumenta za ispis (PrintDocument)
PrintPreviewDialog	dijaloški okvir pregleda dokumenta prije ispisa
ProgressBar	prikaz napretka
RichTextBox	tekstualno polje za upis, s mogućnošću prikaza formatiranog teksta u formatu RTF
Splitter	razdjelnik u sučelju
Timer	štoperica; koristi se za ponavljanje određene radnje u određenom vremenskom periodu (svake sekunde, recimo)
ToolTip	pridruživanje <i>tipova</i> kontrolama koje ih inicijalno nemaju
TrackBar	traka s pomičnim klizačem
TreeView	stablasti prikaz strukture
VScrollBar	okomiti kliznik

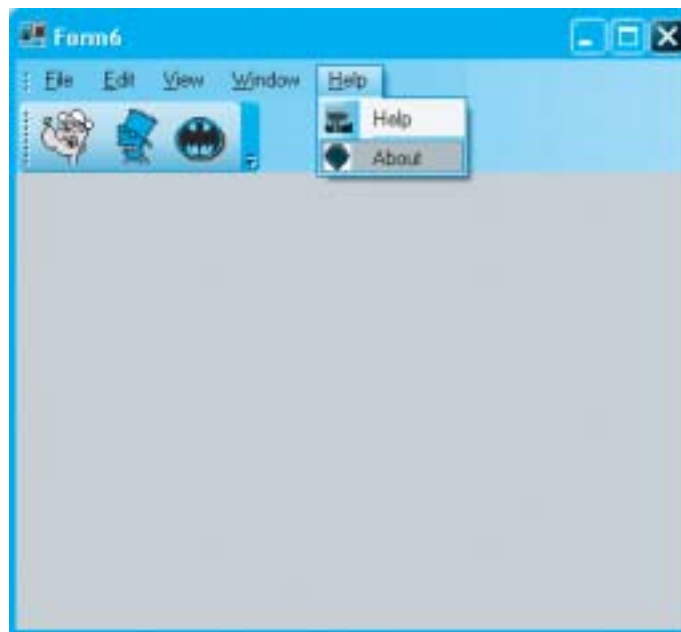
III. DIO: DIJELOVI .NET-A

Slika 8-37:
Službena stranica zajednice Windows Forms na kojoj, između ostalog, možete pronaći i dodatne kontrole – adresa je <http://www.windowsforms.net>



se kontrola koristi. (Ti primjeri ponekad su u Visual Basicu .NET pa vam u “prijevodu” puno može pomoći dvojezičnost trećeg poglavlja.)

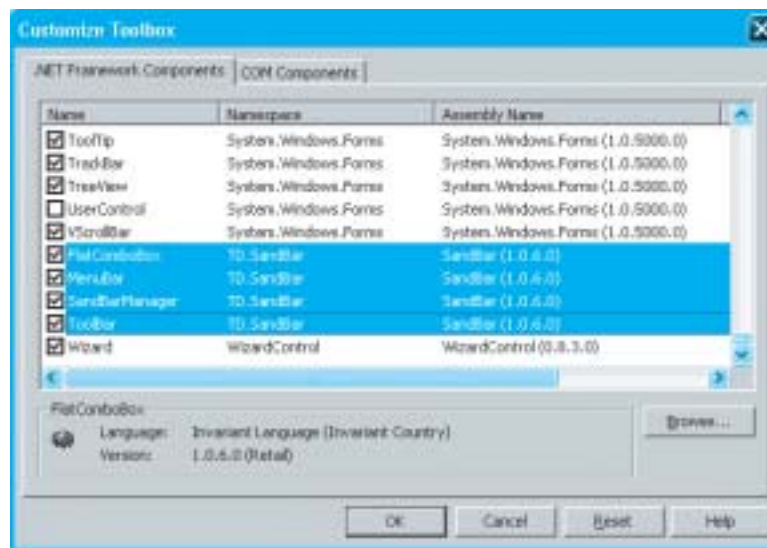
Slika 8-38:
Skupina kontrola SandBar omogućava puno efektivnije izbornike i alatne trake od standardnih.



8. POGLAVLJE: WINDOWS FORMS

Tu datoteku, koja se u našem slučaju zove SandBar.dll, treba smjestiti na neko sigurno mjesto gdje neće smetati. Predlažemo vam da otvorite posebnu mapu na disku u kojoj ćete držati komponente, najbolje zajedno s primjerima i dokumentacijom. Nebitno je gdje se ta mapa nalazi u odnosu na ostale datoteke, glavno da vi znate gdje je i kako do nje doći.

Sljedeći korak je dodavanje kontrola u pomoćni prozor ToolBox kako bi nam bile pri ruci kada ih trebamo. To radimo tako da kliknemo desnom tipkom miša negdje u tom prozoru i iz padajućeg izbornika izaberemo stavku Add/Remove Items. Otvorit će nam se prozor Customize Toolbox u kojemu valja kliknuti na gumb Browse i pronaći datoteku DLL koju smo maloprije smjestili na sigurno.



Slika 8-39:
Prozor Customize
Toolbox, koji služi za
dodavanje kontrola u
Toolbox

Nakon potvrde odabira, pojavit će nam se informacije o kontrolama koje su u toj datoteci pronađene. U našem se primjeru, kao što vidite na slici 8-39, radi o četiri kontrole koje su automatski označene i spremne za dodavanje na listu s ostalim kontrolama. Još pritisak na OK i možete ih početi koristiti...

Sve što dalje s njima radili izgledat će kao da se radi o najobičnijim kontrolama. Naravno, uvijek možete naići na kontrole koje nemaju dobru integraciju s Visual Studiojem, no kod kvalitetnijih nećete imati takvih problema.

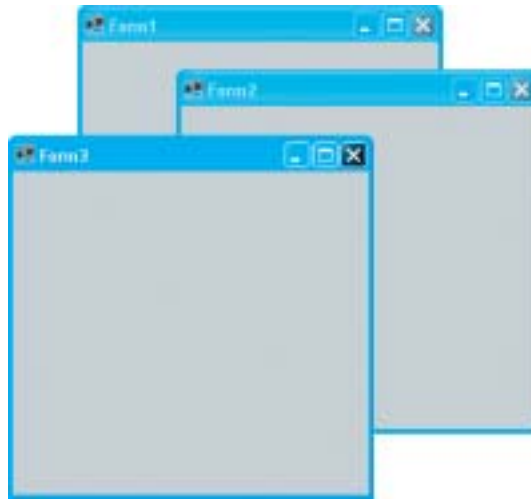
Svaki put kada odvučete kontrolu na formu, u pozadini će Visual Studio iskopirati datoteku s nastavkom ".dll" u mapu *bin* aplikacije u kojoj se koristi. Naravno, vaš jedini zadatak je da kod distribucije ne zaboravite spakirati i tu datoteku, a ne samo izvršnu datoteku aplikacije.

III. DIO: DIJELOVI .NET-A

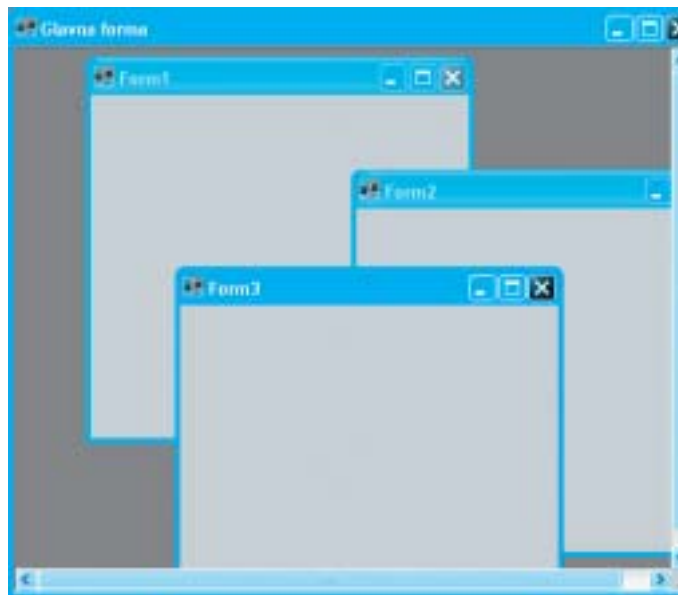
Rad s više formi

Rijetke su aplikacije koje imaju samo jednu formu. Makoliko primjer bio jednostavan, često će vam zatrebati još koja, bilo kao pomoćni prozor za neku radnju, bilo kao punopravni prozor aplikacije.

Slika 8-40:
Single Document Interface



Slika 8-41:
Multiple Document Interface



8. POGLAVLJE: WINDOWS FORMS

Ovisno o načinu otvaranja formi, aplikacija može imati Single Document Interface (SDI) ili Multiple Document Interface (MDI). U aplikaciji sa SDI-jem svaka je forma samostalna i prikazuje se u *taskbaru* operativnog sustava (iako se potonje, kao što smo već pokazali, dadne isključiti). Najbolji primjer ovakvog ponašanja je Internet Explorer – radi se o istoj aplikaciji, no kada otvaramo stranicu u novom prozoru, otvara se nova forma.

Primjer aplikacije sa MDI-jem je Microsoft Access. Učitamo li neku bazu i zatim otvorimo nekoliko njezinih tablica, svaka će se tablica otvarati u zasebnom prozoru (formi), no sve će se događati unutar klijentskog područja istoga, glavnog prozora. Taj glavni prozor naziva se spremnik MDI-ja (engl. *MDI container*).

Dodavanje i korištenje nove forme

Isprobajmo to na primjeru. Pretpostavimo da imate već napravljenu neku formu u jednoj aplikaciji i dođe vrijeme da dodate drugu. Jednostavno krenete u izbornik Visual Studija i pod glavnom stavkom Project odaberete Add Windows Form. Otvorit će vam se nova, prazna forma, kojoj možete dodavati kontrole i funkcionalnosti na jednak način kao što ste to radili s prvom.

Izvjesno je da ćete u nekom trenutku te forme željeti povezati odnosno da ćete iz prve željeti pozvati drugu. Recimo da ste za tu svrhu na prvoj postavili gumb klikom na koji bi druga forma trebala biti otvorena. Funkcija vezana uz događaj klika izgledat će ovako (naravno, isti kôd možete vezati uz bilo koji drugi događaj):

```
private void button1_Click(object sender, System.EventArgs e)
{
    Form2 Fo2 = new Form2();
    Fo2.Show();
}
```

Da biste razumjeli što smo upravo napravili, treba se sjetiti da slaganjem forme u Visual Studiju zapravo radimo novu klasu. Klasa (forma) koju smo preko izbornika dodali automatski je nazvana Form2. Sjetimo se također da je klasa samo nacrt za objekt koji će biti kreiran, pa stoga u prvoj liniji radimo upravo to – stvaramo objekt Fo2 na temelju nacrta klase Form2 (sintaksu za stvaranje objekta, vjerujemo, prepoznajete). Nakon što je objekt stvoren, treba ga prikazati, a to činimo metodom Show().

Dijaloški okviri su forme. Ugrađeni dijaloški okviri su forme s već gotovom funkcionalnošću, a forme koje otvaramo metodom ShowDialog() možemo zvati dijaloški okviri.



III. DIO: DIJELOVI .NET-A

Otvaranjem nove forme na ovaj način primijetit ćete jednu zanimljivu karakteristiku. Naime, obje su forme ravnopravne i možete raditi malo u jednoj, malo u drugoj. Međutim, puno češće nam treba funkcionalnost kojom ćemo primorati korisnika da prvo obavi sav posao u novootvorenoj i da se tek onda smije vratiti na osnovnu. Takvo smo ponašanje već susreli kod dijaloških okvira (kod snimanja i učitavanja datoteka), a kod formi se postiže na isti način, istom metodom:

```
private void button1_Click(object sender, System.EventArgs e)
{
    Form2 Fo2 = new Form2();
    Fo2.ShowDialog();
}
```



Ako otvaramo formu metodom ShowDialog(), kôd nakon te metode neće biti izvršen sve dok korisnik ne zatvori otvorenu formu i vrati se na osnovnu. Ako koristimo metodu Show(), kôd će se izvršiti odmah po otvaranju forme, bez čekanja zatvaranja ili vraćanja na osnovnu.

Zatvaranje otvorene forme možete vršiti iz nje same, i to na sljedeći način:

```
Close();
```

Kao što smo već spomenuli, zatvaranje osnovne forme rezultirat će izlazom iz aplikacije.

Otvaranje formi u okruženju MDI nešto je složenije. Za početak, osnovnu formu moramo proglasiti spremnikom za MDI pomoću svojstva IsMDIContainer. Nakon toga treba kreirati novu formu (ili više njih, na isti način) te ih iz osnovne pozivati na sljedeći način:

```
private void menuNovi_Click(object sender, System.EventArgs e)
{
    Form2 Fo2 = new Form2();
    Fo2.MdiParent = this;
    Fo2.Show();
}
```

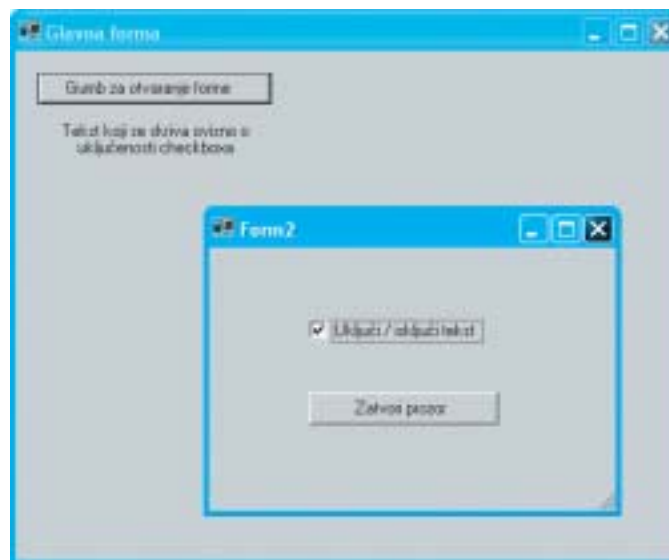
Primjećujete da smo dodali još jednu liniju kojom definiramo roditeljsku formu onoj koju otvaramo. Ključnom riječju *this* mislimo na formu u kojoj se trenutno nalazimo (dakle, kako formu otvaramo iz osnovne forme, mislimo na nju).

Dvojba koja se u ovom slučaju javlja jest na koji događaj povezati otvaranje forme u MDI okruženju. Naime, neozbiljno bi bilo staviti gumb (iako to, u testne svrhe, možete napraviti) jer će on biti prikazan iznad svih MDI-formi ili bilo kakvu drugu kontrolu tog tipa. Zato se najčešće u roditeljske MDI-forme stavljaju samo izbornik i statusna traka koje su pričvršćene uz rub forme (engl. *dock*) pa tako ne smetaju.

Prenošenje vrijednosti među formama

Kako sve forme pripadaju određenoj aplikaciji, vrlo je vjerojatno da će međusobno morati izmjenjivati određene podatke. To možemo postići na nekoliko načina, a mi ćemo vam pokazati najjednostavniji među njima.

Cijela priča temelji se na teoretskim osnovama obrađenima u petom i šestom poglavlju. Mi smo među iskoristivim rješenjima izabrali ono koje se temelji na javnim (*public*) članovima klase, konkretno varijablama.



Slika 8-42:
Primjer za demonstraciju
prenošenja vrijednosti među
formama

Uzmimo za primjer da imamo dvije forme – Form1 i Form2 te da je prva među njima osnovna. Na prvoj se nalazi gumb koji će nam poslužiti za otvaranje druge forme i kontrola tipa Label, a na drugoj se nalazi *checkbox* koji će služiti za pokazivanje i sakrivanje kontrole Label na prvoj formi. Da bismo mogli iz prve forme pristupiti stanju *checkboxa*, uvest ćemo javnu varijablu *checkboxVrijednost*.

III. DIO: DIJELOVI .NET-A

U kôd druge forme (Form2) dodat ćemo sljedeće (najbolje odmah nakon konstruktora klase):

```
public bool checkBoxVrijednost
{
    get
    {
        return checkBox1.Checked;
    }
    set
    {
        checkBox1.Checked = value;
    }
}
```

Riječju *public* određujemo da varijabla tipa *boolean* bude dostupna svima koji koriste tu klasu (vidi šesto poglavlje). Zatim ključnim riječima *get* i *set* definiramo kako će se varijabla ponašati kada čitamo njenu vrijednost (*get*) odnosno mijenjamo (*set*). Tako kažemo: kada netko bude tražio vrijednost naše varijable, vrati mu (*return*) vrijednost svojstva `checkBox1.Checked`. S druge strane, kada neko pridružuje novu vrijednost našoj varijabli, tu vrijednost (*value*) pridruži svojstvu `checkBox1.Checked`. Na taj način ćemo postići potpunu sinkroniziranost (javne) varijable `checkBoxVrijednost` i (privatnog) svojstva `checkBox1.Checked`.



Sve kontrole na formi automatski su dosega *private* i zato im nije moguće pristupiti iz ostalih klasa (formi).

Mogli smo izabrati i drugačiji pristup – varijablu `checkBoxVrijednost` samo definirati, a brigu o njezinoj vrijednosti prepustiti događajima kontrole `checkBox1`. Dakako, takav pristup bio bi znatno nespretniji.

Vratimo se sada na prvu, osnovnu formu i razmotrimo funkciju vezanu uz događaj klika na gumb:

```
private void button1_Click(object sender, System.EventArgs e)
{
    Form2 Fo2 = new Form2();
    Fo2.checkBoxVrijednost = label1.Visible;
    Fo2.ShowDialog();
    label1.Visible = Fo2.checkBoxVrijednost;
}
```

8. POGLAVLJE: WINDOWS FORMS

Prije nego što prikazujemo formu, namještamo vrijednost javne varijable pridružujući joj vrijednost svojstva `label1.Visible`. Ukoliko je kontrola `label1` vidljiva, *checkbox* na drugoj kontroli će biti označen. Vrijednost će prvo biti pridružena varijabli `checkBoxVrijednost`, a zatim će, zbog takve definicije varijable, biti pridružena svojstvu `checkBox1.Checked` koji će utjecati na označenost te kontrole.

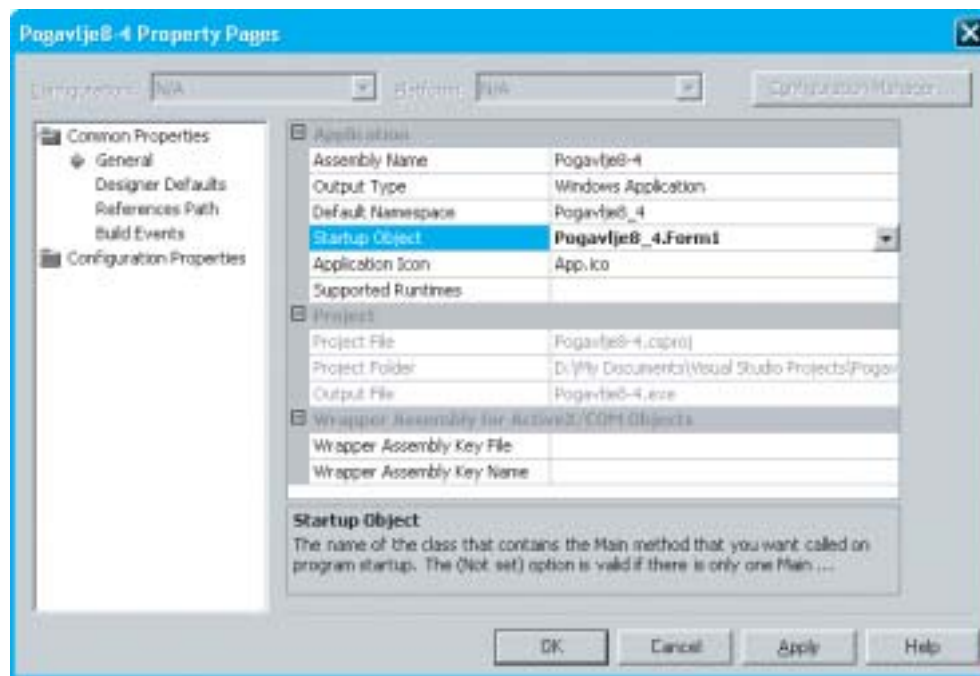
Nakon toga pozivamo formu u obliku dijaloškog okvira. Nakon što se korisnik poigra s *checkboxom* i zatvori formu, vraćamo se u gornji kôd i vrijednost varijable `checkBoxVrijednost` pridružujemo svojstvu `label1.Visible`. I ovdje se događa slična priča – u definiciji varijable odredili smo da vrijednost koja će biti vraćena zapravo bude uzeta iz svojstva `checkBox1.Checked`.

Načina za prenošenje vrijednosti među formama ima još. Možemo, primjerice, varijable prenositi kao parametre konstruktora klase, no to vam ostavljamo na samostalno proučavanje.

Promjena osnovne forme

Prilikom pokretanja aplikacije pokreće se metoda `Main()`. Ta se metoda automatski kreira prilikom otvaranja prve forme u projektu, koja ujedno postaje i osnovna forma. Metoda `Main()` izgleda ovako:

Slika 8-43:
Prozor za podešavanje svojstva *Startup Object* za slučaj postojanja više metoda *Main()* u aplikaciji



III. DIO: DIJELOVI .NET-A

```
[STAThread]
static void Main()
{
    Application.Run(new Form1());
}
```

Želite li promijeniti formu koja se prva otvara, cijelu metodu Main (uključujući i izraz u uglatim zagradama) izrežite iz forme koja je dotad bila osnovna i prebacite je u drugu. Također, nemojte zaboraviti promijeniti ime forme koja se navodi kao parametar metodi Application.Run jer u protivnom nećete ništa postići.

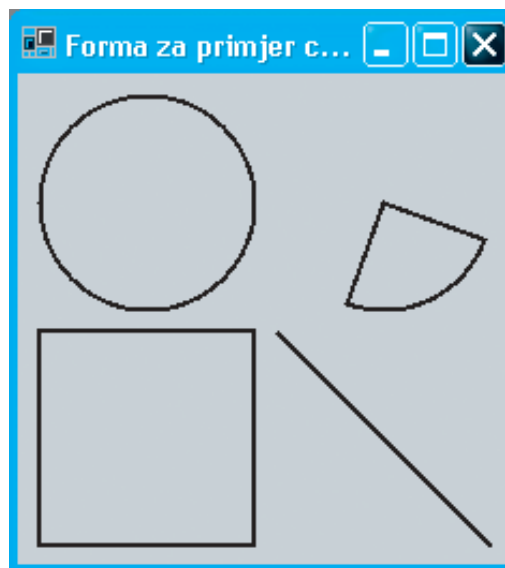
Ako u cijeloj aplikaciji postoji samo jedna forma s metodom Main(), onda je sasvim jasno da će ona biti izvršena. Ukoliko pak više formi ima spomenutu metodu, onda ćete u izborniku, stavka Project > ImeProjekta Properties trebati odrediti vrijednost svojstva Startup Object. U protivnom će vam kompajler prijaviti grešku jer neće znati koja je od postojećih metoda prava.

Grafika

Postoji vjerojatnost da ni postojeće ni dodatne kontrole neće zadovoljiti vaše potrebe, posebno što se vizualnih zahtjeva tiče. Neke takve probleme moći ćete riješiti slikama, no ponekad će biti puno jednostavnije neke stvari na ekranu – nacrtati. O crtanju ovog tipa moglo bi se napisati cijelo poglavlje, no mi ćemo se ovdje osvrnuti tek na osnove osnova.



Slika 8-44:
Primjer crtanja na formi



8. POGLAVLJE: WINDOWS FORMS

Za crtanje po ekranu (i, kao što ćemo kasnije vidjeti, nekim drugim izlaznim jedinicama) brine se Graphical Device Interface (skraćeno GDI) koji u .NET-u dolazi u upravljanoj i prilagođenoj obliku nazvanom GDI+. To nam je sučelje dostupno preko *namespacea* System.Drawing.

Važno je znati da crtanje nije moguće u bilo koje vrijeme. Ne možete se u nekoj funkciji samo sjetiti nešto nacrtati, pozvati određenu metodu i očekivati da bude nacrtano. Sva se crtanja moraju vršiti prilikom iscrtavanja kontrole, što znači za vrijeme događaja Paint.

Evo primjera pomoću kojeg ćemo nacrtati niz geometrijskih oblika na formi, poput onoga na slici 8-44:

```
private void Forma_Paint (object sender, System.Windows.Forms.PaintEventArgs e)
{
    Pen crta = new Pen(Color.Black, 2);
    e.Graphics.DrawEllipse(crta, 10, 10, 100, 100);
    e.Graphics.DrawRectangle(crta, 10, 120, 100, 100);
    e.Graphics.DrawPie(crta, 120, 10, 100, 100, 20, 90);
    e.Graphics.DrawLine(crta, 120, 120, 220, 220);
}
```

Funkcija vezana uz događaj Paint kao drugi parametar ima objekt *e* tipa System.Windows.Forms.PaintEventArgs. U njemu se, među ostalima, nalazi objekt Graphics preko kojega pristupamo površini i u kojem se nalaze metode za crtanje. U primjeru koristimo četiri najjednostavnije, a svaka od njih, osim potrebnih koordinata, kao prvi parametar ima varijablu tipa Pen koja određuje kako će izgledati crta kojom će oblik biti nacrtan. Mi smo se odlučili za crnu boju i širinu od dva piksela, no Pen skriva i druge, naprednije mogućnosti.

Također, svaka od metoda je preopterećena, što znači da ćete umjesto parametara iz primjera moći koristiti i neki drugi skup parametara. Mi u te sitnice ovdje nećemo ulaziti – dovoljno je u kodu započeti pisati “e.Graphics.” i zahvaljujući IntelliSensu pojaviti će se popis svih dostupnih metoda, a kasnije i popis parametara koje možete koristiti, sve s objašnjenjima.

Pozabavit ćemo se još dvjema metodama – jednom koja omogućava ispis slova u određenom fontu i drugom koja omogućava prikaz slike.

```
Font slova = new Font("Arial", 10);
Brush kist = Brushes.Green;
e.Graphics.DrawString("Microsoft .NET", slova, kist, 10, 230);
```

Jedna od kombinacija parametara metode DrawString koja služi za ispis teksta je ova iz primjera. Prvo navodimo *string* koji želimo ispisati, zatim tip slova, tip “kista” te na kraju koordinate na kojima želimo ispisati tekst.

III. DIO: DIJELOVI .NET-A

Istu stvar možemo pisati i na sljedeći način, iako je on puno nepregledniji:

```
e.Graphics.DrawString("Microsoft .NET", new Font("Arial", 10),
    new SolidBrush(Color.Brown), 10, 230);
```

Metoda za prikaz slike izgleda ovako:

```
Image slika = Image.FromFile(@"D:\mapa\slika.bmp");
e.Graphics.DrawImage(slika, 230, 10);
```



Znak “\” je tzv. *escape*-znak, koji u kombinaciji s ostalim znakovima predstavlja neke specijalne znakove (sjetite se “\n” za prelazak novi redak). Kako se on koristi u definiranju putanja do određene mape, znakom “@” smo kompajleru dali do znanja da ga u *stringu* tretira normalno, a ne kao *escape*-znak.

Naravno, i ova metoda je preopterećena (i to čak 30 puta!) pa postoji velik broj načina na koji možete prikazati sliku, što vam i preporučujemo jer često ne možete biti sigurni u putanju datoteke.



Crtanje po kontrolama često se koristi kada se želi modificirati postojeća kontrolu odnosno na temelju postojeće napraviti vlastita. U tim slučajevima treba naslijediti neku kontrolu (najčešće će to biti osnovna, kontrola Control), prekoračiti metodu OnPaint i ubaciti svoje crtarije.

Ispis na pisač

Kod ispisa na pisač situacija nije tako bajna kao s izradom korisničkog sučelja. Kao što možete zaključiti po smještaju ovog odlomka, stvar se svodi na rad s GDI+-om. No, krenimo redom...

Za ispis na pisač potrebna nam je kontrola PrintDocument. Nakon što je dovučemo na kontrolu, bit će nam dostupan objekt printDocument1 koji ima događaje BeginPrint, EndPrint i PrintPage. Uz prva dva bismo definirali stvari koje treba podesiti na početku i na kraju printanja, a uz PrintPage treba nacrtati ono što želimo ispisati. Kao što ćete iz primjera što slijedi primijetiti, stvar je vrlo slična crtanju po kontrolama:

8. POGLAVLJE: WINDOWS FORMS

```

private void printDocument1_PrintPage
    (object sender, System.Drawing.Printing.PrintPageEventArgs e)
{
    Pen crta;
    crta = new Pen(Color.Black, 2);
    e.Graphics.DrawEllipse(crta, 10, 10, 100, 100);
    e.Graphics.DrawRectangle(crta, 10, 120, 100, 100);
    e.Graphics.DrawPie(crta, 120, 10, 100, 100, 20, 90);
    e.Graphics.DrawLine(crta, 120, 120, 220, 220);
    e.HasMorePages = false;
}

```

Jedina razlika među primjerima je posljednja linija, u kojoj koristimo svojstvo `HasMorePages`. Pomoću njega dajemo do znanja ima li još stranica nakon ove. Ako ima, događaj `PrintPage` nastupit će još jednom. Na vama je zadatak da se pobrinete da svaka sljedeća stranica bude drugačije ispisana. Jedno banalno i ne uvijek prihvatljivo rješenje tog problema može biti ovo:

```

private int stranica = 1;

private void printDocument1_PrintPage(...)
{
    if (stranica == 1)
    {
        // crtanje prve stranice
        e.HasMorePages = true;
    }
    else
    {
        // crtanje druge stranice
        e.HasMorePages = false;
    }
    stranica++;
}

```

Kada prijeći na novu stranicu morat ćete izračunati sami. Naime, kako veličina papira u printeru može biti različita, tako se mijenja i površina na koju možete pisati. Veličinu cijelog papira možete dobiti kroz svojstvo `e.PageBounds`, dok su dimenzije te površine, umanjene za margine na koje ne možete pisati, zapisane u `e.MarginBounds`. Oba svojstva su tipa `Rectangle`, što znači da, između ostalog, možete koristiti sljedeće vrijednosti:

```

e.MarginBounds.Width;
e.MarginBounds.Height;

```

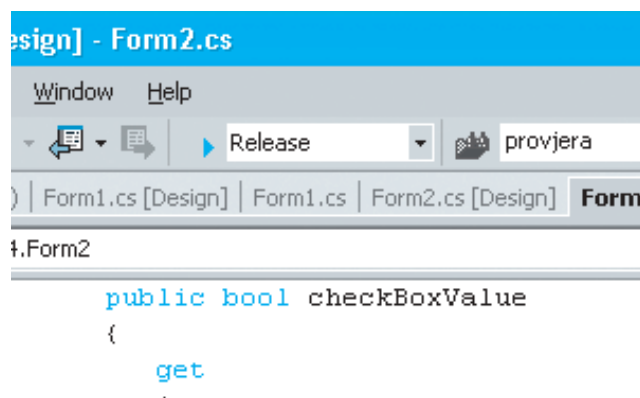
III. DIO: DIJELOVI .NET-A

Naravno, crtanje i ispis skrivaju još velik broj tajni i trikova, za koje nažalost nemamo mjesta. Ipak, nadamo se da su vam ovi početnički primjeri pojasnili način na koji stvari rade i poslužili kao podloga za proučavanje dokumentacije ili primjera na Internetu.

Distribucija

Nakon što napravite prozorsku aplikaciju, preostaje vam distribuirati je korisnicima. Najbolje kod aplikacija pisanih na platformi .NET je što u većini slučajeva možete jednostavno kopirati izvršnu datoteku s nastavkom ".exe" i eventualne dodatne datoteke koje koristite. Ipak, prije toga projekt morate kompajlirati u načinu Release (za razliku od načina Debug, koji je po *defaultu* aktivan). Takav način kompajliranja rezultirat će bržom i manjom aplikacijom jer neće sadržavati stvari potrebne za *debugiranje*. Prebacivanje možete napraviti u alatnoj traci Visual Studija, u padajućem izborniku, desno od ikone Start (vidi sliku 8-45).

Slika 8-45:
Prebacivanje kompajliranja u način Release



Međutim, ljudi su navikli na instalacijske procedure koje će im, osim kopiranja potrebnih datoteka, kreirati mape, prečace u popisu programa i slično. Ponekad prilikom instalacije treba napraviti i neke dodatne poslove, poput inicijalizacije baze podataka ili zapisivanja podataka u *registry*. Bilo kako bilo, u tome vam može pomoći kreiranje posebnog instalacijskog projekta.

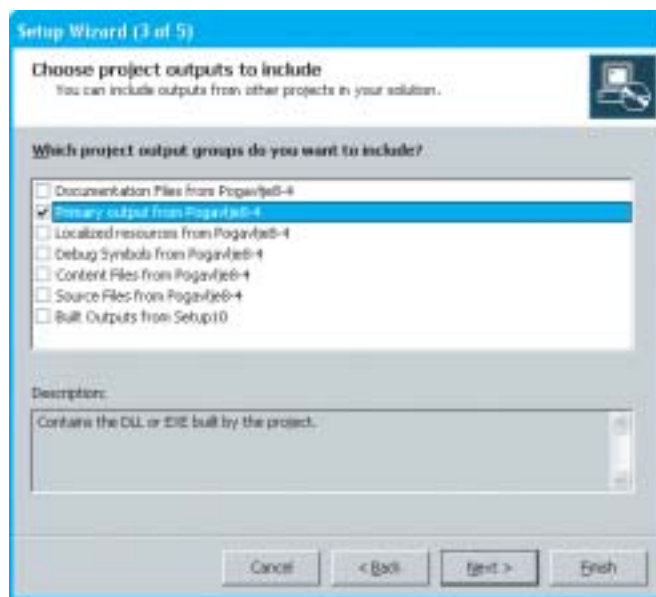
Želite li kreirati instalacijski projekt za svoju aplikaciju, prvo je morate otvoriti i kompajlirati u načinu Release. Zatim treba kreirati novi instalacijski projekt, no tako da ga dodate u rješenje u kojem se nalazi aplikacija (vidi sliku 8-46). Tu postoji mogućnost odabira više tipova instalacija, a mi smo odabrali najjednostavniju koja uključuje instalacijskog čarobnjaka.

8. POGLAVLJE: WINDOWS FORMS



Slika 8-46:
Kreiranje instalacijskog projekta
(vočite odabranu opciju Add to Solution)

Nakon odabira instalacijskog projekta, pokrenut će se čarobnjak u kojem ćemo definirati određene parametre instalacije. Prvo biramo tip aplikacije, zatim projektne grupe koje ćemo unijeti u instalaciju (svakako treba izabrati Primary output koji uključuje izvršnu datoteku, a prema potrebi i ostale stavke), a na kraju nam se nudi mogućnost dodavanja ostalih datoteka (poput *readme*-dokumenata).



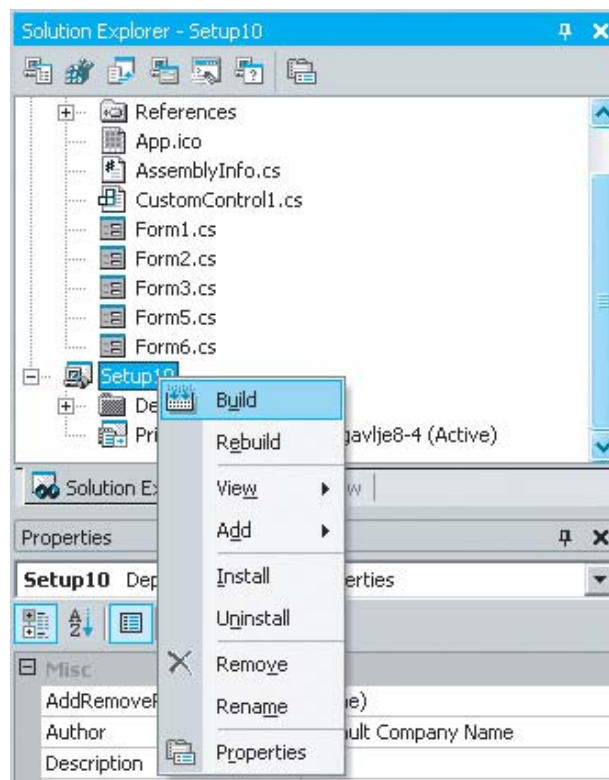
Slika 8-47:
U instalaciju možete uključiti
sve pa i izvorni kôd, no
najčešće ćete uključiti samo
Primary output.

III. DIO: DIJELOVI .NET-A

Slijedi igranje u sučelju pomoću kojeg možete dodati stavke u izbornik Programs, postavljati prečace na Desktop ili smještati datoteke u druge sistemske i programske mape.

U prozoru Solution Explorer treba označiti ime instalacijskog projekta kako bismo došli do njegovih svojstava. Među njima možemo pronaći mogućnost upisa imena aplikacije, inačice, autora, proizvođača i brojnih drugih parametara koji utječu na funkcionalan i vizualan način. Svakako im posvetite dovoljno vremena i upišite adekvatne vrijednosti.

Slika 8-48:
Solution Explorer i izbornik
preko kojega kompajliramo i
testiramo instalacijski projekt



Nakon što ste podesili sve parametre, vrijeme je da kompajlirate instalacijski projekt i isprobate ga. To ćete učiniti tako da u Solution Exploreru kliknete na instalacijski projekt desnom tipkom miša i odaberete stavku Build. Zatim u istom izborniku pronađite Install i vidite kako stvari funkcioniraju.

Kad budete zadovoljni instalacijskom procedurom, kompajlirajte projekt u načinu Release i potražite instalacijske datoteke koje je stvorio, a koje ćete vi distribuirati. One se nalaze u podmapu "Release", koja se pak nalazi u mapi instalacijskog projekta (ovisno o tome gdje ste ga smjestili prilikom kreiranja).

8. POGLAVLJE: WINDOWS FORMS



Slika 8-49:
Radite li sustav
pomoći za pro-
zorske aplikacije,
adresa koja vas
zanima je
[http://www.mshelp](http://www.mshelpwiki.com)
[wiki.com](http://www.mshelpwiki.com)

Osim izrade instalacijske procedure, svaka ozbiljna aplikacija zahtjeva sustav pomoći. Mi u tu problematiku nećemo ulaziti jer izlazi iz okvira programiranja, a one koje zanima upućujemo na internetsku adresu uz sliku.

Ima li još?

Naravno da ima. O programiranju prozorskih aplikacija može se još štošta reći, no zbog namjere da u knjizi pokrijemo sve aspekte programiranja u .NET-u ostaje dosta stvari koje ćete morati otkrivati sami. Sre-

ćom, sve se svodi na iste principe, tako da će vam primjeri koje smo u ovom poglavlju obradili, uz dozu iskustva i redovito druženje s dokumentacijom, omogućiti da riješite i one probleme kojih se ovdje nismo dotaknuli. Sretno!

