O autoru

Kent Reisdorph je glavni inženjer za softver u preduzeću TurboPower Software Co., a samostalno radi i konsultacije. Kent je saradnik časopisa C++Builder Developer's Journal firme Cobb Group i redovno piše članke za Delphi Developer's Journal. Takođe je član grupe TeamB, Borland-ove volonterske grupe za podršku. Kao član grupe TeamB, Kent provodi veliki broj sati u toku nedelje odgovarajući na pitanja u Borland-ovoj grupi za vesti, uglavnom o C++Builder-u i Windows programiranju. Autor je knjiga Sams naučite C++Builder za 21 dan i Sams naučite C++Builder 3 za 21 dan. Kent živi u Colorado Springs-u, država Colorado, sa ženom Jennifer i šestoro dece James, Mason, Mallory, Jenna, Marshall i Joshua.

Posveta

Ova knjiga je posvećena mojoj ženi Jennifer. Ne mogu da pomislim da je posvetim bilo kojoj drugoj osobi. Kao i obično, hvala Jen što održavaš sve, dok sam zatvoren u svom svetu.

Reči zahvalnosti

Ovaj deo knjige mi je veoma lak. Lako je setiti se ljudi koji su učestvovali u završavanju projekta. Prvo bih želeo da se zahvalim Brian Gill-u za njegov predan rad na projektu. U par navrata sam pokušao da prodrmam Brian-a, ali se nije ni pomerio (ili nisam primetio da se pomerio!). Takođe, želim se zahvalim Kezia Endsley za njen rad na ovoj knjizi. Kezia je odradila veliki deo posla na razvoju. Siguran sam da mi je rad sa njom veoma pomogao. Od ljudi u firmi Macmillan Publishing, želeo bih da se zahvalim Dana Leash i Heather Urschel.

Postoji nekoliko ljudi u Inprise Corporation (nekada Borland International) kojima želim da se zahvalim. Iako nisam imao puno direktnog kontakta sa Nan Borreson u toku ovog projekta, znao sam da je, radeći u senci, uradila svoj uobičajeni posao veoma dobro. Želeo bih da se zahvalim tehničkim urednicima Bill Fisher i Ellie Peters. Oboje su uradili dobar posao. Ne mogu da napomenem Ellie, a da ne dodam da mi je drago što mi je Ellie prijatelj i tehnički urednik. Takođe, želim da se zahvalim Steve Teixeira, Steve Trefthen i Ryder Rishel koji su brzo odgovarali na pitanja koja sam im postavljao u toku rada na ovom projektu.

Na kraju, želim da se zahvalim svojoj ženi Jennifer. Ovo je treći projekt koji sam odradio, a Jennifer mi je uvek bila snažna podrška. Daleko je dogurala i navikla je da me vidi sa "glavom na dole i slušalicama na glavi". Ovih dana ću joj to nadoknaditi, obećavam.

Recite nam šta mislite

Kao čitalac, Vi ste najvažniji kritičar i komentator ove knjige. Mi procenjujemo Vaše mišljenje, jer želimo da saznamo, šta smo uradili dobro, šta smo mogli bolje, koje oblasti biste želeli da obradimo u našim izdanjima i kao i sva ona druga mudra zapažanja koja želite da nam uputite. Vi možete da nam pomognete da napravimo jaku knjigu, koja će zadovoljiti Vaše potrebe i da Vam ponudimo računarski vodič kakav ste želeli.

Da li imate pristup Word Wide Web-u? Pristupite našem sajtu na adresi:

`http://www.kombib.co.yu`

Preuzmite kompletne sadržaje knjiga, kompletna poglavlja. Saznajte koji su naslovi u pripremi. I još mnogo toga...

Sajt se menja dva puta nedeljno. Možete nas kontaktirati na sledeći način:

E-mail: `kombib@emi.yu`

Uvod: Vi se nalazite ovde

Zar nije zgodno kada strelica na mapi pokazuje mesto na kom se nalazite? Pa, Vi ste ovde! Možda se ovde nalazite zato što ste i ranije koristili Delphi i želite da vidite šta ima novo u Delphi-ju 4. Možda ste ovde zato što Vam je šef rekao da treba da budete ovde. Možda ste ovde i zato što ste apsolutni početnik koji bi želeo da istraži divan svet Windows programiranja.

Bez obzira na razlog Vašeg dolaska, dobro došli! Mogu Vas uveriti da će ovaj put biti interesantan. Bez sumnje ćete uživati u njemu. Uključiće malo rada, ali će biti i zabave, nešto kasnije. Verujte mi da se ništa ne može porediti sa pretvaranjem Vaših misli u Windows program. Nadam se da će Vas uhvatiti groznica programiranja i da ćete uroniti u sate i sate rada na programima.

Ohrabrujem Vas da eksperimentišete u toku rada sa ovom knjigom. Odlaganje knjige i igranje može da bude dobar način učenja. Prolazak kroz knjigu nije trka. Prvi koji stigne do kraja ne dobija nagradu. Bolje da provedete 21 nedelju učeći Delphi programiranje, nego da jurite i da ne posvećujete vreme konceptima koji su obrađeni u ovoj knjizi. Bez sumnje, moje iskustvo mi govori da je najbolji način za učenje, rad na aplikaciji. Zamislite Vašu aplikaciju i radite na njoj u toku čitanja knjige. Rešavanje pravih problema je jedan od načina za učenje.

Zbog toga i nije važan razlog Vašeg dolaska. Mnogo je važnije da ste ovde. Drago mi je što ste došli i nadam se da ćete uživati u Delphi iskustvu. Opustite se, podignite noge na sto i zabavljajte se učeći kako da koristite Delphi. Siguran sam da će Vam prijati.

Kratak sadržaj

Sadržaj

Dan 1

Početak rada sa Delphijem

Čestitamo - odabrali ste jedan od trenutno najaktuelnijih alata za programiranje! Pre nego što počnete da koristite sve što Delphi može da ponudi morate naučiti nešto o Delphi IDE i o Object Pascal-u. U ovom poglavlju ćete naći:

- Brz pregled Delphija.
- Uvod u jezik Object Pascal.
- Činjenice o Pascal junitima (units), promenljivima i tipovima podataka.
- Diskusiju o nizovima.
- Informacije o stringovima u Pascal-u.

Šta je Delphi?

Do sada ste verovatno znali da je Borlandov Delphi najbolje prodavani proizvod za brzo razvijanje aplikacija (RAD - Rapid Application Development) koji se koriti za pisanje Windows aplikacija. Sa Deflijem možete pisati Windows programe brže i jednostavnije nego što je to bilo moguće ranije. Možete kreirati Win32 konzolne aplikacije, odnosno Win32 programe za grafički korisnički interfejs (GUI - Graphical User Interface). Kada kreirate Win32 GUI aplikacije koristeći Delphi, dostupna Vam je sva snaga istinskog prevodioca (Object Pascal programski jezik) ubačenog u RAD okruženje. Ovo znači da možete da kreirate korisnički interfejs za program (*korisnički interfejs* znači menije, dijalog bokseve, glavni prozor, itd.) korišćenjem tehnike prevuci-i-pusti (drag-and-drop) koja se koriste kod pravih okruženja za brzi

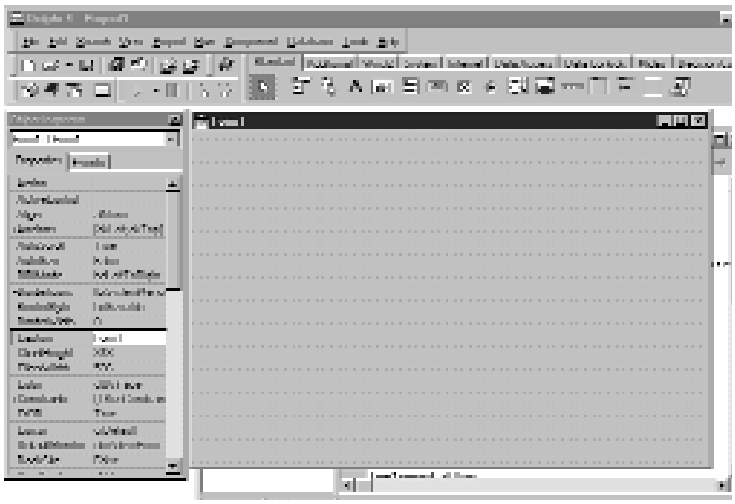
razvoj aplikacija. Takođe, možete da koristite ActiveX kontrole u Vašim formama kako biste kreirali specijalizovane programe poput Web browsera za par minuta. Delphi Vam daje sve to bez dodatnih troškova: Ne morate žrtvovati brzinu izvršavanja programa, pošto Delphi generiše brz prevedeni kod.

Mogu Vas čuti kako govorite, "Ovo će biti kul!" I pogađate, u pravu ste! Ali pre nego što se suviše uzbudite, moram Vam napomenuti da još uvek treba da radite i učite o Pascal programiranju. Ne želim da Vam se učini da možete da kupite program poput Delphija i postanete majstor za Windows programiranje preko noći. Potrebno je puno raditi da biste postali dobar Windows programer. Delphi obavlja veliki deo posla radeći u pozadini sa detaljima niskog novoa koji čine osnovu Windows programa, ali Delphi ne može pisati programe umesto Vas. Na kraju Vi još uvek morate biti programer i to znači da morate naučiti da programirate. Ovo može biti dugo penjanje uz brdo jednog dana. Dobre vesti su da Delphi može da učini Vaš put uglavnom bezbolnim i čak zabavnim. Da, možete raditi i zabavljati se uz rad!

Zato zasucite rukave i obujte cipele za šetnju. Delphi je odličan proizvod, zato se zabavljajte.

Kratak pogled na Delphi okruženje (Delphi IDE)

Ovo poglavlje sadrži kratak pregled Delphi integrisanog okruženja za razvoj (Integrated Development Environment - IDE). Sada ću Vam ukratko predstaviti okruženje, a detaljnije ću ga objasniću u lekciji dana 4, "Istraživanje Delphi okruženja". Pošto ste se prihvatili Windows programiranja, pretpostavljam da ste dovoljno napredovali da shvatite kako da startujete Delphi. Kada ste prvi put startovali program, pred Vama se našla prazna forma i okruženje, kao što je prikazano na slici 1.1.



Slika 1.1
Delphi okruženje i inicijalna prazna forma

Delphi okruženje je podeljeno na tri dela. Gornji prozor se može uzeti kao glavni prozor. On sadrži trake za alate (tool bars) i paletu komponenti (component palette). Delphi traka za alate Vam omogućava da jednim klikom miša pokrenete poslove kao što je otvaranje, snimanje i prevođenje projekta. Paleta komponenti sadrži širok spektar komponenti koje možete ubaciti u Vaše forme (komponente su tekst natpisi, kontrole za editovanje, list boksevi, dugmad i slično). Radi udobnosti komponente su podeljene u grupe. Da li ste zapazili jezičke duž gornjeg dela palete komponenti? Kliknite na ove jezičke i istražite koje su Vam komponente dostupne. Da biste postavili komponentu na Vašu formu jednostavno kliknite na dugme komponente u okviru palete komponenti a zatim kliknite na mesto u okviru forme gde biste želeli da se pojavi Vaša komponenta. Nemojte brinuti o tome što još uvek ne znate kako se koriste komponente. Ovo ćete naučiti za kratko vreme. Kada ste završili sa istraživanjem, kliknite na jezičak sa natpisom Standard, pošto će Vam ubrzo biti potreban.

NOVI TERMIN Komponenta *(component)* je odvojena softverska komponenta koja izvršava određenu unapred definisanu funkciju, kao što je tekst natpis, edit kontrola, odnosno okvir za liste (list box).

Inspektor objekata (object Inspector)

Ispod glavnog prozora na levoj strani ekrana se nalazi Object Inspector. Koristeći Object Inspector možete menjati karakteristike komponenti i događaja. U toku rada sa Delphijem uvek ćete koristiti Object Inspector. Object Inspector ima dva jezička: jezičak *Properties* (karakteristike) i jezičak Events (događaji). Karakteristike komponenata upravljaju radom komponenti. Na primer, promenom karakteristike `Color` određene komponente menja se boja pozadine ove komponente. Lista karakteristika koje su dostupne razlikuje se od komponente do komponente, iako komponente uglavnom imaju nekoliko zajedničkih elemenata (karakteristike `Width` i `Height` - širina i visina, na primer).

NOVI TERMIN Karakteristika *(property)* određuje operaciju komponente.

Jezičak Events sadrži listu događaja za određenu komponentu. Događaji se pojavljuju kada korisnik interaguje sa komponentom. Na primer, kada se klikne na komponentu, generiše se događaj koji govori o tome da je na komponentu kliknuto. Možete pisati kod koji odgovara na ove događaje, izvršavajući posebna dejstva kada se događaj pojavi. Kao što je to slučaj sa karakteristikama i događaji na koje komponenta može reagovati se razlikuju od komponente do komponente.

NOVI TERMIN Događaj (*event*) je nešto što se pojavljuje kao rezultat interakcije komponente sa korisnikom, ili sa Windows operativnim sistemom.

NOVI TERMIN Upravljač događajima *(event handler)* je deo koda koji se poziva u okviru Vaše aplikacije kao odgovor na događaj.

Delphi radni prostor (workspace)

Glavni deo Delphi okruženja je radni prostor. Radni prostor inicijalno prikazuje dizajner forme (Form Designer). Ne treba da Vas iznenadi činjenica da Vam dizajner formi (Form Designer) omogućava da kreirate forme. U Delphiju *forma* predstavlja prozor u okviru Vašeg programa. Forma može biti glavni prozor programa, dijalog boks, odnosno bilo koji drugi tip prozora. Možete koristiti Form Designer da postavite, pomerite, odnosno promenite veličinu komponente u toku procesa njenog kreiranja.

Iza Form Designer-a se krije editor koda (Code Editor). U editor koda (Code Editor) upisujete kod kada pišete Vaše programe. U toku kreiranja aplikacije Object Inspector, Form Designer, Code Editor i Component Palette se koriste za interaktivni rad.

Sada kada ste videli od čega se sastoji Delphi okruženje, krenimo da uradimo nešto.

Vaš prvi program: `Hello World`

Ovo je tradicionalno. Gotovo sve knjige o programiranju za početak Vam nude da napravite program koji prikazuje na ekranu natpis `Hello World`. Bio sam u iskušenju da uradim nešto drugo, ali tradicija nije snaga na koju ne treba računati, zato evo programa `Hello World`. Potrebno je da uradite neke poslove unapred o kojima ćete učiti u narednim poglavljima, zato ću Vam pružiti osećaj Delphijevih dobrih strana pre nego što počnete da učite, očito manje glamurozne, osnove Pascal jezika. Zato se prvo zabavite. Delphi (i njegov rođak C++Builder) Vam daje verovatno najbrži način da stignete do `Hello World` u odnosu na bilo koje Windows programsko okruženje današnjice.

Kreiranje programa

Verovatno Vam je do sada Delphi pokrenut i verovatno već gledate u praznu formu. Ova forma je generički nazvana `Form1`. (Naziv forme je važan u Delphiju, ali ću Vam nešto više reči o tome malo kasnije.) Na levoj strani forme, Object Inspector pokazuje karakteristike forme. Kliknite na naslovnu traku Object Inspector-a. Karakteristika `Caption` je istaknuta i kursor stoji na njoj, čekajući da nešto uradite (ako karakteristika Caption nije vidljiva, možda ćete morati da pomerite prozor Object Inspector-a da biste je pronašli. Karatkeristike su poređane abecednim redom.) Kucajte `Hello World!`, da biste promenili naslov forme.

Kako modifikujete karakteristike, Delphi istovremeno prikazuje ove rezultate, ukoliko je moguće da se promena karakteristike prikaže. Kada menjate naslov, zapazite da se naslov prozora forme menja kao odraz teksta koji ste ukucali.

Sada kliknite na dugme Run u okviru trake sa alatima (dugme sa zelenom strelicom). (Možete isto tako pritisnuti taster F9, odnosno izabrati opciju Run/Run iz glavnog menija.) Pre nego što budete shvatili šta se dešava, Delphi je napravio program. Forma je prikazana, a naslov ispisuje `Hello World!`. U ovom slučaju pokrenuti program izgleda potpuno isto kao prazna forma. Možda ćete sa užasom primetiti da se program prikazuje na istoj lokaciji gde je prikazana forma u okviru Form Designer-a. (Postoji razlika u pojavi pošto Form Designer prikazuje mrežu za poravnavanje, dok je pokrenuti program ne prikazuje.) Čestitamo- upravo ste napisali Vaš prvi Windows program na Delphiju. Hej, ovo je bilo lako!

"Ali šta je to?" pitaćete. Ovo baš i nije nešto. Slažem se, ali ovo je pravi Windows program. Probajte ga i uverićete se. Glavni prozor programa može biti pomeren hvatanjem za naslovnu traku, može mu biti promenjenja veličina, može biti umanjen, može biti maksimiziran i može se zatvoriti klikom na taster Close. Možete čak pronaći ovaj program u okviru Windows Explorer-a (verovatno će biti u Vašem direktorijumu `\Delphi40\Bin` pod nazivom `Project1.exe`) i pokrenuti ga dvostrukim klikom.

Izmena programa

U redu, možda je prikazivanje natpisa `Hello World` u okviru naslova bila mala varka. Hajde da ga malo proširimo. Ukoliko je Vaš program `Hello World` još uvek aktivan, zatvorite ga klikom na dugme Close u gornjem desnom uglu prozora. Form Designer je prikazan ponovo i omogućeno Vam je da izmenite formu (a kao rezultat toga izmenite i program).

Da biste poboljšali program dodajte tekst u središte prozora. Ovo možete učiniti dodavanjem tekst naslova formi:

1. Prvo kliknite na jezičak Standard u okviru palete komponenti. Treće dugme komponenti na paleti sadrži slovo A. Ako postavite kursor miša na ovo dugme, oblačić za savet (tooltip-mali prozor) će prikazati natpis `Label`.
2. Kliknite na dume za natipis a zatim kliknite na bilo koje mesto u okviru forme. Komponenta natpis će biti postavljena na formu sa generičkim naslovom `Label1`.
3. Sada prebacite pažnju na Object Inspector. On sada prikazuje karakteristike komponente `Label1` (zapamtite da je prethodno prikazivao karakteristike komponente `Form1`). Ponovo je karakteristika `Caption` istaknuta.
4. Kliknite na naslovnu traku Object Inspector-a, odnosno na karakteristiku `Caption` i ukucajte `Hello World!`. Sada natpis na formi prikazuje `Hello World!`.

5. Koliko god želite možete menjati veličinu teksta natipisa. Dva puta kliknite na karakteristiku `Font`. Karakteristika će se proširiti kako bi prikazali dodatni atributi fonta.

6. Pronađite karakteristiku `Size` unutar karakteristike Font i promenite veličinu fonta na 24 (trenutno je podešeno na 8). Ubrzo nakon što ste pritisnuli taster Enter, odnosno kliknuli na formu, natpis će promeniti veličinu.

Pošto natpis najverovatnije neće biti centriran, možda ćete poželeti da ga pomerite. Da biste pomerili komponentu, jednostavno kliknite na nju i odvucite je na mesto na kom želite da se nalazi. Kada se natpis bude nalazio na mestu na kom želite da bude, spremni ste da ga ponovo prevedete i pokrenete program. Ponovo kliknite na dugme Run i u sledećem trenutku program će biti pokrenut. Sada vidite natpis `Hello World!` koji je prikazan u sredini forme na isti način na koji je upisan u karakteristiku `Caption`. Slika 1.2 prikazuje rad programa `Hello World!`.

Slika 1.2. Vaš program `Hello World!` u radu

Zatvaranje programa

Sa ovim kratkim osvrtom na Delphi možete videti da je pisanje Windows programa korišćenjem Delphija puno interesantnije nego što je to bilo u starim dobrim vremenima. Da biste se pripremili za ono što ćete raditi sledeće, treba da zatvorite tekući projekt u okviru Delphi okruženja. Odaberite opciju File➥Close All iz glavnog menija. Kliknite na dugme No kada budete upitani da snimite promene u okviru projekta `Project1`, odnosno snimite projekt, ako Vam je stalo do Vaše nove kreacije.

Vaš drugi program `Hello World, deo II`

Pre nego što pređemo na učenje jezika Pascal potrebno Vam je još malo informacija o tome kako Delphi radi. Ove informacije su Vam potrebne da biste testirali različite mogućnosti Pascal jezika koje ćete obraditi u narednih nekoliko dana. Ovo poglavlje će sadržati samo kratak pogled na snagu Delphija. U lekcijama dana 4, 5 i 6 učićete detaljnije o tome kako Delphi radi.

Pisanje programa Hello World II

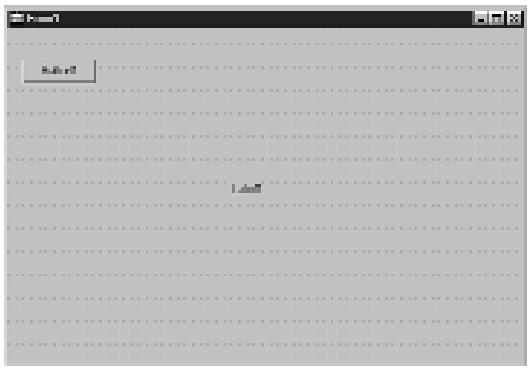
Cilj ove vežbe je da se tekst `Hello World, Part II` pojavi na ekranu nakon pritiska na dugme. Ova vežba će Vam dati obrazac koji treba da pratite kada budete testirali različite delove koda, koje ćete obrađivati u narednih nekoliko dana. Izvršite sledeće radnje:

1. Odaberite opciju File➥New Application iz glavnog menija da biste pokrenuli novu aplikaciju (kliknite na No ukoliko budete upitani da snimite tekući projekt).
2. Kliknite na jezičak Standard u okviru palete komponenti a zatim kliknite na ikonu koja ima taster sa natpisom OK (komponenta `Button`).
3. Pomerite kursor bilo gde u okviru forme, a zatim kliknite. Na formi će biti prikazano dugme.
4. Odaberite komponentu `Label` i postavite je blizu centra forme.

U ovom trenutku Vaša forma bi trebalo da bude slična formi na slici 1.3. Primetite da komponenta Label ima generički naslov `Label1`, a dugme generički naslov `Button1`.

Izmena programa Hello World II

U prvoj verziji programa `Hello World`, koristili ste Object Inspector da biste promenili karakteristiku `Caption` komponente Label. Ova promena je bila usvojena u toku dizajniranja i mogla se isto tako videti u toku rada programa. U ovom primeru, moćićete da promenite naslov komponente Label koristeći programski kod.



Slika 1.3
Nova forma, nakon što su postavljene komponente Button i Label

NAPOMENA Kada menjate karakteristike komponenti koristeći Object Inspector i Form Designer, rećićete da činite promene u toku dizajniranja. Kada menjate karakteristike preko programskog koda (ove izmene se izvršavaju u toku rada programa), rećićete da radite izmene u toku rada programa (runtime).

Da biste promenili karakteristiku `Caption` u toku rada programa, izvršite sledeće korake:

1. Dva puta kliknite na dugme u okviru Vaše forme. Ubrzo nakon što ste ovo uradili, Delphi generiše upravljač događajima za događaj `OnClick` koji se odnosi na dugme. Generisani kod izgelda otprilike ovako:

```
procedure TForm1.Button1Click(Sender: TObject);
begin

end;
```

2. U ovom trenutku ne treba da budete zabrinuti svim onim što ovde vidite. Jedino što treba da razumete je da je upravljač događajima `OnClick` deo koda koji će biti izvršen svaki put kada se klikne na dugme (sve dok program radi, ovo će biti aktuelno). Kursor za editovanje se nalazi između iskaza `begin` i end, i čeka da ukucate kod. Ubacite ovaj kod na mesto kursora:

```
Label1.Caption := "Hello World, Part II";
```

   Programske linije uvek uvlačim za dva prazna mesta (što mnogi programeri smatraju da je odgovarajuće za kodiranje) pa bi kompletan upravljač događajima izgledao ovako:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  Label1.Caption := 'Hello World, Part II';
end
```

   Ovaj kod je veoma jednostavan. On jednostavno dodeljuje vrednost `Hello World, Part II` karakteristici `Caption`, koja pripada natpisu (karakteristika `Caption` se koristi da dodeli tekst koji ovaj natpis prikazuje).

3. Sada kliknite na dugme Run u okviru trake za alat, da biste pokrenuli program. Kada pokrenete program, zapazite da je natpis zadržao stari naslov `Label1`. Kliknite na dugme forme i naslov natpisa će biti promenjen u `Hello World, Part II`. Hej, kako se ovo dogodilo! Magija? Ne, samo su Delphi na delu!

U narednih nekoliko dana, radićete mnogo sličnih vežbi, tako da ćete imati dosta iskustva u postavljanju natpisa, dugmadi i ostalih komponenti na formu. Shvatam da nisam u potpunosti objasnio šta se ovde događa u pozadini, ali ne želim da govorim o tome sada, pošto sam sačuvao objašnjenje za kasnije.

Pregled jezika Object Pascal

Pre nego što naučite nešto o Delphijevim RAD mogućnostima, potrebno je da naučite osnove jezika Object Pascal. Ovaj deo knjige možda neće biti najuzbudljiviji za Vas, ali Vam je potrebno osnovno razumevanje Object Pascal-a pre nego što krenete dalje.

Bilo bi dobro ukoliko bi predstavljanje jezika Object Pascal moglo da se odvija sekvencijalno. Ovo nije moguće pošto su sve mogućnosti koje ćete učiti međusobno ispreplitane. Prikazaću Vam pojedine delove slagalice, a zatim ih uklapati.

Na kraju lekcije dana 3, uglavnom ćete kompletirati sliku jezika Object Pascal. Nemojte se zabrinuti ukoliko trenutno ne možete da zapamtite svaki koncept koji je predstavljen. Neki od njih zahtevaju potpuno razumevanje jezika Object Pascal i mogu se shvatiti tek nakon iskustva sa realnim aplikacijama.

U toku narednih nekoliko dana videćete kratke isečke kodova koji ilustruju pojedine mogućnosti jezika Object Pascal. Takođe ćete raditi vežbe koje će Vam omogućiti da testirate Vaše novostečeno znanje. U prvih nekoliko dana videćete Vaše Delphi aplikacije kao male isečke. Ne bih želeo da istrčavam i ulazim suviše duboko u Delphi okruženje, odnosno biblioteku vizuelnih komponenti (VCL) u ovoj ranoj fazi. Moraćete da se zadovoljite parčićima i delovima, da biste kasnije u knjizi počeli da sagledavate kompletnu sliku. Kod možete skinuti sa Web sajta knjige i on sadrži kompletne programe za neke od vežbi koje ćete raditi u toku narednih nekoliko dana. (Prebacite se na: `http://www.mcp.com/info i kucajte 0-672-31286-7.`)

Na početku...

Vratimo se u 1994. godinu. Borland počinje rad na RAD alatu pod šifrovanim nazivom Delphi. Kada je odlučeno da arhitektura modela komponenti predstavlja najbolji način za inplementaciju RAD, bilo je neophodno opredeliti se za programski jezik koji će biti srce sistema.

U to vreme Borland je bio jedina firma na tržištu prevodilaca koja je prodavao Pascal prevodioce. Borland je isto tako bio poznat kao kompanija koja proizvodi najbolje alate za Pascal. Ako ste bili Pascal programer, verovatno ste koristili Borlandov TurboPascal u jednom, ili drugom obliku. Borland je više ili manje "posedovao" Pascal. Naravno, legalno gledano Borland nije posedovao Pascal, ali nije bilo sumnji da njegova pozicija u svetu Pascala dozvoljava određene slobode u implementaciji novih mogućnosti i usavršavanja. Dodatno, nije postojao komitet za Pascal standarde, niti je bilo koji pisani standard definisao Pascal jezik. Zato je Borland napravio Delphi koristeći Pascal kao osnovni jezik (Borlandov interni šifrovani naziv je ujedno postao službeni naziv proizvoda).

Pre nego što je Delphi počeo da postoji, Borland je već izmenio Pascal u pozitivnom pravcu. Na primer, Borland je već proširio Pascal kreiranjem novog jezika pod nazivom Object Pascal. Moglo bi se reći da je Object Pascal u odnosu na Pascal isto što i C++ u odnosu na C. Object Pascal je dodao klase Pascalu , i tako svrstao Pascal u svet objektno orijentisanih jezika (Object-Oriented Programing language - OOP). Kada je Delphi bio razvijen, dodata su nova ponašanja i ključne reči koje su imale veze sa modelom komponenti. Dodate su ključne reči poput published i property, između ostalog. Ovo je omogućilo Borlandu da u potpunosti implementira snagu modela komponenti. Modifikujući jezik Pascal tako da se uklopi u model kompone-

nti, Borland je bio u mogućnosti da na ispravan način inplementira RAD. U osnovi jezik Object Pascal je modifikovan koliko je to bilo potrebno, dok su stavke vezane za dizajn došle nakon razvoja tada nepoznatog proizvoda nazvanog Delphi. Kao rezultat dobio se jezik koji radi sa modelom komponenti.

Iako je izmena jezika Pascal bila razmatrana kao veliki korak za Borland, ovo se nije dogodilo kao presedan. Prethodno je Microsoft modifikovao jezik Basic i napravio novi jezik pod nazivom Visual Basic. Ovaj novi jezik je bio skoro neprepoznatljiv u poređenju sa originalnim jezikom Basic, koji je poslužio kao njegova osnova.

Borland je preuzeo rizik da modifikuje Pascal. Nakon svega imao je osnovu lojalnih kupaca koji možda nisu mogli da prihvate izmene jezika koji su upoznali i zavoleli. Ipak Borland je bio u dobroj poziciji na tržištu Pascala i krenuo je napred sa svojim planovima. Rezultat je bio pravi hit, naravno.

Nećete pogrešiti, Object Pascal je veoma moćan programski jezik, a ovu izjavu ne koristim često. Imam iskustva sa programskim jezikom C/C++, i poput ostalih C/C++ programera gledao sam u početku na Delphi sa dozom skepticizma. Ubrzo sam otkrio da je jezik Object Pascal veoma moćan. Ustvari, u rukama prosečnog programera skoro da nema razlike između ova dva jezika u pogledu snage. Object Pascal je jedinstven po tome da je moćan i relativno lak za učenje. Ne želim ni na koji način da odam utisak da Object Pascal nije programski jezik koji pruža puno mogućnosti. Pascal je obično bio shvatan kao manje ozbiljan programski jezik. Ovo nikada nije bilo tačno i nikada nije manje važilo nakon pojave Object Pascal-a.

Nekoliko različitih termina su usvojili Delphi programeri da bi opisali šta rade. Osnovni jezik Delphija je naravno, Object Pascal, i neki ga upravo tako zovu. Ostali mogu reći "Ja programiram u Pascalu", ili, čak, "Ja sam Delphi programer". Na kraju krajeva, na Vama je da odlučite koju ćete terminologiju koristiti. Ja koristim pojam Object Pascal i Pascal neizmenljivo u ovoj knjizi, dok sam rezervisao reč Delphi da bi označio Delphi okruženje, odnosno njegove alate.

Object Pascal Vam omogućava da koristite prednosti objektno-orijentisanog programiranja do maksimuma. OOP nije samo fraza. Ono Vam pruža velike povlastice, pošto Vam omogućava da kreirate objekte koje možete koristiti u Vašim programima i da ih ponovo koristite u Vašim budućnim programima.

Objekat , slično kao što su komponente opisane prethodno, je deo softvera koji izvršava određene programske zadatke. (Komponente su objekti, ali nisu svi objekti komponente. Ovo ću objasniti kasnije.)

Objekt se otkriva korisniku (programeru koji koristi objekat) samo onoliko koliko je to potrebno, tako da je korišćenje objekta pojednostavljeno. Svi interni mehanizmi koje korisnik ne treba da zna su sakriveni od njegovog pogleda. Sve ovo je uključeno u koncept objektno-orijentisanog programiranja. OOP omogućava da se programiranju pristupi modularno, što Vas odvaja od situacije da svaki put iznova pronalazite točak. Delphi programi su veoma objektno-orijentisani, pošto Delphi puno koristi komponente. Nakon što je komponenta kreirana (bilo ona koju ste sami kreirali, ili

ona koja je ugrađena) ona se može ponovo koristiti u bilo kom Delphi programu. Komponenta takođe može biti proširena nasleđem, koje kreira nove komponente sa dodatnim mogućnostima. Najbolje od svega je da komponente kriju njihove interne detalje i omogućavaju programeru da se koncentriše na izvlačenje koristi od komponenti. Objekti i klase će biti obrađeni detaljnije u lekciji dana 3, "Klase i objektno-orijentisano programiranje".

Pascal juniti

Programiranje je više od kucanja programskog koda. Pre svega programiranje je kombinacija planiranja zadatka za programiranje, a zatim kucanja koda koji će izvršavati taj zadatak. Kod koji kucate se upisuje u tekst datoteku. Prevodilac uzima ovu datoteku i prevodi je u mašinski kod koji računar može da razume. Tekst datoteka koju Delphi prevodi u mašinski kod se zove junit (unit).

Junit *(unit)* je tekst datoteka koja može biti prevedena u modul.

Tipovi junita

Delphijeve GUI aplikacije će sadržati najmanje dva junita. Izvorni junit projekta sadrži izvorni kod projekta. Juniti izvornog koda programa imaju nastavak `DPR`. Možete pregledati izvorni junit programa ako odaberete Project➥View Source iz glavnog menija. Uglavnom nije potrebno menjati izvorni junit projekta. Ustvari, ne bi trebali da menjate izvorni junit projekta, ukoliko niste sigurni šta radite. Ako slučajno izmenite izvorni junit projekta na neodgovarajući način može se dogoditi da Vaša aplikacija neće moći da se prevede. (Određene napredne tehnike programiranja zahtevaju izmenu izvornog koda projekta, ali za sada o tome ne treba da brinete.)

Drugi tip junita, koji Delphijeve GUI aplikacije sadrže, je junit glavne forme. Junit forme i njegov naziv ukazuju na junit izvornog koda koji mu je pridružen. Ovaj tip junita ima nastavak `PAS`. Ovaj tip junita ćete u većini slučajeva koristiti u Vašim Delphi programima. Delphijeve GUI aplikacije uvek imaju jedan junit forme (za glavnu formu), ali, takođe, mogu imati jednu, ili više dodatnih junita formi. Na primer, aplikacija koja prikazuje okvir sa objašnjenjem programa (About box) će imati junit glavne forme i junit za okvir sa objašnjenjem programa (About box).

Možda ste zapazili da uvek kažem "Delphijeva GUI aplikacija". Ovo radim zbog toga što želim da razdvojim GUI aplikaciju od aplikacije moda konzole. Aplikacija moda konzole je 32-bitna Windows aplikacija koja radi u prozoru konzole (DOS prozor). Aplikacija konzole nema glavnu formu i može, odnosno ne može sadržati druge forme. Aplikacija konzole naravno može imati jedan, ili više junita.

Postoji i treći tip junita koji možete koristiti u Delphi aplikacijama. Ovaj tip junita je junit koji sadrži samo izvorni kod. Junit koji sadrži samo kod pozivaju drugi juniti u okviru projekta. Ne bih želeo da idem suviše u detalje u ovom trenutku, ali ćete naučiti više o ovom tipu junita u kasnijim poglavljima.

Anatomija Delphijevog junita

Delphi juniti moraju pratiti unapred definisan format. Ovo sigurno nije iznenađenje za Vas. Junit mora biti napisan u unapred definisanom formatu da bi prevodilac mogao da čita junit i prevodi kod junita.

Juniti Delphi projekta sadrže ključnu reč `program`, iza kog sledi naziv junita i kod blok koji se nalazi između ključnih reči `begin` i `end`. Možete videti kako izgleda osnovni junit izborom opcije View➥Project Source u okviru glavnog menija Delphija. Izvorni junit projekta za generički Delphijev projekt izgleda kao na listingu 1.1.

Brojevi linija u listingu 1.1 nisu delovi junita. Stavio sam ih samo zbog reference. Neki od ovih listinga koje vidite u ovoj knjizi će imati brojeve linija kao referencu, a drugi neće. U oba slučaja budite sigurni da Pascal jezik ne koristi brojeve linija kao što to čine neki drugi jezici (uglavnom BASIC).

Listing 1.1: Izvorni kod projekta za generički Delphijev projekt

```
01: program Project1;
02:
03: uses
04:   Forms,
05:   Unit1 in 'Unit1.pas' {Form1};
06:
07: {$R *.RES}
08:
09: begin
10:   Application.Initialize;
11:   Application.CreateForm(TForm1, Form1);
12:   Application.Run;
13: end.
```

U liniji 1, ključna reč `program` identifikuje junit kao glavni izvorni junit programa. Možete videti da se naziv junita, `Project1` nalazi iza ključne reči `program`. (Delphi daje projektu generički naziv sve dok ne snimite projekt pod nazivom koji ima bolje značenje.) Počev od linije 3 vidite deo koji je identifikovan ključnom reči `uses`. Iza ključne reči `uses` se nalaze nazivi junita koje navedeni junit traži da bi mogao da bude preveden. Spisak se završava znakom tačka-zarez. Ključna reč uses će detaljnije biti opisana malo kasnije u poglavlju "Lista `uses`".

U liniji 7 možete videti direktivu prevodiocu koja pokazuje Delphiju da uključi odgovarajuću resursnu datoteku. Resursne datoteke će biti obrađene detaljnije u lekciji dana 8 "Kreiranje aplikacija u Delphiju".

Linija 9 sadrži ključnu reč `begin`, a linija 13 sadrži ključnu reč end. Zapazite da poslednja ključna reč `end` u okviru junita iza sebe ima tačku. (Junit može sadržati više blokova koda označenih sa begin i end, ali samo jedan krajnji end iskaz.) Kod u okviru linija 10, 11 i 12 je kod koji inicijalizuje aplikaciju, kreirajući glavnu formu aplikacije, odnosno startuje aplikaciju. Ne treba da Vas brinu detalji ovog koda koji piše Delphi program.

Ključne reči `begin` i `end` označavaju blok koda. Blok koda može sadržati nekoliko linija koda, odnosno može sadržati nekoliko stotina linija koda (odnosno nekoliko hiljada linija koda). U okviru knjige možete videti ključne reči `begin i end`. Dok budete čitali knjigu naučićete da bolje upravljate time kako i kada treba da koristite ključne reči `begin` i `end`.

Hajde da pogledamo još jedan osnovni junit Pascala. Odaberite opciju File➥New u okviru glavnog menija. Kada dijalog New Items bude bio prikazan, pronađite ikonu sa natpisom `Unit` i dva puta kliknite na nju. Delphi će kreirati novi junit i prikazati ga u editoru koda (Code Editor). Listing 1.2 pokazuje kod koji je generisan za ovaj junit.

Listing 1.2: Prazan Pascal junit

```
01: unit Unit2;
02:
03: interface
04:
05: implementation
06:
07: end.
```

Ovde nema puno stvari zar ne? Ovaj junit ima dve slične stvari sa junitom koji je prikazan u okviru listinga 1.1. Prvo, junit počinje ključnom reči `unit` iza koje sledi naziv junita `Unit2` (opet generički naziv koji kreira Delphi). Znam da kod u listingu 1.1 počinje ključnom reči `program`, a ovaj kod počinje ključnom reči `unit`, ali ovde postoji nekoliko sličnih elemenata: Pascal junit počinje sa jednom od ove dve ključne reči iza koje sledi naziv junita, dok se ključna reč `end` pojavljuje na kraju oba listinga. Ponovo imamo ključnu reč end iza koje sledi tačka koja označava kraj junita.

Kod u okviru listinga 1.2 se razlikuje od listinga 1.1 po tome što sadrži delove pod nazivom `interface` i `implementation`. Junit koji nije glavni izvorni junit programa mora sadržati deo `interface` i deo `implementation`. Ove dve ključen reči će biti opisane detaljnije u poglavlju pod nazivom "Odeljak `interface`" i "Odeljak `implementation`" respektivno. Listing 1.2 se razlikuje od listinga 1.1 i po tome što nema iskaz `begin`. Glavni junit programa mora imati oba iskaza `begin` i `end`, ali izvorni junit jedini može sadržati poslednji iskaz `end`.

Sledeće poglavlje opisuje ključne reči koje se koriste u okviru Pascal junita.

Lista uses

NOVI TERMIN *Lista* `uses` je lista eksternih junita koje navedeni junit poziva.

Pogledajte listing 1.1. Zapazite ključnu reč `uses` u liniji 3. Ključna reč uses označava početak odeljka koji sadrži listu drugih junita od kojih navedeni junit zavisi. Na primer linija 11 u okviru listinga 1.1 izgleda ovako:

```
Application.CreateForm(TForm1, Form1);
```

Ova linija koda sadrži informacije koje se nalaze u drugim junitima i ne mogu biti pronađene u navedenom junitu. Procedura identifikovana kao `Application.CreateForm` se nalazi u Delphijevom junitu pod nazivom `Forms.pas`, a identifikatori `TForm1` i `Form1` se nalaze u glavnom junitu forme projekta pod nazivom `Unit1.pas`. Da li primećujete vezu? Lista `uses` pokazuje Delphiju gde da potraži dodatne informacije koje su potrebne za prevođenje ovog junita. Evo još jednog pogleda na listu `uses`:

```
uses
  Forms,
  Unit1 in 'Unit1.pas' {Form1};
```

Zapazite da lista `uses` sadrži dva naziva junita, `Forms` i `Unit1`. U nekim slučajevima ovo nije dobar primer liste uses pošto drugi junit na listi sadrži dodatni tekst koji se obično ne može naći u listama uses (`Unit1 in "Unit1.pas"{Form1}`).

Navedeni tekst se koristi da definiše formu koja sadrži junit i koristi se samo u okviru glavnog izvornog junita programa. (Tekst između vitičastih zagrada je komentar i koristi se nezavisno od ostatka koda. O komentarima će biti reči u nastavku, u okviru poglavlja "Komentari u okviru koda".)

Postoje dva pravila kojih treba da budete svesni kada kreirate listu `uses`:

- Prvo, svaki junit na listi mora biti odvojen zarezom od drugog junita.
- Drugo, tačka-zarez se mora nalaziti na kraju poslednjeg junita u okviru liste. Znak tačka-zarez označava kraj liste `uses`.

Prirodno je da lista mora sadržati ispravne nazive junita. Lista `uses` je označena ključnim rečima `uses` i završava se znakom tačka-zarez. Izvan toga, nije bitno kako je lista uses organizovana. Na primer, sledeće dve liste uses su za prevodilac identične:

```
uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

uses
  Windows,
  Messages,
```

```
SysUtils,
Classes,
Graphics,
Controls,
Forms,
Dialogs,
StdCtrls;
```

Junit može sadržati neograničen broj listi `uses`. Nije obavezno da svi juniti koji su potrebni navedenom junitu budu u jednoj jedinoj `uses` listi.

U nekim slučajevima, Delphi će dodati junite Vašoj uses listi umesto Vas. Ovo se radi korišćenjem opcije File➥Use Unit u okviru menija. Ova opcija će biti detaljnije obrađena u lekciji dana 4.

Odeljak `interface`

Ponovo pogledajte listing 1.2. Zapazite da ovaj listing ima odeljak označen ključnom reči `interface`. Ova ključna reč označava početak odeljka za interfejs u okviru junita.

Odeljak interface je odeljak junita u okviru kog se deklarišu identifikatori koje navedeni junit izvozi. Identifikator za *izvoz* predstavlja identifikator kome mogu pristupiti drugi juniti u okviru projekta.

Većina junita će sadržati kod koji ostali juniti koriste. Kod može biti implementiran kao klasa, procedura, funkcija, odnosno kao promenljiva. Bilo koji objekat, koji je dostupan ostalim junitima iz navedenog junita, mora biti deklarisan u odeljku za interfejs. Može se reći da odeljak za interfejs sadrži listu stavki u okviru junita koje ostali juniti mogu da koriste. Odeljak za intrerfejs počinje ključnom reči `interface` a završava se ključnom reči `implementation`.

Odeljak `implementation`

Odeljak za implementaciju junita je odeljak koji sadrži aktuelni kod junita.

Odeljak za implementaciju počinje ključnom reči `implementation` a završava se ključnom reči sledećeg junita. Ključna reč sledećeg junita je obično poslednja ključna reč u okviru junita, ali može biti ključna reč `initialization`, ukoliko junit sadrži odeljak za inicijalizaciju. Bilo bi teško reći nešto više od ovog u ovom trenutku, ali postoje drugi aspekti Pascala o kojima bih hteo da pišem pre nego što sve ovo povežemo. Dozvolite mi da Vam dam primer koji će ilustrovati korišćenje odeljaka `interface` i `implementation`.

Recimo da kreirate junit koji sadrži proceduru pod nazivom `DoSomething`. Recimo da unapred želite da junit `DoSomething` bude dostupan drugim junitima u okviru Vašeg projekta. U tom slučaju trebalo bi da deklarišete proceduru `DoSomething` u okviru odeljka interfejs a zatim definišete navedenu proceduru u odeljku za implementaciju. Kompletan junit bi izgledao kao što je to prikazano na listingu 1.3.

Listing 1.3: Junit sa javnom funkcijom

```
unit Unit2;

interface

  procedure DoSomething;

implementation

  procedure DoSomething;
  begin
    { Code for DoSomething goes here. }
  end;

end.
```

Zapazite da je procedura `DoSomething` deklarisana u odeljku za interfejs a zatim definisana u odeljku za implementaciju. Shvatam da sam u ovom trenutku malo požurio. O funkcijama i procedurama će biti više reči u lekciji sutrašnjeg dana, pa ću detaljnije objasniti deklaracije i definicije u tom trenutku.

Odeljci `initiallization` i `finalization`

Odeljci `initialization` i `finalization` se mogu koristiti za izvršavanje početnog koda, odnosno koda za čišćenje koji su potrebni junitu. Bilo koji kod u okviru odeljka `initialization` će biti izvršen kada navedeni junit bude učitan u memoriju. Suprotno tome bilo koji kod u okviru `finalization` odeljka će biti izvršen pre nego što junit bude izbačen iz memorije. Možete imati samo odeljak `initialization`, ali ne možete imati odeljak `finalization` bez odeljka `initialization`. Odeljci `initialization` i `finalization` su opcioni.

Dodatne ključne reči koje se koriste u junitima

Pascal junit može sadržati druge opcione ključne reči koje označavaju odeljke koji se ređe koriste. Neke od ovih reči imaju višestruku namenu. Naredna poglavlja opisuju one ključne reči koje imaju veze sa junitima.

Ključna reč const

Junit opciono može sadržati jedan ili više odeljaka `const`. Odeljak const je određen ključnom reči `const`. Odeljak `const` opisuje listu promenljivih koje su poznate kao konstante.

Konstanta je identifikator koji se ne može menjati. Na primer, recimo da imate određene vrednosti koje program koristi veoma često. Za ove vrednosti možete

definisati konstantne promenljive. Da bi ovo ilustrovali dodajmo odeljak `const` u program na listingu 1.3. Dodaćete jedan odeljak `const` za konstante koje su javne (dostupne drugim junitima) i drugi odeljak `const` koji sadrži konstante koje su dostupne samo tekućem junitu. Listing 1.4 prikazuje junit koji sadrži dva odeljka za konstante.

Listing 1.4: Junit sa dodatkom odeljka za `konstante`

```
unit Unit2;

interface

const
  AppCaption = 'My Cool Program 1.0';

  procedure DoSomething;

implementation

const
  BaseX = 20;
  BaseY = 200;

  procedure DoSomething;
  begin
    { Code for DoSomething goes here. }
  end;

end.
```

Pošto je konstanta `AppCaption` deklarisana u odeljku za `interfejs` isto se može koristiti bilo gde u okviru junita, a takođe i u drugim junitima koji u listi `uses` imaju naveden tekući junit. Konstante `BaseX` i `BaseY` su dostupne jedino u okviru navedenog junita pošto su deklarisane u odeljku za `implementaciju`.

Ključna reč `const` ima druge primene osim navedene. O tome će biti više reči u lekciji sutrašnjeg dana u okviru poglavlja "Vrednosti, konstante i parametri promenljivih".

Ključna reč type

NOVI TERMIN Ključna reč `type` se koristi da deklariše nove tipove koje će koristiti Vaš program.

Deklarisanje novih tipova je ezoterična tehnika programiranja koju bi bilo teško objasniti u ovoj fazi igre, ali možda će primer pomoći. Recimo da Vaša aplikacija zahteva *niz* (skup vrednosti) od 20 bajtova i da ovaj tip niza može biti korišćen neograničeno. Možete deklarisati novi tip kao što je navedeno:

```
type
  TMyArray = array [0..19] of Byte;
```

Sada možete da koristite identifikator `TMyArray` umesto kucanja `array [0.19] of Byte` svaki put kada Vam se ukaže potreba za nizom od 20 bajtova. Za sada bih trebao ovo da ostavim, ali više primera o deklaraciji tipova možete pronaći u nastavku knjige.

Ključna reč var

Ključna reč `var` se koristi da deklariše odeljak koda u kom se deklarišu promenljive.

Ključnu reč `var` ste koristili za deklaraciju promenljivih (promenljive su obrađene detaljnije u poglavlju pod nazivom "Promenljive"). Odeljak var možete deklarisati na nekoliko mesta u okviru programa. Možete postaviti odeljak `var` u nivou junita, možete imati odeljak `var` za proceduru, funkciju odnosno oba dela programa istovremeno. Takođe možete imati višestruke odeljke za promenljive u okviru junita. Listing 1.5 prikazuje primer junita sa dodatim odeljcima `type i var`.

Listing 1.5: Junit sa dodatim odeljcima `type` i `var`

```
unit Unit2;

interface

type
  TMyArray = array [0..19] of Byte;

const
  AppCaption = 'My Cool Program 1.0';

var
  X : Integer;
  MyArray : TMyArray;

  procedure DoSomething;

implementation

const
  BaseX = 20;
  BaseY = 200;

  procedure DoSomething;
  begin
    { Code for DoSomething goes here. }
  end;

end.
```

Kao što je to slučaj kod ključne reči `const`, ključna reč `var` takođe ima više primena. Ključna reč var može biti korišćena za deklarisanje parametara funkcije i procedure u okviru kojih se nalaze kao parametarske promenljive. Da se ne bih upuštao u objašnjenje, sačuvaću ga za sutrašnju lekciju kada ćete moći da čitate o funkcijama i procedurama.

NAPOMENA Odeljci opisani ključnim rečima `var`, `const` i `type` počinju nakon ključne reči a završavaju se sledećom ključnom reči u okviru junita.

Komentari u okviru koda

Pre nego što obradimo jezik Pascal detaljnije, dozvolite mi da Vam ukratko objasnim kod za komentar. *Komentari* su linije teksta u okviru izvornog koda koji se u kodu nalaze sa svrhom da ga dokumentuju. Komentari mogu da se koriste za opisivanje šta kod radi, da daju informacije o autorskim pravima, odnosno da označe napomene koje ste naveli za sebe, odnosno za druge programere.

Komentari mogu biti upotrebljeni na tri različita načina. U nastavku su navedene sve ispravne linije sa komentarima:

```
{ Don't forget to free this memory! }
{
 ADTAPI.PAS 2.50
 Copyright (c) TurboPower Software 1996-98
}
(* Mason needs to fix this section of code *)
// This is really good code!
{ This code needs to be reworked later }
```

Možda najčešći tip komentara koji se koristi u Delphi programima sadrži vitičaste zagrade kao što je to ilustrovano u prva dva slučaja u prethodnom primeru. Otvorena zagrada se koristi kao početak komentara, a zatvorena zagrada se koristi kao kraj komentara. Sledeći tip komentara koristi `(*` za početak komentara, a `*)` za kraj komentara. Postoji jedna razlika između komentara naznačenih na ovaj način i komentara koji koriste vitičaste zagrade: Par `(* /  *)` se može koristiti da označi blok velikih delova koda koji između ostalog sadrže i linije sa komentarima. Ova dva tipa komentara se mogu koristiti za komentarisanje u okviru samo jedne linije koda, odnosno komentarisanje koda u više linija.

NAPOMENA Vitičaste zagrade imaju još jednu primenu u Pascalu. Kada se koriste u sprezi sa oznakom dolara ($), zagrade označavaju direktivu prevodiocu. Da biste naredili prevodiocu da ne generiše napomene prevodioca, možete ubaciti u izvorni kod ovakvu liniju:

```
{$HINTS OFF}
```

Kada prevodilac naiđe na ovu liniju, prestaje sa generisanjem napomena u okviru junita sve do pojave direktive `{$HINTS OFF}`. Direktive prevodioca ću zasebno obraditi u različitim delovima knjige kada se za to ukaže potreba.

Treći tip komentara je označen kao dvostruka kosa crta. Ovakvi komentari se obično nazivaju komentari u C stilu, pošto se koriste u programskom jeziku C odnosno C++. Ovakav tip komentara se može koristiti u okviru jedne linije koda. Trebalo bi da znate da ovaj tip komentara nije primenljiv u svim verzijama Delphija. Ako pišete kod koji će najverovatnije biti korišćen u Delphiju 1, isto kao i u kasnijim verzijama, trebalo bi da se uverite da se ovakav stil komentara ne pojavljuje.

Ja koristim vitičaste zagrade za komentarisanje proizvodnog koda (kod koji će drugi videti). Za brzo komentarisanje jedne ili dve linije u svrhu testiranja koristim dvostruku kosu liniju, ali samo kao privremenu meru. Retko koristim komentare u stilu (* / *).

Kompajler ignoriše tekst komentara. Ako koristite generička podešavanja Delphijevog IDE, sve linije sa komentarima će biti prikazane italik plavim tekstom. Ovo omogućava da se lako i brzo pronađu linije sa komentarima.

Ako radite u okviru tima za programiranje, verovatno treba da čitate kod Vaših saradnika, odnosno da Vaši saradnici čitaju Vaš kod. Koncizni komentari u okviru koda mogu da uštede sate rada programera koji treba da čita i održava kod nekog drugog programera. Čak i ako radite samostalno, komentarisanje Vašeg koda je dobra ideja. Iznenadićete se kako brzo možete zaboraviti šta bi kod koji ste napisali trebao da radi. Dobro komentarisanje koda može da uštedi Vama i Vašim saradnicima sate, zato nemojte da zaboravite da komentarišete Vaš kod!

Promenljive

Promenljive treba da budu deklarisane pre nego što se koriste. Promenljive treba da deklarišete u posebnom odeljku koda koji je naznačen ključnom reči `var`, kao što je to opisano ranije; sledi primer:

```
var
  X : Integer;  { variable X declared as an integer variable }
  Y : Integer;  { variable Y declared as an integer variable }
```

U prethodnom delu sam objasnio ključnu reč `var` i njenu ulogu u Pascal junitu. U ovom poglavlju sam objasnio da se promenljive koje se koriste u junitu deklarišu u okviru odeljka `var`. To je tačno, ali odeljak var možete imati takođe i u okviru funkcije ili procedure. Ovo Vam omogućava da deklarišete promenljive u funkcijama i procedurama isto kao što ih deklarišete u junitima. Sledi primer odeljka `var` u okviru procedure.

```
procedure TForm1.Test;
var
  S : string;
begin
  S := 'Hello World!';
  Label1.Caption := S;
end;
```

Nakon što ste deklarisali promenljive možete da ih koristite za upravljanje podacima u memoriji. Ovo Vam verovatno ne izgleda logično, pa mi zato dozvolite da Vam dam par primera. Isečci koda koji slede koriste promenljive pod nazivom X i Y deklarisane ranije. Na kraju svake linije koda je komentar koji opisuje šta se dešava kada se linija izvršava:

```
X := 100;       { 'X' now contains the value 100 }
X := X + 50;    { 'X' now contains the value 150 }
Y := 150;       { 'Y' now contains the value 150 }
X := X + Y;     { 'X' now contains the value 300 }
Inc(X);         { Increment. 'X' now contains the value 301 }
```

NOVI TERMIN *Promenljiva* je neka vrednost koju sadrži lokacija u memoriji računara.

Želeo bih da Vam objasnim nekoliko stvari u vezi ovog koda. Prvo, zapamtite da se vrednost promenljive X menja u zavisnosti od manipulacije promenljivom. (Nešto kasnije ću objasniti operatore Object Pascal-a, njegove funckije i procedure koje se koriste za upravljanje promenljivama.) Možete videti da se promenljivama može dodeliti vrednost, mogu se sabirati, inkrementirati i slično.

Zapazite takođe da se svaki iskaz u okviru segmenta koda završava znakom tačka-zarez. Znak tačka-zarez se koristi na kraju svakog iskaza u okviru Pascal programa.

NAPOMENA Veoma rano u procesu učenja jezika Pascal, novopečeni programer mora naučiti razliku između izraza i iskaza. Službena definicija iskaza je: izraz iza koga sledi znak tačka-zarez. Izraz je deo koda koji izračunava neku vrednost. Zbunjeni? Razmislite o sledećem iskazu:

```
C := A+B;
```

U ovom primeru, deo sa desne strane operatora za dodelu (A+B) je izraz. Kompletna linija je iskaz. Moglo bi se reći da je izraz podskup iskaza. Jedan iskaz može biti sastavljen od nekoliko izraza. Znam da ovo može biti u ovom trenutku zbunjujuće, ali će Vam biti jasnije kako budete odmicali sa učenjem. Za sada zapamtite da iza iskaza sledi znak tačka-zarez. (Postoje slučajevi kada se na kraju linije ne koristi znak tačka-zarez, ali ovo ne pobija pravilo da se znak tačka-zarez stavlja na kraju svakog iskaza. Ove izuzetke ću Vam objasniti kasnije u nastavku knjige kada budemo došli do njih.)

Nazivi promenljivih slede pravila koja su opisana za *identifikatore*. Kao dodatak promenljivama , identifikatori se koriste za definisanje naziva funkcija, procedura polja u okviru slogova, naziva junita i tako dalje. Identifikatori mogu mešati mala i velika slova i mogu u sebi sadržati brojeve i znak za podvlačenje (_), ali ne mogu u sebi sadržati prazna mesta odnosno druge specijalne karaktere. Identifikatori moraju počinjati karakterom ili znakom za podvlačenje. Maksimalna dužina identifikatora nije definisana, ali sve duže od 255 karaktera će biti ignorisano. U realnosti, sve duže od 20 karaktera je veoma dugo da bi bilo korisno. Slede primeri ispravnih naziva promenljivih.

```
aVeryLongVariableName : Integer; { a long variable name }
my_variable : Integer; { a variable with an underscore }
x : Integer; { single digit variable name }
X : Integer; { same as above }
Label2 : string;  { a variable name containing a number }
```

Na jezik Pascal ne utiče veličina slova (case sensitive). Sledeći iskazi su ispravni:

```
var
  XPos : Integer;

{ ...later }
XPos := 20;
XPOS := 200;
xpos := 110;
XpoS := 40;
```

Ako ste prešli sa jezika u kom je bitna veličina slova (C ili C++ na primer), tolerantna priroda jezika Object Pascal u vezi veličine slova će Vam možda biti strana u početku, ali ćete se na nju veoma brzo navići.

Iako je Pascal jezik u kom veličina slova nema ulogu, treba da težite kozistentnom označavanju u Vašim programima. Korišćenje odgovarajućih velikih slova čini Vaš program lakšim za čitanje i može Vas spasiti brojnih glavobolja ukoliko želite da prebacite Vašu aplikaciju na drugi programski jezik (prilagođavanje Delphi programa jeziku C++Builder, na primer).

Tipovi podataka Object Pascal-a

U Object Pascal-u *tipovi* podataka definišu način na koji prevodilac upisuje podatke u memoriju.

U nekim programskim jezicima možete proći sa dodeljivanjem bilo kog tipa vre-dnosti promenljivoj. Primera radi, pogledajte sledeći primer na jeziku BASIC:

```
X = -1;
X = 1000;
X = 3.14;
```

U programskom jeziku BASIC interpreter brine o dodeli dovoljno prostora da bi se u memoriju ubacio bilo koji tip broja.

Deklaracija promenljive

U Object Pascal-u morate deklarisati tip promenljive pre nego što koristite promenljivu:

```
var
  X1 : Integer;
  X  : Integer;
  Y  : Double;
  Z  : Byte;
```

```
{ ...later }
X1 := -1;
X  := 1000;
Y  := 3.14;
Z  := 27;
```

Ovo omogućava prevodiocu da proveri tip promenljive kako bi bio siguran da će sve biti u redu kada program bude bio pokrenut. Neodgovarajuće korišćenje tipa podataka rezultuje greškom prevodioca, odnosno upozorenjem koje može biti analizirano i ispravljeno, tako da možete uočiti problem pre nego što se pojavi.

Neki tipovi podataka mogu sadržati znak *(signed)*, dok drugi ne sadrže znak *(unsigned)*. Tipovi podataka sa znakom mogu sadržati i negativne i pozitivne brojeve, dok tipovi podataka bez znaka mogu sadržati samo pozitivne brojeve. Tabela 1.1 prikazuje osnovne tipove podataka u jeziku Object Pascal, memorijski prostor koju tipovi podataka zauzimaju i opseg vrednosti u kom se može naći svaki tip podataka. Ova tabela ne uključuje podatke tipa string. O ovome će biti reči kasnije u poglavlju "Stringovi".

Tabela 1.1: Tipovi podataka koji se koriste u jeziku Object Pascal (32-bitni programi)

Tip podataka	Dužina u bajtovima	Mogući opseg vrednosti
ShortInt	1	-128 do 127
Byte	1	0 do 255
Char	1	0 do 255 (isto kao Byte)
WideChar	2	0 do 65.535 (isto kao Word)
SmallInt	2	-32.768 do 32.767
Word	2	0 do 65535
LongInt	4	-2.147.483.648 do 2.147.483.647
Int64	8	-9.223.372.036.854.775.808 do 9.223.372.036.854.775.807
Integer	4	Isto kao LongInt
Cardinal	4	0 do 2,147,483,647
Single	4	1.5 ' 10-45 do 3.4 ' 1038
Double	8	5.0 ' 10-324 do 1.7 ' 10308
Real	8	5.0 ' 10-324 do 1.7 ' 10308 (isto kao Double)
Extended	10	3.4 ' 10-4932 do 1.1 ' 104932
Comp	8	-9,223,372,036,854,775,808 do 9,223,372,036,854,775,807
Currency	8	-922,337,203,685,477.5808 do 922,337,203,685,477.5807

nastavlja se

Tabela 1.1: Tipovi podataka koji se koriste u jeziku Object Pascal (32-bitni programi) *nastavak*

Tip podataka	Dužina u bajtovima	Mogući opseg vrednosti
Boolean	1	True or False
Variant	16	Varies

Možda ste primetili pregledajući tabelu 1.1, da je tip podataka `Integer` isti kao i `LongInt`. Pa zašto Object Pascal ima dva različita tipa podataka, kada su oba potpuno ista? Suština se nalazi u prošlosti. U 16-bitnom okruženju, tip `Integer` je zahtevao 2 bajta memorije, a `LongInt` 4 bajta memorije.

U 32-bitnom okruženju oba tipa zahtevaju 4 bajta memorije i imaju isti opseg. Delphi 4 pravi samo 32-bitne programe, pa su tipovi `Integer` i `LongInt` identični. Većina programera radije koristi `Integer` nego `LongInt`.

Možda ste, takođe, primetili da `Int64` i `Comp` (Computational) tipovi imaju identičan opseg. Razlika između ova dva tipa je način na koji ih kompajler interno obrađuje. Tip `Int64` je ceo broj, dok je `Comp` tip realan broj. Možda ćete imati veoma malo razloga da koristite tip `Comp` u Vašim programima.

Zapazite, takođe, da su tipovi `Real` i `Double` identični. U prethodnim verzijama Delphija, tip `Real` je sadržavao promenljive duge 6 bajtova. Sada je njihova dužina 8 bajtova. Ove izmene tipa `Real` su učinjene zbog kompatibilnosti sa današnjim procesorima. Tip `Real` se smatra zastarelim i treba da koristite tip Double umesto tipa `Real` u Vašim Delphi aplikacijama.

Tip podataka Int64 je novina u Delphiju 4. Postoji puno razloga za ceo broj ove veličine. Najveći zahtev za uvođenje ove celobrojne vrednosti su bili današnji tvrdi diskovi velikog kapaciteta. Na primer, Windows sadrži funkciju pod nazivom `GetDiskFreeSpaceEx` koji može vratiti vrednost mnogo veću od 2.147.483.647 (maksimalna vrednost tipa Integer). 64-bitni celobrojni tip je potreban upravo iz ovih razloga.

Tipovi podataka `Single`, `Double`, `Extended` i `Currency` koriste brojeve u pokretnom zarezu (brojevi sa decimalnim mestima). Ostali tipovi podataka rade samo sa celim brojevima. Dodeljivanje vrednosti koja sadrži decimalni deo, tipu podataka definisanom kao ceo broj, nije moguć. Primera radi, sledeći kod će generisati grešku prevodioca:

```
var
  X : Integer;
{ Later... }
X := 3.75;
```

O sličnim stvarima ne morate puno da brinete pošto je prevodilac veoma dobar i upozorava Vas šta možete, a šta ne možete da radite. Primera radi, bićete iznenađeni koliko malo brojeva u pokretnom zarezu postoji u većini Windows programa.

Konverzija tipova podataka

Object Pascal radi konverziju različitih tipova podataka, ukoliko je to moguće. Uzmite kao primer sledeći isečak koda:

```
var
  Res : SmallInt;
  Num1 : Integer;
  Num2 : Integer;

{ Later... }
  Num1 := 200;
  Num2 := 200;
  Res  := Num1 * Num2;
```

U ovom slučaju pokušao sam da dodelim rezultat množenja dva broja tipa `Integer` tipu `SmallInt`. Iako ova formula meša dva tipa podataka, Object Pascal je u mogućnosti da izvrši konverziju. Da li biste želeli da pogodite rezultat? Možda ćete biti iznenađeni kada otkrijete da je rezultat `-25.536`. Šta!? Ako pogledate u tabelu 1.1, videćete da `SmallInt` može imati maksimalnu vrednost `32.767`. Šta se događa ako na vrednost definisanu kao `SmallInt` koja iznosi `32.767` dodate 1? Dobićete vrednost `-32.768`. Ovo je potpuno isto kao kod brojača kilometara koji se nakon 99.999 prebaci na 00,000. Da bismo ovo ilustrovali izvršite sledeće korake:

1. Startujte novu aplikaciju i ubacite natpis i dugme u formu.
2. Dvostrukim klikom miša kreirajte upravljač događaja za događaj dugmeta `OnClick`.
3. Izmenite upravljač događaja da izgleda ovako:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X : SmallInt;
begin
  X := 32767;
  X := X + 1;
  Label1.Caption := IntToStr(X);
end;
```

4. Pokrenite program i kliknite na dugme.

Trebalo bi da vidite kako se menjaju brojevi komponente natpis u broj `-32.768` kada kliknete na dugme (u slučaju da se pitate, funkcija `IntToStr` prevodi cele brojeve u string). Ova vežba ilustruje da je `32.767` plus 1 jednako `-32.768`! U redu, možda i nije tako.

Ovaj primer u stvarnosti ilustruje ono što je poznato kao *overflow* (prekoračenje), odnosno premotavanje *(wrapping)*. Morate paziti na maksimalne vrednosti koje Vaše promenljive mogu imati i odabrati tipove podataka koji su dovoljno veliki da bi

promenljiva sigurno mogla da upiše vrednost bez premotavanja. U većini slučajeva nećete pogrešiti ukoliko odaberete tip `Integer` kao izbor. Malo je verovatno da ćete imati problem premotavanja pošto tip promenljive `Integer` daje brojeve u opsegu od približno -2 milijarde do +2 milijarde.

Gde sam ono stao? Da, govorio sam o automatskim tipovima konverzije. U nekim slučajevima Object Pascal ne može da izvrši konverziju. Ukoliko je ovo slučaj, prevodilac će prijaviti grešku sa porukom koja izgleda otprilike ovako: `Incompatible types: 'Integer' and 'Real'`. (Nekompatibilni tipovi: `'Integer' i 'Real'`) Ova greška prevodioca Vas upozorava da ste pokušali da dodelite vrednost koja ne može biti upisana u navedeni tip podataka. Još jedna greška prevodioca koju ćete možda videti ima veze sa takozvanom proverom opsega *(range checking)*. Pogledajte deo ovog koda kao primer:

```
var
  X : Byte;
begin
  X := 1000;
end;
```

Ovaj kod će generisati grešku prevodioca koja glasi "`Constant expression violates subrange bands.`" (Izraz za konstantu narušava ograničenja podopsega.) Prevodilac Vam poručuje da ne možete dodeliti promenljivoj X vrednost `1000`, pošto je promenljiva X deklarisana kao `Byte`, a tip promenljive `Byte` može sadržati samo vrednosti od `0` do `255`.

Naučite da tretirate savete i upozorenja prevodioca kao greške. Prevodilac pokušava da Vam saopšti da nešto ne štima u Vašem kodu i da treba da uvažite ovo upozorenje. Na kraju, možete da pokušate sa prevođenjem bez upozorenja. U retkim slučajevima, upozorenja ne mogu biti izbegnuta, ali budite sigurni da ste detaljno ispitali sva upozorenja. Učinite sve da shvatite razloge za upozorenja i da ih ispravite ukoliko je to moguće.

Operatori jezika Object Pascal

Operatori se koriste za manipulaciju podacima. Operatori izvršavaju operacije računanja, proveru jednakosti, dodeljivanja, manipulišu promenljivama odnosno izvršavaju druge ezoterične zadatke koje većina programera nikad ne radi. U jeziku Object Pascal postoji veliki broj operatora. Predstavićemo Vam operatore koji se najćešće koriste umesto nabrajanja svih tipova operatora. Tabela 1.2 sadrži listu ovih operatora.

Tabela 1.2: Često korišćeni operatori u jeziku Object Pascal

Operator	Opis	Primer
		Matematički operatori
+	sabiranje	x :=y + z;
-	oduzimanje	x := y - z,
*	množenje	x := y * z;
/	deljenje realnog broja	x := y / 3.14;
div	celobrojno deljenje	x := y div 10;
		Operatori dodeljivanja
:=	dodeljivanje	x :=10;
		Logički operatori
and	logičko "i"	if (x=1) and (y=2) then...
or	logičko "ili"	if (x=1) or (y=2) then...
		Operatori jednakosti
=	jednako	if (x=10) then...
<>	nije jednako	if (x<>10) then...
<	manje je od	if (x<10) then..
>	veće je od	if (x>10) then...
<=	veće ili jednako	if (x<=) then...
>=	manje ili jednako	if (x>=) then...
		Unarni operatori
^	pointer operator	MyObject.Data^;
@	adresa operatora	ptr := @MyRecord;
and"i"	na nivou bita	x := x and $02;
or"ili"	na nivou bita	x := x or $FF;
not"ne"	na nivou bita	x := x and not $02;
not	logičko "ne"	if not Valid then ...
		Ostali operatori
$	operator heksadecimalne vrednosti	X := $FF;
[]	operator indeksa niza	x := MyArray [5];
.	operator pripadanja (tačka)	x := Record.Data;

Kao što možete videti lista operatora je veoma opširna. Nemojte se brinuti i pokušavati da ih sve zapamtite. Kako budete radili sa jezikom Object Pascal, postepeno ćete naučiti kako da koristite sve ove operatore. Neke operatore ćete retko, a možda i nikada koristiti, dok ćete ostale upotrebaljavati veoma često.

Zapazili ste da su ključne reči `and`, `or` i `not` korišćene u dva konteksta: logičke i na nivou bita. Na primer, ključna reč `and` se može koristiti za definisanje logičke "i" operacije, odnosno "i" operacije na nivou bita. Pogledajte ovaj kod:

```
if (Started = True) and (X > 20) then
  Z := X and Y;
```

U ovom primeru ključna reč `and` se koristi u dva potpuno različita konteksta. Nesumljivo će Vam ovo u početku biti zbunjujuće. Kasnije ćete se uveriti da prevodilac zna kako se ključna reč koristi i uradiće pravu stvar. Izgleda sam malo požurio na početku knige, zato ne brinite ako Vam ovo sada ne izgleda smisleno. Kasnije će Vam gotovo sigurno sve imati puno više smisla nego sada.

Čitajući ovu knjigu, videćete puno primera koji koriste ove operatore. Umesto da pamtite funkciju svakog operatora, pokušajte da pažljivo prostudirate primere programa i isečke koda.

Konstante

Kao što sam ranije rekao, konstante su identifikatori dodeljeni vrednosti koja se ne menja. Termini "promenljiva" i "konstanta" nisu slučajno odabrani. Vrednost promenljive može menjati programer, dok se vrednost konstante ne može menjati. Konstante se deklarišu korišćenjem ključne reči const. Da biste deklarisali konstantu, jednostavno ukucajte naziv konstante i njenu vrednost - na primer:

```
const
  DefaultWidth  = 400;
  DefaultHeight = 200;
  Description   = 'Something really cool.';
```

Zapazite da je prilikom deklarisanja konstanti korišćen znak jednakosti, a ne operator dodeljivanja (:=). Možete takođe videti da nije definisan tip podataka. Prevodilac određuje tip podataka konstante na osnovu vrednosti koja joj je dodeljena. Konstante zatim mogu biti korišćene u Vašem kodu tamo gde normalno koristite njihove nazive.

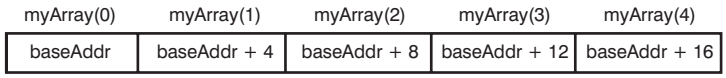
Pažljivo korišćenje konstanti čini program lakšim za izmenu ukoliko se kasnije ispostavi da su promene neophodne. Da biste izmenili ponašanje programa, neophodno je da izmenite samo vrednosti jedne, odnosno više konstanti na početku junita, umesto da u junitu tražite, primera radi, svaku pojavu broja `100` i da ga menjate u `120`.

Nizovi

U niz možete ubaciti bilo koji osnovni tip podataka jezika Object Pascal. *Niz* je jednostavno skup vrednosti. Na primer, recimo da želite da napravite niz `celih` brojeva koji sadrži 5 celobrojnih vrednosti. Ovaj niz možete deklarisati:

```
var
  MyArray : array[0..4] of Integer;
```

U ovom slučaju prevodilac rezerviše memoriju za niz kao što je to ilustrovano na slici 1.4. Pošto svaka celobrojna vrednost zahteva 4 bajta za zapisivanje, ceo niz će zauzeti 20 bajtova u memoriji.



Slika 1.4 Rezervisanje memorije za niz od 5 celobrojnih elemenata

Sada kada ste deklarisali niz, možete ga popuniti vrednostima koristeći *operator* za indeks (`[]`):

```
MyArray[0] := -200;
MyArray[1] := -100;
MyArray[2] := 0;
MyArray[3] := 100;
MyArray[4] := 200;
```

Kasnije u Vašem programu možete pristupiti zasebnim elementima niza korišćenjem operatora indeksa:

```
X := MyArray[3] + MyArray[4];  { result will be 300 }
```

Višedimenzionalni nizovi

Nizovi mogu biti višedimenzionalni. Da kreirate dvodimenzionalni niz celih brojeva, koristite kod poput ovog:

```
var
  MdArray : array[0..2, 0..4] of Integer;
```

Ovo rezerviše prostor za 15 `celih` brojeva (ukupno 60 bajtova, ako beležite). Elementima pristupate kao i kod jednostavnih nizova, sa očiglednom razlikom da morate upisati dva operatora indeksa. Postoje dva načina da ovo uradite. Sledeće dve linije daju iste rezultate:

```
X := MdArray[1][1] + MdArray[2][1];
X := MdArray[1, 1] + MdArray[2, 1];
```

Slika 1.5 ilustruje kako dvodimenzionalni niz može izgledati u memoriji.

	mdArray()(0)	mdArray()(1)	mdArray()(2)	mdArray()(3)	mdArray()(4)
MdArray(0)()	baseAddr	baseAddr + 4	baseAddr + 8	baseAddr + 12	baseAddr + 16
MdArray(1)()	baseAddr + 20	baseAddr + 24	baseAddr + 28	baseAddr + 32	baseAddr + 36
MdArray(2)()	baseAddr + 40	baseAddr + 44	baseAddr + 48	baseAddr + 52	baseAddr + 56

Slika 1.5 Dvodimenzionaln i niz u memoriji

Pod normalnim okolnostima provera opsega će Vas sprečiti da pokušate upisivanje podataka posle kraja niza. Na primer, sledeći kod će rezultovati greškom prevodioca:

```
var
  MyArray : array[0..4] of Integer;
  X : Integer;
begin
  X := MyArray[3] + MyArray[5]; { Oops! 5 outside of
range. }
end
```

Greška će prijaviti "Constant expression violates subrange bounds" (Izraz konstante narušava granice podopsega.), pošto je `MyArray[5]` van deklarisanog opsega niza.

Opseg niza je definisan kada deklarišete niz. Na primer, ako želite da kreirate niz sa donjom granicom 10, i gornjom granicom 20, deklarisaćete ga ovako:

```
var
  MyArray : array[10..20] of Integer;
```

Sada će jedini elementi niza kojima se može pristupiti biti u opsegu od 10 (prvi element niza) do 20 (poslednji element niza). Sve konstante niza moraju biti deklarisane i inicijalizovane istovremeno. Sintaksa izgleda ovako:

```
const
  myArray : array[0..4] of Integer = ( -200, -100, 0, 100, 200 );
```

Funkcije Low i High

Funkcije `Low` i `High` se često koriste u radu sa nizovima. Kao što sam napomenuo ranije, niz može biti deklarisan korišćenjem promenljivih koje definišu donju i gornju granicu. Funkcija `Low` vraća donju granicu niza, dok funkcija `High` vraća gornju granicu niza. Na primer,

```
var
  X, I, Lower, Upper : Integer;
  MyArray : array[10..20] of Integer;
begin
  { Code to initialize MyArray here. }
  Lower := Low(MyArray);  { Lower now contains 10 }
  Upper := High(MyArray); { Upper now contains 20 }
  X := 0;
```

```
  for I := Lower to Upper do
    X := X + MyArray[I];
  { Now do something with X. }
end;
```

Korišćenjem funkcija `Low i High` osiguravate se od pokušaja pristupa elementu niza koji se nalazi van opsega.

Dinamički nizovi

Delphi 4 uvodi koncept dimaničkih nizova. Dinamički niz se deklariše bez inicijalne dužine pa za njega nije ostavljen prostor u memoriji u trenutku deklarisanja. Niz može biti kreiran kasnije definisanjem veličine korišćenjem funkcije `SetLength`. Evo kako to izgleda:

```
var
  BigArray : array of Integer; { no size }
  X : Integer;
begin
  X := GetArraySize;  { function which returns the needed
size }
  SetLength(BigArray, X); { dynamically allocate array }
  { Now fill in and use BigArray }
end;
```

Dinamički niz je niz za koji se rezerviše memorija u toku izvršavanja programa. Dinamički niz može biti veći ili manji u zavisnosti od potre ba programa.

Značajno je da niz može biti smešten u memoriju u zavisnosti od broja elemenata koje niz sadrži. Da bi ovo ilistrovali recimo da Vam je potreban niz sastavljen od celih brojeva. Pretpostavimo da je u nekim slučajevima potrebno rezervisati dovoljno memorije za 10 celih brojeva, a u drugim slučajevima ćete možda imati potrebu da rezervišete memoriju za 1000 celih brojeva.

Vaš program u toku prevođenja nema informaciju o tome koliko Vam je potrebno elemenata- broj elemenata nije poznat sve do pokretanja programa. Pre uvođenja dimaničkih nizova bili ste primorani da deklarišete niz dužine 1000 celih brojeva trošeći veliku količinu memorije ukoliko Vaša aplikacija zahteva 10, 20, odnosno 30 celih brojeva. Sa dinamičkim nizovima možete da rezervišete onoliko prostora koliko je potrebno u određenom trenutku.

Korišćenjem funkcije Copy možete da prebacite niz. Primera radi, recimo da ste inicijalno kreirali niz koji sadrži 100 elemenata, a sada Vam je potrebno da uvećate niz da bi sadržao 200 elemenata. U tom slučaju kod bi izgledao ovako:

```
Copy(BigArray, 200);
```

Sadržaj niza je ostao netaknut, a njegova veličina se povećala za 100 elemenata, tako da je ukupna dužina niza 200 elemenata.

Dvodimenzionalni dinamički nizovi se kreiraju uglavnom na isti način. Da biste kreirali dvodimenzionalni niz, možete koristiti sledeći kod:

```
var
  BigArray : array of array of Integer;
begin
  SetLength(BigArray, 20, 20);
  BigArray[0][0] := 200;
  { More code here. }
end;
```

Nakon što je dinamički niz kreiran elementima niza se može pristupiti na isti način kao kod normalno definisanog niza.

Stringovi

Stringovi se veoma često koriste u programiranju. Object Pascal definiše tri različita tipa: dugi string (long string), kratki string (short string) i široki string (wide string). Dodatno, Pascal koristi null string. Opisaću kratko svaki od ovih tipova, a zatim ću objasniti neke funkcije za manipulaciju stringovima.

Kratki string (short string)

Tip kratkog stringa (short string) ima fiksnu dužinu koja maksimalno iznosi 255 karaktera. Deklarisanje katkih stringova se može uraditi na dva načina. Jedan od načina je korišćenje prethodno definisanje tipa promenljive kao `ShortString` da bi deklarisali kratki `string` dužine 255 bajtova. Takođe možete koristiti ključnu reč string uz dodatak operatora indeksa koji određuje dužinu stringa:

```
var
  S1 : ShortString; { 255 characters long }
  S2 : string[20];  { 20 characters long }
```

Manipulacija stringovima korišćenjem kratkih stringova je brža pošto se količina memorije rezervisane za stringove ne menja. Ipak smatra se da je kratki string zastareo i preporučuje se korišćenje dugih stringova. Kratki stringovi se još mogu nazvati bajt stringovima, pošto prvi element stringa sadrži dužinu stringa (broj karaktera u okviru stringa). Na primer, da biste odredili dužinu stringa možete pročitati vrednost prvog elementa kratkog stringa.

```
var
  S   : ShortString; { 255 characters long }
  Len : Integer;
begin
  S   := 'Hello';
  Len := Ord(S[0]); { 'L' now contains the length of S, or 5 }
end;
```

Ovaj primer čita vrednosti stringa `S[0]` da bi odredio dužinu stringa. Da bi odredili dužinu kratkog stringa možete takođe koristiti funkciju `Length`. Funkcija `Length` će biti obrađena nešto kasnije.

Da biste konvertovali vrednost tipa `Char` u celobrojnu vrednost (ordinalnu vrednost) treba da koristite funkciju `Ord`. Funkcija `Ord` se koristi i kod numerisanja.

Da biste tačno definisali dužinu stringa ukoliko je to potrebno, možete u string upisati prvi element. Određene situacije zahtevaju ovakav pristup, ali ih neću bliže objašnjavati. Dodao bih samo da u principu, korišćenje kratkog stringa dužine 0 bajta spada u napredne tehnike programiranja i ne preporučuje se programerima početnicima.

Dugi string (long string)

Tip podataka deklarisan kao dugi string (long string) dinamički rezerviše memoriju za dati objekat. Dužina dugog stringa je ograničena samo veličinom slobodne memorije. Object Pascal rezerviše, odnosno oslobađa memoriju u zavisnosti od potreba stringa. Dugi stringovi su veoma fleksibilni ali ponekad i sporiji u odnosu na kratke stringove, naročito kada se sa njima intenzivno manipuliše. Razlog leži u dinamičkom rezervisanju mesta u memoriji za duge stringove i njihovoj izmeni sadržaja. Sve dok brzina izvršavanja operacija nije bitna, trebalo bi da se držite korišćenja dugih stringova u aplikacijama.

Da biste deklarisali dugi string koristite ključnu reč `string` bez navođenja parametara dužine:

```
var
  S : string;  { long string, dynamically allocated }
```

Pošto su stringovi dinamički raspoređeni, možete ih menjati po želji, a da istovremeno ne brinete šta se u pozadini događa. Korišćenje dugih stringova je veoma jednostavno pošto ne morate da brinete o rezervisanju prostora za string, odnosno da li ćete prekoračiti dozvoljenu dužinu. Sve je manje više automatizovano.

Dugi stringovi za razliku od kratkih stringova nemaju element 0. Pokušaji pristupa nultom elementu kod dugih stringova kao rezultat daje grešku prevodioca. Umesto toga dužinu dugog stringa možete dobiti korišćenjem funkcije `Length`, odnosno dužinu možete dodeliti procedurom `SetLength`. Funkcije za manipulaciju stringovima sam obradio u poglavlju "String funkcije".

Široki stringovi (wide string)

Tip podataka široki string (wide string) se koristi u sprezi sa Windows API funkcijama koje zahtevaju dvobajtne karakter stringove (Unicode karakter stringove). Široki string ima sličnosti sa dugim stringom pošto je njegova dužina ograničena raspoloživom memorijom, odnosno memorijom koju je string dinamički rezervisao. Široke stringove neću detaljnije opisivati pošto se retko koriste; uglavnom za rad sa OLE funkcijama.

Null stringovi: `PChar` i niz karaktera (`Array of Char`)

Za razliku od jezika Object Pascal, jezici C i C++ nemaju prave string tipove podataka. U jezicima C i C++ stringovi su inplementirani kao nizovi karaktera koji se završavaju terminirajućom nulom (`0` na kraju stringa). Karakter nizovi nemaju bajt dužine, pa se terminirajuća nula koristi da označi kraj stringa karaktera. Pošto je operativni sistem Windows pisan na jeziku C, većina Windows funkcija zahteva karakter niz kao parametar. Pascal stringovi nisu karakter nizovi, pa je potrebno definisati karakter niz ukoliko se želi postići da Pascal stringovi rade sa Windows funkcijama. Tip `PChar` eliminiše ovaj nedostatak. `PChar` se može koristiti tamo gde su potrebni karakter nizovi. Primer je Windows funkcija `MessageBox`. Ova funkcija koja prikazuje standardni Windows dijalog za poruke, ima sledeću deklaraciju:

```
function MessageBox(hWnd: HWND; lpText, lpCaption: PChar; uType:
UINT): Integer;
```

Drugi i treći parametar zahteva korišćenje pointera (ukazivača) na karakter niz (drugi za tekst okvira za poruke, a treći za naslov okvira za poruke). Da bi pozvali ovu funkciju iz Delphi programa, morate koristiti tip `PChar` na sledeći način:

```
var
  Text    : string;
  Caption : string;
begin
  Text := 'This is a test.;
  Caption := 'Test Message';
  MessageBox(0, PChar(Text), PChar(Caption), 0);
end;
```

U ovom slučaju tip `PChar` je korišćen za dodeljivanje Pascalovog dugog stringa nultom terminirajućem stringu. Tip `PChar` takođe možete samostalno koristiti. Ovo je ilustrovano sledećim primerom.

```
var
  Text : PChar;
begin
  Text := 'This is a test.';
  MessageBox(0, Text, 'Message', 0);
end;
```

Pošto Pascal stringovi imaju snažnu podršku za manipulaciju stringovima, verovatno nećete često koristiti tip `PChar`. Obično ćete `PChar` koristiti za konverziju dugih stringova u stringove terminisane nulom, kao što je navedeno u prethodnom primeru. Uočite da Windows API funkciji možete proslediti string literal (string karaktera unutar jednostrukog znaka navoda), u očekivanju tipa `PChar`.

Na kraju, umesto tipa `PChar` možete koristiti niz karaktera. Da bi ovo ilustrovali prethodni isečak koda je ponovo izmenjen:

```
var
  Text : array [0..20] of Char;
begin
  Text := 'This is a test.';
  MessageBox(0, Text, 'Message', 0);
end;
```

Nije važno koji od ovih metoda koristite. Samo zapamtite da za pozive Windows API funkcija ne možete koristiti Pascalov tip podataka string pošto Windows API funkcije zahtevaju kao parametar stringove terminisane nulom. U ovakvim slučajevima treba da koristite tip `PChar`, odnosno niz karaktera.

Osnovne stvari o stringovima

Tipovi Pascalovih stringova imaju nekoliko zajedničkih elemenata. U ovom poglavlju će biti opisane osnovne operacije koje se odnose na sve tipove stringova.

Povezivanje stringova korišćenjem operatora +

Uobičajen zadatak prilikom programiranja je povezivanje stringova (dodavanje stringova). Stringovi mogu biti povezani korišćenjem operatora +; na primer:

```
var
  S1 : string;
  S2 : string;
begin
  S1 := 'Mallory Kim';
  S2 := 'Reisdorph';
  Label1.Caption := S1 + ' ' + S2;
end;
```

Ovaj kod povezuje tri stringa (promenljivu `S1`, string koji sadrži prazno mesto i promenljivu `S2`), a zatim dodeljuje rezultat karakteristici `Caption` natpisa `Label1`. Bilo koji izraz odnosno funkcija koja kao rezutat daje string može se koristiti za povezivanje stringova. Evo još jednog primera:

```
var
  X :  Integer;
begin
  X := 199;
```

```
  Label1.Caption := 'The result is: ' + IntToStr(X);
end;
```

U ovom slučaju funkcija `IntToStr` vraća tip `string` koji možete da koristite kao rezultat funkcije bilo gde, gde se traži string.

Operator indeksa

Još jedan uobičajen slučaj Pascalovih stringova je operator indeksa (`[ ]`). Korišćenjem operatora indeksa možete izdvojiti pojedinačni karakter iz stringa, kao u primeru:

```
var
  S1 : string;
  S2 : Char;
begin
  S1 := 'Hello World!';
  S2 := S1[1];
  Label1.Caption := S2;
end;
```

Promenljiva `S2` u ovom primeru je tipa `Char`, ali bi mogla da bude i dugi string, kratki string, odnosno široki string. Object Pascal konvertuje u odgovarajući format u pozadini, pa na nivou aplikacije nema potrebe da radite sa različitim tipovima stringova. Operator indeksa je od koristi kada Vam se ukaže potreba da pretražujete string karakter po karakter. Stringovi počinju brojem 1: prvi karakter stringa je `S[1]`. Zapamtite da nulti element kratkog stringa (`S[0]`) sadrži dužinu stringa, a ne prvi karakter stringa. U slučaju dugih stringova, odnosno širokih stringova nije moguće pristupiti elementu `S[0]`.

Kontrolni karakteri u stringu

Object Pascal omogućava ubacivanje kontrolnih karaktera u stringove. Ovo je korisno ukoliko imate potrebu za dodavanjem karaktera koji neće biti štampani. Ovi karakteri mogu biti jednostavni, poput karaktera za novu liniju u okviru stringa, odnosno složeni, kao što je ubacivanje kontrolnih karaktera u string koji se šalje serijskom uređaju.

Dodavanje kontrolnih karaktera stringu se vrši korišćenjem karaktera `#`. Ukoliko bi, na primer, želeli da dodate karakter escape (ASCII 27) u Vaš string, ovo možete učiniti na sledeći način:

```
S := 'This is a test. Escape follows.'#27'Finished.';
```

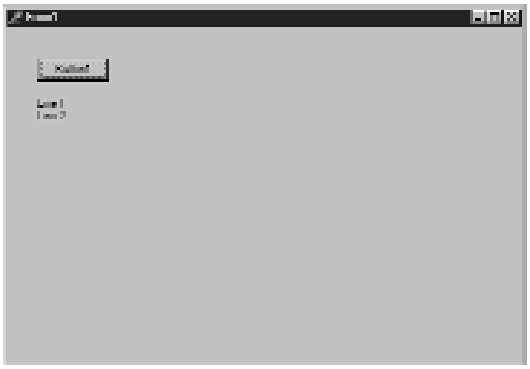
Uočite da je dodati karakter `#27` smešten van znakova navoda, i da između umetnutog karaktera i stringova ne postoji prazno mesto. Da biste koristili umetnute karaktere morate pratiti ovu strukturu. Umesto literala možete koristiti takođe i `string` promenljive:

```
S1 := 'This is a test. Escape follows.';
S2 := 'Finished.';
S3 := S1 + #27 + S2;
```

Ovu teoriju možete jednostavno testirati. Ubacite dugme i natipis u formu. Kliknite dva puta na dugme i dodajte sledeću liniju u okviru opcije `OnClick` upravljača događajima:

```
Label1.Caption := 'Line 1' + #10 + 'Line 2';
```

Sada pokrenite program i kliknite na dugme. Natpis će sadržati dve linije teksta, kao što je to prikazano na slici 1.6. Ovaj kod jednostavno ubacuje karakter za prelaz u sledeći red (ASCII 1O) u string deleći na taj način natips na dve linije.



Slika 1.6
Natpis sa dve linije

Proširivanje stringa na više programskih linija

Često je potrebno ispisati string u okviru dve , odnosno više programskih linija da bi se poboljšala čitljivost i upravljivost koda. Dugi tekstovi poruka na primer, mogu sadržati i preko 200 karaktera. Sve ove karaktere možete staviti u jednu liniju koda (maksimalna dužina linije Delphi editora koda je 1024 karaktera), ali bi ovo učinilo kod skoro nečitljivim. Umesto da delite string na više linija, potrebno je da koristite operator +, kao u primeru:

```
MessageBox(0, 'This is a very, very long message ' +
  'that seems to go on and on forever. In order ' +
  'to make the code more readable the message has ' +
  'been split across several lines of code.', 'Message', 0);
```

Da li se sećate dela kada smo obrađivali znak tačka-zarez i završetke segmenta koda? Ovo je primer kada se iskaz prostire na više programskih linija. Ovo je još uvek jedan iskaz po pitanju prevodioca, pa znak tačka-zarez treba da stoji na kraju *iskaza* a ne na kraju svake linije.

Poređenje stringova

Poređenje stringova se vrši korišćenjem operatora za poređenje. Tabela 1.3 prikazuje najčešće korišćene operatore i njihove opise.

Tabela 1.3: Operatori za poređenje stringova

Operator	Opis
=	jednako
<>	nije jednako
<	manje je od
>	veće je od
<=	manje ili jednako
>=	veće ili jednako

Zapamtite da ovi operatori porede stringove na bazi njihovih ASCII vrednosti. U većini slučajeva ćete koristiti opeatore jednakosti da biste proverili da li je string jednak odnosno da li se razlikuje od navedene vrednosti. Ukoliko sortirate stringove, verovatno ćete koristiti i druge operatore za poređenje stringova. Sledeći primer proverava da li string sadrži određenu vrednost:

```
if FileName = 'TEST.TXT' then
  OpenFile(FileName)
else
  ReportError;
```

Funkcije za manipulaciju stringovima

Object Pascal sadrži veliki broj funkcija i procedura za manipulaciju stringovima. Tabela 1.4 prikazuje nekoliko funkcija i procedura koje su najčešće u upotrebi. Za kompletnu listu funkcija i procedura za manipulaciju stringovima pogledajte Delphi Online help.

Tabela 1.4: Funkcije i procedure za manipulaciju stringovima

Naziv	Opis
`Copy`	Vraća deo stringa.
`Delete`	Briše deo stringa.
`Format`	Formatira i vraća string na osnovu formata stringa i prosleđenih argumenata.
`Insert`	Ubacuje tekst u string.
`IntToStr`	Konvertuje celobrojnu vrednost u string.
`Length`	Vraća dužinu stringa.
`LowerCase`	Konvertuje sva slova stringa u mala slova.
`Pos`	Vraća poziciju traženog stringa koji je sadržan u navedenom stringu.

`StringOfC`	harVraća string popunjen navedenim brojem definisanih karaktera.
`StrPas`	Konvertuje string terminiran nulom (`PChar`, odnosno karakter niz) u Pascalov string.
`StrPCopy`	Konvertuje Pascalov string u string terminiran nulom.
`StrToInt`	Konvertuje string u ceo broj. Ukoliko string ne može biti konvertovan, izbacuje izuzetak.
`StrToIntDef`	Konvertuje string u ceo broj i daje generičku vrednost u slučaju da string ne može biti konvertovan. Ukoliko string ne može biti konvertovan, ne izbacuje izuzetak.
`StrToXXX`	Dodatna funkcija za konverziju koja konvertuje string u broj sa pokret nim zarezom, odnosno u vrednosti `Currency`, `Date`, `Time`.
`Trim`	Odseca prazna mesta na početku i na kraju stringa.
`UpperCase`	Konvertuje sva slova stringa u velika slova.
`XXXToStr`	Dodatna funkcija za konverziju koja konvertuje broj u pokretnom zarezu, odnosno vrednosti `Currency`, `Data`, `Time` u string.

Object Pascal sadrži dodatni set funkcija koje rade sa stringovima koji su terminisani nulom. Ovde ih nećemo obraditi pošto ćete uglavnom koristiti Pascalove stringove, a ne stringove terminisane nulom. Proverite Delphi help ukoliko tražite dodatne informacije o ovim funkcijama. Većina funkcija koje operišu sa stringovima terminisanim nulom počinju sa Str.

Nekoliko funkcija i procedura navedenih u tabeli 1.4 zahtevaju poseban ostvrt. Funkcija `StrToInt` konvertuje string u celobrojnu vrednost. Recimo da imate komponentu za editovanje koja se koristi da prihvati celobrojnu vrednost od korisnika. Pošto komponenta za editovanje može prihvatiti samo tekst, morate tekst konvertovati u ceo broj. Ovo možete učiniti na sledeći način:

```
Value := StrToInt(Edit1.Text);
```

I druge `StrToXXX` funckije (`StrToFloat`, `StrToDate` itd.) rade na potpuno isti način. Zapazite da ove funkcije izbaciju izuzetak ako konverzija nije uspela. Ukoliko na primer, korisnik unese `S123`, biće izbačen izuzetak pošto se slovo *S* ne može konvertovati u ceo broj. O izuzecima još nisam pisao, pa ih u ovom trenutku neću detaljnije objašnjavati.

Funkcija `Format` omogućava kreiranje stringa prosleđivanjem željenog stringa i dodatnih argumenata. Sledeći primer koji dodaje dva broja koristi funkciju `Format` da bi kreirao string koji prikazuje rezultat:

```
var
  S : string;
  X : Integer;
begin
  X := 10 * 20;
  S := Format('The result is: %d', [X]);
  Label1.Caption := S;
end;
```

Kada se izvrši ovaj deo koda, natpis sadrži sledeći tekst:

```
The result is: 200
```

U ovom primeru znak `%d` predstavlja funkciju za `formatiranje`: "Celobrojna vrednost se nalazi ovde." Na kraju formatiranog stringa dodata je promenljiva `X` koja ukazuje funkciji `Format` koju vrednost da stavi na poziciju stringa (sadržaj promenljive `X`).

Funkcija `Format` je jedinstvena po tome što može da primi različit broj argumenata. (Iz tog razloga se promenljiva `X` nalazi u uglastoj zagradi; argument se prosleđuje u formi `niza konstanti`.) String za formatiranje mora biti obezbeđen, a broj argumenata koji se nalaze iza stringa za formatiranje je promenljiv. Evo primera funkcije `Format` koja koristi još tri dodatna argumenta:

```
var
  X : Integer;
  Y : Integer;
begin
  X := 20;
  Y := 5;
```

Kada se izvrši ovaj deo koda, rezultat prikazan u okviru natipisa će biti:

```
20 + 5 = 25
```

Uočite da sam u ovom slučaju dodelio povratnu vrednost iz funkcije `Format` direktno karakteristici `Caption` komponente Label1. U prethodnom primeru sam dodelio povratnu vrednost iz funkcije `Format` promenljivoj, mada ovaj korak nije striktno neophodan.

Dodatni elementi za definisanje formata se koriste za prikaz broja u pokretnom zarezu u naučnoj notaciji, kao heksadecimalni broj, odnosno za prikaz karaktera i stringova. U slučaju prikazivanja brojeva u pokretnom zarezu možete definisati broj decimalnih mesta koji će biti prikazan, odnosno u slučaju celobrojnih vrednosti možete definisati broj cifara koje će biti prikazane. Pogledajte deo "Formatiranje stringova" u okviru Delphi help-a ukoliko želite detaljnije informacije.

Zaključak

Danas smo obuhvatili prilično veliku oblast. Prvo ste se zamislili oko Delphijevog okruženja (Delphi IDE) pišući program `Hello World!`, nakon ovog je sledilo malo interesantnog programiranja stvaranjem aplikacije `Hello World!, Part II`. Nakon početnog igranja, počeli ste sa radom na osnovama jezika Object Pascal. Ovo poglavlje sadrži dosta materijala koje treba da upijete. Nemojte se osećati loše ukoliko ne zapamtite sve. Vratite se i pregledajte sve stvari koje su Vam ostale nejasne u toku današnjeg dana.

Radionica

Radionica sadrži kviz pitanja koja će Vam pomoći da učvrstite Vaše razumevanje obrađenog materijala i vežbe kojima stičete iskustvo u korišćenju onoga što ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A "Odgovori na kviz pitanja".

Pitanja i odgovori

P **Kakva je razlika između Win32 GUI aplikacije i Win32 aplikacije komandnog moda?**

O GUI aplikacija je tradicionalni Windows program. Obično sadrži traku sa naslovom, traku sa menijem i oblast prozora. Aplikacija komandnog moda je 32-bitna aplikacija koja radi u MS-DOS prozoru u okviru Windowsa. Aplikacija komandnog moda liči na DOS program.

P **Nakon što sam deklarisao konstante da li mogu menjati njihove vrednosti u svom programu?**

O Ne. Konstante su ono što im sugeriše i sam naziv: konstante. Ako želite da menjate vrednosti, trebalo bi da koristite promenljive a ne konstante.

P **Da li u okviru mojih junita treba da definišem i odeljak interface?**

O Da. (Ovaj odeljak treba da sadrže svi juniti izuzev izvornog koda projekta.) Odeljak interface će možda biti prazan, ali mora postojati.

P **Da li da koristim kratke stringove ili duge stringove u svojim Delphi aplikacijama?**

O Za većinu slučajeva možete koristiti duge stringove. Dugi stringovi prividno nemaju ograničenja u pogledu dužine i dodeljivanje memorije se obavlja automatski. Kratki stringovi mogu biti brži kada radite sa mnogobrojnim manipulacijama stringovima ali u većini slučajeva razlika u brzini nije velika.

P **Da li mogu dodeliti broj koji ima decimalna mesta celobrojnoj promenljivoj?**

O Ne. Ne možete dodeliti broj u pokretnom zarezu celobrojnoj promenljivoj.

P **Da li će Object Pascal sprečiti da upišem podatke preko postojećih u memoriji ukoliko pokušam da upišem podatke nakon kraja niza?**

O U većini slučajeva, da. Provera opsega u toku prevođenja će osigurati da ne pokušate upis van opsega niza.

Kviz

1. Koja je ekstenzija Pascalovog junita?
2. Koji je naziv ključne reči koja markira odeljak u kom se deklarišu promenljive?
3. Šta radi funkcija `IntToStr`?
4. Koja je svrha liste `uses` u Pascalovom junitu?
5. Da li su sledeće dve deklaracije različite? Objasnite zašto da, odnosno zašto ne?

```
var
  top : Integer;
  Top : Integer;
```

6. Kako grupišete Pascal stringove?
7. Kako možete ubaciti kontrolne karaktere u string?
8. Koja je maksimalna dužina kratkog stringa?
9. Pogledajte ovu liniju koda:

```
MyArray : array [0..10] of Byte;
```

   Koliko bajtova rezeviše ovaj niz?

10. Koji broj označava prvi element niza; 0 ili 1?

Vežbe

1. Napišite Windows program koji prikazuje reči "`Welcome to Delphi!`" u okviru prozora programa.
2. Ponovo napišite program iz koraka 1 i promenite tekst u "`Hello there!`" (savet: treba samo da promenite karakteristiku `Caption` komponente `Label`.)
3. Napišite program koji deklariše dve promenljive i dodeljuje vrednosti ovim promenljivama. Pomnožite dva broja i prikažite njihov rezultat na ekranu.
4. Napišite program koji dodeljuje string, "`Ovde ima tuce jaja.`" promenljivoj a zatim ubacuje string "`12`" na određeno mesto u okviru stringa. Prikažite rezultat u komponenti Label.
5. Napišite program koji kreirsa završni rezultujući string u primeru 4, ali formatiran funkcijom `format`.

Dan 2

Više o Pascalu

Ovo je veoma dobar početak za učenje jezika Object Pascal. U ovom poglavlju nastavičete sa učenjem jezika Object Pascal istražujući još više osnove ovog jezika. Danas čete učiti o:

- Ključnim rečima `if, then i else`
- Petljama: `for, while i repeat`
- Iskazu `case`
- Opsezima
- Slogovima
- Funkcijama i procedurama

`if, then, else`

Postoje neki aspekti programiranja koji su zajednički za sve programske jezike. Jedan od zajedničkih aspekata jezika Object Pascal i drugih programskih jezika je iskaz `if`. Iskaz `if` se koristi za proveru uslova i izvršavanje dela koda u zavisnosti od toga da li je uslov tačan `(true)` ili netačan `(False)`. Evo primera:

```
var
  X : Integer;
begin
  X := StrToInt(Edit1.Text);
  if X > 10 then
```

```
    Label1.Caption := 'You entered a number greater than 10.';
end
```

Ovaj kod hvata sadržaj kontrole za editovanje i smešta je u celobrojnu promenljivu. Ako je broj veći od 10, izraz `X>10` vraća `True` i prikazuje se poruka; u suprotnom ništa nije prikazano. Zapazite da kada je uslovni izraz tačan `(True)`, iskaz koji sledi odmah nakon izraza `if...then` će biti izvršen. Uslovni deo iskaza `if` se uvek nalazi iza ključne reči `then`.

Iskaz `if` se koristi za ispitivanje uslova i izvršava jednu, odnosno više linija koda ukoliko je iskaz tačan `(True)`.

Izvršavanje višestrukih instrukcija

Pretpostavimo da imate više linija koda koje treba da se izvrše ukoliko je uslovni izraz tačan. U tom slučaju treba da koristite ključne reči `begin` i `end`, kako bi ove linije smestili u blok:

```
if X > 10 then begin
  Label1.Caption := 'You entered a number greater than 10.';
  DoSomethingWithNumber(X);
end;
```

Ukoliko je uslovni izraz netačan, blok koda dodeljen izrazu `if` se ignoriše i izvršavanje programa se nastavlja počev od prvog iskaza koji sledi iza bloka koda.

Object Pascal sadrži nekoliko prečica. Jedna od ovih prečica se koristi za iskaze kada je vrednost Bulove promenljive `True`. Pogledajte u ovaj kod:

```
if FileGood then ReadData;
```

Ovaj metod je prečica za dužu formu koja je prikazana sledećom linijom:

```
if FileGood = True then ReadData;
```

Ova prečica važi samo za Bulove promenljive. Možete proveriti promenljivu da li je `False` korišćenjem ključne reči `not` ispred naziva promenljive:

```
var
  FileGood : Boolean;
begin
  FileGood := OpenSomeFile;
  if not FileGood then ReportError;
end;
```

Učenje prečica jezika Object Pascal pomaže u pisanju elegantnijeg koda. Poznavanje prečica će Vam takođe pomoći da shvatite Object Pascal programe koje čitate u primerima.

Dodavanje iskaza else

U nekim slučajevima možete poželeti da izvršite neku radnju kada je uslovni izraz tačan, odnosno drugu radnju kada je uslovni izraz netačan. Ovo možete ostvariti inplementacijom iskaza `else`:

```
if X = 20 then
  DoSomething(X)
else
  DoADifferentThing(X);
```

NOVI TERMIN Iskaz `else` se koristi u sprezi sa iskazom `if` i označava deo koda koji će biti izvršen ukoliko je iskaz `if` pogrešan `(False)`.

U ovom primeru jedna od dve funkcije će biti pozvana u zavisnosti od vrednosti promenljive `X`, ali ne i obe.

Želim da Vam napomenem nešto u vezi prethodnog primera. Linija koja sledi iza iskaza `if` se ne završava znakom tačka-zarez, pošto se kompletan deo koda `if ...then...else` posmatra kao jedinstven iskaz. Izostavićete znak tačka-zarez iz prve linije iskaza `if`, *samo* ukoliko postoji jedna linija koda (zato što ne koristite blok `begin...end` nakon iskaza `if`). Evo nekoliko primera ispravne `if ...the....else` sintakse:

```
if X = 20 then
  DoSomething(X)        { no semi-colon here because }
else                    { it's a single line of code }
  DoADifferentThing(X);

if X = 20 then begin
  DoSomething(X);       { semi-colon needed here because }
end else begin          { of the begin/end block }
  DoADifferentThing(X);
end;

if X = 20 then begin
  DoSomething(X);       { Multiple lines, use semi-colons }
  X := 200;             { at the end of each line. }
  Y := 30;
end else begin
  DoADifferentThing(X);
  X := 100;
  Y := 15;
end;
```

NAPOMENA Nije važno gde stavljate ključne reči then, begin i else. Sledeća dva bloka koda su potpuno identična po pitanju prevodioca:

```
{ One way to do it... }
if X = 20 then begin
  DoSomething(X);
```

```
  X := 200;
  Y := 30;
end else begin
  DoADifferentThing(X);
  X := 100;
  Y := 15;
end;

{ Same code, different layout... }
if X = 20
then
begin
  DoSomething(X);
  X := 200;
  Y := 30;
end
else
begin
  DoADifferentThing(X);
  X := 100;
  Y := 15;
end;
```

Na Vama je odluka koji ćete stil kodiranja koristiti. Sve dok je stil kodiranja stvar ukusa, koristite onaj stil koji čini Vaš kod jednostavnim za čitanje.

Zapamtite da je operator jednakosti predstavljen znakom jednakosti (=), a da je operator dodeljivanja dvotačka i jednako (:=). Najčešća greška prilikom kodiranja je korišćenje operatora dodeljivanja kada se želi korišćenje operatora jednakosti. Srećom, prevodilac će prijaviti grešku ukoliko ovo uradite.

Ugnježdeni iskazi `if`

Možete, ako je to potrebno, iskaze `if` ugnjezditi. Ugnježdavanje nije ništa drugo do postavljanje iskaza `if` iza drugog iskaza `if`.

```
if X > 10 then
  if X < 20 then
    Label1.Caption := 'X is between 10 and 20';
```

Zapamtite da su ovo pojednostavljeni primeri. U normalnom radu, možete se izgubiti u lavirintu `begin` i `end` iskaza koji odvajaju jedan blok koda od drugog. Primera radi, pogledajte ovaj isečak koda:

```
if X > 100 then begin
  Y := 20;
  if X > 200 then begin
    Y := 40;
    if X > 400 then begin
      Y := 60;
```

```
      DoSomething(Y);
    end;
  end;
end else if X < -100 then begin
  Y := -20;
  if X < -200 then begin
    Y := -40;
    if X < -400 then begin
      Y := -60;
      DoSomething(Y);
    end;
  end;
end;
```

Čak i ovo je prilično jednostavan primer, ali Vam je sigurno dao ideju.

Kada deo koda sadrži više od dva ili tri uzastopna iskaza if koji proveravaju različite vrednosti iste promenljive, ovaj deo koda može biti kandidat za iskaz case. Iskaz case će biti obrađen kasnije u ovom poglavlju u delu "Iskaz case".

Do sada sam koristio samo jedan uslovni izraz u `if` primerima. Kada imate samo jedan uslovni izraz, možete, a ne morate da koristite zagrade u okviru kojih se izraz upisuje. Ukoliko imate više od jednog uslovnog izraza, morate svaki izraz posebno staviti u zagrade. Na primer:

```
if (X = 20) and (Y = 50) then
  DoSomething;
```

Ako zaboravite zagrade, prevodilac će Vas, naravno, obavestiti o grešci.

Iskaz `if` se veoma često upotrebljava u programiranju na jeziku Object Pascal. Ovaj iskaz je jednoznačan, pa ne bi trebalo da imate problema sa njim. Najvažnije je da ispravno koristite blokove `begin/end`.

Iskaz `if...then...else`, forma 1


```
if cond_expr then
     true_statement
    else
        false_statement;
```

Ukoliko je uslovni izraz `cond_expr` tačan, biće izvršena linija koda predstavljena sa `true_statement`. Ukoliko je definisana klauzula else, linija koda koju predstavlja `false_statement` će biti izvršena ukoliko je uslovni izraz `cond_expr` netačan.

Iskaz `if...then....else`, forma 2

```
if cond_expr_1 then begin
    true_statements;
end else begin
    false_statements;
end;
```

Ukoliko je uslovni izraz `cond_expr_1` tačan, blok koda predstavljen izrazom `true_statements` će biti izvršen. Ukoliko je netačan, blok koda predstavljen izrazom `false_statements` će biti izvršen.

Korišćenje petlji

Petlja je uobičajen element u svim programskim jezicima. Može se koristiti za iteraciju kroz niz, da izvrši određene aktivnosti navedeni broj puta, da čita datoteku sa diska... Mogučnosti su neograničene. U ovom poglavlju ću objasniti petlju for, `while i repeat`. U večini slučajeva ove petlje rade na veoma sličan način. Sve petlje sadrže sledeće zajedničke elemente:

- Početak
- Telo petlje, koje se obično nalazi između ključnih reči begin i end u okviru kojih se nalaze iskazi koji se izvršavaju u svakom prolazu.
- Završetak.
- Provera uslova koji određuje da li treba da se završi petlja.
- Opciono sadrže procedure `Break` i `Continue`.

NOVI TERMIN *Petlja* je element programskog jezika koji se koristi za višestruko izvršavanje akcije, sve dok je zadovoljen uslov.

Početna tačka petlje je predstavljena iskazom za petlju jezika Object Pascal (`for`, `while`, ili `repeat`). Telo petlje sadrži iskaze koji se izvršavaju prilikom svakog prolaza kroz petlju. Telo petlje može sadržati bilo koji ispravan kod jezika Object Pascal i može biti napisan kao jedna linija odnosno kao blok koda. Ukoliko telo petlje sadrži više linija koda, kod mora biti označen iskazima `begin i end` (u slučaju petlje `for` i petlje `while`) odnosno ključne reči `until` (u slučaju petlje `repeat`). Kada je telo petlje jedna jedina linija koda, nije potrebno koristiti ključne reči `begin` i `end`.

Večina petlji radi na približno ovaj način: nakon ulaska u petlju izračunava se uslov za proveru. Ako je uslov netačan, izvršava se telo petlje. Kada se dostigne kraj petlje u toku izvršavanja programa, izvršavanje se prebacuje na početak petlje i ponovo se izračunava uslov. Ukoliko je uslov još uvek netačan, ceo proces se ponavlja. Ukoliko je uslov tačan, izvršavanje programa se prenosi na liniju koda koji se nalazi iza bloka petlje. Izuzetak je petlja `repeat`, koja testira uslov na kraju petlje, a ne na početku.

Uslov petlje pokazuje kada treba prestati sa izvršavanjem petlje. Efektivno, uslov petlje može da glasi: "Nastavi da radiš ovo, sve dok je X jednako 10", odnosno "Nastavi da čitaš datoteku, sve dok ne stigneš do kraja datoteke." Nakon ulaska u petlju izvršava se telo petlje sve dok uslov petlje ne bude tačan.

Veoma lako Vam se može dogoditi da greškom napišete petlju u kojoj uslov nikada ne postaje tačan. Kao rezultat program će biti blokiran ili će "visiti". Jedini način da izađete iz ove situacije je da pritisnete Ctrl+Alt+Del i na taj način prekinete izvršavanje programa. Okvir za zatvaranje programa u okviru Windowsa (odnosno Task Manager u okviru Windows NT) će biti prikazan a pored Vašeg programa će stajati natpis da ne radi (`Not Responding`). Odaberite Vaš program u okviru liste i kliknite na dugme End Task (Završi posao) da biste prekinuli izvrašvanje programa.

U Delfiju je uobičajeno da pokrećete program korišćenjem dugmeta Run u okviru trake za alate, odnosno pritiskom na taster F9. Ukoliko treba da prekinete rad programa koji je pokrenut u okviru IDE, možete odabrati opciju RunÐProgram Reset iz glavnog menija odnosno pritisnuti tastere Ctrl+F2. Zapamtite takođe da Windows 95 ne podnosi prekidanje programa opcijom Program Reset, pa postoji mogućnost da krahira ukoliko više puta resetujete program (Windows NT je ovde tolerantniji). Uvek izvršavajte Vaše programe do kraja ukoliko je to moguće, naročito ako program razvijate na Windows 95 platformama.

Nakon opšteg pregleda, pogledajmo zasebno svaku od navedenih petlji.

Petlja `for`

Petlja `for` se verovatno najčešće koristi. Ima dva parametra: početnu vrednost i krajnju vrednost. Ukoliko se uvećava vrednost u okviru petlje koristi se ključna reč to. Ukoliko se vrednost u okviru petlje umanjuje koristi se ključna reč `downto`.

Petlja `for`, iskaz se uvećava


```
for initial_value to end_value do begin
    statements;
end;
```

Petlja `for` ponavlja izvršavanje bloka koda pod nazivom `statements`, sve dok ne dostigne vrednost `end_value`. Stanje petlje se inicijalizuje iskazom `initial_value`. Promenljiva naznačena kao `initial_value` se uvećava prilikom svakog prolaska kroz petlju. Ukoliko u okviru petlje postoji samo jedan iskaz, nije potrebno dodavati iskaze `begin i end`.

Petlja `for`, iskaz se umanjuje

```
for initial_value downto end_value do begin
    statements;
end;
```

Petlja `for` ponavlja izvršavanje bloka koda pod nazivom `statements` sve dok se ne dostigne vrednost `end_value`. Stanje petlje se inicijalizuje iskazom `inital_value`. Promenljiva naznačena kao `initial_value` se umanjuje prilikom svakog prolaska kroz petlju. Ukoliko u okviru petlje postoji samo jedan iskaz, nije potrebno dodavati iskaze `begin i end`.

Ukoliko su Vam sintaksni iskazi pomalo nejasni, verovatno će Vam pomoći primeri. Prvo pogledajte tipičnu petlju `for` koja uvećava iskaz:

```
var
  I : Integer;
begin
  for I := 0 to 9 do begin
    Memo1.Lines.Add('This is iteration ' + IntToStr(I));
  end;
end;
```

Ovaj kod rezultuje desetostrukim izvršavanjem iskaza u okviru zagrade. Prvi parametar, `I:= 0`, definiše da petlja `for` počne sa radom od vrednosti `0`. Drugi parametar, `9`, označava da se petlja izvršava sve dok promenljiva `I` ne dostigne vrednost `9`. Ključna reč to definiše da će vrednost promenljive `I` biti uvećana prilikom svakog prolaska kroz petlju.

Korišćenje naziva promenljive `I` ima svoje korene u jeziku FORTRAN i njegovih tradicionalnih petlji `for`. Prirodno, bilo koji naziv promenljive se može koristiti, ali ćete primetiti da se najčešće za petlje `for` upotrebljava promenljiva I.

Pogledajmo jednu varijaciju ovog koda. Sledeći isečak koda će postići suprotan efekat u odnosu na prethodni primer:

```
var
  I : Integer;
begin
  for I := 9 downto 0 do begin
    Memo1.Lines.Add('This is iteration ' + IntToStr(I));
  end;
end;
```

U ovom slučaju počinjemo od broja `9` i zaustavljamo se kada promenljiva `I` dostigne vrednost `0`. Ključna reč downto određuje da će vrednost promenljive `I` biti umanjena prilikom svakog prolaska kroz petlju. Ovo je primer petlje koja računa unazad (umanjuje vrednost).

U prethodnim primerima, nije izričito potrebno koristiti ključne reči `begin` i `end`. Ukoliko se ne koriste ključne reči `begin` i `end`, izraz koji se nalazi iza iskaza `for` se smatra telom petlje. Na Vama je da odlučite, da li da koristite begin i end ključne reči u slučaju kada postoji jedan jedini iskaz, iako se ovo smatra lošim načinom rada.

Primer petlje for

Napišimo kratak program koji ilustruje korišćenje petlje `for`. U toku rada ću objasniti druge Defli komponente, komponentu `Memo` (korišćena je u prethodnom delu ovog poglavlja). Izvršite sledeće korake;

1. Otvorite novu aplikaciju (File➥New Application).
2. Postavite dugme na formu.
3. Pronađite komponentu `Memo` u delu Standard palete komponenti (trebala bi da bude levo od komponente `Button`). Kliknite na dugme, a zatim kliknite na formu. Na taj način ste postavili `Memo` polje na Vašu formu.
4. Uvećajte Memo komponentu povlačenjem crnog pravougaonika za promenu veličine u donjem desnom uglu komponente.
5. Kliknite dva puta na dugme da biste kreirali `OnClick` upravljač događajima za dugme. Upišite sledeći kod, tako da upravljač događajima izgleda ovako:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
begin

  Memo1.Lines.Clear;

  for I := 0 to 5 do
    Memo1.Lines.Add('This is iteration ' + IntToStr(I));

  Memo1.Lines.Add('');

  for I := 5 downto 0 do
    Memo1.Lines.Add('This is iteration ' + IntToStr(I));

end;
```

Pokrenite program. Kada kliknete na dugme, u memo polje će biti dodat tekst. Slika 2.1 prikazuje ovaj program u toku rada .

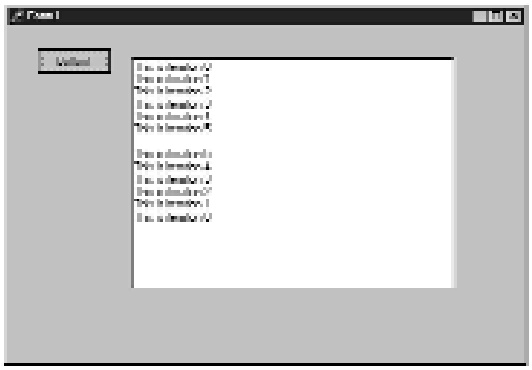
Kao što sam naveo ranije, promenljiva u petlji će biti uvečana prilikom svakog prolaska kroz petlju. Za razliku od drugih programskih jezika, Pascal ne pruža mogućnost iteracije kroz `for` petlju sa vrednostima koje su različite od jedan. Na primer, nije moguće koristiti petlju u opegu od 0 do 100 u koracima od 10. Da bi ovo postigli, morate koristiti druge promenljive. Evo primera:

```
var
  I : Integer;
  X : Integer;
begin
  X := 0;
```

```
  Memo1.Lines.Clear;

  for I := 0 to 9 do begin
    Memo1.Lines.Add('Iteration value: ' + IntToStr(X));
    Inc(X, 10);
  end;
end;
```

Ovaj kod će u memo polju ispisati sledeće:

```
Iteration value: 0
Iteration value: 10
Iteration value: 20
Iteration value: 30
Iteration value: 40
Iteration value: 50
Iteration value: 60
Iteration value: 70
Iteration value: 80
Iteration value: 90
```



Slika 2.1
Izlazni ekran vežbe sa petljom `for`

Pogledajte korišćenje funkcije Inc u prethodnom isečku koda. Ova funkcija uvećava datu promenljivu (x u ovom primeru) za navedenu vrednost (`10`). Funkcija `Inc` se koristi bez parametra za inkrementiranje ukoliko se vrednost promenljive uvećava za jedan. Na primer:

```
Inc(X); {X se uve~ava za 1. Isto kao X:=X+1}
```

Funkcija `Inc` ima svoj par, čiji je naziv, naravno, `Dec`. Evo primera funkcije `Dec`:

```
Dec(X); {X se umanjuje za 1}
Dec(X,10); {X se umanjuje za 10}
```

Korišćenje funkcija `Inc` i `Dec` je bolje od korišćenja dužih verzija (`X:=X+1`, na primer).

Funkcije Pred i Succ

Često ćete primetiti da se u okviru for petlji koriste funckije `Pred` i `Succ`. Funkcija Pred vraća prethodnika prosleđenog argumenta. Primera radi, `Pred(10)` će vratiti vrednost 9, `Pred(100)` će vratiti vrednost 99, i tako dalje. Na osnovu ovih informacija, sledeće tri petlje `for` su identične:

```
var
  X : Integer;
begin
  X := 10;

  for I := 0 to 9 do
    DoSomething;

  for I := 0 to X - 1 do
    DoSomething;

  for I := 0 to Pred(X) do
    DoSomething;
end;
```

Kada počinjete sa inicijalnom vrednošću 0, normalno je da napravite grešku kojom ćete izvršiti iteraciju više u okviru petlji. Korišćenjem funkcije `Pred`, rešavate ovaj problem na elegantniji način, nego da koristite `X-1`. Funkcija `Succ` prirodno vraća naslednika argumenta koji je prosleđen. Ovo je korisno kada se broji unazad:

```
for I := 100 downto Succ(X) do
    DoSomething;
```

Sada kada ste videli primere za petlju `for`, neće Vam biti teško da primenite iste koncepte u petljama `while` i `repeat`. Pogledajmo sad i njih.

Petlja while

Petlja `while` se razlikuje od petlje `for` po tome što sadrži uslov za ispitivanje koji se proverava na početku svake iteracije. Sve dok je uslov za ispitivanje tačan, izvršava se petlja.

```
var
  X : Integer;
begin
  X := 0;
  while X < 1000 do begin
    X := DoSomeCalculation;
    DoSomeMoreStuff;
  end;
  { ...more code }
end;
```

U ovom primeru pozivam funkciju za koju pretpostavljam da će vratiti vrednost veću, ili jednaku 1.000. Sve dok je vrednost koju vraća funkcija manja od 1.000. petlja while nastavlja sa radom. Kada promenljiva `X` dobije vrednost veću ili jednaku 1.000, uslov za ispitivanje postaje netačan i izvršavanje programa se prebacuje na prvu liniju, nakon tela petlje `while`. Uobičajena i inplementacija petlje `while` koristi promenljivu za ispitivanje uslova tipa `Boolean`. Stanje promenljive za ispitivanje uslova može biti definisano u okviru tela petlje:

```
var
  Done : Boolean;
begin
  Done := False;
  while not Done do begin
    DoSomeStuff;
    Done := SomeFunctionReturningABoolean;
    DoSomeMoreStuff;
  end;
end;
```

Od određene tačke očekuje se da promenljiva `Done` dobije vrednost `True`, pa da se petlja završi. Uradimo još jedan jednostavan primer programa koji ilustruje korišćenje petlje `while`. Pokrenite novu aplikaciju i postavite dugme i komponentu `Memo` na formu. Dva puta kliknite na dugme i izmenite upravljač događajima da izgleda ovako:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
begin
  I := 5;
  Memo1.Lines.Clear;
  while I > -1 do begin
    Memo1.Lines.Add('Today I have ' +
      IntToStr(I) + ' problems to worry about.');
    Dec(I);
  end;
  Memo1.Lines.Add('Yeah!');
end;
```

Kada pokrenete program i kliknete na dugme forme videćete sledeći tekst u okviru komponente `Memo`:

```
Today I have 5 problems to worry about.
Today I have 4 problems to worry about.
Today I have 3 problems to worry about.
Today I have 2 problems to worry about.
Today I have 1 problems to worry about.
Today I have 0 problems to worry about.
Yeah!
```

Ovaj program deklariše promenljivu `I` i inicijalno joj dodeljuje vredost 5. Nakon toga pokreće se petlja `while`. Test se dodaje u komponentu `Memo` u svakom prolazu kroz petlju, dok se promenljiva `I` umanjuje za jedan. Kada `I` dobije vrednost -1, petlja staje i dodaje se poslednja linija u komponentu `Memo`.

Iskaz petlje `while`

SINTAKSA

```
while cond_expr do begin
    statements;
end;
```

Petlja `while` ponavlja izvršavanje bloka koda naznačenog delom `statements` sve dok je uslovni izraz `cond_expr` tačan. Stanje petlje se mora inicijalizovati pre iskaza `while`, a izmena stanja se eksplicitno mora nalaziti u okviru bloka koda. Kada uslovni izraz `cond_expr` postane netačan, prekida se izvršavanje petlje. Ukoliko se telo petlje sastoji od jednog jedinog iskaza, nije potrebno dodavati iskaze `begin i end`.

Petlja `repeat`

Petlja `repeat` je skoro identična petlji `while`. Razlika između ove dve petlje je takođe važna. Kao što ste mogli da vidite u poslednjem primeru, petlja `while` proverava uslovni iskaz na početku petlje. U slučaju petlje `repeat`, uslovni iskaz se proverava na kraju petlje. Prethodni primer je prerađen tako što se sada koristi petlja `repeat` umesto petlje `while`:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
begin
  I := 5;
  Memo1.Clear;
  repeat
    Memo1.Lines.Add('Today I have ' +
      IntToStr(I) + ' problems to worry about.');
    Dec(I);
  until I = -1;
  Memo1.Lines.Add('Yeah!');
end;
```

Ovaj kod kao rezultat prikazuje isti tekst u okviru `Memo` komponente kao u prethodnom primeru. Zapazite da korišćenje ključnih reči `begin` i `end` nije neophodno, pošto ključna reč `repeat` označava početak bloka koda, a ključna reč `until` označava kraj bloka koda. Da li ćete koristiti petlju `while` ili petlju `repeat` zavisi od toga šta sama petlja radi.

Iskaz petlje repeat

SINTAKSA

```
while cond_expr do begin
    statements;
end;
```

Petlja `repeat` ponavlja izvršavanje bloka koda označenog kao `statements`, sve dok je uslovni izraz `cond_expr` netačan. Stanje petlje se mora inicijalizovati pre iskaza `repeat`, a izmena stanja se eksplicitno mora nalaziti u okviru bloka koda. Kada uslovni izraz `cond_expr` postane tačan, prekida se izvršavanje petlje.

NAPOMENA Zbog načina na koji petlja repeat radi, kod u okviru tela petlje će biti izvršen još jednom, najmanje, bez obzira na uslov petlje (pošto se uslov izračunava na kraju petlje). U slučaju petlje `while`, uslov petlje se izvršava na početku petlje, pa se telo petlje možda nikad neće izvršiti.

Iskaz goto

Spominjem iskaz goto samo da bi znali da postoji. Iskaz `goto` Vam omogućava da prebacite izvršavanje programa na oznaku koju ste prethodno deklarisali ključnom reči `label`. Oznaka se u kod upisuje tako što se iza naziva oznake postavlja znak dvotačka (:). Sledeći isečak koda ovo ilustruje:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  I: Integer;
label
  MyLabel;
begin
  Memo1.Clear;
  I := 0;
MyLabel:
  Inc(I);
  Memo1.Lines.Add(IntToStr(I));
  if I < 5 then
    goto MyLabel;
end;
```

U ovom slučaju nije neophodno da koristite oznake bloka `begin` i `end`, pošto će biti izvršene sve linije koda između iskaza `goto` i oznake.

NAPOMENA Iskaz `goto` se smatra kao loš način pisanja programa u jeziku Object Pascal. Sve što možete postići iskazom `goto`, možete postići i petljama `while`, odnosno `repeat`. Samo mali broj programera na jeziku Object Pascal koji drže do sebe koriste iskaz goto u svojim programima. Ako prelazite na jezik Object Pascal sa nekog drugog jezika koji koristi iskaze `goto`, primetićete da osnovna struktura jezika Object Pascal čini nepotrebnim korišćenje iskaza `goto`.

Iskaz goto

SINTAKSA

```
label
    label_name;

    goto label_name
    .
    .
    .
label_name:
```

Iskaz `goto` bezuslovno prebacuje izvršavanje programa na oznaku predstavljenu kao `label_name`.

Procedure Continue i Break

Pre nego što završimo sa pričom o petljama, treba da upoznate dve procedure koje pomažu u kontroli izvrašvanja programa u okviru petlji. Procedura `Continue` se koristi za bezuslovno preskakanje programskih linija do kraja petlje. Na primer, možda ćete imati deo petlje koji ne želite da izvršite ukoliko određena promenljiva vrati vrednost `True`. U tom slučaju treba da koristite proceduru `Continue`, da biste izbegli izvršavanje bilo kog koda koji se nalazi iza određene tačke.

```
var
  Done  : Boolean;
  Error : Boolean;
begin
  Done := False;
  while not Done do begin
    { some code }
    Error := SomeFunction;
    if Error then Continue;  { jumps to the bottom of the loop }
    { other code that will execute only if no error occurred }
  end;
end;
```

Procedura `Break` se koristi da prekine izvršavanje petlje pre nego što se dostigne definisani uslov. Na primer, možda ćete pretraživati niz celih brojeva tražeći određeni broj. Prekid izvršavanja petlje za traženje broja će se dogoditi kada traženi broj bude pronađen, pa možete proslediti indeks elementa niza gde se broj nalazi:

```
var
  MyArray      : array [0..99] of Integer;
  Index        : Integer;
  SearchNumber : Integer;
  I            : Integer;
begin
  FillArray; { procedure which fills the array }
  Index := -1;
```

```
  SearchNumber := 50;
  for I := 0 to High(MyArray) do begin
    if MyArray[I] = SearchNumber then begin
      Index := I;
      Break;
    end;
  end;
  if Index <> -1 then
    Label1.Caption := 'Number found at index ' + IntToStr(Index)
  else
    Label1.Caption := 'Number not found in array.';
end;
```

Procedure `Continue` i `Break` se koriste samo u okviru petlji `for`, `while` i `repeat`. Ako pokušate da koristite ove procedure van petlje, prevodilac će prijaviti grešku u prevođenju koja glasi: "`BREAK or CONTINUE outside of loop.`" (BREAK ili CONTINUE van petlje).

Postoji mnogo situacija u kojima su procedure `Continue` i `Break` korisne. Kao što je to slučaj i sa prethodnim stvarima koje sam naveo, biće Vam potrebno iskustvo u programiranju na jeziku Object Pascal pre nego što otkrijete sve moguće načine da koristite ove dve procedure.

Iskaz case

Iskaz `case` može biti smatran kao prošireni iskaz `if`. Omogućava Vam da izvršite jedan od mnogo blokova koda baziranih na rezultatu izraza. Iskaz može biti promenljiva, rezultat poziva funkcije, odnosno bilo koji odgovarajući kod jezika Object Pascal koji izračunava izraz. Evo primera za iskaz `case`:

```
case AmountOverSpeedLimit of
   0  : Fine := 0;
   10 : Fine := 20;
   15 : Fine := 50;
   20,
   25,
   30 : Fine := AmountOverSpeedLimit * 10;
  else begin
    Fine := GoToCourt;
    JailTime := GetSentence;
  end;
end;
```

Iskaz `case` čini nekoliko delova. Prvo što možete uočiti je izraz koji u prethodnom primeru predstavlja promenljiva `AmountOverSpeedLimit` (zapamtite, upozorio sam Vas na promenljive sa dugim imenima!). Zatim, iskaz `case` ispituje da li je tačna jednakost izraza. Ukoliko je promenljiva `AmountOverSpeedLimit` jednaka `0 (0 :)`, vrednost `0` se dodeljuje promenljivoj `Fine`. Ukoliko je promenljiva

`AmountOverSpeedLimit` jednaka `10`, promenljivoj Fine se dodeljuje vrednost `20`, i tako dalje. U svakom od prva tri slučaja vrednost dodeljena promenljivoj `Fine` i izvršavanje koda se prebacuje na iskaz `case`, što znači da slučaj odgovara izrazu koji je pronađen, a ostatak iskaza `case` može biti ignorisan.

Zapazite da slučajevi 20 i 25 u nastavku imaju zarez, ali ne i iskaze. Ukoliko se za izraz `AmountOverSpeedLimit` dobije vrednost `20`, ili `25`, ovi slučajevi će propustiti izvršavanje programa na sledeći blok koda, kako bi isti bio izvršen. U ovoj situaciji, vrednosti `20, 25` ili `30` kao rezultat daju izvršavanje istog bloka koda.

Na kraju ćete primetiti iskaz `else`. Kod blok koji se nalazi iza iskaza `else` će biti izvršen, ukoliko se ne pronađe odgovarajući slučaj. Uključivanje iskaza `else` nije obavezno. Iskaz `case` možete pisati bez iskaza `else`:

```
case X of
  10 : DoSomething;
  20 : DoAnotherThing;
  30 : TakeABreak;
end;
```

Kao što sam napomenuo ranije, možda ćete poželeti da koristite iskaz `case` ukoliko imate nekoliko povezanih iskaza `if`. Iskaz `case` je bitno jasniji za čitanje ukoliko neko drugi pregleda Vaš program.

NAPOMENA Tip izraza u iskazu `case` mora pripadati nekom od standardnih tipova jezika Object Pascal (`Integer, Word, Byte, itd.`) . Sledeći primer iskaza case nije dozvoljen:

```
case SomeStringVariable of
  'One' : { code }
  'Two' : { code }
end;
```

String vrednosti nisu dozvoljene, isto kao i vrednosti u pokretnom zarezu.

Iskaz case

SINTAKSA

```
case expr of
    value_1: statements_1;
    value_2: statements_2;
    .
    .
    .
    value_n: statements_n;
else
    else_statements;
end;
```

Iskaz `case` omogućava izvrašvanje različitih blokova koda u zavisnosti od različitih vrednosti izraza `expr`. Blok koda predstavljen izrazom `statements_1` se izvršava kada je izraz `expr` jednak vrednosti `value_1`, blok koda predstavljen izrazom

`statements_2` kada je izraz `expr` jednak vrednosti `value_2` i tako dalje kroz blok koda, do dela predstavljenog iskazom `statements_n` za izraz `expr` koji je jednak vrednosti `value_n`. Ukoliko izraz `expr` nije jednak nijednoj vrednosti od `value_1` do `value_n` blok koda definisan izrazom `else_statements` će biti izvršen. Iskaz `else` nije obavezan.

Opseg (scope)

Pojam opseg (scope) označava vidljivost promenljive u različitim delovima programa. Većina promenljivih ima lokalni opseg *(local scope)*, što znači da je promenljiva vidljiva samo u okviru bloka koda u kom je deklarisana. Pogledajte program u listingu 2.1. (Ovo je prvi pogled na kompletan junit koji Delphi generiše. Ovde postoji deo koda sa kojim se do sada niste sretali, a objasniću ga tokom vremena, pa za početak treba da ignorišete delove koji Vam nisu poznati.)

NOVI TERMIN Pojam opseg *(scope)* se odnosi na vidljivost promenljivih u okviru različith delova Vašeg programa.

Listing 2.1: SCOPEU.PAS

```
01: unit ScopeU;
02:
03: interface
04:
05: uses
06:   Windows, Messages, SysUtils, Classes, Graphics, Controls,
      Forms,➥·Dialogs,
07:   StdCtrls;
08:
09: type
10:   TForm1 = class(TForm)
11:     Button1: TButton;
12:     Memo1: TMemo;
13:     procedure Button1Click(Sender: TObject);
14:     procedure FormCreate(Sender: TObject);
15:   private
16:     { Private declarations }
17:   public
18:     { Public declarations }
19:   end;
20:
21: var
22:   Form1: TForm1;
23:
24: implementation
25:
26: var
27:   X : Integer;
```

```
28
29: {$R *.DFM}
30:
31: procedure TForm1.Button1Click(Sender: TObject);
32: var
33:   X : Integer;
34:
35:   procedure Test;
36:   var
37:     X : Integer;
38:   begin
39:     X := 300;
40:     { This X variable has a value of 300. }
41:     Memo1.Lines.Add('Local Function X: ' + IntToStr(X));
42:   end;
43:
44: begin
45:   X := 100;
46:   Memo1.Lines.Clear;
47:   { Local X has a value of 100. }
48:   Memo1.Lines.Add('Local X: ' + IntToStr(X));
49:   { Unit scope X has a value of 200. }
50:   Memo1.Lines.Add('Global X: ' + IntToStr(ScopeU.X));
51:   { Call the Test procedure. }
52:   Test;
53: end;
54:
55: procedure TForm1.FormCreate(Sender: TObject);
56: begin
57:   { Initialize the unit variable X. }
58:   X := 200;
59: end;
60:
61: end.
```

Prva stvar koju ćete možda zapaziti (ukoliko ste još uvek budni) je da je promenljiva `X` deklarisana tri puta u okviru programa. Ona je deklarisana u liniji 27 u okviru odeljka `implementation`, zatim u liniji 33 u okviru metoda `Button1Click` i u liniji 37 u okviru lokalne procedure pod nazivom `Test`. Ukoliko greškom deklarišete promenljivu više puta, prevodilac izbacuje grešku koja glasi: `Identifer redeclared: 'X'`, (identifikator ponovo deklarisan: `'X'`) i prevođenje se zaustavlja. Ipak ovaj program se uredno prevodi i izvršava. Zašto? Pošto svaka promenljiva `X` u listingu 2.1 ima različit opseg.

Pažljivije pogledajte Listing 2.1. Deklaracija za `X` u okviru linije 37 se nalazi unutar procedure `Test`, pa promenljiva ima lokalni opseg u okviru blok koda. (Shvatam da nisam objasnio lokalne funkcije i procedure, pa sada po malo istrčavam. Imajte strpljenja sa mnom; lokalne funkcije sam objasnio u jednom od narednih poglavlja "Lokalne funkcije i procedure".) Efektivno promenljiva `X` deklarisana u okviru linije 37

ne postoji van procedure `Test`. Promenljiva ima lokalni opseg. Slično je i sa deklaracijom promenljive `X` u okviru linije 33, koja ima lokalni opseg u okviru metoda `Button1Click` i ne postoji izvan funkcije.

Sada pogledajte promenljivu `X` deklarisanu u odeljku implentation. Promenljiva je vidljiva u okviru junita. Razmislite o ovome na trenutak. U jednom trenutku unutar procedure `Button1Click` postoje dve promenljive sa nazivom `X` (jedna je deklarisana u odeljku implementation, a druga je deklarisana u metodi `Button1Click`), a obe su u opsegu. Šta će se dogoditi ako želite da pristupite promenljivoj `X` iz procedure `Button1Click`, a ne lokalnoj promenljivoj `X`? Treba da *kvalifikujete* promenljivu. Linija 50 Listinga 2.1 izgleda ovako:

```
Memo1.Lines.Add('Global X: ' + IntToStr(ScopeU.X));
```

Kao što možete da vidite promenljiva `X` je kvalifikovana nazivom junita (`ScopeU`) iza kog sledi operator tačka. Kvalifikovanjem promenljive preko naziva junita poručujete: "Daj mi promenljivu `X` junita, a ne lokalnu promenljivu `X`." (Operator tačka se takođe koristi kod slogova i klasa, ali će o tome biti više reči kasnije, kada budem objašnjavao klase.)

Kao što sam napomenuo, kada se deklariše promenljiva junita `X` u okviru odeljka `implementation`, promenljiva dobija opseg junita. Ako želite da promenljiva bude dostupna ostalim junitima u okviru projekta, trebalo bi da deklarišete promenljivu u odeljku `interface` (promenljiva `Form1` u Listingu 2.1 je deklarisana na ovaj način). Promenljiva deklarisana u odeljku `interface` je dostupna iz bilo kog drugog junita u okviru projekta. Promenljiva deklarisana na ovaj način se obično naziva globalna promenljiva *(global variable)*. Da biste pristupili promenljivoj koja je deklarisana u odeljku `interface` tekućeg junita, potrebno je da dodate naziv junita u listu `uses`. Zatim možete pristupiti promenljivoj kao bilo kojoj drugoj. Ukoliko bilo koji junit u okviru projekta ima promenljive sa istim nazivom, promenljiva mora biti kvalifikovana nazivom junita, kao što je to opisano ranije.

Upravo sam rekao da promenljiva deklarisana u `interface` odeljku junita obično biva definisana kao globalna promenljiva. Ovo nije u potpunosti tačno, pošto promenljivu ne mogu automatski koristiti drugi juniti u okviru projekta - treba da dodate junit koji sadrži promenljive na listu uses u okviru junita iz kog želite da koristite promenljivu. Prava globalna promenljiva je promenljiva koju možete koristiti iz bilo kog junita u programu, bez potrebe da upisujete junit koji sadrži promenljive na listu `uses`. U Delphiju postoji nekoliko globalnih promenljivih koje postavlja startni kod prevodioca. Prave globalne promenljive niste u mogućnosti da definišete.

Slogovi

Slog je skup međusobno povezanih podataka postavljenih u jedinicu za smeštanje podataka. Na primer, pretpostavimo da želite da sačuvate listu pošte. Bilo bi pogodno da koristite jednu jedinu promenljivu koja sadrži sva polja potrebna za karakterističnu listu pošte. Slog Vam omogućava da to uradite. Prvo deklarišete slog, a

zatim kreirate slučaj sloga, kada želite da navedeni slog koristite. Slog se deklariše ključnom reči `record`:

```
MailingListRecord = record
  FirstName : string;
  LastName : string;
  Address : string;
  City : string;
  State : string;
  Zip : Integer;
end;
```

Elementi sloga se nazivaju polja *(field)*. Zapazite da svako polje mora biti deklarisano kao promenljiva u okviru blok koda. Ovaj primer sloga sadrži pet string polja i jedno celobrojno polje. (Izvinjavam se svojim prijateljima u svetu ukoliko ovo liči na listu pošte koja naginje standardu SAD.) Polje za zip kod/poštanski kod je moglo biti definisano kao string, takođe, ali sam želeo da vam pokažem da u okviru sloga može biti više različitih tipova podataka.

NOVI TERMIN Slog *(record)* je zbirka međusobno povezanih podataka koji su identifikovani kao jedinstvena jedinica za skladištenje podataka. Nakon što se deklariše slog, kreira se slučaj sloga za korišćenje. Deo sloga se naziva polje *(field)*.

NAPOMENA Primer sloga koji je naveden odgovara stvarima koje u ovoj knjizi objašnjavam, ali nije idealan za čuvanje podataka u datoteci. Kada smeštate slogove u datoteke, svaki slog mora imati istu dužinu u bajtovima. Pošto slog u ovom primeru koristi duge stringove kao polja, ne postoji način koji će garantovati da će svi slogovi biti iste dužine. Kada kreirate slogove koji će biti smešteni u datoteku, trebalo bi da koristite kratke stringove, ili pak nizove karaktera, umesto dugih stringova. O ovome će više biti reči u sutrašnjoj lekciji kada ću objasniti čitanje i pisanje u datoteku u okviru poglavlja "Rad sa binarnim podacima".

Nakon što ste deklarisali slog možete početi sa njegovim korišćenjem. Prvo što je potrebno da uradite, je kreiranje slučaja sloga. Evo kako to izgleda:

```
var
  MLRecord : TMailingListRecord;
```

Ovaj iskaz rezerviše memoriju za slog i dodeljuje memoriju promenljivoj pod nazivom `Record`. Sada kada postoji slučaj sloga, mogu da dodeljujem vrednosti poljima:

```
MLRecord.FirstName := 'Bruce';
MLRecord.LastName  := 'Reisdorph';
MLRecord.Address   := '123 Inspiration Pt.';
MLRecord.City      := 'Merced';
MLRecord.State     := 'CA';
MLRecord.Zip       := 99999;
```

Ovaj isečak koda sadrži sintaksu koju još niste upoznali (mada je veoma slična ranijim primerima u kojima sam objašnjavao kvalifikovanje promenljivih). Da bi pristupili

polju sloga, potrebno je da uposlite operator izbora elementa strukture *(structure memeber selector)*, koji se uobičajeno naziva operator tačka. Operator tačka je ustvari tačka koja se nalazi između naziva promenljive i naziva polja. Ako zaboravite da dodate operator pripadanja slogu, verovatno će se prevodilac požaliti na nedefinisan simbol. Operator elementa sloga Vam omogućava da pristupite određenom elementu sloga - bilo da čitate vrednost polja, ili da ga menjate. Evo primera koji postavlja sadržaj određenog polja u slog u okviru natpisa na formi:

```
Label1.Caption := MLRecord.LastName;
```

Iskaz record

SINTAKSA

```
name = record
    field_1 : data_type;
    field_2 : data_type;
    .
    .
    .
    field_n : data_type;
end;
```

Iskaz `record` deklariše grupisanje polja (`field_1, field_2,...,field_n`) i obezbeđuje naziv za ovu grupu (`name`).

Iskaz with

Dok objašnjavam slogove, dozvolite mi da predstavim iskaz `with`. Korišćenje iskaza `with` nije ograničeno na slogove, ali je ovo dobra prilika da se ilustruje način korišćenja. Ranije sam dao primer kako se popunjava struktura:

```
MLRecord.FirstName := 'Bruce';
MLRecord.LastName  := 'Reisdorph';
MLRecord.Address   := '123 Inspiration Pt.';
MLRecord.City      := 'Merced';
MLRecord.State     := 'CA';
MLRecord.Zip       := 99999;
```

Iskaz `with` se može koristiti kao pomoć pojednostavljenju ovog koda. Stoga, evo primera istog koda sa implementiranim iskazom `with`:

```
with MLRecord do begin
  FirstName := 'Bruce';
  LastName  := 'Reisdorph';
  Address   := '123 Inspiration Pt.';
  City      := 'Merced';
  State     := 'CA';
  Zip       := 99999;
end;
```

Iskaz `with` poručuje: "Sa (with) ovim objektom (`MLRecord`) uradi sledeće...". Zapazite da, ukoliko implementirate iskaz `with`, ne morate da kvalifikujete nazive polja identifikatorima sloga i operatorom tačka. Podrazumeva se da sve što se nalazi u okviru bloka `begin` i `end` pripada objektu `MLRecord`, pa je kvalifikovanje naziva polja nepotrebno. Iskaz `with` Vam može uštedeti dosta kucanja i Vaš kod može učiniti čitljivijim.

Nizovi slogova

Kao što imate nizove celih brojeva, karaktera, odnosno reči, možete isto tako imati nizove slogova. Deklarisanje i korišćenje nizova slogova nije strašno komplikovano.

```
var
  MLRecord : array [0..9] of MailingListRecord;
begin
  MLRecord[0].FirstName := 'Bruce';
  MLRecord[0].LastName  := 'Reisdorph';
  MLRecord[0].Address   := '123 Inspiration Pt.';
  MLRecord[0].City      := 'Merced';
  MLRecord[0].State     := 'CA';
  MLRecord[0].Zip       := 99999;

  MLRecord[1].FirstName := 'Georgia';
  MLRecord[2].LastName  := 'Burleson';
  MLRecord[3].Address   := '999 Fortitude';
  MLRecord[4].City      := 'Denver';
  MLRecord[5].State     := 'C0';
  MLRecord[6].Zip       := 80888;

  Label1.Caption := MLRecord[0].LastName;
  { More code here. }
end;
```

Ovo je samo malo komplikovanije od korišćenja nizova kod integralnog tipa podataka. Zapazićete da se zajedno koriste operator indeksa i operator elementa sloga, kako bi obezbedili vrednost polja koje se nalazi na navedenoj poziciji u okviru niza.

Priključene datoteke (include files)

Ponekad programeri na Pascalu koriste priključene datoteke. Priključena datoteka može da sadrži bilo koji kod koji ne želite da se nađe u glavnom junitu. Karakteristično je da se priključene datoteke koriste za konstante, odnosno direktive prevodiocu koje su namenjene za korišćenje u drugim datotekama projekta. Priključena datoteka nije ništa drugo do tekst datoteka sa produžetkom `.INC`. Produžetak `INC` nije obavezan, ali je uobičajeno da se koristi. Listing 2.2 pokazuje primer priključene datoteke.

Listing 2.2: EST.INC

```
const
  DefWidth  = 500;
  DefHeight = 300;

type
  MailingListRecord = record
    FirstName : string;
    LastName : string;
    Address : string;
    City : string;
    State : string;
    Zip : Integer;
  end;
```

Da biste kreirali priključenu datoteku, jednostavno otvorite novu tekst datoteku, a zatim je snimite sa produžetkom INC. Prvo odaberite opciju File➥New u okviru glavnog menija. Zatim dva puta kliknite na ikonu Text u okviru dijaloga New Items. U okviru editora koda (Code Editor) će biti kreirana i otvorena tekst datoteka. Upišite kod, a zatim snimite datoteku birajući opciju File➥Save As u okviru glavnog menija. Uverite se da ste datoteci dali produžetak INC, u protivnom će datoteka biti snimljena sa produžetkom TXT, koji se generički dodeljuje.

Da biste koristili priključenu datoteku, treba da upišete $I direktivu prevodiocu u sve junite koji zahtevaju korišćenje deklaracija definisanih u priključenoj datoteci. Primer izgleda ovako:

```
unit Unit2;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs,
  StdCtrls;

{$I Test.inc}

{ ... rest of unit follows }
```

Direktiva prevodiocu $I, poručuje prevodiocu da u junit prevede sadržaj priključene datoteke počev od definisane pozicije. Efekat je isti kao da ste ubacili priključenu datoteku u junit počev od navedene pozicije. Morate biti sigurni da je kod u okviru priključene datoteke sintaksno ispravan, u protivnom će prevodilac generisati grešku. Nemojte se zabrinuti ukoliko Vam sve ovo, u ovom trenutku izgleda pomalo zbunjujuće. Verovatno će Vam biti potrebno više iskustva u pisanju realnih programa da bi sve ovo leglo na svoje mesto.

Funkcije, procedure i metode

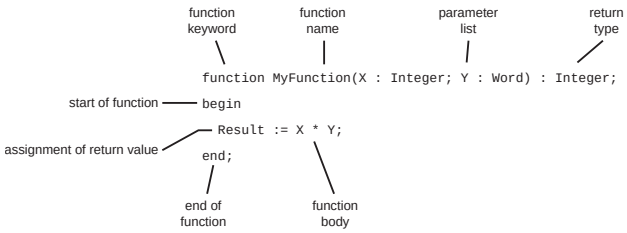
Funkcije i procedure su delovi koda izdvojeni od glavnog programa. Ovaj deo koda se izvršava kada se ukaže potreba za posebnom akcijom u okviru programa. Na primer, možda imate funkciju koja uzima dve vrednosti, izvršava kompleksne matematičke operacije sa navedenim vrednostima i vraća rezultat. Možda Vam je potrebna funkcija koja uzima string, raščlanjuje ga i vraća delove raščlanjenog stringa. Ove funkcije možete pozvati (koristiti) bilo kada u Vašem programu.

Funkcije i procedure se zajednički mogu nazvati podprogrami *(subroutines)*. (Pošto pojam podprogram nije uobičajen za Pascal, ovo je prikladnija reč koja obuhvata i funkcije i procedure, pa ću je koristiti.) Podprogrami su važan deo bilo kog programskog jezika , pa i Object Pascal nije izuzetak. Najjednostavniji tip podprograma ne sadrži parametre i ne vraća vrednosti. Drugi podprogrami mogu sadžavati jedan, ili više parametara i mogu vraćati vrednost. Pravila za dodeljivanje naziva funkcijama i procedurama su ista kao i kod prethodno pomenutih promenljivih.

NOVI TERMIN Funkcija *(function)* je deo koda izdvojen iz glavnog programa koji izvršava neku akciju i vraća vrednost.

NOVI TERMIN Parametar *(parameter)* je vrednost koja se prosleđuje funkciji, odnosno proceduri koja se koristi da izmeni delovanje parametra (promenljive), odnosno naznači proširenje njegovog delovanja.

Slika 2.2 prikazuje anatomiju funkcije.



Slika 2.2
Anatomija funkcije

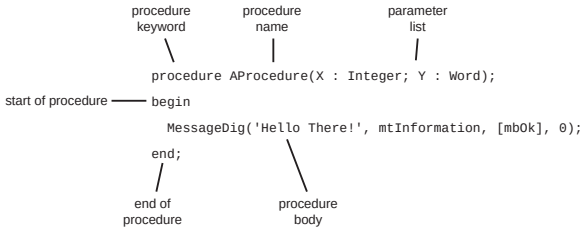
28670202.eps
31286-7
p2/v4

NOVI TERMIN Procedura *(procedure)* je deo koda izdvojen iz glavnog programa koji izvršava neku akciju, ali ne vraća vrednost.

Slika 2.3 prikazuje anatomiju procedure.

NOVI TERMIN Metoda *(method)* je funkcija, odnosno procedura koja pripada klasi.

Kao što ste mogli da vidite iz ovih opisa, jedina razlika između funkcije i procedure je u tome što funkcija vraća vrednost, dok je procedura ne vraća.



Slika 2.3
Anatomija procedure

28670203.eps
31286-7
p2/v4

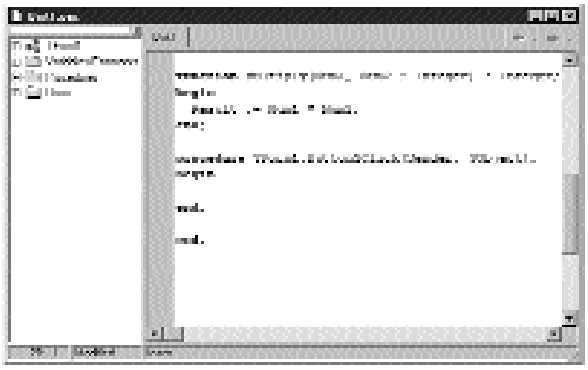
Hajde da napišemo program koji koristi funkciju. Još jednom počnimo sa novom aplikacijom. Izvršite sledeće korake:

1. Postavite komponentu dugme i komponentu natpis na formu.
2. Dvostrukim klikom na dugme kreirajte upravljač događajem `OnClick`.
3. Koristite taster strelica na gore na tastaturi, da bi pomerili kursor za editovanje do upravljača događaja koji ste kreirali.
4. Kucajte sledeću funkciju u editor koda: (Code Editor):

```
function Multiply(Num1, Num2 : Integer) : Integer;
begin
  Result := Num1 * Num2;
```

```
  end;
```

Vaš editor koda bi trebao da izgleda slično kao na slici 2.4.



Slika 2.4
Editor koda (Code Editor) prikazuje funkciju `Multiply`

5. Pomerite se na dole do upravljača događajem `OnClick` i kucajte kod sve dok upravljač događajima ne bude izgledao ovako:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X : Integer;
begin
  X := Multiply(10, 20);
  Label1.Caption := IntToStr(X);
end;
```

Pokrenite program i kliknite na dugme. Natpis će se promeniti u `200` kada kliknete na dugme. Evo kako program radi: kada kliknete na dugme, upravljač događajem `Button1Click` će biti pozvan. Nakon toga poziva se funkcija `Multiply` (Pomnoži) koja prihvata vrednosti `10` i `20` kao parametre. Rezultat se smešta u promenljivu X, koja se zatim prikazuje na natpisu.

Ovaj primer ilustruje korišćenje zasebne funkcije u okviru Delphi programa. Normalno je da sam ovu funkciju učinio delom klase glavne forme, ali pošto još uvek nismo obradili klase, malo ću požuriti ukoliko koristim ovu tehniku .

Možda mislite "U redu, ali kako da vratimo proizvod dva broja iz funkcije?" Ponovo pogledajte funkciju `Multiply`:

```
function Multiply(Num1, Num2 : Integer) : Integer;
begin
  Result := Num1 * Num2;
end;
```

Svaka funkcija jezika Object Pascal ima lokalnu promenljivu pod nazivom `Result`. Ovu promenljivu skriveno deklariše prevodilac, a ona se koristi da sačuva vrednost

koju vraća funkcija. Da bi vratili određenu vrednost iz funkcije, potrebno je da dodelite vrednost promenljive `Result` promenljivoj u okviru funkcije.

Postoji još jedan način definisanja vrednosti koju vraća funkcija. Bolje od dodeljivanja povratne vrednosti promenljivoj `Result`, je dodeljivanje povratne vrednosti nazivu funkcije. Na primer:

```
function Multiply(Num1, Num2 : Integer) : Integer;
begin
  Multiply := Num1 * Num2;
end;
```

Možda ste ovaj metod koristili u starim Pascal programima, ili prilikom prilagođavanja Turbo Pascal programa Delphiju.

Funkcija `Multiply` se može koristiti na jedan od sledečih načina. Možete proslediti promenljive znakovne vrednosti, ili pak, rezultate drugih poziva funkcija. Na primer:

```
X := Multiply(2, 5);   { passing literal values }
X := Multiply(A, B);   { passing variables }
{ return value used as a parameter for another function }
Label1.Caption := IntToStr(Multiply(X, Y));
Multiply(X, Y);        { return value ignored }
```

Zapazite da u prethodnom primeru vrednost koja se vraća nije korišćena. U ovom slučaju nema mnogo smisla pozivati funkciju `Multiply` i ignorisati vrednost koja se vraća, ali ignorisanje vrednosti koja se vraća je nešto što se često radi u programiranju na jeziku Object Pascal. Postoji mnogo funkcija koje izvršavaju određenu akciju, a zatim vraćaju vrednost koja ukazuje na status poziva funkcije. U nekim slučajevima, vrednost koja se vraća nije bitna za Vaš program, pa je ignorišete. Ukoliko ne radite ništa sa vrednošću koja se vraća, jednostavno je odstranite i to vam neće naškoditi.

Dodajmo proceduru programu koristeći sledeće korake:

1. Dva puta kliknite na dugme u okviru forme. Upravljač događajem `OnClick` će biti prikazan onako kako ste ga ostavili.
2. Pomerite kursor za editovanje nekoliko linija na gore da bude između funkcije `Multiply` i upravljača događajem `OnClick`. Kucajte sledeći kod:

```
procedure SayHello;
begin
  MessageDlg('Hello There!', mtInformation, [mbOk], 0);
end;
```

3. Pomerite se nekoliko linija na dole i dodajte jednu liniju koda na kraj upravljača događajem `OnClick` tako da izgleda ovako:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X : Integer;
```

```
begin
  X := Multiply(10, 20);
  Label1.Caption := IntToStr(X);
  SayHello;
end;
```

Sada ponovo pokrenite program. Ovaj put kada ste pokrenuli program, rezultat funkcije `Multiply` se prikazuje u natpisu kao i u prethodnom slučaju, a zatim se pojavljuje okvir za poruke. Okvir za poruke koji je prikazan predstavja rezultat poziva procedure `SayHello`. Pozivanje procedure `SayHello` je ekstremno jednostavno, pošto procedura ne preuzima parametre. Veoma je važno da shvatite da kod u funkciji, odnosno proceduri biva izvršen samo ukoliko eksplicitno pozovete funkciju, odnosno proceduru u bilo kom delu Vašeg koda.

Ukoliko otkrijete da se Vaš kod ponavlja nekoliko puta u okviru programa, razmislite o tome da taj deo koda prebacite u podprogram. Nakon toga možete pozvati podprogram, svaki put kada treba da izvršite željeni deo koda.

Podprogrami mogu pozivati druge podprograme (i najčešće se koriste za to). Podprogrami, čak, mogu pozivati sami sebe. Ovo se naziva *rekurzija* i predstavlja jedan od načina da zapadnete u nevolje prilikom programiranja! Rekurziju bi u početku rada sa jezikom Object Pascal trebali izbegavati.

Rekurzija je proces u kom procedura, ili funkcija poziva samu sebe.

Deklaracija i definicija

Funkcije i procedure često sadrže deklaraciju i uvek sadrže definiciju.

Deklaracija je iskaz koji opisuje naziv metoda i parametre, ukoliko postoje. U slučaju funkcije, deklaracija takođe ukazuje na tip vrednosti koja se vraća.

Definicija funkcije, odnosno procedure je ustvari osnova funkcije, odnosno procedure u odeljku `implementation` tekućeg junita.

Deklaracija je neophodna u tri slučaja:

- Kada funkciju odnosno proceduru koriste drugi juniti.
- Kada se definicija funkcije odnosno procedure nalazi ispod poziva funkcije, odnosno procedure.
- Kada je funkcija odnosno procedura element klase.

Do sada nisam koristio deklaracije, koristio sam samo definicije, zbog toga što definicija funkcije uvek dolazi ispred mesta u kodu gde se funkcija koristi. Kao primer uzmite funkciju `Multiply`. Da sam napisao deklaraciju ove funkcije, izgledala bi ovako:

```
function Multiply(Num1, Num2 : Integer) : Integer;
```

Kao što možete videti deklaracija funkcije opisuje samu funkciju.

Deklaracije funkcija i procedura se smeštaju u odeljak `interface`. Postavljanjem deklaracije u odeljak `interface` automatski čini da funkcija, odnosno procedura bude dostupna drugim junitima (takoreći, čini je javnom). Ukoliko ne želite da funkcija, ili procedura bude dostupna drugim junitima, ne treba da koristite deklaraciju. Umesto toga treba da budete sigurni da funkcija, odnosno procedura bude definisana blizu početka odeljka `interface`, da bi bila dostupna drugim metodama u okviru junita koji treba da je koriste. Kao što sam rekao, primeri funkcija i procedura su dosada koristili ovaj metod. Mogao sam definisati funkcije i procedure na drugi način, korišćenjem i deklaracija i definicija. Evo dela junita koji sadrži deklaraciju za funkciju `Multiply`, metod `Button1Click` koji poziva funkciju `Multiply`, i definiciju funkcije `Multiply`:

```
unit Unit1;

interface

{ some code removed… }

function Multiply(Num1, Num2 : Integer) : Integer;

implementation

procedure TForm1.Button1Click(Sender: TObject);
var
  X : Integer;
begin
  X := Multiply(10, 20);
end;

function Multiply(Num1, Num2 : Integer) : Integer;
begin
  Result := Num1 * Num2;
end;

end.
```

U ovom slučaju deklaracija je neophodna pošto je funkcija `Multiply` definisana *nakon* metoda `Button1Click`, koji je poziva. Deklaracija ukazuje prevodiocu da se funkcija nalazi u nastavku junita. O deklaracijama funkcija ćete više naučiti u sutrašnjoj lekciji, kada budemo obrađivali metode u okviru klasa.

Ako deklarišete funkciju, a zaboravite da je definišete, prevodilac će prijaviti grešku: `Unsatisfied forward or external decleration: 'Multiply'.` (nezadovoljavajuće prosleđivanje ili eksterna deklaracija: `'Multiply'.`)

Vrednosti, konstante i referentni parametri

Parametri funkcija, ili procedura mogu biti definisani na tri različita načina (ustvari, više od tri, ali objasniću samo tri načina).

Vrednosni paramatri (value parameters)

Kao prvo, parametri mogu biti vrednosni paramatri *(value parameters)*. Svi parametri koje ste do sada sretali su vrednosni parametri. Vrednosni paramatri se ponašaju kao lokalne promenljive u okviru funkcije, ili procedure. Promenljivu u okviru funkcije možete menjati, dok originalna promenljiva ostaje nepromenjena. Hajde da kreiramo novu funkciju koja će ilustrovati ovaj slučaj.Ova funkcija će se zvati `SquareAndMultiply`. Funkcija uzima dva broja, izračunava njihove kvadrate, množi ih i vraća rezultat. Evo primera:

```
function SquareAndMultiply(Num1, Num2 : Integer) : Integer;
begin
  Num1 := Num1 * Num1;
  Num2 := Num2 * Num2;
  Result := Num1 * Num2;
end;
```

Hajde da pogledamo kod koji poziva ovu funkciju:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X : Integer;
  Y : Integer;
  Z : Integer;
begin
  X := 2;
  Y := 3;
  Z := SquareAndMultiply(X, Y);
  Label1.Caption := IntToStr(Z);
end;
```

Ako želite, možete da unesete ovaj kod da bi ga testirali. Dve vrednosti se prosleđuju funkciji `SquareAndMultiply`. Ove dve vrednosti se menjaju unutar funkcije `SquareAndMultiply`, pošto brojeve prvo treba podići na drugi stepen, a tek onda ih pomnožiti. Originalne vrednosti `X` i Y u okviru metoda Button1Click se ne menjaju. Kada funkcija koristi vrednosni parametar, prevodilac prvo načini kopiju promenljive koja se prosleđuje funkciji, a zatim šalje kopiju navedenoj funkciji. Originalna promenljiva ostaje nepromenjena, pošto se u funkciju šalje kopija, a ne promenljiva.

Konstantni parametri (constant parameters)

Drugi način da se proslede vrednosti funkciji je korišćenje *konstantnih parametara* (constant parameters). Konstantni paramatri ne mogu biti menjani u okviru same funkcije. Evo primera procedure koja koristi konstantne parametre:

```
procedure SaySomething(const S : string);
begin
  S := S + 'Test';
  ShowMessage(S);
end;
```

Ovo je jedan od nekoliko primera u knjizi koji (nadam se) sadrže grešku. Prevodilac će prijaviti grešku na prvoj liniji procedure. Greška prevodioca glasi: `Left side cannot be assigned to`. (leva strana ne može biti dodeljena) Greška je generisana pošto ključna reč `const` određuje da promenljiva S ne može biti menjana. Bilo koji pokušaj izmene konstantnih parametara rezultuje greškom prevodioca. Ukoliko želite da prosleđujete promeljive koje se ne mogu menjati unutar funkcije, možete koristiti konstantne parametre.

Referentni parametri (reference parameters)

Treči način prosleđivanja vrednosti funkcijama je korišćenje *referentih parametara* (reference paramaters). Kada koristite referentne parametre, prevodilac ne pravi kopiju objekta kao u slučaju vrednosnih parametara. Umesto toga prosleđuje se aktuelna promenljiva. To znači, da ukoliko se izmeni promenljiva u okviru funkcije ili procedure, originalna promenljiva će takođe biti izmenjena. Sledi primer procedure koja koristi referentne parametre (prikazana je procedura i način na koji se procedura koristi):

```
procedure Square(var Number : Integer);
begin
  Number := Number * Number;
end;

procedure TForm1.Button1Click(Sender: TObject);
var
  X : Integer;
begin
  X := 20;
  Square(X);
  Label1.Caption := IntToStr(X);
end;
```

Prvo pogledajte proceduru `Square`. Uočite da je parametar promenljive definisan korišćenjem ključne reči `var`. Pošto se ključna reč koristi za deklarisanje referentnih parametara, obično se nazivaju promenljivi parametri *(var paramaters)*. Ove termine ću koristiti naizmenično u ovom poglavlju.

NAPOMENA Mnoge ključne reči jezika Object Pascal imaju dvostruku namenu. U ovom slučju, ključna reč `var` se koristi da deklariše referentni parametar. Ranije ste mogli da vidite kako se ključna reč `var` koristi za deklarisanje promenljivih u okviru funkcija, procedura, odnosno junita. Prevodilac otkriva u kom kontekstu se koristi ključna reč, pa pronalazi način da ispravno prevede kod.

Uočite da je u osnovi funkcije promenljiva Number, koja modifikuje samu sebe, množeći je sa sobom. Takođe, pogledajte kod u metodu `Button1Click` koji se naziva `Square`. Prvo se promenljivoj `X` dodeljuje vrednost. Zatim se promenljiva prosleđuje proceduri `Square`. Nakon izvršavanja procedure `Square`, vrednost promenljive `X` će biti 400. Pošto procedura Sqare uzima referentni parametar, promenljiva se prosleđuje proceduri (u ovom slučaju `X`) i biva izmenjena. Ukoliko želite da procedura, ili funkcija izmene promenljivu, koristite referentne parametre.

Pošto procedura `Square` koristi referentne parametre, morate proslediti promenljivu koja je istog tipa kao i referentni parametar. Primera radi, ovo ne možete uraditi:

```
Square(30);
```

Ovaj kod će generisati grešku prevodioca, pošto ne možete proslediti brojnu vrednost referentnom parametru. Ovo, takođe, neće biti prevedeno:

```
var
  X : Word;
begin
  X := 20;
  Square(X);
```

U ovom slučaju `X` je deklarisano kao `Word`, a referentni parametar procedure `Square` je deklarisan kao `Integer`. Prevodilac če generisati grešku, pošto se tipovi ne slažu. Greška prevodioca glasi: `Types of actual and formal var parameters must be identical`. (Tipovi aktuelnog i formalnog referentnog parametra moraju biti identični.)

SAVET Zapamtite da funkcije mogu vratiti samo jednu vrednost. Korišćenjem referentnih parametara, možete postići isti efekat kao kad funkcija vraća više od jedne vrednosti. Funcija i dalje vraća samo jednu vrednost, ali objekti prosleđeni po referenci se osvežavaju, pa funkcija efektivno vraća više vrednosti.

Lokalne funkcije i procedure

Lokalna funkcija, ili procedura je podprogram koji se nalazi u okviru drugog podprograma. Evo primera:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X : Integer;
```

```
  { A local procedure. }
  procedure Test;
  begin
    Memo1.Lines.Add('Local Function, X = ' + IntToStr(X));
  end;

begin
  X := 100;
  Memo1.Lines.Clear;
  Memo1.Lines.Add('Main Function, X = ' + IntToStr(X));
  Test;
end;
```

Uočite da se procedura nazvana `Test`, nalazi u odeljku var procedure `Button1Click`. Ukoliko je procedura deklarisana na ovaj način, naziva se lokalna procedura pošto se nalazi u okviru funkcije i procedure na lokalnom nivou. Lokalni podprogram se može pozvati samo iz rutine koja ga sadrži; nije moguće pozvati ga bilo gde u okviru programa.

Važna činjenica za lokalne procedure i funkcije je da promenljive koje pripadaju proceduri, koja se nalazi u okviru podprograma, mogu biti dostupne samo u okviru lokalnog podprograma. U ovom primeru promenljiva `X` je dostupna u glavnom delu procedure `Button1Click` i u okviru lokalne procedure. Kada se izvršava kod komponenta memo će sadržati sledeći tekst:

```
Main Function, X = 100
Local Function, X = 100
```

Ovo ilustruje da je promenljiva `X` dostupna u lokalnoj proceduri isto kao i u glavnoj proceduri.

Preopterećenje metoda (method overloading)

Počev od Delphija 4, jezik Object Pascal Vam omogućava da radite sa funkcijama koje imaju isti naziv, ali koriste različite parametre.

NOVI TERMIN Preopterećenje metoda *(method overloading)* nastaje kada postoje dve, odnosno više metoda, ili funkcija sa istim nazivom, ali različitim parametrima.

Metode koje dele zajednički naziv se nazivaju *preopterećene metode*.

U prethodnom delu sam Vam pokazao primer programa koji sadrži funkciju pod nazivom `Muliply`. Ne bez razloga, ova funkcija međusobno množi dve vrednosti. Funkcija uzima dva cela broja, množi ih, a zatim vraća rezultat. Šta bi bilo kada bi funkcija množila dve promenljive deklarisane kao `Double`, ili `Word`? Prethodne verzije Delphija bi zahtevale da imate nekoliko funkcija:

```
{ declarations for a program written in Delphi 1, 2, or 3 }
function MultiplyInt(Num1, Num2 : Integer) : Integer;
```

```
function MultiplyDouble(Num1, Num2 : Double) : Double;
function MultiplyWord(Num1, Num2 : Word) : Word;
```

Zar ne bi bilo lakše ukoliko biste imali samo jednu funkciju pod nazivom `Multiply` koja će biti dovoljno inteligentna da zna kada želite da pomnožite promenljive deklarisane kao `Integer`, `Double`, odnosno `Word`? Zahvaljujuči preopterećenju funkcije, ovo je sada moguće uraditi u Delphiju. Evo kako izgledaju deklaracije za preopterećenu funkciju:

```
{ declarations in Delphi 4 }
function Multiply(Num1, Num2 : Integer) : Integer; overload;
function Multiply(Num1, Num2 : Double) : Double; overload;
function Multiply(Num1, Num2 : Word) : Word; overload;
```

Još uvek treba da pišete odvojene funkcije za svaku deklaraciju, ali možete koristiti isti naziv funkcije. Prevodilac brine o pozivu prave funkcije bazirane na parametrima koje ste joj prosledili. Na primer:

```
var
  X, Y, Z : Double;
begin
  X := 1.5;
  Y := 10.5;
  Z := Multiply(X, Y);
end;
```

Prevodilac primećuje da su dve promenljive, deklarisane kao `Double`, prosleđene funkciji i poziva verziju funkcije `Multiply`, koja je uzima kao parametre dve promenljive tipa `Double`. Slično, ukoliko se proslede dve promenljive tipa `Integer`, prevodilac poziva verziju funkcije `Multiply` koja uzima dve celobrojne vrednosti (Integer).

Ono što pokreće preopterećene funkcije je lista parametara. Možete menjati ili tip, ili broj parametara koje funkcija uzima (odnosno i tip i broj parametara), ali ne možete kreirati preopterećenu funkciju menjajući samo vrednost koja se vraća. Na primer, na sledeći način se ne definiše preopterećena funkcija:

```
function DoSomething : Integer; overload;
function DoSomething : Word; overload;
```

Ako pokušate da prevedete program sa sledećim linijama, prevodilac će prijaviti grešku: `Declaration of 'DoSomething' differs from previous declaration`. (Deklaracija 'DoSomething' se razlikuje od prethodne deklaracije.) Dve funkcije se ne mogu razlikovati po vrednosti koju vraćaju da bi bile preopterećene funkcije.

Generički parametri za funkcije

NOVI TERMIN Procedura, ili funkcija može imati generičke parametre (*default parameters)* koji kao što im samo ime kaže, pružaju generičke vrednosti parametra ukoliko u pozivu funkcije nije definisana vrednost parametra.

Funkcija koja sadrži generičke parametre, može izgledati ovako:

```
{ Procedure declaration. }
{ Parameter 'EraseFirst' will be false by default. }
procedure Redraw(EraseFirst : Boolean = False);

{ Procedure definition. }
procedure Redraw(EraseFirst : Boolean);
begin
  if (EraseFirst) then begin
    { erase code }
  end;
  { drawing code }
end;
```

Ovu funkciju možete pozvati sa, ili bez parametara. Ukoliko se prilikom poziva funkcije definiše parametar, funkcija se ponaša kao i bilo koja druga funkcija. Ukoliko parametar nije definisan prilikom poziva funkcije, automatski se uzima generički parametar. Prema ovom primeru, sledeće dve linije koda su identične:

```
Redraw;
Redraw(False);
```

Kao što možete videti, kada parametar ima generičku vrednost, isti može biti izostavljen iz poziva funkcije:

```
{ declaration }
function PlayWaveFile(Name : string;
  Loop : Boolean = False; Loops : Integer = 10) : Integer;

{ calls to PlayWaveFile }
R := PlayWaveFile('chime.wav');          { does not loop sound }
R := PlayWaveFile('ding.wav', True);     { plays sound 10 times }
R := PlayWaveFile('bell.wave', True, 5); { plays sound 5 times }
```

Generički parametri su korisni iz nekoliko razloga. Prvo, čine Vaš život jednostavnijim. U 99 posto slučajeva ćete koristiti funkcije koje pozivate sa istim parametrima. Dodeljivanjem generičkih parametara, skraćujete kucanje koje je potrebno svaki put kada pozivate funkciju. Ukoliko želite da prenesete parametre koji se razlikuju od generičkih, sve što treba da uradite je da dodelite druge vrednosti generičkim parametrima.

NAPOMENA Bilo koji generički parametar se mora nalaziti na kraju liste parametara funkcije. Sledeća deklaracija nije ispravna:

```
procedure MyProcedure(X : Integer; Y : Integer = 10; Z :
Integer);
```

Da bi se navedena deklaracija funkcije mogla prevesti, generički parametar mora biti prebačen na kraj liste:

```
procedure MyProcedure(X : Integer; Z : Integer; Y :
Integer = 10);
```

Ako ne postavite generičke parametre na kraj liste parametara, prevodilac će generisati grešku.

Zaključak

Ovo poglavlje sadrži osnovne informacije nekih od fundamentalnih operacija jezika Object Pascal. Da biste programirali na Delfiju potrebno je da shvatite šta je ovde prezentirano. Prvo ste naučili različite tipove petlji jezika Object Pascal, a zatim ste naučili iskaz case i način na koji se koristi. Napisao sam nešto i o opsegu i šta on znači za Vaše promenljive. Zatim ste mogli pronaći nešto o slogovima i kako se slogovi mogu koristiti u Vašim programima. Dan ste završili učeći o funkcijama i cedurama.

Radionica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstite Vaše znanje vezano za materijal koji je obrađen. Takođe sadrži vežbe koje Vam omogučavaju da steknete iskustvo u korišćenju stvari koje ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A, "Odgovori na kviz pitanja".

Pitanja i odgovori

P **Na koliko nivoa mogu ugnjezditi iskaz `if`?**

O Ne postoji ograničenje. Naravno, praktično ograničenje ipak postoji. Ako imate suviše ugnježdenih iskaza `if`, veoma je teško kontrolisati ih!

P **Da li će se izvršavanje petlje automatski prekinuti, ukoliko nešto krene naopako?**

O Ne. Ukoliko greškom napišete beskonačnu petlju, ista će nastaviti da se izvršava sve dok ne učinite nešto da je zaustavite. Program koji se zaglavio u beskonačnoj petlji možete zaustaviti pozivom Windows Task Manager-a (ili, okvira za zatvaranje programa - Close Program box) i na taj način zaustaviti posao koji pravi grešku. Ukoliko izvršavate program iz Delphi okruženja (Delphi IDE), možete odabrati Run➥Program Reset u okviru glavnog menija i na taj način zaustaviti izvršavanje programa.

P **Da li case iskaz mora da sadrži else sekciju?**

O Ne. else sekcija je opciona.

P Da li mogu da posedujem više od jedne promenljive istog naziva?

O Da, ako se nalaze u različitim oblastima važenja. Na primer, možete imati globalnu promenljivu X i lokalnu promenljivu istog imena.

P Zašto se koristite overload procedure i funkcije?

O Ove funkcije pružaju mogućnost kreiranja nekoliko funkcija koje obavljaju iste osnovne operacije, ali sa različitim parametrima. Na pirmer, možete definisati overload funkciju `DrawObject`. Jedna verzija ove funkcije može uzeti klasu `Circle` kao parametar, dok druga može uzeti klasu `Square` kao parametar, a treća klasu `Polygon` kao parametar. Sa ove tri funkcije istog imena, nema potrebe imati tri različite funkcije.

P Da li mogu koristiti zapis bez prethodnog kreiranja instance zapisa?

O Ne. Pre upotrebe zapisa, morate kreirati instancu zapisa i pristupiti zapisu preko njegove insance.

Kviz

1. Koji se iskazi izvršavaju u slučaju kada rezultat `if` izraza ima vrednost `True`?
2. Koliko vrednosti funkcija može da vrati?
3. Pored sintakse, koja je razlika između `while` i `repeat` petlje?
4. Šta je funkcija `Break` i `Continue` procedura?
5. Šta je globalna promenljiva?
6. Da li zapis može posedovati kombinovane podatke različitih tipova (`Char`, `Integer`, `Word` itd.)
7. Kako se pristupa članovima zapisa?
8. Koliko funkcija i procedura može imati u programu?
9. Da li funkcija može pozvati drugu funkciju ili proceduru?
10. Da li je dozvoljeno kreirati niz sa zapisima?

Vežbe

1. Napišite proceduru pod imenom `Test2` koja menja mala u velika slova i obrnuto Label komponente. Postavite dugme (button) na formu i podesite da `OnClick` događaj dugmeta poziva `Test2` proceduru.
2. U programu iz vežbe 1 dodajte proceduru `Test1` koja poziva `Test2` proceduru. Promenite da `OnClick` događaj poziva proceduru `Test1` umesto `Test2`.

3. Napravite program koji prikazuje tekst Nikad više neću pričati `sa majkom`, 20 puta u `Memo` komponenti.
4. Formirajte zapis koji sadrži polja za informaciju o zaposlenima. Uključite ime, prezime, adresu, datum zapošljavanja i indikaciju da li radnik poseduje osiguranje u firmi.

Dan 3

Klase i objektno orijentisano programiranje

Danas ćete preći na dobre stvari. U ovom poglavlju ćete učiti o klasama. Klase su srž jezika Object Pascal, kao i drugih objektno orijentisanih jezika. Klase su takođe srž Visual Component Library (biblioteka vizuelnih komponenti - VCL), koje ćete koristiti kada počnete pisati prave Windows aplikacije. (VCL će biti detaljnije obrađen u lekciji dana 5, "Model vizuelnih komponenti".) Danas ćete otkriti šta su to klase i kako treba da ih koristite. U toku rada ćete naučiti značenje termina jezika Object Pascal, kao što su: nasleđivanje (*inheritance*), objekti (*objects*) i apstrakcija podataka (*data abstraction*). Pre nego što pređemo na ove pojmove, hteo bih da obradim još nekoliko aspekata jezika Object Pascal koji do sada nisu bili obrađeni.

Skupovi (sets)

Skupovi se često koriste u Delphiju, pa je potrebno da znate šta su to skupovi i kako funkcionišu.

NOVI TERMIN Skup (*set*) je grupa vrednosti koje pripadaju istom tipu.

Ovaj opis Vam ne kazuje puno, zar ne? Primer koji pada na pamet je karakteristika `Style` VCL font objekta. Karakteristika može uključiti jednu, ili više navedenih vrednosti:

- `fsBold`
- `fsItalic`
- `fsUnderline`
- `fsStrikeout`

Font može sadržati bili koju kombinaciju stilova, odnosno ne mora sadržati ni jedan stil posebno. Skup stilova fonta može da ne sadrži ni jednu vrednost, odnosno može sadržati sve vrednosti, odnosno može sadržati bilo koju kombinaciju vrednosti.

Pa, kako da koristite skup? Koristiću karakteristiku `Style`, da bih Vam ovo ilistrovao. U toku dizajniranja aplikacije možete menjati vrednosti pojedinih karakteristika `Style`. Ponekad, pak, imate potrebu da menjate karakteristiku `Style` u toku rada programa. Na primer, pretpostavimo da stilu fonta želite dodati atribute bold i italic. Jedan od načina je deklarisanje promenljive tipa `TFontStyles`, a zatim dodate stilove `fsBold` i `fsItalic` Vašem skupu. Evo kako to izgleda:

```
var
  Styles : TFontStyles;
begin
  Styles := Styles + [fsBold, fsItalic];
end;
```

Ovaj kod dodaje elemente `fsBold` i `fsItalic` skupu `Styles`. Elementi su napisani u uglastoj zagradi, kojom se naznačava dodavanje elemenata skupu. Uglasta zagrada se, kada je koristite na ovaj način, zove konstruktor skupa (*set constructor*). Uočite da ovaj kod ustvari ne menja stil fonta; on samo kreira skup i dodaje dva elementa navedenom skupu. Da biste promenili stil fonta, treba da pridružite novokreirani skup karakteristici `Font.Style` bilo koje komponente:

```
Memo.Font.Style = Styles;
```

Pretpostavimo da sada želite da font bude podebljan (bold), ali ne i kurziv (italic). U tom slučaju treba da uklonite stil kurziv iz skupa:

```
Styles := Styles - [fsItalic];
```

Stil sada sadrži samo vrednost `fsBold`, pošto je vrednost `fsItalic` uklonjena.

Često ćete poželeti da proverite da li se određena stavka nalazi u skupu. Ukoliko želite da proverite, da li je font trenutno podešen kao podebljan (bold), ovo možete otkriti korišćenjem elementa `fsBold` i korišćenjem ključne reči `in`:

```
if fsBold in Styles then
  DoSomething;
```

Ponekad će biti porebno da se uverite da počinjete sa praznim skupom. Možete isprazniti skup dodeljujući mu prazan skup preko promenljive. Ovo se može uraditi praznim konstruktorom skupa - na primer:

```
{{ start with an empty set }
Styles := [];
{ now add the bold and italic styles }
Styles := Styles + [fsBold, fsItalic];
```

U ovom primeru svi stilovi fonta su izbrisani, a zatim su dodati stilovi za podebljani i kurziv font. Isti efekat se može postići na malo drugačiji način dodelom stila direktno u skup:

```
Styles := [fsBold, fsItalic];
```

Da bi promenili stil fonta, ne morate striktno kreirati promeniljivu `TFontStyles`. Možete raditi direkno sa karakteristikom - na primer:

```
Memo.Font.Style := [];
Memo.Font.Style := Memo.Font.Style + [fsBold, fsItalic];
```

Skup će biti deklarisan korišćenjem ključne reči set. Karakteristika `TFontStyles` je deklarisana u okviru VCL izvorne datoteke `GRAPHICS.PAS` na sledeći način:

```
TFontStyle = (fsBold, fsItalic, fsUnderline, fsStrikeOut);
TFontStyles = set of TFontStyle;
```

U ovom primeru prva linija deklariše specifikaciju tipa pod nazivom `TFontStyle`. (*Specifikacija* predstavlja listu mogućih vrednosti.) Druga linija kreira skup `TFontStyles` koji predstavlja skup vrednosti `TFontStyle`.

Skupovi se često koriste u VCL i programiranju na Delphiju. Većina karakteristika komponenti je definisana preko skupova. Brzo ćete se privići na skupove tokom rada na Delphiju.

Raspoređivanje

NOVI TERMIN Rasporediti (*cast*), znači saopštiti prevodiocu da tretira jedan tip podataka kao da pripada drugom tipu podataka. Drugi naziv za raspoređivanje je raspoređivanje tipova (*typecast*).

Evo primera raspoređivanja tipa podataka `Char` u tip podataka `Integer`:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  AChar : Char;
  AnInteger : Integer;
begin
  AChar := 'A';
  AnInteger := Integer(AChar);
  Label1.Caption := IntToStr(AnInteger);
end;
```

U ovom primeru raspoređeni `Integer` (`AChar`) pokazuje prevodiocu da konvertuje vrednost promenljive `AChar` u celobrojni tip podataka (`Integer`). Raspoređivanje je

neophodno pošto ne možete dodeliti vrednost promenljive čiji je tip `Char`, tipu `Integer`. Ako pokušate da dodelite vrednost, a da ne koristite raspoređivanje, prevodilac će prijaviti grešku koja glasi: `Incompatible types: 'Integer' and 'Char'`.

```
Incompatible types: 'Integer' and 'Char'.
```

Uzgred, u toku izvršavanja navedenog koda, na natpisu će biti prikazan tekst: `65` (65 je celobrojna vrednost karaktera A).

Nije uvek moguće rasporediti jedan tip u drugi. Uzmimo na primer sledeći kod:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Pi : Double;
  AnInteger : Integer;
begin
  Pi := 3.14;
  AnInteger := Integer(Pi);
  Label1.Caption := IntToStr(AnInteger);
end;
```

U ovom slučaju sam pokušao da rasporedim tip podataka `Double` u `Integer`. Ovo nije dozvoljeno raspoređivanje, pa prevodilac javlja grešku koja glasi: `Invalid typecast`. (Pogrešan tip dodeljivanja). Da biste konvertovali vrednost u pokretnom zarezu u celobrojnu vrednost, koristite funkcije `Trunc`, `Floor`, odnosno `Ceil`. Ove funkcije rade upravo ono na šta njihovi nazivi ukazuju, pa ih ubuduće neću objašnjvati. Za detaljnije informacije o funkcijama pogledajte Delphijev Help.

Pointeri se mogu rasporediti u drugi tip korišćenjem operatora `as`. (O pointerima će biti više reči u sledećem poglavlju.) Operator as ću opisati u poglavlju "Ključne reči klasa: `is` i `as`."

Pointeri (pointers)

Pointeri su aspekti jezika Object Paskal koji Vas mogu najviše zbuniti. Pa, šta su to pointeri? To su promenljive koje sadrže adresu druge promenljive. Ovo i nije tako loše, zar ne? Voleo bih da je sve tako jednostavno! Pošto pointer sadrži adresu druge promenljive, kaže se da on ukazuje (point to) na drugu promenljivu. Ovo se zove *indirektnost* pošto pointer ne sadrži direktan, već indirektan priključak za aktuelni podatak.

NOVI TERMIN *Pointer* je promenljiva koja sadrži adresu druge promenljive.

Pogledajmo primer. Recimo da imate slog čiju adresu je potrebno proslediti proceduri koja zahteva pointer. Adresu sloga uzimate korišćenjem operatora `@`. Evo kako to izgleda:

```
var
  MLRecord : TMailingListRecord;
APtr : Pointer;
begin
  { Fill MLRecord with data. }
  APtr := @MLRecord;
  SomeFunction(APtr);
end;
```

Promenljiva `APtr` (čiji je tip `Pointer`) se koristi za čuvanje memorijske adrese sloga `MLRecord`. Ovaj tip pointera se naziva pointer bez tipa (*Untyped pointer*), pošto tip podataka `Pointer` sadrži memorijsku adresu. Sledeći tip pointera je pointer koji je deklarisan kao ukazivač na određeni tip objekta. Recimo da ste kreirali novi tip, pointer na slog `TMailingListRecord`. Deklaracija će izgledati ovako:

```
type
  PMailingListRecord = ^TMailingListRecord;
  TMailingListRecord = record
    FirstName : string;
    LastName : string;
    Address : string;
    City : string;
    State : string;
    Zip : Integer;
  end;
```

Tip `PMailingListRecord` je deklarisan kao pointer na `TMailingListRecord`. Obično ćete se sretati sa slogovima i njima pripadajućim pointerima koji su deklarisani na isti način. Možda se pitate čemu ovo služi. Pređimo na sledeće poglavlje i pokazaću Vam jedan od načina na koji možete koristiti pointere.

Za slogove obično ne koristim duge stringove kao što sam ih koristio u prethodnom primeru za slog `TMailingListRecord`. Obično koristim niz karaktera umesto dugog stringa. Razlog leži u činjenici da se dugi stringovi dinamički raspoređuju i nisu fiksne dužine. Polja fiksne dužine su važna ukoliko zapisujete slogove na disk. U slučaju sloga `TMailingListRecord` sam koristio dugi string, pošto nisam želeo da Vas zbunjujem pričom o slogovima sa fiksnom dužinom u ovom delu knjige.

Lokalno korišćenje memorije nasuprot dinamičkom korišćenju memorije

U jučerašnjoj lekciji sam dao neke primere u delu koji se odnosi na slogove. Svi primeri koriste lokalno dodeljivanje objekata. To znači da memoriju potrebnu za promenljivu sloga obezbeđuje stek programa.

NOVI TERMIN *Lokalno dodeljivanje* znači da memoriju potrebnu promenljivoj, odnosno objektu obezbeđuje stek programa.

NOVI TERMIN Stek (*stack*) je deo radne memorije koju program rezerviše na početku rada.

Memorija koju program zahteva za stvari kao što su lokalne promenljive, pozivi funkcija itd., se uzima od steka programa. Ova memorija se rezerviše kada se ukaže potreba, a oslobađa se kada prestane potreba; ovo se obično dešava kada program poziva funkciju, odnosno neki drugi lokalni kod blok. Memorija za lokalne promenljive koje funkcija koristi se rezerviše u trenutku poziva funkcije. Prilikom izlaska iz funkcije, kompletna memorija rezervisana za funkciju će biti oslobođena. Sve ovo se odvija automatski; Vi ne morate da brinete o tome kako se memorija oslobađa, odnosno da li je memorija u potpunosti oslobođena.

Lokalno rezervisanje memorije ima svoje dobre i loše strane. Dobra strana je što memorija steka može biti rezervisana brzo. Loša strana je da je stek fiksne dužine i da dužinu nije moguće menjati u toku rada programa. Ukoliko Vaš program ostane bez slobodnog prostora u okviru steka, počeće da se dešavaju čudne stvari. Vaš program će možda krahirati, možda će početi da se čudno ponaša, odnosno možda će se ponašati normalno, ali će krahirati kada poželite da ga zatvorite. Ovo je manji problem u 32-bitnom programiranju nego u 16-bitnom programiranju, ali ga treba uzeti u obzir.

Za promenljive čiji su tipovi već dati u programskom jeziku, odnosno za male nizove, nema potrebe rezervisati memoriju na drugi način, osim lokalno. Ukoliko želite da koristite duge slogove, verovatno ćete poželeti da od samog početka koristite dinamičko rezervisanje raspoložive memorije. Količina raspoložive memorije zavisi od zbira ukupne slobodne fizičke memorije računara (RAM) i slobodnog prostora na tvrdom disku. Drugim rečima, lako možete rezervisati 100MB raspoložive memorije na prosečnom računaru koji radi pod Windows-ima. Dobra vest je, da za Vaše programe teoretski možete koristiti neograničeno mnogo memorije. Loša vest je da memorija rezervisana dinamički zahteva dodatna objašnjenja i znatno je sporija od memorije rezervisane preko steka. U većini programa nije potrebno posebno naznačiti dinamičko korišćenje memorije. Dodatna poteškoća kod dinamičkog rezervisanja memorije je zahtev za dodatnim pisanjem koda, ali ne toliko puno kao što ste mislili.

NOVI TERMIN *Dinamičko rezervisanje* memorije znači da se memorija za objekte rezerviše sa gomile.

NOVI TERMIN *Gomila* u Windows programima predstavlja kompletnu virtuelnu memoriju računara.

Dinamičko rezervisanje memorije i pointeri

U Object Paskal programima memorija se dinamički može rezervisati na nekoliko različitih načina. Možda je najbolji način korišćenje funkcije `AllocMem`. Funkcija `AllocMem` rezerviše memoriju i popunjava je nulama. (Drugi način dinamičkog rezervisanja memorije uključuje proceduru `GetMem` i funkciju `New`.) Ukoliko sve uzmemo u obzir, funkcija `AllocMem` verovatno pruža najbolji način za dinamičko rezervisanje memorije. Vratimo se na slog `TMailingListRecord`. U prethodnom primeru sam rezervisao memoriju steka za jedan ovakav slog na sledeći način:

```
var
  MLRecord : TMailingListRecord;
begin
  { Fill MLRecord with data. }
  MLRecord.FirstName := 'Per';
  MLRecord.LastName  := 'Larsen';
  { etc. }
end;
```

Sada ću slog kreirati dinamički, umesto lokalno:

```
var
  APtr : PMailingListRecord;
begin
  APtr := AllocMem(SizeOf(TMailingListRecord));
APtr.FirstName := 'Per';
  APtr.LastName  := 'Larsen';
  { Do some other things. }
  FreeMem(APtr);
end;
```

Zapazite da sam ovaj put deklarisao `PMailingListRecord` (pointer na `TMailingListRecord`), umesto `TMailingListRecord`. Takođe, uočite, da sam rezervisao memoriju za strukturu pozivom funkcije `AllocMem`. Parametar prosleđen funkciji `AllocMem` predstavlja veličinu memorije koju treba rezervisati. Funkcija `SizeOf` vraća dužinu sloga, pa sam ovu funkciju koristio kako bih odredio koliko memorije treba rezervisati. Poziv funkcije `AllocMem` rezerviše memoriju i inicijalizuje pointer dinamički kreirajući novi slučaj sloga `TMailingListRecord`. Nakon što rezervišete memoriju možete koristiti pointer promenljivu kao što korisite normalne promenljive. Na kraju, treba da primetite da sam po završetku rada sa objektom oslobodio memoriju dodeljenu objektu korišćenjem procedure `FreeMem`. Greška koja nastaje ukoliko ne pozovete proceduru `FreeMem` koja oslobađa dinamički rezervisane objekte, će kao rezultat dati program koji troši memoriju uzima memoriju koju nikad ne oslobađa).

Dinamičko rezervisanje memorije za slogove i nizove nije obavezno. Kod klasa se preporučuje korišćenje dinamičkog rezervisanja memorije. O ovome će biti više reči u delu koji opisuje klase.

Ovo je proces u kom dinamički kreirate i pristupate slogovima u okviru jezika Object Pascal. Verovatno ćete retko koristiti dinamičko rezervisanje, ali ponekad je to neophodno, pa bi trebalo da znate kako se to radi.

Ključna reč `nil` se koristi da definiše pointer koji nema vrednosti. Ako želite da oslobodite pointer, to jest da obrišete vrednost pointera, možete koristiti ključnu reč `nil` na sledeći način:

```
SomePointer := nil;
```

Takođe, možete koristiti nil da biste proverili da li je u pointer upisana neka vrednost:

```
if SomePointer = nil then
SomePointer := AllocMem(Size);
```

Ovaj kod proverava da li je pointeru dodeljena neka vrednost. Ako pointeru nije dodeljena vrednost, rezerviše se memorija za isti.

Uklanjanje reference pointera

Ponekad je potrebno da uklonite referencu pointera.

Uklanjanje reference (dereferencing) pointera znači vraćanj objekta na koji pointer pokazuje.

Recimo da ste dinamički kreirali slog za listu pošte (mailing list record) kao što je ranije opisano. Sada želite da dodelite sadržaj sloga za listu pošte drugoj promenljivoj sloga liste pošte koja se nalazi na steku. Za sada imate sledeći kod:

```
var
  APtr : PMailingListRecord;
  Rec  : TMailingListRecord;
begin
  APtr := AllocMem(SizeOf(TMailingListRecord));
```

Pretpostavimo da sada želite da kopirate sadržaj promenljive `APtr` u promenljivu `Rec`. Promenljiva `APtr` je pointer na `TMailingListRecord`, a promenljiva `Rec` sadrži `TMailingListRecord`. Možete probati:

```
Rec := APtr;
```

Ovo neće raditi pošto `APtr` sadrži memorijsku adresu a ne `TMailingListRecord`. Da bistepešno izveli dodeljivanje morate ukloniti referencu pointera korišćenjem pointer operatora (`^`), što izgleda ovako:

```
Rec := APtr^;
```

Kada uklanjate referencu pointera dajete na znanje prevodiocu: ”Daj mi objekt na koji ukazuje pointer, a ne vrednost pointera.”

Šta je klasa?

Klasa je skup međusobno povezanih polja i metoda (funkcija i procedura) koje se koriste za izvršavanje određenog programskog zadatka. U ovom slučaju može se reći da klasa *enkapsulira* zadatak. Klase imaju sledeće mogućnosti:

- mogućnost da kontrolišu pristup
- konstruktore
- destruktore
- polja
- metode (procedure i funkcije)
- skrivene, posebne pointere pod nazivom `Self`

Pre nego što pređemo na objašnjenje ovih mogućnosti, dozvolite mi da Vam dam kratak primer načina na koji se klasa može koristiti. Uzmimo tipičnu Windows kontrolu kao primer - polje za potrvrdu. Klasa koja predstavlja polje za potvrdu (check box) može sadržati polja za natpis i stanje (potvrđeno, ili nepotvrđeno) polja za potvrdu. Ova klasa, takođe, može imati metode koje Vam omogućavaju da podesite i ispitate stanje i naslov polja za potvrdu. Ove metode mogu imati naziv: `GetCheck`, `SetCheck`, `GetCaption` i `SetCaption`. Nakon što napišete klasu, možete kreirati slučaj klase koja kontroliše polje za potvrdu u okviru Windows-a. (Ovo baš nije tako jednostavno, ali ipak, ovo je samo primer.) Ukoliko imate tri polja za potvrdu, treba da definišete tri primerka klase `CheckBox` koji će se koristiti za pojedinačno kontrolisanje svakog polja za potvrdu.

```
var
  Check1 : TMyCheckBox;
  Check2 : TMyCheckBox;
  Check3 : TMyCheckBox;
begin
  Check1 := TMyCheckBox.Create(ID_CHECK1);
  Check2 := TMyCheckBox.Create(ID_CHECK2);
  Check3 := TMyCheckBox.Create(ID_CHECK3);
  Check1.SetCaption('Thingamabob Option');
  Check1.SetCheck(True);
  Check2.SetCaption('Doohickey Options');
  Check2.SetCheck(False);
  Check3.SetCaption('Whodyacallum Options');
  Check3.SetCheck(True);
  if Check1.GetCheck then DoThingamabobTask;
  if Check2.GetCheck then DoDoohickeyTask;
  { etc. }
end;
```

U ovom primeru svaki slučaj klase je zaseban objekt. Svaki slučaj ima svoja polja, a objekti funkcionišu nezavisno jedan od drugog. Svi objekti pripadaju istom tipu, ali su razdvojeni u memoriji. Nakon ovog kratkog uvoda, zasucite rukave još jednom i krenite na učenje klasa.

NAPOMENA Prethodni primer bi možda bio jasniji da sam koristio karakteristike umesto metoda pod nazivom `SetCheck`, `GetCheck` i `SetCaption`. Karakteristike nisam koristio, pošto još nije vreme da se ovaj deo detaljno obrađuje. U suštini, većina ovog poglavlja će biti posvećena klasama bez isticanja karakterisika. Detaljnije ćemo obraditi karakteristike u lekciji dana 5.

Anatomija klase

Klasa, kao što je to slučaj kod slogova, ima svoju deklaraciju. Deklaracija klase se uvek nalazi u odeljku `type`.

Nivoi pristupa klasi

Klase mogu imati četiri nivoa pristupa:

- privatni (private)
- javni (public)
- zaštićeni (protected)
- najavljeni (published)

Svaki od navedenih nivoa pristupa se definiše u navedenom odeljku.

Nivoi pristupa klasama kontrolišu način korišćenja klase. Ukoliko samostalno programirate, možda nećete biti samo kreator klasa nego i njihov korisnik. U timu programera, jedan programer može biti kreator klasa, a ostali korisnici. Da biste shvatili ulogu nivoa pristupa u operacijama sa klasama, prvo treba da razumete kako se klase koriste.

U svakoj klasi postoji javni (*public*) deo klase, kome se može pristupiti spolja, a takođe postoji i privatni deo klase. *Privatni* deo klase predstavlja interna inplementacija klase - takozvani unutrašnji rad.

Deo dobro dizajniranih klasa sadrži skriveni deo, koji korisnik klase ne treba da vidi.

NOVI TERMIN *Apstrakcija podataka* (data abstraction) je sakrivanje internih inplementacija koje se nalaze unutar klase od pristupa van klase.

Apstrakcija podataka sprečava korisnika da sazna više od onoga što je potrebno da zna o klasi, i takođe sprečava korisnika da se meša u stvari u koje ne bi smeo. Na primer, kada uđete u auto i okrenete ključ za pokretanje motora, da li bi trebali da znate detaljno kako automobil radi? Naravno da ne. Treba samo da znate onoliko

koliko je potrebno da bi automobil bezbedno radio. Po ovoj analogiji, upravljač, pedale, ručica menjača, brzinomer i slično predstavljaju javni interfejs između automobila i vozača. Vozač zna kojom komponentom treba da manipuliše da bi se automobil ponašao onako kako on želi.

Suprotno, motor, upravljački sistem i električne instalacije automobila su skrivene od pogleda vozača. Motor je skoro skriven, pošto ga najverovatnije nećete gledati ukoliko to ne želite. To je detalj koji Vas ne interesuje, pa je zato sakriven - definisan privatno (private), ukoliko Vam se više sviđa. Zamislite kakav bi problem bila vožnja ukoliko biste morali da znate sve što automobil radi: Da li karburator dobija dovoljno benzina? Da li diferencijal ima dovoljno ulja? Da li alternator proizvodi dovoljno struje da bi ubrizgavanje goriva i radio radili istovremeno? Da li se usisni ventili otvaraju pravovremeno? Kome to treba! Na isti način klasa čuva svoju internu implementaciju privatno, da korisnik klase ne mora da brine o svemu što se dešava u unutrašnjosti. Interni rad klase ostaje privatan a korisnički interfejs je javan.

Zaštićeni (*protected*) nivo pristupa je malo teže objasniti. Zaštićenim elementima klase, kao što je to slučaj i kod privatnih elemenata klase, ne mogu pristupiti korisnici klase. Njima mogu, naravno, pristupiti klase koje su izdvojene iz navedene klase. Nastavimo sa analogijom kod automobila: Recimo da želite da proširite auto (bukvalno) praveći izduženu limuzinu. Da biste ovo uradili, treba da znate nešto o donjem podstroju automobila. Morate znati kako da što jednostavnije modifikujete upravljački sistem i ram automobila. U ovom slučaju će biti potrebno da uprljate ruke, i da kao dizajner limuzine dođete do delova automobila koji su prethodno bili neinteresantni za Vas (zaštićeni delovi).

Unutrašnji rad sa motorom još uvek ostaje privatan, pošto ne morate znati kako motor radi da biste proširili ram automobila. Opet će većina javnih delova automobila ostati ista, ali ćete morati da dodate nove javne elemente kao što je kontrola za interfon. Zastaću na trenutak ovde i omogućiti Vam da zavirite u nešto što se zove nasleđivanje (*inheritance*), ali za sada neću detaljnije komentarisati. O zaštićenom pristupu ću govoriti kasnije u poglavlju "Metode", a o nasleđivanju u poglavlju "Nasleđivanje". Suština ovoga je da zaštićena oblast klase sadrži delove klase koje neko ko proširuje klasu može poželeti da upozna.

Najavljeni nivo pristupa se koristi prilikom kreiranja komponenti. Svaka komponenta deklarisana u odeljku published, će se pojaviti u Object Inspector-u u toku dizajniranja. O odeljku published će više biti reči u lekciji dana 20, "Kreiranje komponenti".

Jezik Object Pascal sadrži četiri ključne reči koje se odnose na pristup klasama. Ključne reči su (ne slučajno) `public`, `private`, `protected` i `published`. Nivo pristupa klasi definišete kada deklarišete klasu. Klasa se deklariše ključnom reči `class`. Evo primera:

```
TVehicle = class
private
CurrentGear : Integer;
  Started : Boolean;
```

```
  Speed : Integer;
  procedure StartElectricalSystem;
  procedure StartEngine;
protected
  procedure StartupProcedure;
public
  HaveKey : Boolean;
  Start : Boolean;
  procedure SetGear(Gear : Integer);
  procedure Accelerate(Acceleration : Integer);
  procedure Brake(Factor : Integer);
  procedure Turn(Direction : Integer);
  procedure ShutDown;
end;
```

Uočite da je organizacija klasa podeljena na tri nivoa pristupa. Ne morate koristiti sve nivoe pristupa za Vaše klase. U ovom primeru nisam koristio nivo pristupa `published`. Nije potrebno da koristite bilo koji od nivoa pristupa ukoliko to ne želite, ali ćete najverovatnije i u najjednostavnijem slučaju imati odeljke `public` i `private`.

Konstruktori

Klase u jeziku Object Pascal sadrže specijalne metode koje se zovu konstruktori.

Konstruktor (*constructor*) su metode koje se koriste za kreiranje slučajeva klase.

Konstruktor se koristi za inicijalizaciju bilo koje promenljive klase, za rezervisanje memorije koja je potrebna klasi, odnosno za obavljanje zadataka prilikom pokretanja klase. Primer `TVehicle` koji ste upravo razmatrali nema konstruktor. Ukoliko ne obezbedite konstruktor, možete koristiti osnovni konstruktor klasa kada kreirate klase. (Ako drugačije nije definisano, sve klase Object Pascal-a se granaju iz `TObject`. Klasa `TObject` ima konstruktor pod nazivom `Create`, pa se ovaj konstruktor poziva u slučaju da konstruktor nije obezbeđen. O osnovnim klasama i nasleđivanju će biti reči kasnije u poglavlju "Nasleđivanje".) Sve dok koristite osnovni konstruktor klasa što je u redu kod jednostavnih klasa, uglavnom ćete obezbeđivati konstruktor za klase koje imaju nekog značaja. Konstruktor može imati proizvoljan naziv, ali mora biti deklarisan korišenjem ključne reči `constructor`. To će ga izdvojiti kao konstruktor. Na osnovu toga, dodajmo deklaraciju konstruktora klasi `TVehicle`:

```
TVehicle = class
private
  CurrentGear : Integer;
  Started : Boolean;
  Speed : Integer;
  procedure StartElectricalSystem;
```

```
  procedure StartEngine;
protected
  procedure StartupProcedure;
public
  HaveKey : Boolean;
  Start : Boolean;
  procedure SetGear(Gear : Integer);
  procedure Accelerate(Acceleration : Integer);
  procedure Break(Factor : Integer);
  procedure Turn(Direction : Integer);
  procedure ShutDown;
  constructor Create;  { the constructor }
end;
```

Uočite da je konstruktor poseban tip metoda. Pošto ne vraća vrednost, ne sadrži tip za vraćanje. Ukoliko želite da dodate tip za vraćanje u deklaraciju konstruktora, prevodilac će prijaviti grešku.

Klasa može imati više od jednog konstruktora. Ovo može biti ostvareno na dva različita načina. Prvi način je jednostavno dodeljivanje drugačijeg naziva konstruktoru - na primer,

```
TVehicle = class
  { rest of class deleted }
  constructor Create;
  constructor CreateModel(Model : string);
end;
```

Ovaj primer prikazuje dva konstruktora, jedan sa nazivom `Create`, a drugi sa nazivom `CreateModel`.

Još jedan način za deklarisanje višestrukih konstruktora je preko preopterećenja metoda, o čemu je bilo reči u jučerašnjoj lekciji. Evo primera koji koristi konstruktore sa istim nazivom, ali različitim parametrima:

```
TVehicle = class
  { rest of class deleted }
  constructor Create; overload;
  constructor Create(AOwner : TObject);  overload;
end;
```

Pošto je preopterećenje metoda novina u Delphiju 4, ne očekujem da se koriste višestruki konstruktori često u Delphi programima. Tradicionalni metod deklarisanja konstruktora sa različitim nazivima će, verujem, nastaviti da bude trend bar još neko vreme. Ipak, oba metoda su ispravna i bilo koji od njih se može koristiti.

Ako kreirate komponente za prodaju, treba da se uverite da konstruktori Vaših komponenti imaju različite liste parametara. Ovo će osigurati da Vaše komponente mogu raditi zajedno sa jezikom C++Builder, isto kao što rade sa Delphijem (C++Builder nema zasebne nazive za konstruktore, pa se koristi preopterećenje metoda za odvajanje konstruktora). Čak, iako ne planirate da prodajete Vaše komponente na tržištu programskog jezika C++Builder, bilo bi pametno da unapred predvidite ovu mogućnost.

Koja je svrha višestrukih konstruktora? Višestruki konstruktori omogućavaju različite načine kreiranja klasa. Na primer, klasa može imati konstruktor koji ne uzima parametre, a može imati konstruktor koji uzima jedan, ili više parametara, da bi inicijalizovala polja određenim vrednostima. Na primer, recimo da imate klasu pod nazivom `TMyRect` koja enkapsulira pravougaonik (pravougaonici se često koriste u Windows programiranju). Ova klasa može imati nekoliko konstruktora. Može imati generički konstruktor koji setuje sva polja na `0`, dok drugi konstruktori omogućavaju da se polja klase setuju kroz konstruktor. Prvo pogledajmo kako bi mogla izgledati deklaracija klase:

```
TMyRect = class
private
  Left : Integer;
  Top : Integer;
  Right : Integer;
  Bottom : Integer;
public
  function GetWidth : Integer;
  function GetHeight : Integer;
  procedure SetRect(ALeft, ATop, ARight, ABottom : Integer);
  constructor Create;
  constructor CreateVal(ALeft, ATop, ARight, ABottom : Integer);
end;
```

Definicija konstruktora može izgledati ovako:

```
constructor TMyRect.Create;
begin
  inherited Create;
  Left   := 0;
  Top    := 0;
  Right  := 0;
  Bottom := 0;
end;

constructor TMyRect.CreateVal(ALeft, ATop, ARight, ABottom :
Integer);
begin
  inherited Create;
  Left   := ALeft;
  Top    := ATop;
  Right  := ARight;
  Bottom := ABottom;
end;
```

Prvi konstruktor jednostavno inicijalizuje svako polje na `0`. Drugi konstruktor uzima parametre koji su prosleđeni i dodeljuje ih odgovarajućim poljima klase. Nazivi promenljivih u listama parametara su deklarisani kao lokalne promenljive u okviru konstruktora, pa svaki naziv promenljive počinje sa A da bi ih razlikovali od lokalnih

promenljivih i polja klase (korišćenje početnog slova A je uobičajeno u Delphi programima). Uočite korišćenje ključne reči `inherited` u konstruktorima. O ključnoj reči `inherited` će biti reči kasnije u poglavlju "Nasleđivanje". Želeo bih ovo da napomenem samo da Vas ne bih ostavio u neznanju.

Nije obavezno da se polja inicijalizuju sa `0` kao što je to slučaj sa konstruktorom `Create`. Sva polja se automatski inicijalizuju na `0` prilikom kreiranja objekta klase.

Kreiranje slučaja je kreiranje objekta klase koji nazivamo slučajem (*instance*).

Pa, kako da koristite jedan od ovih konstruktora umesto drugog? Ovo možete raditi kada kreirate slučaj u okviru klase. Sledeći isečak koda kreira dva slučaja klase `TMyRect`. Prvi koristi konstruktor `Create`, a drugi konstruktor `CreateVal`:

```
ar
  Rect1 : TMyRect;
  Rect2 : TMyRect;
begin
  Rect1 := TMyRect.Create;
  Rect2 := TMyRect.CreateVal(0, 0, 100, 100);
end;
```

Možete kreirati koliko god želite konstruktora, ali svi konstruktori moraju imati različite nazive, odnosno ako želite da kreirate preopterećene konstruktore, isti moraju biti kreirani po pravilima preopterećenih metoda.

Želeo bih da napomenem jednu stvar vezanu za prethodni primer: oba slučaja klase `TMyRect` su dinamički raspoređena u memoriji. U prethodnom delu knjige sam napomenuo da možete dinamički rezervisari memoriju za objekt pozivom procedure `GetMem`. Možda će Vam se učiniti da protivurečim samom sebi, mada u suštini to nije tačno. Razlog je što se memorija za klase jezika Object Pascal uvek rezerviše dinamički. Mada to nije tačno kod slogova, tačno je kod klasa. To, takođe, znači da prethodni isečak koda troši memoriju pošto nije oslobodio memoriju dodeljenu dvema klasama. O tome će biti reči kasnije. Pošto sve klase jezika Object Pascal bivaju kreirane odjednom, sve promenljive klase su, naravno, pointeri. Promenljive `Rect1` i `Rect2` u prethodnom primeru su pointeri klase `TMyRect`.

Destruktori (destructors)

Destruktor (*destructor*) je poseban metod koji se automatski poziva nakon uništenja objekta.

Destruktor se može posmatrati kao suprotnost konstruktoru. Destruktori se obično koriste da oslobode memoriju koju su rezervisale klase, odnosno da odradi neki drugi posao čišćenja. Nije neophodno da klasa poseduje destruktor, pošto se može koristiti osnovni destruktor klasa. Kao što je to slučaj kod konstruktora, destruktori ne vraćaju vrednost.

Iako klase mogu imati više destruktora, uobičajeno je da se ova mogućnost ne primenjuje. Ukoliko imate samo jedan destruktor, trebalo bi da ga nazovete `Destroy`. Ovo je više nego tradicija. Kada oslobodite slučaj klase (uklonite je iz memorije), pozivate metod `Free`. Metoda `Free` pripada klasi `TObject` koja poziva `Destroy` metodu klase, trenutak pre no što klasa bude uklonjena iz memorije. Ovo je uobičajen način oslobađanja memorije koja je pridružena klasi. Evo primera:

```
Rect1 := TMyRect.Create;
{ Do some things with Rect1. }
{ ... }
{ Now delete Rect1. }
Rect1.Free;
```

Primer u poglavlju "Konstruktori" rasipa memoriju pošto oba objekta `TMyRect` nisu oslobođena.

Sledeći primer pokazuje prepravljeni kod klase `TMyRect`, zajedno sa destruktorom (deo koda je uklonjen zbog skraćivanja primera):

```
TMyRect = class
private
  Left : Integer;
  Top : Integer;
  Right : Integer;
  Bottom : Integer;
  Text : PChar;     { new field }
public
  function GetWidth : Integer;
  function GetHeight : Integer;
  procedure SetRect(ALeft, ATop, ARight, ABottom : Integer);
  constructor Create;
  constructor CreateVal(ALeft, ATop, ARight, ABottom : Integer);
  destructor Destroy; override;
end;

constructor TMyRect.Create;
begin
  inherited Create;
  { Allocate memory for a null-terminated string. }
  Text := AllocMem(1024);
end;

destructor TMyRect.Destroy;
begin
  { Free the allocated memory. }
  FreeMem(Text);
  inherited Destroy;
end;
```

Modifikovana verzija klase `TMyRect` rezerviše mesto za string terminisan nulom (`PChart`) pod nazivom `Text`, svojim konstruktorom, a oslobađa rezervisano mesto

destruktorom. (Boljeg primera od klase koja upravlja pravougaonicima za prihvat tekst polja ne mogu da se setim, ali nikad se ne zna! Ovo je ipak samo primer.)

Pogledajte detaljnije destruktor u deklaraciji klase `TMyRect`. Ona izgleda ovako:

```
destructor Destroy; override;
```

Uočite ključnu reč `Override` na kraju deklaracije. Ova ključna reč ukazuje prevodiocu da preopterećujete metod koji se takođe može pronaći u baznoj klasi. Opet Vam govorim nešto što ćete tek učiti, o ovom slučaju će više biti reči u poglavlju pod nazivom "Nasleđivanje". (Pošto ovo stalno napominjem, kladim se da očekujete da će ovo poglavlje biti veoma dobro!)

Uobičajeno je da iskaz inherited koristite kao prvi iskaz u okviru konstruktora, odnosno kao zadnji iskaz u okviru destruktora.

Polja podataka (data fields)

Polja podataka (*data fields*) klasa su jednostavne promenljive koje se deklarišu u okviru deklaracije klase; one se mogu smatrati promenljivima koje imaju opseg klasa. Polja u klasama su u osnovi ista kao i polja u slogovima, s tom razlikom što se pristup poljima u okviru klasa može kontrolisati deklarisanjem istih kao privatna, javna, odnosno zaštićena. Nezavisno od načina pristupa, polja klasa su dostupna bilo kom metodu u okviru klase. U zavisnosti od nivoa pristupa poljima klase, ista mogu biti vidljiva i van klase. Privatna i zaštićena polja, na primer, su vidljiva samo u okviru klase i ne može im se pristupiti van klase u kojoj su definisana. Uzmite kao primer prethodno deklarisanu klasu `TMyRect`. Ona ne sadrži javna polja. Ukoliko pokušate da joj pristupite na način koji je naveden u sledećem primeru, prevodilac će prijaviti grešku:

```
Rect := TMyRect.CreateVal(0, 0, 100, 100);
Rect.Left := 20; { compiler error! }
```

Greška prevodioca glasi: `Undeclared identifier: 'Left'`. (Nedefinisan identifikator: `'Left'`.) Prevodilac želi da Vam saopšti da je `Left` privatno polje, i da ne možete da mu pristupite. Ukoliko se polje `Left` deklariše u odeljku `public` u okviru deklaracije klase, ovaj kod ćete moći prevesti.

Prethodna tvrdnja vezana za privatna polja podataka je tačna, ukoliko je klasa `TMyRect` deklarisana u odvojenom junitu, ali nije tačna ukoliko je klasa `TMyRect` deklarisana u junitu u kom se koristi. Klase koje se nalaze u istom junitu imaju nešto što se zove privilegije prijatelja (friend privileges), što znači da klase mogu pristupati privatnim poljima drugih klasa. Ovo važi samo za klase koji su deklarisane u istom junitu.

Object Pascal koristi karakteristike da bi kontrolisao pristup privatnim poljima. Karakteristike mogu biti čitaj/piši (read/write), samo - čitanje (read - only), odnosno samo - piši (write - only) (iako su karakteristike samo - pisanje retke).

Karakteristike mogu imati metodu za čitanje koji se poziva kada se karakteristika čita, a metodu za pisanje kada se karakteristika zapisuje. Nijedna od metoda nije neophodna, pošto karakteristike mogu imati direktan pristup privatnim poljima. Ove metode za čitanje i pisanje se pozivaju svaki put kada se želi pristupiti karakteristici. Metoda za pisanje je naročitio važna pošto se može koristiti za proveru unosa, odnosno može nositi bilo koji drugi zadatak kada se karakteristici dodeljuje vrednost. Na ovaj način privatnom polju se nikad ne pristupa direktno, već uvek preko karakteristike. Opet pričam unapred, pa ću detaljnije objašnjenje ostaviti za kasnije. O karakteristikama će biti više reči i detaljnije će biti obrađene u lekciji dana 5.

Neki OOP ekstremisti tvrde da polja nikad ne bi trebalo da budu javna. Oni će Vas savetovati da koristite karakteristike za pristup svim poljima. Na drugoj strani je grupa koja preporučuje da sva svoja polja deklarišete kao javna. Istina se nalazi negde u sredini. Neka polja nisu kritična i mogu biti ostavljena kao javna ukoliko je to zgodno. Ostala polja kod kojih je kritičan način na koji klasa upravlja njima ne treba da budu deklarisana kao javna. Ukoliko pravite grešku, bolje je da napravite grešku deklarišući polja kao privatna.

Kada kreirate slučaj klase, svaka klasa mora imati sopstvene podatke. Promenljivoj `Left` možete dodeliti vrednost u jednom slučaju klase, a neku drugu vrednost u drugom slučaju klase; na primer:

```
Rect1 := TMyRect.CreateVal(100, 100, 500, 500);
Rect2 := TMyRect.CreateVal(0, 0, 100, 100);
```

Ovaj kod kreira dva slučaja klase `TMyRect`. Iako su ova dva slučaja identična po strukturi, oni su potpuno odvojeni u memoriji. Svaki slučaj ima sopstvene podatke. U prvom primeru, polje `Left` će imati vrednost `100`. U drugom slučaju će imati vrednost `0`. Ovo liči na nova kola u izlogu. Svaki primerak određenog tipa kola izlazi iz istog kalupa, ali su svi primerci zasebni objekti. Oni se mogu razlikovati po boji, unutrašnjem uređenju, karakteristikama i slično.

Metode (methods)

Metode (*methods*) su funkcije i procedure koje pripadaju Vašoj klasi. One su lokalne i ne postoje van klase. Metode se mogu pozvati samo iz klase, odnosno preko slučaja klase. Metode imaju pristup svim javnim, zaštićenim i privatnim poljima klase. Metode mogu biti deklarisane u `private`, `protected`, ili `public` odeljku klase. Dobar dizajn klase zahteva da razmislite o odeljcima u kojima će se naći Vaše metode.

- Javne metode (*public methods*) zajedno sa karakteristikama predstavljaju korisnički interfejs klase. Preko javnih metoda korisnici mogu pristupiti klasi kako bi im bile dostupne sve mogućnosti koje klasa pruža. Uzmimo na primer, da imate klasu koja svira i snima audio zapise. Javne metode mogu uključiti metode pod nazivom `Open`, `Play`, `Record`, `Save`, `Rewind` itd.

- Privatne metode (*private methods*) su metode koje klasa interno koristi da radi stvari za sebe. Ove metode nisu predviđene za pozivanje od strane korisnika klase; one su privatne u cilju njihovog sakrivanja od spoljašnjeg sveta. često klasa sadrži početni posao koji se izvršava nakon što je klasa kreirana (na primer, već ste videli da se konstruktor poziva nakon što je klasa kreirana). Kod nekih klasa početak izvršavanja može biti pozamašan, zahtevajući dosta linija koda. Da otklonimo zbrku oko konstruktora; klasa može imati metod `Init` koji se poziva iz konstruktora, kako bi izvršio početne zadatke. Ovaj metod nikad ne bi trebao da bude pozvan od strane korisnika klase. Ustvari, više nego očigledno je da se loše stvari mogu desiti ukoliko korisnik pozove ovaj metod u pogrešnom trenutku, stoga je ovaj metod privatan kako bi zaštitio integritet klase i korisnika.
- Zaštićene metode (*protected methods*) su metode kojima se ne može pristupiti iz spoljašnjeg sveta, ali moguće im je pristupiti klasama nastalim iz tekuće klase. Još nisam govorio o klasama koje su izdvojene iz drugih klasa. O ovome će biti više reči kasnije, kada to bude imali više smisla. Izdvojene klase će biti obrađene u poglavlju "Nasleđivanje".

Metode mogu biti deklarisane kao klasne metode (*class methods*). Klasne metode operišu više kao regularne funkcije, odnosno procedure, nego kao metode klasa. Karakteristično je da klasne metode ne mogu pristupati poljima, odnosno, drugim metodama u okviru klasa. (Nešto kasnije ću Vam saopštiti zašto ova restrikcija postoji.) U većini slučajeva nećete koristiti klasne metode, pa ih neću detaljnije objašnjavati.

O polju `Self`

NOVI TERMIN Sve klase imaju skriveno polje pod nazivom `Self`. Polje `Self` je pointer na slučaj klase koji se nalazi u memoriji.

Očigledno je da ova definicija zahteva objašnjenje. Prvo pogledajmo kako bi klasa `TMyRect` izgledala ukoliko polje `Self` ne bi bilo skriveno:

```
TMyRect = class
private
  Self : TMyRect;
  Left : Integer;
  Top : Integer;
  Right : Integer;
  Bottom : Integer;
  Text : PChar;
public
  function GetWidth : Integer;
  function GetHeight : Integer;
  procedure SetRect(ALeft, ATop, ARight, ABottom : Integer);
  constructor Create;
```

```
  constructor CreateVal(ALeft, ATop, ARight, ABottom : Integer);
  destructor Destroy; override;
end;
```

Ovako klasa `TMyRect` izgleda prevodiocu. Kada se kreira objekt class, pointer `Self` se automatski inicijalizuje na adresi na kojoj se nalazi klasa u okviru memorije:

```
Rect := TMyRect.CreateVal(0, 0, 100, 100);
{ Now 'Rect' and 'Rect.Self' have the same value   }
{ because both contain the address of the object in memory. }
```

Sigurno postavljate pitanje: "Šta u stvari predstavlja `Self`?" Zapamtite da svaki slučaj klase ima sopstvene kopije polja klase, ali svi slučajevi klasa dele isti skup metoda za klasu (nema svrhe duplirati ovaj kod za svaki slučaj klase). Kako prevodilac shvata koji slučaj ide sa kojim pozivom metoda? Sve metode klase imaju skriven parametar `Self` koji im je priključen. Da bi ovo ilustrovali, recimo da imate funkciju za klasu `TMyRect` pod nazivom `GetWidth`. To bi izgledalo otprilike ovako:

```
function TMyRect.GetWidth : Integer;
begin
  Result := Right - Left;
end;
```

Ovako funkcija izgleda nama. Prevodiocu, pak, ona izgleda drugačije:

```
function TMyRect.GetWidth : Integer;
begin
  Result := Self.Right - Self.Left;
end;
```

Ovo baš i nije tačno sa tehničkog aspekta, ali je dovoljno blisko tačnom objašnjenju. U ovom kodu možete videti da `Self` radi u pozadini, da bi Vam sve bilo jasnije. Ne treba da brinete o tome kako se sve ovo odvija, ali treba da znate da se ipak dešava.

Nikad nemojte modifikovati pointer `Self` .Možete ga koristiti da prosledite pointer Vaše klase drugim metodama, odnosno kao parametar u konstruisanju drugih klasa, ali nemojte menjati njegovu vrednost. Naučite da tretirate pointer `Self` kao promenljivu koja se može samo čitati (read -only).

Sve dok pointer `Self` radi u pozadini, možete pristupiti u okviru klase. Kao ilustraciju kratko pogledajmo VCL. Uglavnom ćete kreirati komponente u VCL-u prebacujući ih na formu u toku dizajniranja. Kad ovo budete radili, Delphi kreira pointer na komponentu i radi sve vrste poslova za spremanje umesto Vas, omogućavajući Vam da se brinete samo o tehničkim pojedinostima. Ponekad ćete, naravno, kreirati komponente u toku izvršavanja programa. VCL insistira (kao što to čine svi dobri kosturi programa), na praćenju veza između objekata; odnosno koji objekt pripada kom objektu roditelju. Na primer, recimo da želite da kreirate dugme na formi nakon što ste kliknuli na drugo dugme. Potrebno je da saopštite VCL-u šta je roditelj novog dugmeta. Kod bi trebao da izgleda ovako:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Button : TButton;
begin
  Button := TButton.Create(Self);
  Button.Parent  := Self;
  Button.Left    := 20;
  Button.Top     := 20;
  Button.Caption := 'Click Me';
end;
```

U ovom kodu možete primetiti da se `Self` koristi kao konstruktor (setuje karakteristiku `Owner` dugmeta,; ovo će biti obrađeno kasnije u lekciji dana 7, "VCL komponente", kada budu obrađene komponente VCL-a) i da je dodeljen karakteristici Parent novokreirane komponente dugme. Na ovaj način ćete kreirati pointer `Self`, kako bi uštedeli većinu vremena na kreiranju Vaših Delphi aplikacija.

U prethodnom delu knjige sam objasnio da klasne metode ne mogu pristupiti poljima klase. Razlog zašto je ovo tačno, leži u činjenici da klasne metode nemaju skriveni parametar `Self`, za razliku od običnih metoda. Ukoliko metode nemaju parametar `Self`, ne mogu pristupiti poljima klase.

Nemojte suviše brinuti u ovom trenutku o parametru `Self`. Kada budete počeli da koristite VCL, brzo će Vam biti jasno kada je potrebno da u svojim aplikacijama koristite parametar `Self`.

Primer klasa

Bilo bi dobro ukoliko biste u ovom trenutku mogli da vidite primer klase. Listing 3.1 pokazuje junit koji sadrži klasu pod nazivom `TAirplane`. Ova klasa se može koristiti za program kontrole aviona. Klasa Vam omogućava da upravljate avionom šaljući mu poruku. Možete da naredite avionu da uzleti, sleti, odnosno promeni kurs, visinu, ili brzinu. Prvo pogledajte junit, a zatim ćemo prodiskutovati šta se dešava u okviru klase.

Listing 3.1: `AIRPLANU.PAS`

```
unit AirplanU;

interface

uses
  SysUtils;

const
  { Airplane types. }
  Airliner     = 0;
  Commuter     = 1;
  PrivateCraft = 2;
```

nastavlja se

Listing 3.1: AIRPLANU.PAS *nastavak*

```
  { Status constants. }
  TakingOff    = 0;
  Cruising     = 1;
  Landing      = 2;
  OnRamp       = 3;

  { Message constants. }
  MsgChange    = 0;
  MsgTakeOff   = 1;
  MsgLand      = 2;
  MsgReport    = 3;

type

  TAirplane = class
  private
    Name     : string;
    Speed    : Integer;
    Altitude : Integer;
    Heading  : Integer;
    Status   : Integer;
    Kind     : Integer;
    Ceiling  : Integer;
  protected
    procedure TakeOff(Dir : Integer); virtual;
    procedure Land; virtual;
  public
    constructor Create(AName : string; AKind : Integer = Airliner);
    function SendMessage(Msg : Integer; var Response : string;
       Spd : Integer; Dir : Integer; Alt : Integer) : Boolean;
    function GetStatus(var StatusString : string) : Integer; over-
load;
➥virtual;
    function GetStatus : Integer; overload;
    function GetSpeed : Integer;
continues
    function GetHeading : Integer;
    function GetAltitude : Integer;
    function GetName : string;
  end;

implementation

constructor TAirplane.Create(AName : string; AKind : Integer);
begin
  inherited Create;
```

```
  Name     := AName;
  Kind     := AKind;
  Status   := OnRamp;
  case Kind of
    Airliner : Ceiling := 35000;
    Commuter : Ceiling := 20000;
    PrivateCraft : Ceiling := 8000;
  end;
end;

procedure TAirplane.TakeOff(Dir : Integer);
begin
  Heading := Dir;
  Status  := TakingOff;
end;

procedure TAirplane.Land;
begin
  Speed    := 0;
  Heading  := 0;
  Altitude := 0;
  Status   := OnRamp;
end;

function TAirplane.SendMessage(Msg : Integer; var Response : string;
   Spd : Integer; Dir : Integer; Alt : Integer) : Boolean;
begin
  Result := True;

  { Do something based on which command was sent. }
  case Msg of
    MsgTakeOff :
      { Can't take off if already in the air! }
      if status <> OnRamp then begin
        Response := Name + ': I''m already in the air!';
        Result := False;
      end else
        TakeOff(dir);
    MsgChange :
      begin
        { Check for bad commands and exit if any found. }
        if Spd > 500 then
          Response := 'Command Error: Speed cannot be more than
500.';
        if Dir > 360 then
          Response := 'Command Error: Heading cannot be over 360
➥degrees.';
        if Alt < 100 then
          Response := Name + ': I''d crash!';
        if Alt > Ceiling then
```

nastavlja se

Listing 3.1: AIRPLANU.PAS *nastavak*

```
            Response := Name + ': I can''t go that high.';
          if (Spd = 0) and (Dir = 0) and (Alt = 0) then
            Response := Name + ': Huh?';
          if Response <> '' then begin
            Result := False;
            Exit;
          end;

          { Can't change status if on the ground. }
          if status = OnRamp then begin
            Response := Name + ': I''m on the ground.';
            Result := False;
          end else begin
            Speed := Spd;
            Heading := Dir;
            Altitude := Alt;
            Status := Cruising;
          end;
        end;
      MsgLand :
        { Can't land if already on the ground. }
        if status = OnRamp then begin
          Response := Name + ': I''m already on the ground.';
          Result := False;
        end else
          Land;
      MsgReport :
        begin
          GetStatus(Response);
          Exit;
        end;
    end;

    { Standard response if all went well. }
    if Result then
      Response := Name + ': Roger.';
  end;
  continues

  function TAirplane.GetStatus(var StatusString : string) : Integer;
  begin
    StatusString := Format('%s, Altitude: %d, Heading: %d, ' +
      'Speed: %d', [Name, Altitude, Heading, Speed]);
    Result := Status;
  end;

  function TAirplane.GetStatus : Integer;
  begin
```

```
  Result := Status;
end;

function TAirplane.GetSpeed : Integer;
begin
  Result := Speed;
end;

function TAirplane.GetHeading : Integer;
begin
  Result := Heading;
end;

function TAirplane.GetAltitude : Integer;
begin
  Result := Altitude;
end;

function TAirplane.GetName : string;
begin
  Result := Name;
end;
```

Prvo pogledajte deklaraciju klase u okviru odeljka `interface`. Uočite da klasa `TAirplane` sadrži preopterećenu funkciju pod nazivom `GetStatus`. Ukoliko je pozivate string parametrom, klasa `GetStatus` će vratiti statusni string, a isto tako i statusne podatke(parametar string je promenljivi parametar- promenljiva). Ukoliko funkciju `GetStatus` pozivate bez parametara, ista će vratiti samo promenljivu `Status`. Uočite da je jedini način za pristupanje privatnim poljima, pristup preko javnih metoda. Na primer, možete menjati brzinu, visinu, odnosno, naziv aviona jedino ako mu pošaljete poruku. Iskoristimo analogiju sa kontrolorom vazdušnog saobraćaja koji nije u stanju da promeni oznaku (naziv) aviona. Najbolji način bi bio da pošalje poruku pilotu i saopšti mu da promeni oznaku (naziv) aviona.

Ova klasa ima najviše koristi od karakteristika. Kao što sam ranije rekao, u ovom trenutku nećemo obrađivati karakteristike detaljnije, pa bih morao da priznam da ova klasa može puno bolje da izgleda i može biti unapređena.

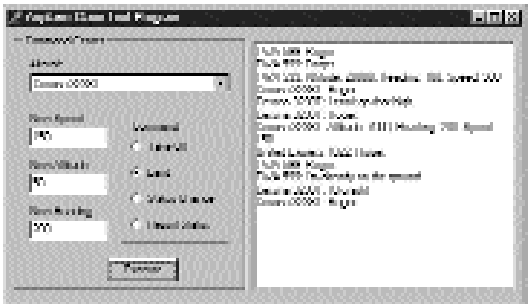
Vratimo se na definiciju klase `TAirplane` u odeljku interface. Konstruktor u ovom delu obavlja poslove inicijalizacije. Verovatno ste zapazili da funkcija `SendMessage` radi većinu posla. Iskaz `case` određuje koja će poruka biti poslata i izvršava odgovarajuće dejstvo. Uočite da procedure `TakeOff` i `Land` ne mogu biti direktno pozvane (zaštićene su), odnosno mogu biti pozvane samo preko funkcije `SendMessage`. Još jednom, u ulozi kontrolora vazdušnog saobraćaja, avionom ne možete da sleteti, odnosno uzleteti, možete mu samo poslati poruku saopštavajući mu šta želite da uradi.

Postoji još nešto o čemu još nije bilo reči. Uočite ključnu reč `virtual`. Ona definiše funkciju kao virtuelna metoda.

NOVI TERMIN Virtuelna metoda (*virtual method*) je metoda koja se automatski poziva ukoliko metoda sa istim nazivom postoji u odvojenoj klasi.

O virtuelnim metodama će više biti reči u sledećem poglavlju, ali želeo sam, da Vam ih u ovom trenutku spomenem.

Programi u okviru knjige sadrže i program pod nazivom `Airport`, koji Vam omogućava da se igrate kontrolora vazdušnog saobraćaja.(izvorne kodove programa u ovoj knjizi možete pronaći na web sajtu `http://www.mcp.com/info`. Ukucajte ISBN broj knjige: 0-672-31286-7.) Program prvo setuje niz klase `TAirPlane` a zatim kreira tri slučaja klase `TAirPlane`. Poruke možete slati avionima odabirajući ih, podešavajući parametre za poruke, a zatim pritiskajući dugme Execute. Klik na dugme rezultira pozivom na funkciju `SendMessage` odabranog aviona. Kada pošaljete poruku, odgovor dobijate od aviona, i taj odgovor se prikazuje u okviru memo komponente. Pokrenite program i igrajte se da biste dobili osećaj kako radi. Slika 3.1 prikazuje kako radi program `Airport`.



Slika 3.1
Program Airport u radu

Nasleđivanje(inheritance)

Jedna od najmoćnijih karakteristika klasa u jeziku Object Pascal je njihovo proširivanje kroz nasleđivanje.

NOVI TERMIN Nasleđivanje (*inheritance*) znači dodavanje novih funkcionalnih osobina klasi izdvajanjem nove klase iz stare.

NOVI TERMIN Klasa od koje se počinje, naziva se osnovna klasa (*base class*), odnosno klasa predak (*ancestor class*), a nova kllasa koju kreirate se naziva izdvojena klasa (derived class).

Da bi ovo ilustrovali vratimo se klasi `TAirplane`. Kao što znate, civilni i vojni pojmovi se veoma razlikuju. Da bi predstavili vojni avion, treba da izdvojimo klasu iz klase `TAirplane` i dodamo joj novu funkcionalnu karakteristiku:

```
TMilitaryPlane = class(TAirplane)
private
  TheMission : TMission;
  constructor Create(AName : string; AType : Integer);
  function GetStatus(var StatusString : string) : Integer; override;
protected
  procedure TakeOff(Dir : Integer); override;
  procedure Land; override;
  procedure Attack; virtual;
  procedure SetMission; virtual;
end;
```

Klasa `TMilitaryPlane` sadrži sve što i klasa `TAirplane`, uz dodatak nekoliko novih mogućnosti. Uočite prvu liniju deklaracije klase. Naziv klase u zagradi, nakon ključne reči `class` se koristi da saopšti prevodiocu da se klasa `TMilitaryPlane` nasleđuje od klase `TAirplane`. Klasa iz koje izdvajam navedenu klasu je osnovna klasa, što je u ovom slučaju klasa `TAirplane`.

Kada izdvajate klasu iz neke druge klase, nova klasa ima sve funkcionalne osobine osnovne klase, uz dodatak mogućnosti koje ste dodali.U novoj klasi možete dodati polja i metode, ali ne možete ukloniti ništa što stara klasa nudi.

Uočićete da u odeljku `private` postoji linija koja deklariše promenljivu klase `TMission`. Klasa `TMission` ovde nije prikazana, ali ista može da enkapsulira sve osobine misije vojnih aviona: cilj, navigacione tačke, ulazne i izlazne visine i njihove nazive, itd. Ovo ilustruje korišćenje polja koje je slučaj druge klase. Ustvari, dosta ćete toga saznati u toku programiranja na Delphiju.

Preskakanje metoda (overriding methods)

Želeo bih da iskoristim ovaj trenutak za diskusiju o virtuelnim metodama. Uočite da je procedura `TakeOff` virtuelna metoda u okviru klase `TAirplane`(pozivam se na Listing 3.1). Uočite da se procedura `TakeOff` poziva iz funkcije `SendMessage` kao odgovor na poruku `MsgTakeOff`. Ukoliko klasa `TMilitaryPlane` ne pruža svoj metod `TakeOff`, metoda osnovne klase pod istim nazivom će biti pozvana. Pošto klasa `TMilitaryPlane` sadrži metodu `TakeOff`, ista će biti pozvana umesto metode osnovne klase.

Zamena metode osnovne klase u okviru izdvojene klase se naziva preskakanje metoda.

Da bi preskakanje funkcionisalo, potpis metode mora biti potpuno isti sa metodom u okviru glavne klase. Drugim rečima, tip promenljive koja se vraća naziv metode i lista parametara koja se vraća moraju biti isti kao kod metode osnovne klase. Dodatno, metode u izdvojenoj klasi moraju biti deklarisane ključnom reči override.

NAPOMENA Object Pascal takođe ima dinamičke metode. Dinamičke metode mogu biti tretirane isto kao i virtuelne metode, a to je stav većine programera. Razlika je u načinu na koji se smeštaju pointeri metoda u tabelu virtuelnog metoda (VMT-virtual method table). Nije važno da shvatite u ovom trenutku razliku, ali želeo bih da saznate nešto o dinamičkim metodama u slučaju da naiđete na njih pregledajući primere Delphi programa, odnosno VCL izvornih kodova. U većini slučajeva, dinamičke metode možete tretirati kao što u svojim programima tretirate virtuelne metode.

Metod možete preskočiti sa namerom da zamenite metod osnovne klase, odnosno možete ga preskočiti da biste izmenili metod osnovne klase. Kao primer, razmotrite metod `TakeOff`. Ako želite da u potpunosti zamenite metod `TakeOff` klase `TAirplane`, možete ga preskočiti i ubaciti kod koji želite:

```
procedure TMilitaryPlane.TakeOff(Dir : Integer);
begin
  { New code here. }
end;
```

Ukoliko želite da Vaš metod ima funkcionalnost osnovne klase na koju se dodaje, prvo treba da pozovete metodu osnovne klase a zatim da dodate novi kod. Pozivanje metoda osnovne klase se vrši ključnom reči inherited. Sledi primer:

```
procedure TMilitaryPlane.TakeOff(Dir : Integer);
begin
  { First call the base class TakeOff method. }
  inherited TakeOff(Dir);
  { New code here. }
end;
```

Pozivanjem metoda osnovne klase, dobićete ponašanje zapisano u osnovnoj klasi. Zatim možete da dodate kod pre, odnosno posle osnovne klase da bi izmenili metod osnovne klase. Zapazite da je metod `TakeOff` deklarisan u okviru odeljka protected, klase `TAirplane`. Ukoliko bi se nalazio u odeljku private, ovo ne bi funkcionisalo, pošto čak ni izdvojena klasa ne može pristupiti privatnim članovima klase pretka. Definisanjem metoda `TakeOff` kao zaštićenog, sakrivate ga od pristupa drugih klasa, dok je isti još uvek dostupan izdvojenoj klasi.

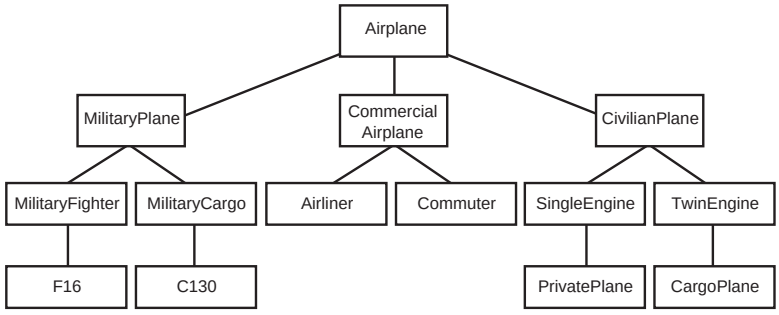
NAPOMENA Kod zaštićenog pristupa u odnosu na privatni pristup postoji izuzetak.Ukoliko je izdvojena klasa deklarisana u istom junitu kao i osnovna klasa, privatna polja i metode osnovne klase su dostupna izdvojenoj klasi. Ukoliko je izdvojena klasa deklarisana u izdvojeni junit, samo zaštićena polja i metode osnovne klase su dostupni izdvojenoj klasi.

Kada izdvajate klasu iz druge klase, morate biti sigurni da pozovete konstruktor osnovne klase kako bi sve klase preci bile adekvatno inicijalizovane. Pozivanje konstruktora osnovne klase se vrši takođe ključnom reči `inherited`. Pogledajte još jednom konstruktor klase `TAirplane`:

```
constructor TAirplane.Create(AName : string; AKind : Integer);
begin
  inherited Create;
```

```
  Name     := AName;
  Kind     := AKind;
  Status   := OnRamp;
  case Kind of
    Airliner : Ceiling := 35000;
    Commuter : Ceiling := 20000;
    PrivateCraft : Ceiling := 8000;
  end;
end;
```

Uočite da se poziva konstruktor osnovne klase `Create`, kako bi se osiguralo ispravno kreiranje klase. ”Hej, sačekaj malo!”- mogu čuti kako neki od Vas kažu. ”Klasa `TAirplane` nema osnovnu klasu!” Ustvari, ima je. Ukoliko osnovna klasa nije definisana, prilikom deklarisanja željene klase, automatski joj se kao osnovna, dodeljuje klasa `TObject`. Uverite se da pozivate konstruktor osnovne u okviru konstruktora Vaše klase. Slika 3.2 ilustruje koncept nasleđivanja:



Slika 3.2
Primer nasleđivanja

28670302.eps
31286-7
p2/v4

Na slici 3.2 možete primetiti da se klasa `F16` nasleđuje iz klase pod nazivom `MilitaryFighter`. Ustvari, klasa `F16` je nasleđena iz klase `TAirplane`, pošto je klasa `TAirplane` osnovna klasa za sve klase aviona (airplane).

Ključna reč class: `is` i `as`

Object Pascal ima dva operatora koja se odnose isključivo na klase. Operator is se koristi da odredi da li klasa pripada određenom tipu. Vratimo se na primer klasa `TAirplane` i `TMilitaryPlane`. Recimo da imate slučaj klase pod nazivom `Plane`. Ova klasa može biti slučaj klase `TAirPlane`, može biti slučaj klase `TMilitaryPlane`, odnosno može biti slučaj neke druge klase u isto vreme. Da biste ovo saznali, možete koristiti operator is. Na primer:

```
if Plane is TMilitaryPlane then
  Attack;
```

Operator is vraća Bulovu vrednost. Ukoliko promenljiva pripada željenom tipu, operator is vraća vrednost `True` (tačno).Ukoliko promenljiva ne pripada željenom tipu, operator is vraća `False` (netačno). Operator is će vratiti rezultat `True` i u slučaju da je željeni tip klase predak promenljive. Na primer, pošto je klasa `TMilitaryPlane` izvedena iz `TAirplane` , sledeći iskaz će biti tačan:

Pošto se sve klase izdvajaju iz klase `TObject`, sledeći iskaz će uvek biti tačan:

```
if AnyClass is TObject then
DoSomething;
```

Operator `is` se ne koristi toliko često kao operator as. Operator as se koristi da dodeli pointer određenom tipu klase. Evo kako to izgleda:

Operator as se obično koristi u sprezi sa operatorom with (tačno je, ova objašnjenja su pomalo zbunjujuća). U ovom isečku koda, promenljiva `Plane` je pointer koji može biti slučaj klase `TAirplane`, klase `TMilitaryPlane`, odnosno nijedne od njih. Operator as se koristi da prosledi pointer tipu klase `TMilitaryPlane`, nakon čega se poziva metod `Attack`. Ukoliko promenljiva `Plane` nije slučaj klase `TMilitaryPlane` (jedna od klasa predaka promenljive), ovo dodeljivanje neće uspeti, pa ni metod `Attack` neće biti pozvan. Ukoliko je naravno, promenljiva `Plane` pointer na slučaj klase `TMilitaryPlane`, dodeljivanje će biti uspešno, pa će metod `Attack` biti pozvan.

Zaključak

U današnjoj lekciji ste učili o klasama u okviru jezika Object Pascal. Dobro dizajnirana klasa je jednostavna za korišćenje i uštedeće Vam sate programiranja. Ići ću toliko daleko da kažem, da se dobro dizajnirana klasa koristi sa užitkom - pogotovo ako je sami kreirate.

Važno je da shvatite lekcije prva tri dana kako biste mogli da nastavite sa čitanjem knjige. Ukoliko Vam do sada sve u potpunosti nije bilo jasno, nemojte očajavati. U toku rada sa narednim lekcijama ćete primetiti da se ovi koncepti ponavljaju i koriste u programima koji imaju više praktičnih primena od kratkih, nekompletnih primera sa kojima ste do sada radili.

Učenje jezika ObjectPascal može prouzrokovati, odnosno sigurno će biti uzrok preopterećenja moždanih vijuga! Ovo je prirodno i stoga ne treba da budete zabrinuti. Možda ćete večeras zatvoriti ovu knjigu, ugasiti svetla i pomisliti: "Nikada ovo neću shvatiti." Verujte mi, hoćete.

Ponekad je neophodno da odmorite nekoliko dana i dopustite da se sve slegne. Ustvari, ukoliko bih mogao da tako nešto uradim, lekciju dana 4 bih ostavio kao prazno poglavlje pod nazivom" dan za odmor". Odmorite se na trenutak, pa ćete jednog od ovih dana biti kao Arhimed - trčaćete po kancelariji, odnosno kući, vičući "Eureka!", pošto se upalilo svetlo u Vašoj glavi. Vodite računa o odeći. Susedi Vas možda posmatraju.

Radionica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstite Vaše znanje o materiji koja je obrađena, odnosno vežbe koje omogućavaju da steknete iskustvo u korišćenju onoga što ste naučili. Odgovore na pitanja kviza možete pronaći u Dodatku A ," odgovori na kviz pitanja".

Pitanja i odgovori

P **Kako mogu da omogućim izdvojenim klasama da pozovu moju metodu, a da za ostale klase ona i dalje bude privatna?**

O Zaštitite je. Zaštićena metoda nije dostupna korisnicima Vaše klase, ali je dostupna izdvojenim klasama.

P **Šta znači apstrakcija podataka (*data abstraction*)?**

O Apstrakcija podataka predstavlja sakrivanje detalja klase koje korisnici iste ne treba da vide. Klasa može imati desetine polja i metoda, ali samo nekoliko polja, odnosno metoda može biti dostupno korisnicima. Učinite vidljivim samo one metode za koje korisnici treba da znaju.

P **Šta je to objekt?**

O Efektivno, objekt je blok koda koji se može tretirati kao zaseban entitet u okviru Vašeg programa. Uopšteno, u okviru jezika Object Pascal, objekt pre-

dstavlja klasu. U Delphiju ova definicija je proširena i uključuje VCL komponente. ActiveX kontrole su objekti, takođe.

P **Da li moje klase mogu imati više od jednog konstruktora?**

O Da.Vaše klase mogu imati koliko god želite konstruktora.

P **Da li su svi VCL objekti pointeri?**

O Da. Pošto su svi VCL objekti rezervisani odjednom, svi VCL objekti su pointeri.

P **Pointeri me zbunjuju. Da li sam jedini?**

O Ne! Pointeri su jedan od aspekata u programiranju jezikom Object Pascal koji najviše zbunjuju. Nakon što budete koristili Delphi određeno vreme, shvatićete pointere.

Kviz

1. Kako možete osloboditi skup svih vrednosti?
2. Koja je svrha uvođenja privatnih polja i metoda?
3. Kako možete da zadržite privatnost polja, a da omogućite korisniku da čita i menja njihove vrednosti?
4. Kada se poziva destruktor klase?
5. Šta znači preskakanje metoda osnovne klase?
6. Kako možete da preskočite metodu osnovne klase, a da još uvek imate koristi od operacija koje metode osnovne klase izvršavaju?
7. Koji operator se koristi da ukloni referencu pointera?
8. Da li klasa može da sadrži ostale slučajeve klasa kao svoja polja?
9. Koja se ključna reč koristi da bi se definisalo da pointer nema vrednost?
10. Za šta se koristi ključna reč as?

Vežbe

1. Napišite klasu koja uzima visinu osobe u inčima i vraća visinu u stopama.
2. Izdvojte klasu iz klase navedene u primeru 1 koja, takođe, vraća visinu u metrima, santimetrima, odnosno milimetrima. (napomena: u jednom inču ima 25,4 milimetra.)
3. Uzmite dan odmora. Zaslužili ste ga!

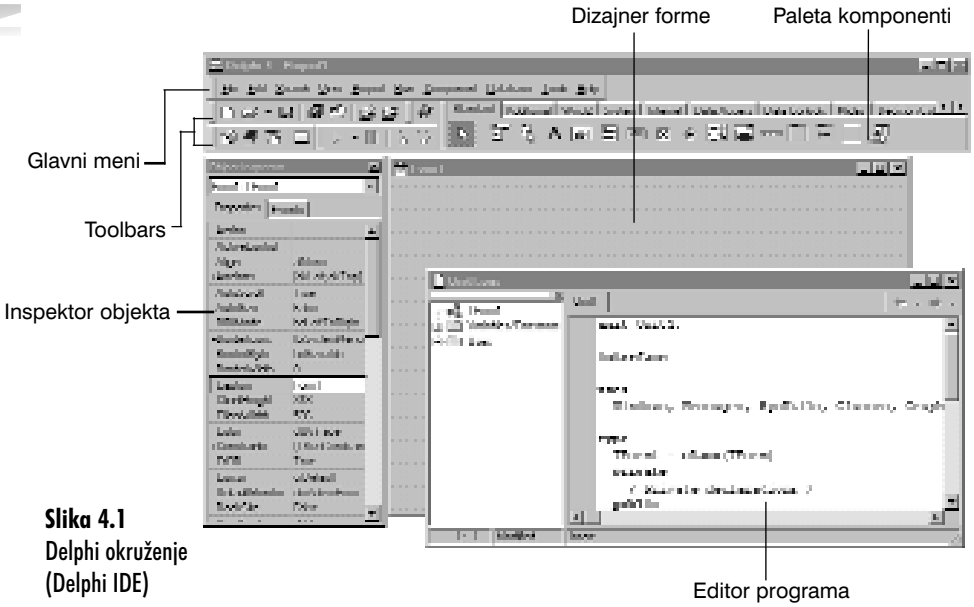
Dan 4

Istraživanje Delphijevog okruženja (Delphi IDE)

Jedan od najtežih aspekata prilikom učenja novog okruženja za programiranje je snalaženje u okruženju: shvatanje strukture osnovnog menija, kako i šta rade opcije menija, odnosno, kako u celini radi okruženje. Ukoliko ste početnik u programiranju, odnosno, počinjete da radite sa jezikom Object Pascal, ovaj zadatak komplikuje činjenica da istovremeno treba da naučite novi program (Delphi okruženje) i novi programski jezik. Ovo može biti ponekad veoma naporno. Učiniću sve da Vam učenje Delphi okruženja bude iskustvo u kom ćete uživati. Većim delom, učićete na primerima, što je mnogo interesantnije (da ne pominjem da je puno efikasnije).

Delphi okruženje (Delphi IDE)

Pre nego što počnemo, pogledajte sliku 4.1, a zatim pređimo na stvar. Uzgred, ako ste već koristili Delphi, možda ćete smatrati ovo poglavlje kao uvod. U tom slučaju, letimično pregledajte ovo poglavlje da bi pronašli pojedinosti koje prethodno niste znali, naročito delove koji su novine Delphija 4.



Slika 4.1
Delphi okruženje (Delphi IDE)

Delphi okruženje se sastoji od sledećih delova:

- glavnog menija i traka sa alatima
- palete komponenti (Component palette)
- Form Designer (dizajner forme)
- Code Editor (editor pro grama)
- Object Inspector (inspektor objekata)
- Code Explorer (istraživač programa)
- Project Manager (administrator projekta)

Sve ove delove ne mogu obraditi u jednom jedinom poglavlju, pa ću Vam u sledećih nekoliko poglavlja objasniti Delphi okruženje i ispitati detaljnije sve mogućnosti okruženja. U današnjoj lekciji ću početi sa obrađivanjem projekata, odnosno objasniću kako se projekti koriste za pisanje Delphi aplikacija. Nakon toga ću preći na Delphijevu traku sa alatima i na paletu komponenti. Zatim ću obraditi forme detaljnije nego u ovom poglavlju.

U toku rada ćete kreirati jednostavne programe kako bi ilustrovali razne mogućnosti Delphija. Takođe će detaljnije biti obrađen i Object Inspector. Ovo će biti zagrevanje za lekciju dana 6, "Rad sa Form Designer-om i Menu Designer-om", kada ćete naučiti sve o Delphijevom Form Designer-u. U današnjoj lekciji ću obraditi dodatne prozore

Delphijevog okruženja. Dodatni prozori Vam pomažu da prilagodite okruženje koje bi zadovoljilo Vaš ukus i Vaš način rada.

Za početnike, pogledajmo kako Delphi vidi aplikacije i kako pojednostavljuje proces kreiranja programa.

Projekti u Delphiju

Kao što do sada znate, dok pišete Delphi aplikacije dosta toga se dešava u pozadini. U stvari, u pozadini se dešava više no što sam Vam objasnio do sada. Nije neophodno da detaljno znate šta se sve dešava u pozadini dok pišete Delphi aplikaciju, ali je dobro da imate opšti pregled.

NOVI TERMIN *Projekt* je skup datoteka koje zajedno kreiraju zasebnu izvršnu datoteku, odnosno dinamički povezanu biblioteku (DLL- dinamic link library).

NOVI TERMIN Pored samostalnih projektata, Delphi dodatno omogućava i kreiranje grupa projekata. Grupa projekata (*project group*) je skup Delphi projekata.

Grupa projekata se koristi za administriranje grupom projekata pisanih na Delphiju, koji zajedno formiraju kompletan softverski proizvod. Grupe projekata će detaljnije biti obrađene u lekciji dana 9, "Projekti, Code Editor i Code Explorer". Za sada, jedino što treba da shvatite je da Delphi kreira novu bezimenu grupu projekata svaki put kada pokrenete Delphi (ukoliko niste uključili opciju za snimanje radne površine - desktop, prilikom zatvaranja Delphija). Bilo koji novi projekt koji kreirate će ući u grupu projekata. Ukoliko želite, možete snimiti grupu projekata, odnosno možete tretirati generičku grupu projekata kao privremenu.

Datoteke koje se koriste u Delphi projektima

Delphi upravlja projektima koristeći nekoliko datoteka za podršku. Da bi ovo ilustrovali, kreirajmo jednostavnu aplikaciju kako bi videli šta se dešava kada Delphi kreira izvršnu datoteku za Vaš program. Pratite sledeće korake:

1. Pre početka, kreirajte novi direktorijum na Vašem hard disku. (Direktorijumu možete dodeliti naziv po želji.)
2. Prvo odaberite opciju glavnog menija File➥Save All, da bi počeli od početka. Sada odaberite opciju File➥New Application u okviru glavnog menija. Na ekranu će biti prikazana prazna forma.
3. Odaberite opciju File➥Save All u okviru glavnog menija. Bićete upitani za naziv datoteke junita. Uverite se da ste odabrali prazan direktorijum koji ste prethodno kreirali.

4. Upišite naziv `MyUnit` za naziv datoteke junita, a zatim kliknite na dugme Save.
5. Sada ćete biti upitani za naziv projekta. Upišite `Test` u polje File Name, a zatim kliknite na dugme Save.
6. Odaberite opciju Project➥Build Test u okviru glavnog menija. Delphi prevodi program. (Prevođenje traje samo par sekundi.)
7. Odaberite opciju File➥Close All u okviru glavnog menija. (Da, ova vežba ima svoju svrhu.)
8. Zatim pokrenite Windows Explorer i pronađite direktorijum gde ste snimili projekt. Primetićete nekoliko datoteka.

Trebalo bi da vidite ukupno osam datoteka (tačan broj datoteka zavisi od opcija Delphijevog okruženja.) Prvo ću Vam objasniti šta se dešava kada Delphi kreira aplikacije; zatim ću objasniti čemu služi svaka datoteka.

Datoteke sa nastavkom koji počinje tildom (`~`) su rezervne kopije datoteka (backup files). Delphi može kreirati nekoliko rezervnih kopija datoteka, u zavisnosti od broja izvornih datoteka u projektu i opcija projekta koje ste definisali. Opcije projekta će biti obrađene u lekciji dana 9.

Kada prvi put kreirate projekt, Delphi će kreirati najmanje četiri datoteke (podrazumeva se tipična Delphi GUI aplikacija):

- Izvorni kod projekta.
- Junit glavne forme.
- Resursna datoteka glavne forme.
- Resursna datoteka projekta.

Izvorna datoteka projekta (*project source file*) je datoteka koja sadrži Delphijev početni kod. Izvornu datoteku projekta možete pregledati odabiranjem opcije Project➥View Source u okviru glavnog menija. Junit glavne forme (*main form unit*) sadrži deklaraciju klasa i definiciju klase glavne forme. Delphi će kreirati dodatnu datoteku junita za svaku novu formu koju kreirate. Resursna datoteka glavne forme (*main form resource file*) i resursna datoteka projekta (*project resource file*) su binarne datoteke koje opisuju glavnu formu i ikonu aplikacije.

Kada izdate komandu Delphiju da prevede projekt, Delphi prevodi izvorni kod projekta, junit glavne forme i ostale junite u okviru projekta. Nekoliko stvari se događa tokom ovog procesa; prvo, Object Pascal prevodilac prevodi junite projekta u binarne objektne datoteke. Zatim prevodilac resursa prevodi resurse, poput ikone programa i datoteke formi u binarnu resursnu datoteku. Zatim program za povezivanje preuzima kontrolu. Program za povezivanje uzima binarne datoteke koje je prevodilac kreirao, dodaje biblioteke komandi ukoliko su potrebne, zatim ih povezuje u

celinu i na kraju kreira izvršnu datoteku. Nakon što se sve završi, pred Vama je zaseban program koji možete pokrenuti na jedan od uobičajenih načina.

U redu, ali čemu sve ove datoteke služe? Tabela 4.1 prikazuje nastavke datoteka koje Delphi koristi i daje opis uloge svakog tipa datoteke.

Tabela 4.1: Tipovi datoteka koji se koriste u Delphiju

Nastavak	Opis.
pos	paslzvorni kod jezika Object Pascal. Za svaki junit xtoji jedna ovakva datoteka, kao i bilo koja datoteka izvornog koda koju dodajete projek tu.
.dfm	Datoteka forme. Ova datoteka je u stvari skrivena binarna resursna datoteka (.res). Ona predstavlja opis forme i svih njenih komponen ti. Svaka forma ima svoju .dfm datoteku.
.dsk	Datoteka radne površine projekta. Ova datoteka čuva podatke o načinu na koji se prikazuje radna površina nakon poslednjeg zapisivanja (odnosno zat varanja) projekta. Dimenzije i pozicije svih otvorenih prozora su snimljene, tako da prilikom ponovnog otvaranja projekta prozor izgleda onako kako ste ga ostavili. Ova datoteka se kreira samo u slučaju da ste odabrali opciju za snimanje Vaše radne površine (Environment Options okvir za dijalog).
.dof	Datoteka opcija projekta. Ova datoteka sadrži opcije projekta koje su definisane u dijalogu Project Options.
.exe	Finalni izvršni program..cfgKonfiguraciona datoteka projekta. Ova datoteka primarno sadrži trenutno odabrane opcije prevodioca i pro grama za povezivanje, koje se odnose na tekući projekt.
.dcu	Prevedena binarna objektna datoteka. Ove datoteke kreira prevodilac u toku prevođenja Vaših Object Pascal junita.
.dpr	Izvorni kod projekta.
.res	Prevedena binarna resursna datoteka.

Delphi može imati i druge nastavke datoteka. Na primer, nastavak .bpg se koristi da odredi grupu projekata, nastavak .dpk se koristi da odredi Delphijevu izvornu datoteku paketa, a nastavak .bpl predstavlja prevedeni paket. Paketi će biti detaljnije obrađeni u lekciji dana 8, "Kreiranje aplikacija u Delphiju", odnosno u lekciji dana 9, gde će biti obrađene grupe projekata.

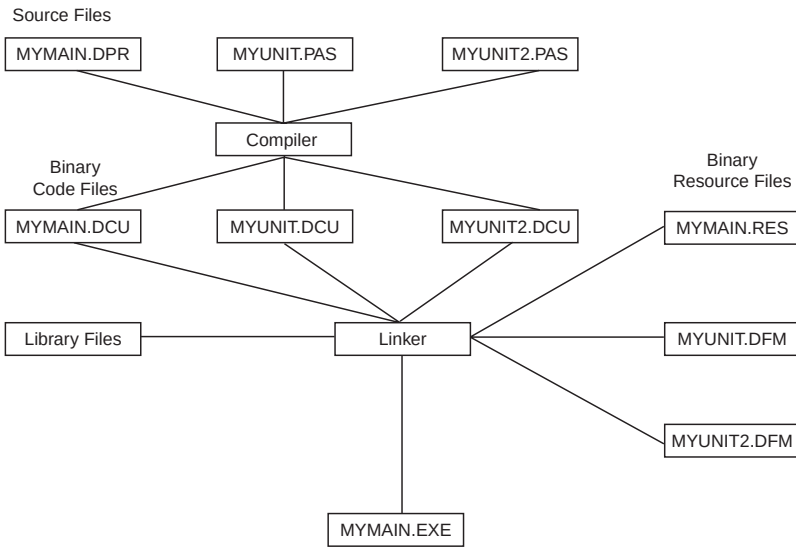
Datoteke koje Delphi kreira mogu biti podeljene u dve kategorije: Datoteke na koje se Delphi oslanja prilikom kreiranja projekata i datoteke koje Delphi kreira kada prevodi *i* povezuje projekt. Ukoliko želite da prenesete svoje izvorne datoteke na drugi računar, nećete morati da prenosite sve, nego samo datoteke koje su potrebne Delphiju da kreira aplikaciju. Zgodno je što su izvorne datoteke, igrom slučaja,

najmanje datoteke u okviru projekta. Izvorne datoteke ne zauzimaju puno prostora na disku, ukoliko želite da napravite njihovu rezervnu kopiju (backup).

Minimalni skup datoteka čine datoteke sa nastavcima `.pas`, `.dfm` i `.dpr`; sve ostale datoteke će Delphi ponovo kreirati nakon prevođenja programa. Datoteka radnog prostora (`.dsk`) će Vam možda biti interesantna pošto čuva podatke o stanju projekta, pre njegovog zatvaranja.

NAPOMENA Kao dodatak izvornim datsotekama, spomenuo sam da neke aplikacije koriste resursne skript datoteke (resursne skript datoteke imaju nastavak .rc). Resursni skriptovi (resource scripts) su datoteke teksta koje se koriste za definisanje resursa kao što su bitmape, ikone, odnosno kursori. Ako ih koristite, uverite se da se resursni skriptovi prilikom prenosa projekta na drugu lokaciju, nalaze zajedno sa ostalim datotekama projekta. Resursne skript datoteke se ne koriste često u Delphi projektima.

Slika 4.2 ilustruje kako Delphi prevodi izvornu datoteku i povezuje je u formu finalne izvršne datoteke.



Slika 4.2
proces prevođenja/povezivanja na Delphiju

31286-7
28670402
LP
5/29

SAVET Ukoliko otkrijete da Vam ponestaje mesta na hard disku, možete obrisati neke Delphi datoteke iz projekta, ukoliko trenutno sa njima ne radite. Bezbedno je obrisati datoteke sa nastavkom

.dcu. Ove datoteke će iznova biti generisane nakon kreiranja projekta, i nema svrhe čuvati ih za projekte koji se trenutno ne koriste.

Nemojte brisati ni jednu datoteku iz Delphi direktorijuma, osim datoteka iz direktorijuma Examples. Ukoliko se dvoumite, nemojte brisati!

Juniti izvornog koda

U prethodnom delu knjige sam spomenuo da većina aplikacija bilo koje veličine ima nekoliko izvornih datoteka, koje se nazivaju juniti (*units*). Svaki put kada kreirate novu formu, Delphi će uraditi sledeće:

- Kreiraće datoteku forme (.dfm).
- Izdvojiće klasu iz forme TForm.
- Kreiraće junit (.pas datoteku) za definiciju klasa.
- Dodaće nove podatke o formi izvornom kodu projekta.

Inicijalno, Delphi formi dodeljuje generički naziv Form1, a Unit1.pas junitu forme. Druga forma, koju kreira projekat, imaće generički naziv Form2, itd. Svaki put kada kreirate novu formu, Delphi će kreirati novi junit (.pas) i datoteku forme (.dfm) koja pripada datoj formi.

Ubrzo, nakon što kreirate novi projekt, trebalo bi da ga snimite, dodeljujući mu razumljiv naziv. Isto tako, kada kreirate novu formu, treba da je snimite sa nazivom koji je jasno opisuje. Na ovaj način ćete lakše pronalaziti forme, odnosno junite, kada poželite da ih menjate. Zapamtite da možete koristiti duga imena kada dodeljujete nazive Vašim junitima.

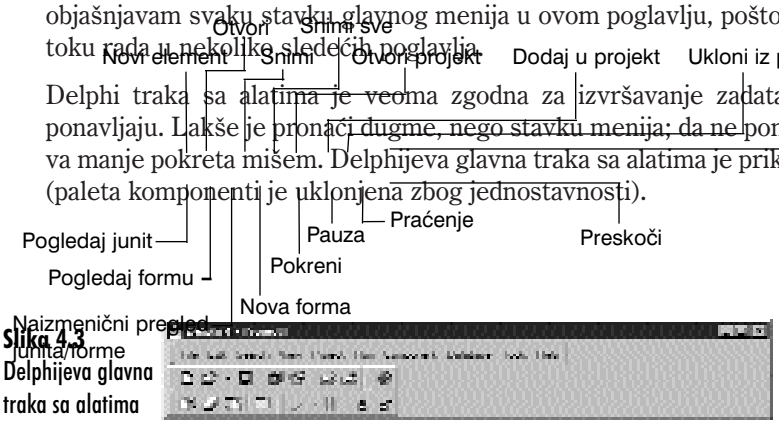
Kada pišete tehničku knjigu, često dolazite u teške situacije. Želeo bih da koristim primere koji imaju svrhu, da bih Vam lakše predstavio materiju. Da bih napisao ove primere, koristio sam tehnike, odnosno metode o kojima još nije bilo reči. Ali, o ovim metodama ne mogu pisati sve dok Vam ne dam neki dobar primer, odnosno primer koji ima smisla. Na žalost to nije moguće... Sada vidite moju dilemu. Zato ću napraviti malu digresiju i obraditi glavni meni, traku sa alatima (toolbar) i paletu komponenti (Component palette). Dok budete čitali sledeće poglavlje, imajte na umu da sam skrenuo sa puta, ali sam to učinio sa dobrim razlogom.

Delphijev glavni meni (main menu) i traka sa alatima (toolbar)

Delphijev glavni meni sadrži sve opcije koje su neophodne za rad sa Delphijem. Pošto je programiranje na Delphiju postupak koji zahteva operacije sa vizuelnim komponentama, možda nećete koristiti glavni meni kao što to možda činite u drugim programskim okruženjima. Ipak, skoro sve što Vam je potrebno je dostupno iz glavnog menija, ukoliko više volite da radite na taj način. Nemam nameru da

objašnjavam svaku stavku glavnog menija u ovom poglavlju, pošto ćete ih pronaći u toku rada u nekoliko sledećih poglavlja.

Delphi traka sa alatima je veoma zgodna za izvršavanje zadataka koji se često ponavljaju. Lakše je pronaći dugme, nego stavku menija; da ne pominjemo da zahteva manje pokreta mišem. Delphijeva glavna traka sa alatima je prikazana na slici 4.3. (paleta komponenti je uklonjena zbog jednostavnosti).



Slika 4.3
Delphijeva glavna traka sa alatima

Ako ste poput mene, obično ćete zaboraviti da koristite traku sa alatima. Ipak Vam kažem: nemojte zaboraviti da naučite opcije i da koristite traku sa alatima! Stara izreka kaže: "Radi onako kako govorim, a ne onako kako ja radim." Ukoliko imate vremena, naučite da koristite traku sa alatima, to će Vam uštedeti vreme i učiniti Vas efikasnijim na duge staze. Jedan od razloga zašto ste kupili Delphi je brzo pravljenje Windows aplikacija, pa ćete možda poželeti da ih u većini slučajeva brzo i napravite.

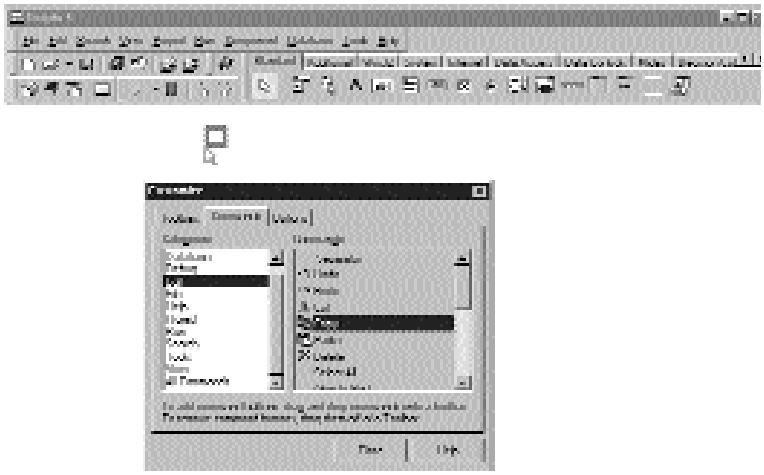
Delphijeva traka sa alatima je potpuno prilagodljiva. Traku sa alatima možete postaviti bilo gde u okviru glavnog prozora. Poziciju menija, trake sa alatima, odnosno palete komponenti možete reorganizovati tako da odgovaraju načinu Vašeg rada.

Prilagođavanje trake sa alatima je veoma jednostavno. Delphi Vam omogućava da dodajete dugmad na traku sa alatima, uklanjate dugmad sa trake, odnosno da rasporedite dugmad kako Vam odgovara. Da biste konfigurisali traku sa alatima kliknite desnim tasterom miša na traku sa alatima kako bi Vam se prikazao meni sadržaja. Odaberite opciju Customize (prilagođavanje) u okviru menija sadržaja. Nakon što odaberete ovu opciju menija, pojaviće se okvir za dijalog Customize.

Okvir za dijalog Customize sadrži tri jezička kartice :

- Prvi jezičak Toolbars (trake sa alatima), pokazuje koje su trake sa alatima dostupne; označena je svaka traka sa alatima koja je trenutno vidljiva. Možete dodavati, odnosno uklanjati postojeće trake sa alatima, odnosno vratiti trake sa alatima na stanje koje odgovara početnom.

- Drugi jezičak kartice sa natpisom Commands (komande), pokazuje svu dostupnu dugmad koju možete koristiti za trake sa alatima. Da biste dodali dugme na traku sa alatima, pronađite opis dugmeta na uokvirenoj listi Commands, a zatim ga prevucite do mesta na koje želite da ga postavite. Da biste uklonili dugme sa trake sa alatima, uhvatite je i prevucite van trake. Ovo je veoma jednostavno. Slika 4.4 prikazuje trenutak dodavanja dugmeta na traku sa alatima. Ukoliko ste napravili zbrku, jednostavno se vratite na karticu Toolbars i kliknite mišem na dugme Reset. Traka sa alatima će se vratiti na početno stanje.



Slika 4.4
Prilagođavanje trake sa alatima

- Treći jezičak kartice, Options (opcije), sadrži opcije poput sledeće: da li je oblačić za savet(tooltip)prikazan i kako se prikazuje.

Slobodno prilagođavajte Delphi okruženje onako kako želite. Ovo okruženje Vam služi za razvoj, zato ga učinite takvim da radi za Vas.

Korišćenje palete komponenti (Component palette)

Delphijeva paleta komponenti (Component palette) se koristi za odabiranje komponenti, odnosno drugih kontrola (kao što su ActiveX kontrole), da bi mogli da ih prebacite na formu. Paleta komponenti je prozor sa više kartica. Jezičci kartica se koriste za odabiranje kartica. Klikom miša na jezičak prikazuju se dostupne komponente, odnosno kontrole na kartici.

Smeštanje komponente na formu predstavlja proces od dva koraka. Prvi korak je odabiranje dugmeta koje predstavlja željenu komponentu na paleti komponenti. Zatim se klikom miša na formu prebacuje komponenta. Komponenta se pojavljuje sa gornjim levim uglom na mestu gde ste kliknuli mišem na formu.

Već ste videli osnovne operacije palete komponenti, mada postoje neke druge mogućnosti koje do sada nisu bile navedene. Sledeće poglavlje objašnjava ove mogućnosti.

Postavljanje više kopija komponenti

Do sada ste na formu odjednom postavljali samo jednu komponentu. Više komponenti istog tipa možete lako postaviti, a da posebno ne odabirate komponente na paleti komponenti. Da biste postavili više komponenti na formu, pritisnite i držite taster Shift u trenutku kada odabirate komponente sa palete komponenti. Nakon što ste odabrali komponente, možete da otpustite taster Shift.

Dugme komponente na paleti komponenti će izgledati kao da je pritisnuto i biće oivičeno plavim okvirom. Kliknite mišem na formu da biste postavili prvu komponentu. Uočite da dugme još uvek ostaje pritisnuto na paleti komponenti. Mišem možete kliknuti onoliko puta koliko želite; nova komponenta će biti postavljena svaki put kada kliknete mišem na formu. Kada poželite da završite sa postavljanjem komponente, kliknite mišem na dugme za izbor na paleti komponenti (dugme sa strelicom). Dugme komponente će iskočiti, da bi pokazalo da ste završili sa postavljanjem komponenti.

"Videti, znači verovati" - kaže izreka; zato pratite sledeće korake:

1. Kreirajte novi projekt.
2. Pritisnite i držite taster Shift na tastaturi, a zatim kliknite na komponentu Label (natpis) u okviru palete komponenti.
3. Tri puta kliknite mišem na formu, pomerajući kursor svaki put kada želite da označite mesto gde će komponenta biti postavljena. Nova komponenta `Label` će biti postavljena na formu nakon svakog klika mišem.
4. Kliknite na dugme sa strelicom u okviru palete komponenti, da biste završili proces, a zatim se vratite na mod za dizajniranje forme.

Najbrži način za istovremeno postavljanje svih komponenti određenog tipa na formu je korišćenje prethodno navedene tehnike. Komponente kasnije možete reorganizovati, odnosno promeniti im veličinu.

Kada postavljate više primeraka određene komponente, lako Vam se može desiti da zaboravite da kliknete mišem na dugme sa strelicom kada završite posao. Ukoliko greškom postavite više komponenti nego što ste želeli, jednostavno obrišite višak.

Postavljanje i centriranje komponenti na formi

Delphi pruža prečice za postavljanje komponenti na formu. Dvostrukim klikom miša na dugme komponente u okviru palete komponenti ćete postaviti komponentu na formu. Komponenta će biti centrirana na formi i vertikalno i horizontalno.

Komponenta postavljena na ovaj način, može biti premeštena na drugi deo forme na potpuno isti način na koji se ostale komponente postavljaju na formu.

NAPOMENA Svaki put kada dva puta kliknete mišem na dugme palete komponenti, komponenta sa palete će biti postavljena u središte forme i imaće svoju generičku veličinu. Ukoliko dva puta kliknete mišem na dugme komponente, i to ponovite nekoliko puta, na formu će biti postavljeno više kopija komponenti. Svaka komponenta će biti smeštena u središte forme preko prethodne. Ovo se događa čak i ako koristite samo jednu komponentu, pa možda nećete shvatiti da na formi nekoliko komponenti zauzima isti prostor. Ukoliko greškom postavite više komponenti, kliknite mišem na neželjene i uklonite ih sa forme.

Meni sadržaja palete komponenti (Component palette context menu)

Kada postavite kursor preko palete komponenti i kliknete desnim tasterom miša, pojaviće se meni koji se odnosi na paletu komponenti (videti sliku 4.5).

Slika 4.5 meni sadržaja palete komponenti

Opcija Show Hint (prikaži oblačić za savet) uključuje, odnosno isključuje oblačić za savet koji se nalazi na dugmetu komponente. Ukoliko stvarno ne volite oblačiće za savet (tooltips) isključite ovu opciju, u protivnom opcija bi trebala da ostane uključena. Opcija Hide (sakriti) u okviru menija sadržaja, sakriva paletu komponenti. Da bi paleta komponenti bila ponovo prikazana, treba da je odaberete koristeći meni sadržaja trake sa alatima.

Opcija Help (pomoć) u okviru menija sadržaja, prikazuje Delphijev program za pomoć sa otvorenom stranom koja objašnjava paletu komponenti. Opcija Properties (karakteristike) prikazuje karticu Palette, koja se nalazi u okviru za dijalog Environment Options (opcije okruženja), gde isto tako možete prilagoditi paletu komponenti. U navedenom okviru za dijalog možete dodavati, odnosno uklanjati kartice palete komponenti. Takođe, možete dodavati, uklanjati, odnosno menjati raspored komponenti na svakoj kartici posebno. Ova mogućnost će detaljnije biti obrađena u lekciji dana 11 "Delphijevi alati i opcije", kada ćete učiti o podešavanju opcija okruženja.

Kretanje po paleti komponenti

Ukoliko je paleta komponenti toliko mala da ne može da prikaže sve jezičke kartica, primetićete dugmad za pomeranje s gornje desne strane. Kliknite na ova dugmad za pomeranje, da biste videli jezičke koji nisu trenutno vidljivi. Isto tako, ukoliko

Dugmad za pomeranje kartica komponenti

određena strana palete komponenti sadrži više dugmadi nego što može stati na prikazani prozor, dugmad za pomeranje će Vam omogućiti da pomerate karticu palete i tako pronađete odgovarajuće dugme. Slika 4.6 prikazuje paletu komponenti na kojoj se nalaze obe vrste dugmadi za pomeranje.

Slika 4.6
Paleta komponenti sa dugmadima za pomeranje

Paleta komponenti nije zastrašujuće komplikovana, ali je njeno osnovno razumevanje, odnosno korišćenje vitalno za programiranje na Delphiju. Nakon što smo na kratko pogledali osnovni prozor Delphija, možemo se ponovo vratiti na glavnu temu.

Aplikacija sa više formi

Da bi ilustrovali kako Delphi koristi junite, možete kreirati aplikaciju koja sadrži više formi. Kreiraćete jednostavnu aplikaciju koja prikazuje drugu formu ukoliko kliknete na dugme:

1. Kreirajte novi projekt izborom opcije File➥New Aplication u okviru glavnog menija.
2. Promenite karakteristiku `Name` (naziv) u `MainForm`, a karakteristiku `Caption` u `Multiple Forms Test Program` (Program sa više formi).
3. Snimite projekt. Snimite junit kao Main, a projekt kao `Multiple`.
4. Sada postavite dugme na formu. Upišite u karakteristiku Name naziv `ShowForm2`, a u karakteristiku `Caption`, `Show Form 2` (Prikaži formu 2).
5. Odaberite opciju File➥New Form u okviru glavnog menija (odnosno kliknite na dugme New Form u okviru trake sa alatima), kako biste kreirali novu formu. Od ovog trenutka nova forma dobija naziv `Form1` i postavljena je preko glavne forme. Pošto želite da nova forma bude manja od glavne, umanjite je i centrirajte približno u odnosu na glavnu formu.
6. Veličina i pozicija nove forme su takve da je nova forma oko pedeset posto manja od glavne i centrirana u odnosu na glavnu formu. Koristeći traku sa naslovom pomerite novu formu. Veličinu nove forme možete promeniti povlačeći donji desni ugao forme.
7. Promenite karakteristiku nove forme `Name` u `SecondForm`, a karakteristiku `Caption` u `Second` (Druga).

8. Odaberite opciju File➥Save u okviru glavnog menija (odnosno kliknite na dugme Save File u okviru trake sa alatima), a zatim snimite novu formu sa novim nazivom, Second.

9. Odaberite komponentu `Label` i prebacite je na novu formu. Promenite karakteristiku `Caption` komponente Label (natpis) u `This is the second form` (Ovo je druga forma). Promenite veličinu natpisa i boju po želji. Centrirajte natpis u okviru forme. Vaša forma sada približno izgleda kao forma na slici 4.7.

Slika 4.7
Izgled forme za sada

10. Kliknite na glavnu formu. Sada glavna forma prekriva drugu formu. Dva puta kliknite mišem na dugme `Show Form 2`. Editor koda (Code Editor) će biti prikazan, a kursor će se nalaziti tačno tamo gde treba da počnete sa pisanjem koda. (Dvostruki klik mišem na dugme je prečica za generisanje `OnClick` upravljača događajem.)

11. Kucajte kod tako da funkcija dobije sledeći izgled (treba da kucate samo jednu liniju koda):

```
procedure TMainForm.ShowFrom2Click(Sender: TObject);
begin
  SecondForm.ShowModal;
end;
```

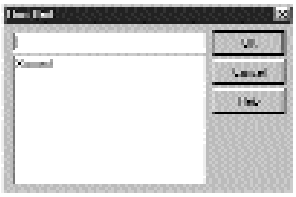
12. Pokrenite program.

Od ovog trenutka ćete imati okvir sa porukom koja glasi: `Form 'MainForm' references form 'SecondForm' declared in unit 'second' which is not in your USES list. Do you wish to add it?` (forma `'Main Form'` ukazuje na formu `'SecondForm'` deklarisanu u junitu `'Second'` koji nije u Vašoj listi `USES`. Da li želite da je dodate?) Kliknite na dugme Yes, pa će Delphi dodati junit `Second`, na listu uses junita `Main`. Kliknite ponovo na dugme Run, pa će ovog puta aplikacija biti pokrenuta. Kada kliknete na dugme Show Form 2 na glavnoj formi,

biće prikazana druga forma. Drugu formu možete zatvoriti klikom na sistemsko dugme za zatvaranje, koje se nalazi na naslovnoj traci forme.

Dodavanje junita

Junite možete ručno dodavati, što je bolje od pojavljivanja Delphijevog upita za dodavanje junita na listu uses. Naziv junita možete manuelno kucati u listu uses željene forme, odnosno možete odabrati opciju File➥Use Unit u okviru glavnog menija. Ukoliko koristite drugi metod, okvir za dijalog Use Unit će biti prikazan onako kako je prikazan na slici 4.8. Okvir za dijalog Use Unit, prikazuje listu svih raspoloživih junita. Odaberite junit koji želite da dodate, a zatim kliknite na dugme OK. Delphi će dodati junit na listu uses tekuće forme. Uočite da okvir za dijalog Use Unit prikazuje samo one junite koji postoje u projektu, a da istovremeno nisu uključeni u tekući. Juniti koji su već uključeni u tekući nisu prikazani na listi dostupnih junita.



Slika 4.8
Okvr za dijalog Use Unit

Ko što ste već mogli videti, Delphijevo administriranje junitima je veoma korisno. Kasnije, kada Vaše potrebe za programiranjem budu dosta veće, moraćete da sami menjate izvorni kod programa, ali u ovom trenutku Delphi može uraditi većinu poslova umesto Vas.

Pogledajmo sada drugi način prevođenja koji Vam je dostupan kada pišete programe u Delphiju.

Prevođenje, kreiranje i povezivanje

Svaki put kada kliknete na dugme Run, Delphi prevodi i povezuje Vaš program. Nije neophodno prevesti svaki junit projekta. Potrebno je prevesti samo one junite koji su bili promenjeni nakon poslednjeg prevođenja. Ova mogućnost Vam štedi vreme, pošto ne morate čekati prevodilac da prevede datoteke koje nisu menjane. Delphi vodi računa o tome koje su datoteke promenjene, odnosno koje nisu, pa ne morate ništa posebno raditi da biste koristili ovu mogućnost, pošto je sve automatizovano.

U većini slučajeva želite da vidite rezultate promena na delu. U ovakvim slučajevima kliknite na dugme Run i program će biti preveden, povezan i izvršen. Ponekad, naravno ne želite da pokrenete program. Na primer, možda ćete poželeti da prevedete program, da biste proverili da li ima grešaka.

U okviru Delphija se nalaze tri opcije menija koje su dodatak opciji Run i pružaju Vam kontrolu procesa prevođenja i povezivanja. Ako odaberete opciju Project u okviru glavnog menija, primetićete tri opcije pod nazivom Compile, Build i Syntax Check (prevedi, kreiraj i provera sintakse). Tekst ove tri opcije menija se menja u zavisnosti od naziva trenutno aktivnog projekta. Na primer, kada prvi put startujete Delphi, ove opcije će glasiti: Compile Project1, Build Project1 i Syntax Check. (Takođe postoje stavke menija pod nazivom Compile All Projects i Build All Projects, ali o ovom će biti više reči u lekciji dana 9, kada će biti obrađene grupe projekata.) Krenimo redom od najjednostavnijih do najkomplikovanijih opcija (iz perspektive prevodioca):

- Syntax Check je opcija koju veoma volim. Ova mogućnost se koristi za prevođenje projekta i izveštavanje o greškama i upozorenjima. Ovo je najbrži način da proverite greške u Vašem kodu. Delphi samo prevodi projekat, ne povezuje ga. Svrha provere sintakse (Syntax Check opcije) je brza provera koda i grešaka u sintaksi. Pošto faza povezivanja zahteva dodatno vreme, opcija za proveru sintakse preskače ovaj korak.
- Compile opcija prevodi junite koji su bili promenjeni nakon poslednjeg prevođenja isto kao i opcija za proveru sintakse, ali dodatno povezuje kompletan projekt. Prirodno je da ova aktivnost traje nešto duže od provere sintakse. Opciju Compile možete koristiti kada želite da prevedete i povežete Vaš program, a ne želite da ga pokrenete.

SAVET Prečica tastature za opciju Compile je `Ctrl+F9`.

- Build opcija se najduže izvršava.Ova opcija prevodi svaki junit u okviru projekta, bez obzira da li je bio menjan nakon poslednjeg kreiranja. Nakon što prevede sve junite, Delphi povezuje kompletan projekt.

Do sada ste dozvoljavali Delphiju da dodaje junite Vašim projektima. Ubuduće ćete možda morati ručno da editujete Vaše izvorne kodove da biste dodavali junite, odnosno direktive prevodioca. Možda ćete čak prestati sa editovanjem izvornog koda projekta. S vremena na vreme stvari mogu da se zapetljaju (svi pravimo greške, zar ne). Izvršavanje opcije Build će sve razrešiti i izbaviti Vas iz problema u koje ste upali. Ponekad opcija Build rešava greške prevodioca i programa za povezivanje, tako da nećete imati potrebu da bilo šta menjate ubuduće.

SAVET Svaki put kada naiđete na neočekivanu (neuobičajenu) grešku prevodioca, ili programa za povezivanje, pokrenite opciju Build. Razlog možda leži u činjenici da je nešto u programu nesinhronizovano, što će se pokretanjem opcije Build popraviti. Ukoliko opcija Build ne reši problem, treba da uradite nešto kako bi otkrili gde se problem nalazi.

Delphi pruža opciju prikazivanja dijalog okvira statusa prevođenja u toku samog prevođenja. Ovu opciju možete uključiti preko okvira za dijalog Environment Options (kartica Preferences). Ukoliko je ova opcija uključena, dijalog okvir statusa

prevođenja će prikazati naziv datoteke svakog junita u trenutku prevođenja. Ukoliko se pojavi greška, okvir za dijalog statusa prevođenja će Vas obavestiti: `There are errors` (ima grešaka), a zatim će ispisati koliko je grešaka otkriveno, odnosno koliko je otkriveno upozorenja.

Slika 4.9 prikazuje dijalog okvira statusa prevođenja, nakon što su otkrivene greške. Delphi Vaše projekte prevodi toliko brzo, da u principu dijalog okvir statusa prevođenja nije neophodan. U stvari, dijalog okvira statusa prevođenja će produžiti vreme prevođenja pošto je potrebno određeno vreme za prikazivanje podataka.

Slika 4.9
Dijalog okvir statusa prevođenja prikazuje greške

Bez obzira na metod koji je odabran za prevođenje projekta, ukoliko je otkrivena greška, editor koda će se pojaviti na vrhu radne površine, a prozor sa porukom u dnu editora koda, će prikazati listu grešaka i upozorenja. Editor koda ističe liniju gde se pojavila prva greška. Nakon što ste uspešno proverili sintaksu, preveli, odnosno kreirali program, istovremeno možete pokrenuti Vaš program koristeći dugme Run, ukoliko želite.

Prevođenje i kreiranje drugih Object Pascal programa

Delphijeva snaga je u okruženju za vizuelno programiranje. Ovo okruženje je povezano direktno za VCL i ne može se odvojiti od njega. Da biste izvukli što više od Delphija, uglavnom ćete pisati aplikacije bazirane na VCL-u. Biće trenutaka, naravno, kada ćete poželeti da pišete druge tipove aplikacija.

NOVI TERMIN *Dinamic link library* (DLL- biblioteka dinamičke veze) je eksterna datoteka koja sadrži kod koji se može izvršiti iz programa, odnosno drugog DLL-a.

Možda je najčešći oblik "drugih" programa, koje ćete poželeti da pišete, DLL (dinamičke biblioteke). Izgledaće Vam možda da su DLL neka vrsta crne magije, ali ustvari ove biblioteke nisu preterano komplikovane: one su delovi prevedenog koda koji možete pozvati iz Vaše aplikacije. Nakon što kreirate DLL, pozivanje funkcije koja se nalazi u DLL-u se ne razlikuje od poziva funkcije koja se nalazi u okviru Vašeg glavnog programa. Ovo je pomalo pojednostavljeno, ali je još uvek tačno. O DLL-ovima će biti više reči u lekciji dana 19, "Kreiranje i korišćenje DLL-a".

Još jedan tip aplikacija koji ćete možda pisati u Delphiju su Win32 aplikacije konzole.

NOVI TERMIN Win32 aplikacije konzole (*console application*) su 32-bitni programi koji se izvršavaju iz DOS prozora u okviru operativnog sistema Windows 95, odnosno Windows NT.

Aplikacije konzole se koriste za pisanje malih pomoćnih programa, programa za servere, kao što je TCP/IP server, odnosno server za poštu (mail server), mada postoji mnogo drugih mogućnosti. U suštini, ukoliko apliikacija ne zahteva grafički interfejs, postaje dobar kandidat za aplikaciju konzole.

Više o Delphijevim formama

Pre nego što nastavim sa opisom Delphijevog okruženja, ukratko ću objasniti forme. Mogli ste da vidite nekoliko formi na delu, dok ste čitali ovu knjigu, a u lekciji dana 6 ćete naučiti sve o dizajneru formi (Form Designer). Pre nego što stignete do ovog dela, potrebne su Vam osnovne informacije o formama, a njih ću obraditi u nastavku.

Forme glavnog prozora

Forme su glavni delovi za kreiranje Delphi aplikacija. Svaka GUI aplikacija ima najmanje jednu formu koja služi u glavnom prozoru. Forma glavnog prozora može biti samo prazan prozor, može sadržati kontrole, odnosno može imati prikazanu bitmapiranu sliku (bitmapu). U tipičnom Windows programu, Vaš glavni prozor bi trebao imati meni. Glavni prozor takođe može imati dekoracije, kao što su traka za alate, odnosno statusna traka. Bilo šta od ovih komponenti se može postaviti prilikom kreiranja glavnog prozora Vaše aplikacije. Svaka aplikacija je jedinstvena i ima različite zahteve.

Forme okvira za dijalog

Forme se takođe koriste kod okvira za dijalog. U suštini, za korisnika nema nikakve razlike između Delphijevih formi koje se ponašaju kao okviri za dijalog i pravih okvira za dijalog. (Pod nazivom "pravi okviri za dijalog" sam mislio na okvire za dijalog koji su kreirani na tradicionalan način, korišćenjem editora resursa i resursnih skript datoteka. Ovo je način kako se okviri za dijalog kreiraju u drugim programskim okruženjima. Delphi ne koristi tradicionalne okvire za dijalog, pa stoga najverovatnije nećete morati da radite sa njima na tradicionalnom nivou.) Okviri za dijalog obično imaju nekoliko osobina koje ih izdvajaju od običnih prozora:

- Veličinu okvira za dijalog obično nije moguće menjati. Oni obično izvršavaju sprecifične funkcije, i promena veličine okvira za dijalog nije korisna, a ni poželjna.
- Okviri za dijalog najčešće imaju dugme OK. Neki okviri za dijalog imaju dugme sa natpisom Close (zatvori) koji izvršava zadatak zatvaranja okvira za dijalog.

Jednostavni okvir za dijalog kao što je About (opis programa) najčešće ima samo dugme OK.

- Okviri za dijalog takođe mogu imati dugme Cancel i dugme Help.
- Okviri za dijalog tipično imaju samo sistemsko dugme za zatvaranje na naslovnoj traci. Obično nemaju dugmad za maksimiziranje i minimiziranje.
- Neki okviri za dijalog su takozvani okviri za dijalog sa karticama (*tabbed dialog boxes*) koji prikazuju nekoliko kartica između kojih korisnik može da bira. Kada kliknete na jezičak kartice, prikazaće Vam se određena kartica okvira za dijalog.
- Prečica za jezičak kartice se može koristiti za pomeranje sa jedne kartice na drugu u većini okvira za dijalog.

Za svako pravilo postoje i izuzeci. Većina okvira za dijalog ima uobičejene karakteristike, ali neki okviri za dijalog izvršavaju posebne zadatke koji se razlikuju od uobičajenih, na jedan ili drugi način.

Dijalozi na stari način

U tradicionalnim Windows programima (programi koji su pisani u Borland Pascal-u, odnosno korišćenjem kostura kao što je OWL) okvir za dijalog se kreira uz pomoć editora okvira za dijalog. U većini slučajeva editor okvira za dijalog je vizuelni alat koji funkcioniše slično Delphijevom alatu Form Designer. Kada korisnik završi sa dizajniranjem okvira za dijalog, vizuelni prikaz okvira za dijalog se konvertuje u definiciju preko resursne skript datoteke. Da bi ovo ilustrovali pogledajmo okvir za dijalog koji je prikazan na slici 4.10.

NOVI TERMIN Resursni skript (*resource script*) je tekst datoteka koja se kasnije prevodi u binarnu resursnu datoteku korišćenjem resursnog prevodioca.

Slika 4.10
Tipični okvir za dijalog About

Slika 4.10 prikazuje tipični okvir za dijalog About. Ovaj okvir sadrži naziv programa, informaciju o autorskim pravima i ikonu aplikacije. Resursna skript definicija okvira za dijalog je prikazana na listingu 4.1.

Listing 4.1: Okvir za dijalog, resursna definicija

```
1: IDD_ABOUT DIALOG 58, 53, 194, 119
```

```
 2: STYLE DS_MODALFRAME or WS_POPUP ¦
 3:   WS_VISIBLE or WS_CAPTION or WS_SYSMENU
 4: CAPTION 'About TMMPlayer Example Program'
 5: FONT 8, 'MS Sans Serif'
 6: {
 7:  DEFPUSHBUTTON 'OK', IDOK, 72, 96, 50, 14
 8:  CTEXT 'TMMPlayer Example Program', -1, 48, 22, 128, 8
 9:  CTEXT 'Copyright © 1996, by Kent Reisdorph', -1, 32, 47, 136, 8
10:  CTEXT 'March 15, 1996', -1, 24, 59, 146, 8
11:  CONTROL '', 99, 'button', BS_GROUPBOX ¦
12:    WS_CHILD or WS_VISIBLE or WS_GROUP, 12, 4, 176, 70
13:  CONTROL 1, 1, 'static', SS_ICON ¦
14:    SS_SUNKEN or WS_CHILD or WS_VISIBLE, 24, 17, 20, 20
15: }
```

Resursni skript sadrži informacije koje Windows koristi da bi kreirao okvir za dijalog u toku rada programa. Ove informacije uključuju broj i tip kontrola na okviru za dijalog, njegovu veličinu, poziciju, tekst, opcije, itd. Naravno, resursni skript takođe uključuje isti tip informacija za aktuelni okvir za dijalog.

Neki Windows programeri uopšte ne koriste editor okvira za dijaloge, pošto više vole da pišu definiciju okvira za dijalog od samog početka koristeći editor teksta. Naravno, ne mogu kriviti ove programere što kreiraju okvir za dijalog na taj način, mogu samo reći da većina programera koji koriste takav pristup nisu 100 posto efikasni. Potrebno je puno više vremena da bi se na ovaj način kreirao okvir za dijalog, nego što je to potrebno vizuelnim pristupom.

Obično sve definicije okvira za dijalog aplikacija sadrže jednu resursnu skript datoteku sa produžetkom (extension) `.rc`. U jednom trenutku, prilikom kreiranja programa, resursni skript se prevede u `.res` datoteku (binarna resursna datoteka), koja se zatim povezuje u `.exe` datoteku, korišćenjem programa za povezivanje. U toku rada programa, okvir za dijalog se prikazuje ili modalno, ili ne-modalno u zavisnosti od svrhe samog okvira za dijalog. Kada se okvir za dijalog pozove, Windows učitava resurse okvira za dijalog iz izvršne datoteke, kreira okvir za dijalog, a zatim ga prikazuje.

Okviri za dijalog na Delphijev način

U Delphiju okviri za dijalog su jednostavne forme. Okvire za dijalog kreirate kao što kreirate formu glavnog prozora, odnosno bilo koju drugu formu. Da biste sprečili da se menja veličina okvira za dijalog, možete promeniti karakteristiku `BorderStyle` u `bsDialog`, odnosno `bsSingle`. Ukoliko koristite `bsDialog`, Vaši okviri za dijalog će imati samo dugme za zatvaranje na naslovnoj traci, što je tradicionalan način prikazivanja okvira za dijalog. U drugom slučaju ne morate raditi ništa posebno da bi se Vaša forma ponašala kao okvir za dijalog. Sve Delphi forme imaju ugrađenu podršku za taster Tab. Da biste definisali redosled tabulatora, izmenite karakteristiku `TabOrder` za svaku kontrolu u okviru za dijalog.

NOVI TERMIN Modalni (*modal*) okvir za dijalog se mora prethodno zatvoriti, da bi korisnik mogao da nastavi sa korišćenjem aplikacije. Glavni prozor aplikacije je deaktiviran ukoliko je ovaj tip okvira za dijalog otvoren. Većina okvira za dijalog je modalna.

NOVI TERMIN Ne-modalni (*modeless*) okvir za dijalog omogućava korisniku da nastavi rad na aplikaciji čak i kada je okvir za dijalog prikazan. Okvir za dijalog Find u nekim tekst procesorima je primer ne-modalnog okvira za dijalog.

Delphijev okvir za dijalog (ustvari bilo koja Delphijeva forma) je modalna, odnosno ne-modalna, u zavisnosti od toga kako je prikazana. Da biste izvršili modalni okvir za dijalog, možete pozvati metod `ShowModal` forme `TForm`. Da biste kreirali ne-modalni okvir za dijalog, pozovite metod `Show`.

Kreiranje forme za dijalog

Sada možete dodati okvir About projektu sa više formi, koji ste ranije kreirali. Ukoliko Vam ovaj projekt nije otvoren, odaberite File➥Open Project u okviru glavnog menija, odnosno kliknite na dugme Open Project u okviru trake sa alatima i pronađite datoteku. (Verovatno ste ga snimili zajedno sa projektom pod nazivom `Multiple`.)

SAVET Delphi čuva listu datoteka i projekata koje ste prethodno koristili. Odaberite File➥Reopen, da biste videli MRU (most recently used - najčešće korišćeno) listu. MRU lista je podeljena na dva dela. Gornji deo prikazuje projekte koje ste skoro koristili, a donji deo prikazuje zasebne datoteke koje ste skoro koristili. Kliknite na jednu od ovih stavki, da biste ponovo otvorili projekt, odnosno datoteku.

Prvo treba da dodate dugme na formu koja prikazuje okvir za dijalog About:

1. Odaberite glavnu formu. Odaberite komponentu `Button` sa palete komponeti i postavite dugme na formu.
2. Uredite oba dugmeta tako da budu skladno postavljena na formu.
3. Promenite karakteristiku `Name` novog dugmeta u `AboutButton`, a karakteristiku `Caption` u `About...`
4. Dva puta kliknite na `AboutButton` koji ste upravo kreirali. Editor koda se prikazuje sa kursorom na funkciji za upravljanje događajem. Dodajte liniju koda na mesto kursora:

```
AboutBox.ShowModal;
```

Još uvek niste kreirali okvir About, ali kada to uradite dodelite mu naziv `AboutBox`, pošto dovoljno znate da biste upisali kod koji će prikazati About okvir.

Sada kreirajte okvir za dijalog, prateći sledeće korake:

1. Kreirajte novu formu (kliknite na dugme New Form u okviru trake za alate), promenite veličinu forme na veličinu karakterističnog About okvira (skoro ista veličina kao forma koju ste nazvali `SecondForm`, a koju ste prethodno kreirali).
2. Promenite karakteristiku `Name` u `AboutBox`, a zatim karakteristiku `Caption` u `About This Program` (o ovom programu).
3. Locirajte karakteristiku `BorderStyle` (iznad karakteristike `Caption`) i promenite je u `bsDialog`.
4. Sada dodajte tri natpisa u okvir. Editujte natpise tako da okvir About izgleda kao što je to prikazano na slici 4.11. (Naravno, možete upisati tekst koji želite). Možete ostaviti generičke nazive koje Delphi generiše za karakteristiku `Name`, tekst natpisa. Pošto ništa posebno nećete raditi sa karakteristikom `Name`, nije potrebno da koristite poseban, razumljiv naziv.

Slika 4.11
Okvir About sa dodatim tekst natpisima

SAVET Simbol za autorska prava (©) ima ASCII vrednost 169 u većini fontova. Da biste kreirali simbol za autorska prava pritisnite i držite taster Alt, a zatim kucajte brojeve 0169 na numeričkom delu tastature (uverite se da je uključen Num Lock). Kada otpustite taster Alt, pojaviće se simbol za autorska prava. Na ovaj način možete uneti ASCII vrednost bilo kog karaktera. Naravno, morate kucati sva četiri broja. Na primer, ASCII vrednost velikog slova A je 65. Da biste ubacili A, treba da pritisnete Alt taster i kucate `0065` na numeričkom delu tastature.

Zatim možete dodati ikonu na okvir About:

1. Kliknite na jezičak Additional u okviru palete komponenti i odaberite komponentu `Image`. Postavite komponentu na levu stranu forme.
2. Pronađite karakteristiku `AutoSize` za komponentu `Image` i promenite je u `True`.
3. Pronađite karakteristiku `Picture` i dva puta kliknite na kolonu Value. Okvir za dijalog Picture Editor (editor slika) će biti prikazan.
4. Kliknite na dugme Load u okviru za dijalog File Open, pronađite direktorijum `\Borland Shared Files\ Images\Icons` i odaberite ikonu sa liste. Kliknite

mišem na dugme Open. Ikona koju ste odabrali će biti prikazana u prozoru Picture Editor. Kliknite mišen na dugme OK, da biste zatvorili editor slika. Ikona će biti prikazana na formi. Uočite da komponenta `Image` ima istu veličinu kao i ikona.

5. Postavite ikonu koju želite.

Od ovog trenutka potrebno Vam je dugme OK na formi. Skrenimo na trenutak sa puta i pogledajmo novu komponentu:

1. Ukoliko se već ne nalazite na jezičku kartice Additional, u okviru palete komponenti, kliknite na nju. Odaberite komponentu `BitBtn` i postavite `BitBtn` na formu u njenom donjem delu i centrirajte je po horizontali.

2. Pronađite karakteristiku `Kind` i promenite je u `bkOK`. Uočite da se pojavio zeleni znak za potvrdu na dugmetu, a karakteristika `Caption` se promenila u `OK`. To je sve što treba da uradite sa dugmetom. Komponenta `BitBtn` već sadrži kod za zatvaranje forme, kada se klikne na dugme OK.

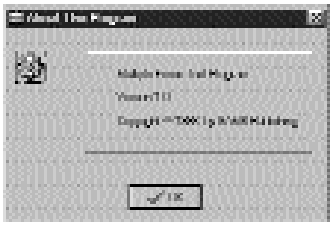
Dodajmo završni izgled okviru About:

1. Pronađite dugme Bevel (na kartici Additional u okviru palete komponenti) i kliknite mišem.

2. Pomerite se na formu, ali umesto da kliknete mišem na nju, raširite okvir tako da obuhvati tri tekst natpisa. Komponenta `Bevel` će se pojaviti kada prestanete sa povlačenjem miša. Ukoliko ovo baš niste potpuno shvatili, dodatno možete promeniti veličinu komponente.

3. Pronađite karakteristiku `Shape` i promenite je u `bsFrame`. Sada imate 3D okvir oko statičnog teksta.

Vaša forma bi trebalo da liči na formu sa slike 4.12. Snimite junit (File➥Save) i dodelite mu naziv `About`.



Slika 4.12
Završen okvir About

Da li ste spremni da prevedete i pokrenete program? Još uvek ne. Treba da dodate junit About u `uses` listu glavne forme. Pratite sledeće korake:

1. Pređite na editor koda (pritisnite F12) i odaberite jezičak kartice `Main.pas`.

2. Odaberite File➥Use Unit u okviru glavnog menija.

3. Odaberite junit About u okviru za dijalog Use Unit, a zatim kliknite na dugme OK.

Sada ste spremni da pokrenete program, stoga kliknite na dugme Run. Kada se program pokrene, kliknite na dugme About, pa će okvir za dijalog About biti prikazan. Uočite da je okvir za dijalog modalan (ne možete se vratiti u glavni prozor sve dok je okvir za dijalog prikazan) i da ne možete menjati veličinu okvira. Forma About se ponaša u svakom pogledu kao vaki regularni Windows okvir za dijalog.

NAPOMENA Uobičajene klase okvira za dijalog (`TOpenDialog`, `TSaveDialog`, `TFontDialog`, itd.) ne predstavljaju okvire za dijalog koji su kreirani kao Delphi forme. Ovi okviri za dijalog su preuzeti od Windows-a kao skup uobičajenih okvira za dijalog koje sve Windows aplikacije mogu koristiti. Aktuelni okviri za dijalog se nalaze u datoteci pod nazivom `COMDLG32.DLL`). Klase okvira za dijalog VCL-a enkapsuliraju uobičajene okvire za dijalog, kako bi njihovo korišćenje bilo lakše. Enkapsuliranje u ovom slučaju znači da se uzima uobičajena funkcionalnost okvira za dijalog i prebacuje u VCL komponentu. Komponenta omogućava mnogo lakše korišćenje okvira za dijalog nego što bi to u drugom slučaju bilo moguće.

NAPOMENA Delphi sadrži nekoliko prethodno kreiranih formi koje možete odabrati, kako biste olakšali kreiranje okvira za dijalog i ubrzali koliko je to moguće. O tome će biti više reči u lekciji dana 8.

Sekundarni prozori nasuprot okvirima za dijalog

Sekundarni prozor (*secondary window*) je forma koja se prikazuje u okviru glavnog prozora. Kada je forma sekundarni prozor, a kada okvir za dijalog? Kada bi detaljnije razmotrili ovu situaciju ne bi bilo razlike između sekundarnog prozora i okvira za dijalog u Delphiju. Možete imati prozore koji sadrže okvire za dijalog, a možete imati druge prozore koji sadrže tradicionalne prozore.

U opširnijoj šemi, sve su to forme i nema puno smisla izdvajati termine okvir za dijalog i *sekundarna forma*. U suštini, sve je to isto. U tradicionalnom okruženju za programiranje, morali ste tačno definisati okvir za dijalog, odnosno morali ste isto tako tačno definisati kreiranje sekundarnog prozora u aplikaciji. Delphi vas oslobađa ovih restrikcija i omogućava Vam da tretirate i okvire za dijalog i sekundarne prozore na potpuno isti način.

Model interfejsa sa višesturkim dokumentima

Do sada ste kreirali samo tip aplikacija koje imaju interfejs sa jednim dokumentom (SDI - single document interface). SDI aplikacije imaju jedan jedini glavni prozor i što je karakteristično, prikazuju okvire za dijalog ukoliko je to potrebno, ali ne prikazuju prozore potomke (child windows).

NOVI TERMIN Neki programi koriste model interfejsa sa višestrukim dokumentima (MDI - multiple document interface). *MDI aplikacije* se sastoje od

glavnog prozora (roditelj MDI) i prozora potomaka (MDI potomci - MDI child).

Primeri programa koji koriste MDI model su Windows System Configuration Editor (`SYSEDIT`) i Program Manager u okviru Windows 3.1.

Jedna od najočiglednijih karakteristika modela MDI je povezanost prozora potomka za prozor roditelj. Prozor potomak možete pomerati u okviru prozora roditelja, ali ga ne možete pomeriti van prozora roditelja. MDI aplikacije uvek imaju stavku: Window, u okviru glavnog menija. Ovaj meni obično sadrži stavke pod nazivom Cascade i Tile, što Vam omogućava da prikažate MDI prozore potomke bilo kaskadno (jedan preko drugog), ili uređeno (jedan pored drugog). Kada je MDI potomak minimiziran, u okviru MDI roditelja se nalazi ikona prozora potomka. Kada je običan (ne-MDI) prozor potomak minimiziran, njegova ikona će biti prikazana na radnoj poovršini Windows-a.

Da biste kreirali MDI aplikacije u Delphiju, morate podesiti karakteristiku glavne forme `FormStyle` na `fsMDIForm`. Svaki od MDI prozora potomaka mora imati karakteristiku `FormStyle` podešenu na `fsMDIChild`. Na stranu sa ovom restrikcijom, potrebno je veoma malo rada da biste kreirali MDI aplikaciju u Delphiju. Jednostavno kreirate formu glavnog prozora, jednu, odnosno više formi koje će biti korišćene kao prozori potomci, i završili ste sa poslom, tako da možete da radite.

Karakteristike Key za forme

Klasa `TForm` ima mnoštvo karakteristika. Neke od ovih karakteristika su neobične i retko se koriste; ostale se često upotrebljavaju. U ovom poglavlju će biti obrađene karakteristike koje se koriste veoma često. Očigledne karakteristike, kao što su `Caption`, `Color`, `Left`, `Top`, `Width` i `Height` neću obrađivati sve dok ne dođemo do osobina zbog kojih bi trebale da se koriste.

Karakteristike u toku dizajniranja

Karakteristike istaknute u ovom poglavlju se mogu podešavati u toku dizajniranja, a takođe i u toku rada programa. Skoro sve navedene karakteristike se mogu očitati u toku rada programa.

`ActiveControl`. Ova karakteristika se koristi za podešavanje fokusa kontrole u trenutku kada se aktivira forma. Na primer, možete poželeti da dodelite fokus određenoj kontroli za editovanje, u trenutku kada se forma okvira za dijalog prikaže na ekranu. U toku dizajniranja forme, kolona Value za karakteristiku `ActiveControl` sadrži listu komponenti koje se nalaze na formi. Možete odabrati bilo koju komponentu sa liste i učiniti je aktivnom u trenutku kada se forma prvi put prikaže.

Autoscroll, HorzscrollBar, VertscrollBar. Sve navedene karakteristike zajedno kontrolišu trake za pomeranje formi. Ukoliko je karakteristika `Autoscroll` definisana kao `True` (generički), trake za pomeranje se automatski pojavljuju, ukoliko je forma suviše mala da bi prikazala sve komponente. Karakteristike `HorzscrollBar` i `VertscrollBar` dodatno imaju nekoliko sopstvenih karakteristika koje kontrolišu operacije sa trakom za pomeranje.

BorderIcons. Ova karakteristika definiše koja sistemska dugmad će se pojaviti u formi u toku rada programa. Izbor uključuje sistemski meni, dugme za minimiziranje, odnosno maksimiziranje forme i dugme za pomoć.

BorderStyle. Ova karakteristika ukazuje na tip okvira koji će forma imati. Generička vrednost je `bsSizeable`, čime se kreira prozor kome se veličina može menjati. Stilovi koji se odnose na forme čija se veličina ne može menjati imaju vrednosti `bsDialog` i `bsNone`.

ClientWidth i ClientHeight. Umesto da definišete širinu i visinu kompletne forme bolje je definisati samo širinu i visinu klijent oblasti (client area), koristeći karakteristike `ClientWidth` i `ClientHeight`. (*Klijent oblast* forme je oblast unutar okvira, ispod naslovne trake i meni trake.) Ove karakteristike možete koristiti kada želite da klijent oblast može dostići određenu veličinu, a da se ostatak prozora samostalno prilagodi. Podešavanje karakteristika `ClientWidth` i `ClientHeight` menja takođe i karakteristike `Width` i `Height`.

Constraints. Ova karakteristika se koristi za definisanje maksimalne i minimalne širine i visine forme. Jednostavnim podešavanjem vrednosti MaxHeight, MaxWidth, MinHeight i MinWidth biće definisane granice veličine.

DefaultMonitor. Ova karakteristika određuje na kom monitoru će se pojaviti forma, ukoliko se aplikacija izvršava u okruženju koje podržava više mionitora (kao što je Windows 98).

DockSite. Ova karakteristika određuje da li će se forma ponašati kao sidrište (dock site) za komponente koje se mogu usidriti (dockable components). Sidrište i usidrene komponente će biti obrađene u lekciji dana 13, "Iza osnova Delphija".

Font. Ova karakteristika definiše font koji će biti korišćen u formi. Važno je da shvatite, da će svaka komponenta postavljena na formu naslediti font od forme. Ovo takođe znači da font koje koriste sve komponente, možete istovremeno menjati promenom fonta forme. Ukoliko zasebno menjate font za svaku kontrolu, promena fonta kontrole se neće dogoditi, ukoliko se promeni font glavne forme.

FormStyle. Ova karakteristika je obično podešena na opciju `fsNormal`. Ukoliko želite da Vaša forma uvek bude na vrhu, koristite opciju `fsStayOnTop`. MDI forme bi trebalo da koriste opciju `fsMDIForm`, a MDI forme potomci bi trebalo da koriste opciju `fsMDIChild`. MDI forme i MDI prozori potomci su bili obrađeni u prethodnom poglavlju, "Model interfejsa sa višestrukim dokumentima".

HelpContext i HelpFile. Karakteristika `HelpContext` se koristi za definisanje identifikacionog broja konteksta za pomoć u okviru forme. Ukoliko je pomoć za kontekst forme aktiviran, pritiskom na funkcijski taster F1 će biti aktiviran Windows sistem za pomoć. Identifikacioni broj konteksta se koristi da bi se sistemu za pomoć saopštilo koju stranu datoteke za pomoć treba prikazati. Karakteristika `HelpFile` definiše naziv datoteke za pomoć, koja će se koristiti ukoliko je pritisnut funkcijski taster F1.

Icon. Ova karakteristika definiše ikonu koja će se koristiti na naslovnoj traci forme u toku izvršavanja programa, odnosno kada se forma minimizira. U nekim slučajevima definisanje ove karakteristike nema efekta. Na primer, kada je `FormStyle` karakteristika definisana kao `fsDialog`, karakteristika Icon se ignoriše.

KeyPreview. Ukoliko je ova karakteristika podešena kao `True`, događaji forme `OnKeyPress` i `OnKeyDown` će biti generisani ukoliko bilo koji taster bude pritisnut na bilo koju komponentu forme. Uobičajeno je da forme ne prihvataju događaje tastature kada se komponenta forme nalazi u fokusu.

Position. Ova karakteristika određuje početnu veličinu i poziciju forme koja se pojavljuje u okviru druge forme. Tri osnovna izbora su: `poDesigned`, `poDefault` i `poScreenCenter`.

- `poDesigned` prikazuje formu na poziciji na kojoj je forma bila dizajnirana.
- `poDefault` omogućava operativnom sistemu Windows da definiše veličinu i poziciju forme u skladu sa uobičajenim Windows algoritmom za Z-redosled. (algoritam za Z-redosled koristi Windows operativni sistem za odlučivanje gde će novi prozor biti prikazan. Ukoliko novi prozor nema tačno određenu poziciju, biće prikazan nešto pomeren na dole i desno u odnosu na prethodno prikazani prozor.)
- `poScreenCenter` opcija prikazuje formu na sredini ekrana.

Visible. Ova karakteristika definiše da li je forma u početku vidljiva. Karakteristika nije posebno korisna u toku dizajniranja, ali u toku rada programa određuje da li je forma trenutno vidljiva. Takođe se može koristiti za sakrivanje, odnosno prikazivanje forme.

WindowState. Ova karakteristika određuje trenutno stanje forme (maksimizirana, minimizirana, odnosno normalna). Takođe se može koristiti za određivanje načina na koji će forma inicijalno biti prikazana. Izbori su: `wsMinimized`, `wsMaximized`, `wsNormal`.

Karakteristike koje se definišu samo u toku rada programa

Nekim karakteristikama je moguće pristupiti samo u toku rada programa, korišćenjem programskog koda. Slede najčešće korišćene karakteristike u toku rada programa:

ActiveMDIChild. Ukoliko se čita, ova karakteristika vraća pointer na trenutno aktivni MDI prozor potomak. Karakteristika se može samo čitati (read only). Ukoliko ne postoji trenutno aktivan MDI prozor potomak, odnosno ukoliko aplikacija nije MDI aplikacija, karakteristika `ActiveMDIChild` vraća nil.

Canvas. Kanvas forme predstavlja površinu forme na koju se mogu postavljati komponente. Karakteristika `Canvas` dozvoljava pristup kanvasu forme. Korišćenjem ove karakteristike na formi možete crtati bitmapirane slike, linije, oblike, odnosno tekst u toku rada programa. Uglavnom ćete koristiti komponentu `Label` za pisanje teksta na formi, a komponentu `Image` za prikazivanje slike, dok ćete za crtanje oblika koristiti komponentu `Shape`. Naravno, postoje situacije kada je potrebno crtati po kanvasu u toku rada programa, što Vam karakteristika `Canvas` i omogućava. Karakteristika `Canvas` se takođe može koristiti za snimanje slike forme na disk. Kanvasi će biti detaljnije obrađeni u lekciji dana 12, "Programiranje grafike i multimedije".

ClientRect. Ova karakteristika sadrži koordinate gornjeg, levog i desnog i donjeg levog i desnog ugla klijent oblasti u okviru forme, što je korisno u pojedinim situacijama prilikom programiranja. Na primer, možda ćete poželeti da znate koja je širina i visina klijent oblasti, da biste postavili bit mapu u središte forme.

Handle. Ova karakteristika vraća upravljač prozora tekuće forme (`HWND`). Ovu karakteristiku ćete koristiti kada poželite da prosledite upravljač prozora Windows API funkciji.

ModalResult. Ova karakteristika se koristi da ukaže na način na koji je modalna forma zatvorena. Ukoliko imate okvir za dijalog sa dugmadima OK i Cancel, karakteristiku `ModalResult` možete podesiti na `mrOK`, ukoliko korisnik klikne na dugme OK, odnosno na `mrCancel`, ukoliko korisnik klikne mišem na dugme Cancel. Pozvana forma zatim može očitati karakteristiku `ModalResult` da bi otkrila nakon pritiska na koje dugme je forma zatvorena. Ostale mogućnosti uključuju opcije `mrYes`, `mrNo` i `mrAbort`.

Owner. Ova karakteristika predstavlja pointer na vlasnika forme. Vlasnik forme je objekt koji je zadužen za njeno brisanje kada prestane potreba da forma bude prikazana. Roditelj komponente, u drugom slučaju je prozor (forma, odnosno druga komponenta) koji se ponaša kao kontejner komponenti. U slučaju glavne forme, objekt aplikacije je i vlasnik i roditelj forme. Kod komponenti vlasnik može biti forma, a roditelj može biti neka druga komponenta, na primer pano.

Parent. Ova karakteristika predstavlja pointer na roditelja forme. Pogledajte opis prethodne karakteristike, u kom je dato objašnjenje između karakteristike `Owner` i karakteristike `Parent`.

Metode forme

Forme su takođe i komponente. U suštini, forme imaju dosta sličnih metoda kao i komponente. Uobičajene metode uključuju `Show`, `ShowModal`, `Invalidate`; da spomenemo samo neke od metoda. Postoje i metode koje su, naravno, specifične samo za forme. Kao i u prethodnom slučaju obradiću samo najčešće korišćene metode.

BringToFront

Ova metoda prebacuje formu ispred svih ostalih u tekućoj aplikaciji.

Close i CloseQuery

Metoda Close zatvara formu nakon poziva metode `CloseQuery`, koja osigurava da se forma može zatvoriti. Funkcija `CloseQuery`, zauzvrat, poziva upravljač događajima `OnCloseQuery`. Ukoliko `Bulova` promenljiva, prosleđena upravljaču događaja `OnCloseQuery`, ima vrednost `False`, forma nije zatvorena. Ukoliko je vrednost `True`, forma je normalno zatvorena. Upravljač događaja `OnCloseQuery` možete koristiti da upitate korisnika da li želi da snimi datoteku koju treba snimiti i da kontrilišete da li se forma može zatvoriti.

Print

Ova metoda štampa sadržaj forme. Štampa se samo klijent oblast, a ne naslov, naslovna traka i okvir. Metoda `Print` je korisna za brzo preslikavanje (screen dump) forme.

ScrollInView

Ova metoda pomera formu tako da omogući prikazivanje navedene komponente forme.

SetFocus

Ova metoda aktivira formu i prenosi je na vrh svih formi. Ukoliko forma sadrži komponente, komponenta definisana karakteristikom `ActiveControl` će primiti ulazni fokus (pogledajte karakteristiku `ActiveControl` u poglavlju "Karakteristike u toku dizajniranja").

Show i ShowModal

Ove metode prikazuju formu. Metoda `Show` prikazuje formu ne-modalno, tako da je moguće aktivirati druge forme kada je tekuća forma vidljiva. Metoda `ShowModal` prikazuje formu modalno. Setite se da modalna forma mora biti zatvorena pre nego što korisnik može da nastavi sa radom u okviru aplikacije.

MDI metode

Nekoliko metoda formi rade isključivo sa MDI operacijama. Metoda `ArrangeIcons` uređuje ikone minimiziranih MDI potomaka u okviru MDI prozora roditelja. Metoda `Cascade` - kaskadno (jedan prozor preko drugog) raspoređuje sve neminimizirane MDI prozore potomke. Metoda `Tile` raspoređuje sve otvorene MDI prozore potomke, tako da se prozori ne preklapaju. Metoda `Next` aktivira (postavlja na vrh) sledeći MDI prozor potomak u okviru liste prozora potomaka, dok metoda `Previous` aktivira prethodni MDI prozor potomak u okviru liste prozora potomaka. MDI metode važe samo za MDI prozore roditelje.

Događaji forme

Forma može odgovoriti na širok spektar događaja. Neki od najčešće korišćenih događaja su opisani u ovom poglavlju.

OnActivate

Ovaj događaj nastaje kada se forma prvi put aktivira. Forma može biti aktivirana kao rezultat njenog početnog kreiranja, odnosno kada korisnik preskače sa jedne na drugu formu. Objekt `Application` takođe ima događaj `OnActivate` koji se generiše kada korisnik prelazi sa neke druge aplikacije na Vašu aplikaciju.

OnClose i OnCloseQuery

Kada se aplikacija zatvori, šalje se događaj `OnClose`. Događaj `OnClose` poziva događaj `OnCloseQuery` koji utvrđuje da li forma može da se zatvori. Ukoliko događaj `OnCloseQuery` vrati vrednost `False`, forma nije zatvorena.

OnCreate

Događaj `OnCreate` se pojavljuje prilikom početnog kreiranja forme. Samo jedan događaj `OnCreate` se pojavljuje za svaki slučaj tekuće forme. Upravljač događajima OnCreate možete koristiti za izvršavanje početnih zadataka koji su potrebni za rad forme.

OnDestroy

Događaj `OnDestroy` je suprotnost događaja `OnCreate`. Ovaj događaj možete koristiti da oslobodite memoriju koju je forma dinamički rezervisala, odnosno da uradite druge poslove spremanja.

OnDragDrop

Događaj `OnDragDrop` se pojavljuje prilikom spuštanja objekta na formu. Ukoliko Vaša forma podržava prevlačenje i puštanje objekata, možete odgovoriti na ovaj događaj.

OnMouseDown, OnMouseMove i OnMouseUp

Na događaje `OnMouseDown`, `OnMouseMove` i `OnMouseUp` možete odgovoriti u slučajevima kada korisnik klikne na miša, odnosno pomeri miša u okviru forme.

OnPaint

Događaj `OnPaint` se pojavljuje ukoliko je potrebno ponovo iscrtati formu, što je posledica različitih događaja. Odgovorite na ovaj događaj, ukoliko je potrebno da je Vaša forma uvek prikazana. U većini slučajeva, zasebne komponente vode računa o sopstvenom osvežavanju (ponovnom iscrtavanju), ali u nekim slučajevima je potrebno da se osveže preko forme.

OnResize

Događaj `OnResize` se šalje svaki put kada se promeni veličina forme. Na ovaj događaj možete poželeti da odgovorite kako bi podesili komponente na formi, odnosno osvežili formu.

OnShow

Događaj `OnShow` se pojavljuje pre nego što forma postane vidljiva. Ovaj događaj možete koristiti za izvršavanje bilo kog procesa potrebnog formi, pre nego što se ista prikaže.

Nakon prikazivanja forme generiše se više događaja. Na isti način, prilikom zatvaranja (uništenja) forme, generiše se nekoliko događaja. Kojim redom se ovi događaji generišu? Prilikom kreiranja forme događaji se pojavljuju sledećim redosledom (konstruktor i virtuelne metode `AfterConstruction` su prikazane kao dodatak događajima):

konstruktor forme

događaj `OnCreate`

metoda `AfterConstruction`

događaj `OnShow`

događaj `OnActivate`

Kada se forma uništi, događaji se generišu sledećim redosledom:

događaj `OnCloseQuery`

događaj `OnClose`

metoda `BeforeDestruction`

događaj `OnDestroy`

destruktor forme

Kod većine aplikacija strogo držanje redosleda nije bitno. U nekim slučajevima, redosled može biti izuzetno važan, šta više, kritičan. Poznavanje redosleda pozivanja upravljača događaja, konstruktora i destruktora Vam može uštedeti frustracije u slučajevima kada je redosled bitan.

Object Inspector (inspektor objekata)

U integralni deo Delphijevog okruženja spada Object Inspector (inspektor objekata). Ovaj prozor u saradnji sa dizajnerom forme pomaže u kreiranju komponenti. Dizajner forme će detaljnije biti obrađen u lekciji dana 6, ali pre toga bih želeo da Vam ukratko opišem Object Inspector.

Object Inspector je mesto gde se podešavaju karakteristike u toku dizajniranja; ove karakteristike utiču na komponente u toku rada programa. Object Inspector se sastoji od tri osnovne oblasti:

- Component Selector (selektor komponenti)
- Properties page (kartica karakteristika)
- Events page (kartica događaja)

Do sada ste pomalo koristili Object Inspector, pa ću Vas podsetiti na ono što već znate i pokazati Vam nekoliko mogućnosti koje možda niste znali.

Selektor komponenti

Normalan način izbora komponente je klik mišem na komponentu koja se nalazi na formi. Selektor komponenti pruža alternativni način izbora komponente, kako bi je mogli videti, odnosno izmeniti. Selektor komponenti je padajuća lista koja se nalazi na vrhu prozora Object Inspector.

Najbrži način za izbor komponente obično predstavlja klik mišem na komponentu koja se nalazi na formi. Izbor komponente sa selektora komponenti je uobičajen u slučajevima kada je komponenta koju tražite skrivena iza neke druge komponente, odnosno nalazi se van vidljivog dela forme.

Selektor komponenti prikazuje naziv komponente i klasu iz koje je komponenta izdvojena. Na primer, memo komponenta pod nazivom Memo se može pojaviti na selektoru komponenti kao:

```
Memo: TMemo
```

Naziv klase nije prikazan u padajućoj listi komponenata; prikazuje se samo u gornjem delu selektora komponenti. Da biste odabrali komponentu, kliknite na dugme kojim se aktivira padajuća lista, a zatim kliknite na komponentu koju želite da odaberete.

Selektor komponenti pokazuje samo komponente koje su dostupne u tekućoj formi, odnosno prikazuje naziv same forme. Druge forme i njihove komponente neće biti prikazane, ukoliko nisu aktivne u okviru dizajnera forme.

Nakon izbora komponente iz selektora komponenti, na formi se komponenta takođe odabira. Sadržaj kartica Properties i Events se menja tako da prikazuje karakteristike i događaje koji se odnose na odabranu komponentu. (Zapamtite da je i forma komponenta.) Slika 4.13 prikazuje prozor Object Inspector sa otvorenom listom selektora komponenti.

Slika 4.13
Lista selektora komponenti

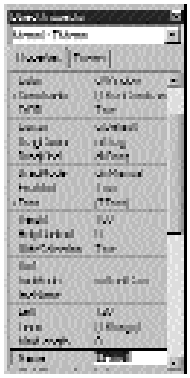
Kartica Properties (karakteristike)

Kartica Properties prozora Object Inspector prikazuje sve karakteristike dostupne u toku dizajniranja za trenutno odabranu kontrolu. Kartica Properties ima dve kolone: kolonu Property (karakteristika) na levoj strani kartice, u kojoj se prikazuje naziv karakteristike; kolona Value (vrednost) se nalazi na desnoj strani kartice, gde se upisuju, odnosno odabiraju vrednosti karakteristika.

Ukoliko odabrana komponenta ima više karakteristika nego što može da stane na prozor Object Inspector, traka za pomeranje će Vam omogućiti da se pozicionirate na neku karakteristiku koja nije vidljiva.

NAPOMENA Ukoliko odaberete više komponenti u okviru forme, Object Inspector će prikazati sve karakteristike koje su zajedničke odabranim komponentama. Ovu mogućnost možete koristiti da biste istovremeno izmenili karakteristike nekoliko komponenti. Na primer, da biste izmenili širinu nekoliko komponenti, možete odabrati sve komponente čiju širinu želite da promenite, a zatim treba da izmenite karakteristiku `Width` u prozoru Object Inspector. Nakon pritiska tastera Enter, odnosno prelaska na drugu karakteristiku, sve komponente koje ste odabrali će imati promenjenu karakteristiku `Width`.

Slika 4.14 prikazuje prozor Object Inspector u trenutku kada je odabrana komponeta `Memo`.



Slika 4.14
Object Inspector prikazuje karakteristike komponente Memo

Karakteristike mogu biti celobrojne vrednosti, specifikacija, skup, neki drugi objekti, stringovi i drugi tipovi. (Karakteristike će biti detaljnije obrađene u lekciji sutrašnjeg dana.) Prozor Object Inspector radi sa svakim tipom karakteristike, u zavisnosti od tipa podataka karakteristike. Delphi sadrži nekoliko ugrađenih editora karakteristika kojima se upravlja unosom podataka za određenu karakteristiku. Na primer, karakteristika `Top` prihvata celobrojnu vrednost (`Integer`). Pošto je tip `Integer` osnovni tip podataka, upravljanje nije komplikovano, pa je editor karakteristika jednostavan. Editor karakteristika za ovaj tip, Vam omogućava da upišete vrednost direktno u kolonu Value za celobrojne karakteristike, kao što su `Top`, `Left`, `Width` i `Height`.

NAPOMENA U većini slučajeva, editor karakteristika proverava parametre za karakteristike u koje se upisuju celobrojne vrednosti. Karakteristika `Width`, na primer, ne može sadržati negativan broj. Ukoliko pokušate da upišete negativnu vrednost u karakteristiku `Width` kontrole, Delphi će ispisati minimalnu vrednost koja je dozvoljena za tekuću kontrolu (obično `0`). Ukoliko upišete string vrednost u karakteristiku koja zahteva celobrojnu vrednost, Delphi će prikazati poruku o grešci. To je upravo i zadatak editora karakteristika, da proveri parametre.

U većini slučajeva editor karakteristika za karakteristike sadrži listu stavki koje možete odabrati. Karakteristike koje imaju specifikaciju, odnosno karakteristike koje imaju Bulovu vrednost, spadaju u ovu kategoriju, ukoliko njihovi osnovni podaci

to zahtevaju. Kada kliknete na kolonu Value sa karakteristikom ovog tipa, editor karakteristika će prikazati dugme padajuće liste na desnoj strani kolone Value. Klikom miša na ovo dugme biće prikazana lista mogućih vrednosti.

SAVET Dvostrukim klikom miša na kolonu Value za određeni tip karakteristika, editor karakteristika će se kretati kroz moguće vrednosti za izbor. Na primer, da biste brzo promenili karakteristiku čija je vrednost `Boolean`, dva puta kliknite mišem na vrednost karakteristike. Pošto su jedini izbor vrednosti `True` i `False`, dvostruki klik miša na vrednost će imati efekat promene vrednosti karakteristike.

Ukoliko bolje pogledate prozor Object Inspector, primetićete da neke karakteristike imaju znak plus ispred naziva. Karakteristike koje predstavljaju skup karakteristika i karakteristike koje su klase, imaju znak plus ispred naziva. Znak plus označava da se karakteristika može proširiti kako bi se prikazao skup, odnosno ako su karakteristike klase, da prikaže karakteristike odabrane klase. Da biste proširili nod karakteristike, dva puta kliknite mišem na kolonu Property za željenu karakteristiku (na naziv karakteristike), odnosno odaberite opciju Expand iz menija sadržaja prozora Object Inspector. Za zatvaranje noda ponovo kliknite dva puta mišem, odnosno odaberite opciju Collapse iz menija sadržaja prozora Object Inspector.

Da biste videli primer podešavanja karakteristike, odaberite formu, a zatim dva puta kliknite mišem na karakteristiku `BorderIcons` u okviru prozora Object Inspector. Nod se proširio i možete videti četiri elementa skupa. Svaki od ova četiri elementa možete uključiti, odnosno isključiti ukoliko je potrebno.

U slučaju karakteristika koje su objekti (slučajevi klase VCL), imate dve mogućnosti editovanja karakteristike. Prva mogućnost je klik mišem na kolonu Value za određenu karakteristiku, a zatim klik mišem na dugme na desnoj strani vrednosti (ukoliko postoji). Ovo dugme ima oznaku tri tačke (`...`) na licu dugmeta. Klikom miša na dugme pozivate editor karakteristike za tekuću kontrolu. Na primer, kliknite mišem na karakteristiku `Font`, a zatim kliknite mišem na dugme koje ima nacrtane tri tačke. Okvir za dijalog Choose Font će biti prikazan kako biste mogli da odaberete font.

Drugi način na koji možete editovati ovaj tip karakteristike je proširivanje noda karakteristike. Biće prikazana karakteristika karakteristike (da, to je tačno), koju možete editovati kao svaku drugu karakteristiku. Ponovo pronađite karakteristiku `Font` i dva puta kliknite mišem na nju. Biće prikazana karakteristika `TFont`. Sada možete menjati veličinu (`Height`), boju (`Color`), naziv (`Name`) fonta, a isto tako možete menjati i druge karakteristike fonta.

Neke karakteristike imaju samo dugme sa tri tačke, što znači da se karakteristika edituje. U prethodnom delu ste koristili komponentu `Image`, da odaberete ikonu za okvir About programa `Multiple`. Kao što ste mogli otkriti, karakteristika Picture komponente `Image`, može biti promenjena samo pozivanjem editora koji pripada karakteristici karakteristike. U ovom slučaju, editor karakteristike je Delphijev `Picture` Editor (editor slika).

Ostatak osigurava da svaka karakteristika zna šta je potrebno uraditi da bi bila predstavljena odgovarajućim editorom karakteristika. Kada se budete upoznavali sa novim komponentama i novim vrednostima, srešćete se sa različitim tipovima editora karakteristika.

Kartica Events (događaji)

Kartica Events prikazuje sve događaje kojima određena komponenta može da upravlja. Korišćenje kartice Events je prilično jednostavno. Da biste kreirali upravljač događajima za željeni događaj, jednostavno kliknite dva puta mišem na kolonu Value koja se nalazi pored događaja kojim želite da upravljate. Nakon toga Delphi umesto Vas kreira funkciju za upravljanje događajima, zajedno sa svim parametrima koji su potrebni za upravljanje željenim događajima. Takođe se prikazuje editor koda sa kursorom postavljenim na upravljač događajem. Sve što treba da uradite je upisivanje koda. Naziv funkcije je generisan na osnovu karakteristike `Name` tekuće komponente i događaja kojim se upravlja. Ukoliko, na primer, imate dugme sa nazivom `OKBtn` i upravljate događajem `OnClick`, generisani naziv funkcije će biti `OKBtnClick`.

Možete dopustiti Delphiju da generiše naziv funkcije za generisanje događajima umesto Vas, odnosno možete sami dodeliti naziv funkciji. Ukoliko dodeljujete naziv funkciji, upišite željeni naziv u kolonu Value, koja se nalazi pored događaja, a zatim pritisnite taster Enter. Biće prikazan editor koda sa kursorom koji se nalazi na upravljaču događajem; upravljač događajem će imati naziv koji ste mu dodelili.

Prilikom pokretanja, prevođenja, odnosno snimanja junita, Delphi će ukloniti sve prazne upravljače događajima. Na primer, recimo da ste kreirali upravljač događajem za događaj OnCreate, ali niste upisali kod. Prilikom sledećeg pokretanja programa, prevođenja, odnosno snimanja junita, Delphi će ukloniti upravljač događajem koji ste upravo kreirali, pošto ne sadrži kod. Na ovaj način je Delphi kreiran, i to ima svoj smisao, mada Vas može zbuniti ukoliko niste sigurni šta se dešava. Ukoliko ne želite da Delphi ukloni upravljač događajem, možete upisati bilo koji kod, odnosno možete upisati komentar, tako da upravljač događajem ne može biti uklonjen.

Nakon kreiranja funkcije za upravljanje događajem za određenu komponentu, ovu funkciju možete koristiti za bilo koju komponentu koja upravlja istim događajima. Ponekad je zgodno imati nekoliko dugmadi koja koriste isti događaj `OnClick`. Pođimo korak dalje, možete imati opciju glavnog menija, opciju padajućeg menija i dugme na traci sa alatima, a da sve navedene komponente koriste isti upravljač događajem, `OnClick`. Korišćenje koda za više elemenata naučićete da cenite, kako budete sticali iskustvo u radu sa Delphijem. Čak, iako radite sa tri različite komponente, one mogu deliti zajednički upravljač događajima `OnClick`. Kolona Value na kartici Events sadrži dugme za padajuću listu koje se može koristiti za prikaz spiska

svih upravljača događajima koji su kompatibilni sa tekućim događajima. Sve što treba da uradite je da odaberete događaj sa liste.

Usidreni IDE prozori (Dockable IDE Windows)

Novina u Delphiju 4 su usidreni prozori (dockable windows).

Usidreni prozor (*dockable window*) je prozor koji se može prevući (korišćenjem miša) sa trenutne pozicije i usidriti na jedno od sidrišta Delphi okruženja.

Sidrište (*dock site*) je određena lokacija na okruženju gde se može postaviti usidreni prozor. Okruženje može imati nekoliko sidrišta.

Skoro svaki prozor u Delphiju može biti usidren. Ovo uključuje Project Manager, Code Explorer, Object Inspector, prozor Watch List, prozor Message itd. U suštini, postoji samo nekoliko Delphijevih prozora koji ne mogu biti usidreni.

Usidreni prozori su me obradovali; da li sam jedini? Ne, nikako. U drugim programima se ni ne interesujem za usidrene prozore. U Delphi okruženju usidreni prozori mi pružaju veću produktivnost i zato ih volim.

Sidrišta

O usidrenim prozorima se ne može diskutovati, a da se ne pomenu sidrišta. Naravno, možete prevlačiti prozor preko ekrana i postavljati ga gde poželite. Ovo može biti uzrok stvaranja gomile razbacanih prozora po celom ekranu. Da bi usidreni prozori imali smisla, morate imati mesto na koje ćete ih usidriti. U Delphijevom okruženju se obično koristi prozor editora koda.

Editor koda sadrži tri sidrišta. Jedno sidrište se nalazi duž leve ivice prozora editora koda. Ovo sidrište se nalazi na mestu gde se pojavio Code Explorer prilikom prvog startovanja Delphija. Drugo sidrište se nalazi duž donje ivice prozora editora koda. Početna konfiguracija Delphija postavlja prozor za poruke na donje sidrište (nije vidljiv sve dok nema poruka koje treba prikazati). Treće sidrište na editoru koda se nalazi duž leve ivice prozora editora koda. Ova tri sidrišta su Vam dovoljna da biste potpuno prilagodili Delphi okruženje svojim potrebama.

Postoji još jedan tip sidrišta koji bih želeo da spomenem. Ukoliko imate otvoren prozor za alate (kao što je Project Manager), na ovaj prozor možete usidriti naredni prozor sa alatima. Ovo omogućava da se dva, odnosno više Delphi prozora smeste u okviru istog prozora sa alatima. Na primer, Code Explorer i Project Manager mogu biti spojeni u jedan plutajući prozor sa alatima. Plutajući prozor sa alatima sadrži pet sidrišta: levo, desno, gornje, donje i centralno.

Kada usidrite prozor u centar alata sa prozorima, ovaj prozor postaje prozor sa karticama. Svaka kartica u okviru prozora sadrži naslov koji je ispisan na jezičku kartice. Možda Vam se čini da ovo nema smisla, pa ću Vam pokazati kako da spojite dva prozora za alate.

Eksperimentisanje sa usidrenim prozorima

Pošto je bolje uraditi vežbu nego objašnjavati veze između različih prozora, počećemo sa osnovnim operacijama za usidravanje i polako prelaziti na kompleksna usidravanja prozora. Ova vežba ne traje dugo, a može se pokazati kao veoma upečatljiv primer. Da počnemo:

1. Kreirajte novu aplikaciju i pređite na editor koda. Uočite da je Code Explorer usidren na levoj strani editora koda.
2. Kliknite na traku koja se nalazi na vrhu Code Explorer-a i povucite je na desno. Uočite da ste prevukli sivi pravougaonik. Ovaj pravougaonik ukazuje na mesto gde će se pojaviti Code Explorer kada prestanete sa prevlačenjem.
3. Prevucite Code Explorer u sredinu editora koda i pustite taster miša. Prozor Code Explorer-a postaje plutajući prozor sa alatima.
4. Kliknite na naslovnu traku prozora Code Explorer i prevucite ga nazad na levu stranu editora koda. Nakon što kursor dostigne levu ivicu editora koda, pravougaonik koji prevlačite će upasti na svoje mesto. Otpustite taster miša, pa će Code Explorer ponovo biti usidren na editor koda.
5. Odvojite ponovo Code Explorer i prevucite ga na dno editora koda. Kada kursor dostigne donju ivicu prozora editora koda, pravougaonik koji se prevlači menja oblik i dobija širinu prozora editora koda. Otpustite taster miša. Code Explorer je usidren na editor koda preko donjeg sidrišta. Uočite da hvatač za prevlačenje postaje vertikalan kada se Code Explorer usidri na donje sidrište.
6. Pomerite Code Explorer ponovo na sidrište koje se nalazi na levoj strani prozora editora koda. Editor koda i Code Explorer izgledaju potpuno isto kao na početku.

Ova vežba je jednostavna, ali Vam može dati ideju šta možete uraditi sa usidrenim prozorima na osnovnom nivou. Sledeća vežba je interesantnija. Pratite sledeće korake:

1. Odvojte Code Explorer i pomerite ga na desno. Postavite ga bilo gde sa desne strane editora koda.
2. Veličina prozora Code Explorer je skoro kvadratnog oblika.
3. Odaberite opciju View➥Project Manager u okviru glavnog menija, nakon čega će biti prikazan Project Manager.

4. Prevucite Project Manager preko prozora Code Explorer. Kada u toku prevlačenja pravougaonik upadne u središte prozora Code Explorer, otpustite taster miša. Plutajući prozor sa alatima bi trebao da izgleda slično kao na slici 4.15. Uočite da plutajući prozor sa alatima postaje prozor sa karticama i dobija naslov Tool Windows. Možete kliknuti na bilo koji jezičak kartice, da biste videli Code Explorer, odnosno Project Manager.

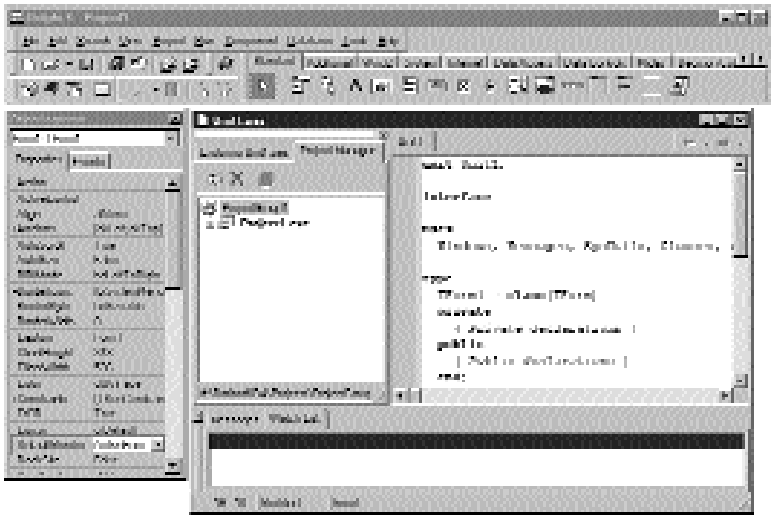
Slika 4.15
Code Explorer i Project Manager zajedno usidreni u prozor sa alatima (tool window)

5. Prevucite prozor sa alatima na levu stranu editora koda i usidrite ga. Sada imate usidren Code Explorer i Project Manager na mestu odakle ih možete lako pozvati kada se za to ukaže potreba.

Da li počinjete da shvatate? U okviru jednog prozora sa karticama možete imati koliko god prozora želite. Uradimo još jednu vežbu sa istim ciljem. Obično želite da Vam prozor debagera bude vidljiv u toku rada sa debagerom (debagiranje je obrađeno u lekciji dana 10 "Debagiranje Vaših aplikacija"). Dozvolite mi da Vam pokažem kako da Vam prozor Watch List bude pri ruci sve vreme. Izvršite sledeće korake:

1. Kliknite desnim tasterom miša na editor koda i odaberite opciju Message View u okviru menija sadržaja. Na donjem sidrištu editora koda će se pojaviti prozor sa porukama.
2. Odaberite opciju View➥Debug Windows➥Watches u okviru glavnog menija. Prozor Watch List će biti prikazan kao plutajući prozor.
3. Prevucite prozor Watch List i usidrite ga u središtu prozora za poruke. Na sidrištu će se pojaviti dva jezička kartica: Messages (poruke) i Watch List.

Sada možete kliknuti na jezičak Watch List u dnu editora koda svaki put kada poželite da pregledate stavke koje ste definisali za pregled. Prozor za poruke (Message window) sam brine o sebi, i pojaviće se, odnosno vratiti u prozor sa karticama svaki put kada postoji poruka koju treba prikazati. Slika 4.16 prikazuje okruženje nakon što je završena poslednja vežba.



Slika 4.16
četiri prozora sa alatima usidrena na editor koda

Zabranjeno usidravanje

Ponekad ćete poželeti da se određeni prozor ne može usidriti. Iako je dobro imati usidrene prozore, ponekad je teško pronaći mesto za prozor koji ne želite usidriti. Čini se da prozor želi da se usidri bilo gde da ga postavite. Dobra vest je da mogućnost usidravanja možete isključiti za bilo koji prozor.

Svaki prozor za alate koji ima mogućnost sidrenja ima stavku menija sadržaja pod nazivom Dockable. Ukoliko je ova opcija odabrana, prozor se može usidriti. Ukoliko ova opcija nije odabrana, prozor nije moguće usidriti i možete ga postaviti bilo gde u okviru Delphijevog okruženja.

Prozori koji se mogu usidriti predstavljaju veoma dobru osobinu Delphijevog okruženja. Prozore sa alatima možete rasporediti uglavnom onako kako Vam odgovara. Više ne morate da lovite Project Manager, Watch List, odnosno Object Inspector koji su skriveni ispod drugih prozora. Prozor koji tražite je udaljen samo nekoliko kliktaja na taster miša.

Primer MDI programa

Da bi učvrstili znanje iz današnje lekcije o projektima i formama, kreirajmo MDI aplikaciju. Ova aplikacija će Vam omogućiti da otvorite i snimite grafičke datoteke kao što su bitmapirane slike, ikone i meta-datoteke. Da biste završili zadatak, treba da razradite plan. Šta je potrebno da se uradi:

1. Kreirajte formu glavnog prozora (MDI roditelj), zajedno sa menijem.
2. Upišite kod za opcije menija File➥Open i File➥Save.

3. Upišite kod za stavke menija Cascade, Tile i Arrange All.
4. Kreirajte MDI formu potomka.
5. Kreirajte okvir About.
6. Zavalite se u stolicu i divite se Vašem radu.

Nema smisla odugovlačiti (vreme je novac!), pa krenimo na posao.

Kreiranje forme glavnog prozora

Prvo kreirajte formu glavnog prozora. Glavni prozor MDI aplikacije mora imati u karakteristiku `FormStyle` upisan podatak `fsMDIForm`. Takođe treba dodati meni aplikacije, kao i okvire za dijalog File Open i File Save. Pratite sledeće korake:

1. Pokrenite Delphi i odaberite opciju File➥New Application u okviru glavnog menija.
2. Za glavnu formu promenite karakteristiku `Name` u `MainForm`.
3. Promenite karakteristiku `Caption` u `Picture Viewer`.
4. Promenite visinu (`Height`) u `450` i širinu (`Width`) u `575` (odnosno koristite vrednosti koje Vam odgovaraju rezoluciji ekrana).
5. Promenite karakteristiku `FormStyle` u `fsMDIForm`.

U redu, sada imate završen osnovni deo forme. Sledi dodavanje menija. Pošto još uvek nismo obradili dizajner menija (Menu Designer), koristićemo lakši način za kreiranje menija aplikacije. Da bi ovo uradili, koristićemo prednost opcije Delphija koja omogućava uvoz unapred definisanog menija:

1. Kliknite na jezičak Standard u okviru palete komponenti, a zatim kliknite na dugme MainMenu.
2. Kliknite na formu, da biste postavili komponentu `MainMenu`. Nije bitno gde postavljate komponentu, pošto ikona koja predstavlja meni samo rezerviše mesto i neće biti prikazana na formi u toku rada programa. Na ovaj način se nevizualne komponente pojavljuju na formi.
3. Promenite naziv karakteristike `Name` u `MainMenu`.
4. Dva puta kliknite mišem na komponentu `MainMenu`. Prikazaće se dizajner menija. (O dizajneru menija ćete učiti detaljnije u lekciji dana 6.)
5. Postavite kursor na dizajner menija i kliknite na desni taster miša. Odaberite opciju Insert from Template u okviru menija sadržaja. Pojaviće se okvir za dijalog Insert Template (ubacivanje obrasca). Slika 4.17 prikazuje okvir za dijalog Insert Template sa dizajnerom menija u pozadini.

Slika 4.17
Dizajner menija sa otvorenim okvirom za dijalog Insert Template

6. Odaberite MDI Frame Menu i kliknite na dugme OK. U okviru dizajnera menija će biti prikazan odabrani meni.

7. Kliknite na sistemski okvir za zatvaranje dizajnera menija i zatvorite prozor.

Sada možete da se vratite na glavnu formu. Uočite da ste napravili meni na Vašoj formi. Da biste videli kompletan meni, možete kliknukti na stavke koje se nalaze na traci menija. Nemojte pokušavati da kliknete mišem na stavke padajućih menija, pošto ćemo to uraditi kasnije. Uočite da postoji prilično mnogo opcija na Vašem meniju. Neće Vam sve biti potrebne, ali pre nego što ih uklonite, sačekajte da završimo sledeći deo vežbe.

Sada je potrebno pripremiti okvire za dijalog File Open i File Save:

1. Kliknite na jezičak Dialogs u okviru palete komponenti. Odaberite komponentu OpenPictureDialog i postavite je na formu. Ikona komponente OpenPictureDialog može biti postavljena bilo gde u okviru forme.

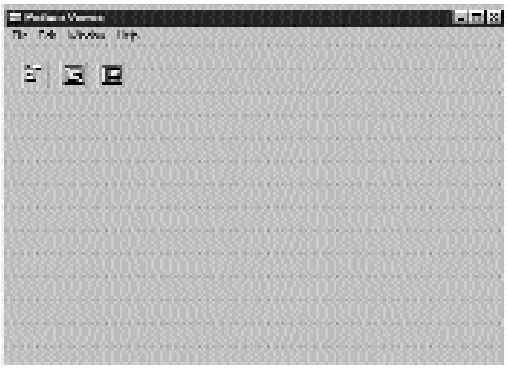
2. Promenite karakteristiku `Name` okvira za dijalog Open u OpenPictureDialog.

3. Promenite karakteristiku `Title` u `Open` a `Picture for Viewing` (Otvaranje slike za pregled).

4. Dodajte komponentu `SavePictureDialog`.

5. Promenite karakteristiku `Name` komponente u `SavePictureDialog` i karakteristiku `Title` u `Save` a `Picture` (Snimanje slike).

Vaša forma bi trebalo da izgleda približno kao i forma na slici 4.18.



Slika 4.18
Tekući izgled forme

Pisanje koda za stavke menija File➥Open i File➥Save As

Sada ste spremni da napišete kod koji će implementirati stavke menija File➥Open i File➥Save As. Delphi omogućava lako i jednostavno pisanje upravljača menija. Do sada niste kreirali MDI formu potomka, ali znate dovoljno da biste napisali kod za upravljanje menijem. Zapamtite da aplikaciju nećete moći prevesti sve dok ne kreirate MDI formu potomka. Stoga počnimo:

1. Na glavnoj formi odaberite opciju File➥Open u okviru menija. Biće kreiran upravljač događaja za ovaj meni, a zatim i editor koda.

2. Upišite kod tako da upravljač događajem izgleda ovako:

```
procedure TMainForm.Open1Click(Sender: TObject);
var
 Child : TChild;
begin
 if OpenPictureDialog.Execute then begin
    Child := TChild.Create(Self);
    with Child.Image.Picture do begin
      LoadFromFile(OpenPictureDialog.FileName);
      Child.ClientWidth := Width;
      Child.ClientHeight := Height;
    end;
    Child.Caption := ExtractFileName(OpenPictureDialog.FileName);
    Child.Show;
  end;
end;
```

Ovaj kod prvo izvršava okvir za dijalog File Open i prihvata naziv datoteke. Ukoliko korisnik klikne mišem na dugme Cancel u okviru za dijalog File Open, funkcija za otvaranje datoteke neće ništa uraditi. Ukoliko korisnik klikne mišem na dugme OK u okviru za dijalog File Open, kreira se novi objekt `TChild` (`TChild` će biti naziv MDI klase potomka koju ćete kasnije kreirati). Datoteka sa slikom se učitava u

komponentu `Image` u okviru forme potomka, a klijent oblast MDI prozora potomka se smanjuje na odgovarajuću veličinu slike. Na kraju, karakteristika `Caption` dobija naziv odabrane datoteke i prozor potomak se prikazuje.

Funkcija `ExtractFileName` u okviru metoda `Open1Click` se koristi da odvoji naziv datoteke iz stringa koji sadrži put i naziv datoteke, a nalazi se u karakteristici `FileName` u okviru komponente OpenPictureDialog. Slične funkcije su: `ExtractFilePath`, `ExtractFileDir` i `ExtractFileExt`.

Prisetite se dela jučerašnje lekcije o pozivu metoda Free za sve objekte koji su dinamički kreirani. Uočite da izgleda kao da sam prekršio pravilo u prethopdnom isečku koda. U stvarnosti to nije tačno, pošto VCL preuzima odgovornost za oslobađanje memorije koju su rezervisali MDI prozori potomci. Uočite da je jedini parametar konstruktora `TChild` parametar `Self`. To pokazuje VCL-u da je vlasnik MDI potomka prozor MDI forme. Kada se uništi MDI forma (kada se zatvori aplikacija), treba biti siguran da su obrisani svi njegovi MDI objekti potomci.

3. Pritisnite funkcijski taster F12 da se ponovo vratite na formu. Zatim odaberite opciju File➥Save As u okviru menija. Biće prikazan upravljač događaja opcije File➥Save As.

4. Upišite kod tako da upravljač događaja File➥Save As izgleda ovako:

```
procedure TMainForm.SaveAs1Click(Sender: TObject);
begin
  if SavePictureDialog.Execute then
    with ActiveMDIChild as TChild do
      Image.Picture.SaveToFile(SavePictureDialog.FileName);
end;
```

Kod za stavku menija File➥Save As je jednostavan. Prve dve linije proveravaju da li je MDI prozor potomak aktivan. Ukoliko je prozor aktivan, biće prikazan okvir dijaloga File Save. Ukoliko korisnik klikne mišem na dugme OK, slika će biti snimljena na disk korišćenjem metoda `SaveToFile` klase `TPicture`.

U prethodnom kodu možete primetiti kako se koristi operator as. Karakteristika `ActiveMDIChild` vraća pointer na objekt `TForm`. Ono što Vam je potrebno u tom slučaju je pointer na objekt `TChild` (Vaša MDI klasa potomak, izdvojena iz klase `TForm`), kako bi mogli da pristupite karakteristici `Image` MDI forme potomka. Operator as raspoređuje promenljivu `ActiveMDIChild` pointeru `TChild`. Ukoliko iz nekog razloga operator as nije u mogućnosti da izvrši raspoređivanje blok koda koji se nalazi iza iskaza, as se ignoriše.

Pre nego što nastavimo, dobro bi bilo da snimimo projekt. Odaberite opciju File➥Save All u okviru glavnog menija. Snimite Unit1 (generički naziv koji Delphi dodeljuje novom junitu) kao `PctViewU`, a projekt kao `PictView`.

Pisanje koda za meni Window

Sada možete upisati kod za meni Window. Ovaj deo je jednostavan:

1. Vratite se na formu pritiskom na funkcijski taster F12. Odaberite opciju Window➥Tile u okviru menija forme.
2. Treba da upišete samo jednu liniju koda u upravljač događajem. Završni upravljač događajem će izgledati ovako:

```
procedure TMainForm.Tile1Click(Sender: TObject);
begin
  Tile;
end;
```

3. Vratite se na formu i ponovite proces za opciju Window➥Cascade. Krajnja funkcija bi trebalo da izgleda ovako:

```
procedure TMainForm.Cascade1Click(Sender: TObject);
begin
  Cascade;
end;
```

4. Ponovite korake za stavku menija Window➥Arrange All. Treba dodati samo jednu liniju koda u osnovni deo funkcije: `ArrangeIcons;`

U redu, završili ste sa radom na glavnoj formi. Možete se prebaciti na kreiranje MDI forme potomka.

Kreiranje MDI forme potomka

MDI forma potomak je iznenađujuće jednostavna. U suštini, ne treba da pišete nikakav kod. Samo pratite sledeće korake:

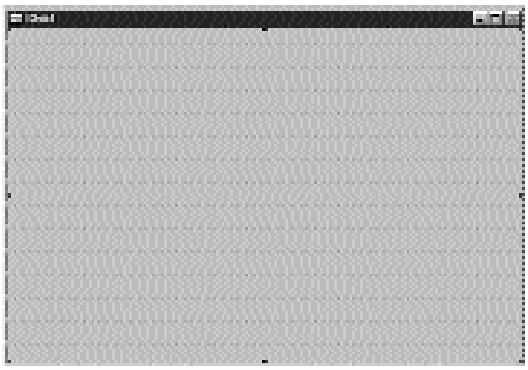
1. Kreirajte novu formu korišćenjem dugmeta New Form u okviru trake sa alatima, odnosno birajući opciju File➥New Form u okviru glavnog menija.
2. Promenite naziv karakteristike `Name` u `Child`. Karakteristika `Caption` može biti ignorisana, pošto ćete naziv okvira za dijalog upisivati u toku rada programa.
3. Promenite karakteristiku `FormStyle` u `fsMDIChild`. Ovo je neophodno da bi forma bila tretirana kao MDI prozor potomak.

Ovo bi bilo sve u vezi forme. Postavimo sada komponentu Image na formu. Komponenta će prikazati grafičku datoteku koju korisnik odabere.

1. Kliknite na jezičak Additional u okviru palete komponenti. Kliknite na dugme Image i postavite komponentu `Image` bilo gde unutar forme.
2. Promenite naziv karakteristike `Name` u `Image`.
3. Promenite karakteristiku `Stretch` u `True`.

4. Promenite karastike `Align` u `alClient`. Komponenta `Image` se proširuje i ispunjava klijent oblast forme.
5. Odaberite opciju File➥Save i snimite junit forme pod nazivom `MDIChild`.
6. Prebacite se na editor koda (pritisnite funkcijski taster F12, da biste preskočili između dizajnera forme i editora koda). Kliknite na jezičak `PctViewU`. Sada odaberite opciju File➥Use Unit u okviru glavnog menija, odaberite junit `MDIChild`, a zatim kliknite na dugme OK. Zbog toga je prevodilac zadovoljan kada ukažete na objekt `TChild`.

Forma je prilično neugledna u ovom trenutku, ali bi trebala da izgleda kao forma na slici 4.19.



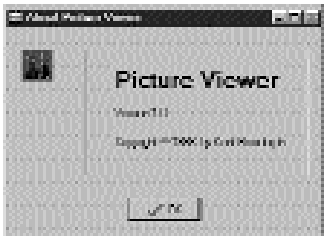
Slika 4.19
MDI forma potomak sa komponentom Image

Preostaje Vam da kreirate okvir About, ali trenutno ste verovatno nestrpljivi da isprobate program. Pokušajte, kliknite mišem na dugme Run. Nakon par trenutaka, program će biti prikazan. Možete odabrati opciju menija File➥Open i otvoriti grafičku datoteku (bilo koja datoteka sa nastavkom `.bmp`, `.wmf`, odnosno `.ico`).

Uočite da se MDI prozor potomak prilagođava veličini otvorene slike. Otvorite nekoliko datoteka, a zatim isprobajte opcije Cascade i Tile u okviru menija Window. Ukoliko želite, možete snimiti datoteku pod drugim nazivom, korišćenjem opcije menija File➥Save As.

Kreiranje okvira About

Do sada ste trebali da naučite dovoljno o jeziku Delphi, da biste samostalno kreirali okvir About. Kreirajte okvir About tako da izgleda slično okviru na slici 4.20. Ukoliko se zaglavite, možete se vratiti nekoliko strana unazad i pogledati korake za kreiranje okvira About u prethodnim poglavljima današnje lekcije. Slobodno kreirajte okvir About da bude prilagođen Vama.



Slika 4.20
Okvir About za aplikaciju

Nakon što ste kreirali okvir, izvršite sledeće korake da biste ga pozvali iz menija:

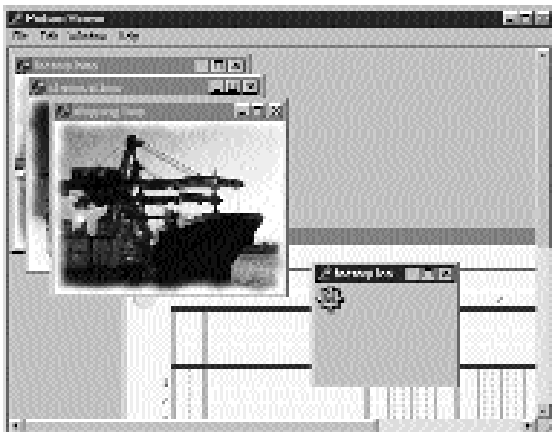
1. Promenite karakteristiku `Name` u `AboutBox`.
2. Snimite junit sa novim nazivom `PVAboutU`.

Delphi podržava duga imena datoteka. Ja koristim 8.3 konvenciju za dodelu naziva u ovoj knjizi iz razloga koji imaju veze sa elektronskim izdavaštvom. Za aplikacije koje Vi pišete, možete koristiti prednosti dugih imena datoteka.

3. Prebacite se na jezičak `PctViewU` u okviru editora koda (pritisnite F12). Odaberite opciju File➥Use Unit u okviru glavnog menija i ubacite ga u junit `PVAboutU`.
4. Pritisnite F12 da biste se vratili na glavnu formu. Odaberite opciju Help➥About iz glavnog menija. Preći ćete u editor koda na upravljač događajem `OnClick`.
5. Dodajte sledeću liniju u upravljač koda:

```
AboutBox.ShowModal;
```

Ovo bi bilo sve za sada. Kliknite na dugme Run i isprobajte opciju About u okviru menija Help. Slika 4.21 prikazujue program Picture Viewer sa nekoliko otvorenih prozora potomaka.



Slika 4.21
Program Picture Viewer u radu

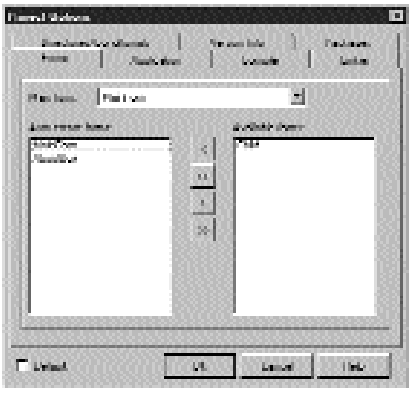
Doterivanje programa

Od ovog trenutka program je funkcionalan, ali nije doteran. Ipak, za programiranje od 30 minuta ovo nije tako loše! Trenutno postoji nekoliko problema koji su vezani za program. Ukoliko pokušate da otvorite datoteku koja nije slika, primetićete da program izbacuje izuzetak. Izuzetke i upravljanje izuzetcima ću obraditi u lekciji dana 14 "Napredno programiranje". Takođe, postoji još dosta opcija menija kojih treba da se oslobode. Ovo će biti pokazano u lekciji dana 6, pošto će tada detaljnije biti obrađen dizajner menija.

Postoje dva problema vezana za ovaj program, za koje mislim da ih možete obraditi, pošto se mogu jednostavno ispraviti. Prvo, da li ste uočili da se prilikom startovanja aplikacije pojavljuje prazan MDI prozor potomak? Uzrok leži u činjenici da Delphi aplikacija automatski kreira sve forme prilikom pokretanja. U slučaju MDI potomka prozor će biti prikazan u trenutku kada aplikacija postane vidljiva. Svaku MDI formu potomak kreirate kada Vam je potrebna, pa će Vam smetati što Delphi automatski kreira formu umesto Vas.

Srećom, uklanjanje MDI prozora potomka iz liste forme za automatsko kreiranje je jednostavno. Izvršite sledeće korake:

1. Odaberite opciju Project➥Options u okviru glavnog menija. Okvir za dijalog Project Options će biti prikazan.
2. Ukoliko je potrebno, kliknite na jezičak Forms. Biće prikazane forme koje se automatski kreiraju.
3. Kliknite na formu potomka, a zatim kliknite na dugme >. Ovo uklanja formu potomka iz liste za automatsko kreiranje i prebacuje formu u listu dostupnih formi. Slika 4.22 prikazuje okvir za dijalog Project Options nakon pomeranja forme potomka u listu formi koje su dostupne.



Slika 4.22
Okvir za dijalog Project Options

Ponovo pokrenite program. Ovaj put neće biti prikazana prazna MDI forma potomak.

Ukoliko uklonite formu iz liste za automatsko kreiranje, morate biti sigurni da kreirate formu koja prethodi njenom korišćenju. Ukoliko ne kreirate ovakvu formu, pointer na formu se ne inicijalizuje, što znači da pointeru još uvek nije dodeljena vrednost koja ima smisao. (Prisetite se da se pointer automatski kreira u okviru Delphija.) Pokušaj korišćenja Delphija će kao rezultat dati pogrešan pristup, odnosno grešku u ponašanju u okviru programa. Nakon što uklonite formu iz liste za automatsko kreiranje, preuzimate odgovornost da proverite da li je forma kreirana pre nego što će se koristiti.

Vaša aplikacija ima još jedan problem koji bih hteo da obradim. Nakon klika mišem na dugme za zatvaranje nekog MDI prozora, primetićete da se prozor minimizira, umesto da se zatvori. Veroveli ili ne, ovo je standardno ponašanje koje preporučuje Microsoft. Standardno ponašanje ili ne, ono je uvrnuto, zato ga treba izmeniti, kako bi se klikom miša na dugme za zatvaranje prozor zatvorio (kao što bi svaka razumna osoba očekivala). Da biste ovo uradili izvršite sledeće korake:

1. Vratite formu prozora potomka u dizajner forme. Uverite se da je odabrana baš forma, a ne komponenta `Image` koja se nalazi na formi (odaberite `Child` u okviru selektora komponenti u gornjem delu Object Inspector-a, ukoliko je potrebno).

2. Dva puta kliknite mišem na kolonu Value, pored događaja `OnClose` u okviru Object Inspector-a. Dodajte liniju koda u upravljač događajem, tako da isti izgleda ovako:

```
procedure TChild.FormClose(Sender: TObject; var Action:
TCloseAction);
begin
  Action := caFree;
end;
```

   Podešavanje dejstva zatvaranja na `caFree` ukazuje VCL-u da zatvori prozor potomak i oslobodi memoriju dodeljenu ovom prozoru. Sada će se prozor potomak ponašati onako kako bi trebalo da se ponaša kada se pritisne dugme za zatvaranje prozora.

3. Ponovo pokrenite program i uverite se da se program ponaša kako je navedeno.

Zaključak

Delphi okruženje može izgledati zastrašujuće sve dok ga ne upoznate. Ukoliko ga učite malo po malo, neće Vam izgledati tako strašno. U današnjoj lekciji ste naučili više o različitim delovima koji sačinjavaju Delphijevo okruženje. Zatim ste naučili kako se koriste projekti za kreiranje izvršnih datoteka. Takođe ste naučili više o formama. Otkrili ste kako Delphi radi sa okvirima za dijalog i drugim prozorima potomcima. Otkrili ste novine vezane za prozor Object Inspector i kako se Object Inspector koristi za promenu karakteristika komponente. Takođe ste naučili o sidrištima u okviru Delphijevog okruženja. Na kraju ste kreirali program koji radi nešto interesantno. U lekciji sutrašnjeg dana ćete naučiti više o modelu vizuelnih komponenti.

Radionoica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstirte razumevanje obrađenog materijala, kao i vežbe koje Vam omogućavaju da steknete iskustvo u korišćenju gradiva koje ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A, "Odgovori na kviz pitanja".

Pitanja i odgovori

P **Delphijeva traka sa alatima nema dugmad za opcije koje često koristim. Da li mogu menjati sadržaj trake sa alatima?**

O Apsolutno. Traka sa alatima se može u potpunosti prilagođavati. Možete dodavati, odnosno uklanjati dugmad, ukoliko Vam to odgovara.

P **Postavio sam više komponenti Label na formu, a zatim pokušao da ih sve odaberem, prevlačeći preko njih mišem. Kao rezultat dobio sam jednu veliku komponentu Label. Šta sam loše uradio?**

O Zaboravili ste da isključite opciju za višestruko postavljanje komponenti. Nakon što postavite više komponenti na formu, potrebno je da kliknete na dugme sa strelicom u okviru palete komponenti, da biste isključili opciju za višestruko postavljanje komponente.

P **Pokušao sam da postavim jedan od prozora Delphi okruženja pored editora koda. Svaki put kada sam pokušao, prozor je pokušao da se usidri na editor koda. Kako da sprečim usidravanje prozora?**

O Kliknite desnim tasterom miša na prozor i isključite opciju Dockable.

P **Kada sam greškom upisao karakter u karakteristiku Top u okviru svoje forme, dobio sam poruku o grešci. Shvatam da bih trebao da upišem broj umesto karaktera, ali odakle je došla poruka?**

O Object Inspector zna koji tip vrednosti je ispravan za određenu karakteristiku. Karakter nije prihvatljiva vrednost za karakteristiku u koju se upisuje celobrojna vrednost, pa se pojavljuje poruka greške. U nekim sklučajevima editor karakteristika proverava vrednost unosa.

P **Šta mi je potrebno da bi moja aplikacija postala MDI aplikacija?**

O Samo treba da se uverite da glavna forma ima karakteristiku `FormStyle` podešenu na `fsMDIForm` i da MDI forme potomci imaju karakteristiku `FormStyle` podešenu na `fsMDIChild`.

P **Koja je razlika između okvira za dijalog i prozora potomka u okviru Delphija?**

O Nema prave razlike. Forma okvira za dijalog može da ima određene promene kao što su okvir okvira za dijalog umesto okvira promenljive veličine; dugmad OK, Cancel i Help; odnosno da nema dugmad za minimiziranje i maksimiziranje forme. Ipak, okviri za dijalog su forme kao i svaka druga. Forma može imati pojavu okvira za dijalog, odnosno pojavu prozora potomka, ali ipak, forma ostaje forma.

P **Da li mogu da proverim da li na junitu na kom radim ima grešaka, a da ne pokrećem program?**

O Da. Odaberite opciju Project➥Syntax Check u okviru glavnog menija. Delphi će prevesti sve junite koji su bili promenjeni nakon poslednjeg prevođenja i prijaviće ukoliko postoji greška.

Kviz

1. Kako da pozovete okvir za dijalog Customize za glavni prozor?
2. Nakon što ste otvorili okvir za dijalog Customize, kako možete dodati dugmad na traku sa alatima?
3. Kako da uklonite dugmad iz trake sa alatima?
4. Koji je najlakši način da postavite više komponenti istog tipa na formu?
5. Koji je najlakši način da postavite komponentu na središte forme?
6. Nabrojte tipove datoteka koji su potrebni za kreiranje Delphi aplikacije.
7. Koji VCL metod koristite da prikažete formu ne-modalnu?
8. Koji VCL metod koristite da prikažete formu modalnu?
9. Kako možete prikačiti događaj na upravljač događajem koji je prethodno bio definisan?
10. Kada koristite Object Inspector, kako možete da pregledate specifikaciju opcija za određenu karakteristiku?

Vežbe

1. Uklonite Pause, Step Over i Trace Into dugmad iz Delphijeve trake za alate View. Dodajte Compile, Build i Syntax Check dugmad na traku sa alatima.
2. Resetujte traku sa alatima na gereričke vrednosti.
3. Provedite neko vreme pregledajući komponente na svakoj strani palete komponenti. Postavite komponente koje Vas interesuju na formu i eksperimentišite sa njima.

4. Kreirajte novi direktiorijum na Vašem hard disku. Kreirajte novu aplikaciju na Delphiju. Dodajte tri nove forme projektu (ukoliko želite, forme mogu biti prazne). Snimite projekt u novi direktorijum koji ste kreirali, a zatim pokrenite program. Zatvorite program. Zatim istražite direktoruijum u kom se nalazi snimljeni projekt. Uporedite datoteke koje ovde vidite sa tipovima datoteka koji se nalaze u tabeli 4.1.
5. Pokrenite program Picture Viewer, koji ste prethodno kreirali. Otvorite nekoliko grafičkih datoteka. Prevlačite MDI prozore potomke preko prozora roditelja. Pokušajte da pomerite prozor potomak van prozora roditelja. Šta se dešava?
6. Eksperimentišite sa usidravanjem različitih prozora Delphi okruženja sa sidrištima na editoru koda.
7. Pokrenite novu aplikaciju postavite nekoliko komponenti na formu. Kliknite na svaku komponentu i pregledajte karakteristike svake komponente u Object Inspector-u.
8. Kreirajte praznu formu. Dva puta kliknite mišem na kolonu Value pored karakteristike `Color`, da biste pozvali okvir za dijalog Color. Odaberite boju, a zatim kliknite na dugme OK.

Dan 5

Model vizuelnih komponenti

U današnjoj lekciji ću Vam objasniti model vizuelnih komponenti i Borlandovu biblioteku vizuelnih komponenti (VCL - Visual Component Library). Pre nego što pređemo na obrađivanje VCL-a, objasniću Vam ukratko kosture klasa (class frameworks). U ovom poglavlju možete pronaći:

- osnove kostura
- pregled biblioteke vizuelnih komponenti (VCL)
- VCL klase, uključujući forme, aplikacije i klase komponenti

Osnove kostura (frameworks)

"U početku beše C..." Pa, ne baš. Kada je u pitanju programiranje za operativni sistem Windows, ova izjava je tačna. U početku, velika većina Windows programa su bili pisani na jeziku C. U suštini, Windows Application Programming Interface (API - interfejs za progranmiranje aplikacija) je jedna velika zbirka C funkcija - ima ih na stotine. Još uvek hiljade programera oko nas, bez sumnje, piše Windows programe na jeziku C.

Tokom vremena, ljudi u Borlandu su odlučili: "Sigurno postoji lakši način." (Ustvari, revolucija kostura programa je možda počela na nekoliko različitih frontova, ali je sigurno da je lider firma Borland.) Očigledno je da je Windows programiranje veoma dobro prilagođeno objektno orijentisanom programiranju. Kreiranjem klasa koje enkapsuliraju osnovne Windows zadatke za programiranje, programer može biti veoma produktivan. Nakon kreiranja klase koja enkapsulira različite poslove prozora,

ista klasa se, primera radi može koristiti na više mesta, odnosno i u drugim programima. Revolucija programskih kostura je počela.

Ali, još uvek Vam nisam objasnio šta su programski kosturi.

NOVI TERMIN Kostur (*framework*) programa je zbirka klasa koja pojednostavljuje programiranje u okviru Windows-a, enkapsulirajući tehnike programiranja koje se često koriste. Kosturi se često nazivaju i biblioteke klasa (*class libraries*). *Enkapsuliranje* predstavlja preduzimanje kompleksnih programskih zadataka koji se na ovaj način pojednostavljuju obezbeđivanjem pojednostavljenog interfejsa.

Popularni programski kosturi sadrže klase koje enkapsuliraju prozore, kontrole za editovanje, okvire za listanje, grafičke operacije, bitmape, trake za pomeranje teksta, okvire za dijalog, itd.

Zašto onda treba da brinem o kosturima programa?

Ovo je dobro pitanje. Podloga svega je što kosturi programa čine Windows programiranje mnogo lakšim nego što bi to bilo moguće korišćenjem čistog jezika C, asemblera, odnosno originalnog Pascala (Pascala koji je nastao pre jezika Object Pascal). Daću Vam primer. Listing 5.1 sadrži deo Windows programa napisanih na jeziku C++. Ovaj deo koda učitava bitmapiranu datoteku sa diska i prikazuje je u središtu ekrana. U ovom trenutku Vam ništa neće biti jasno, ali budite strpljivi.

Listing 5.1: C++ kod koji učitava i prikazuje bitmapu

```
HPALETTE hPal;
BITMAPFILEHEADER bfh;
BITMAPINFOHEADER bih;
LPBITMAPINFO lpbi = 0;
HFILE hFile;
DWORD nClrUsed, nSize;
HDC hDC;
HBITMAP hBitmap;
void *bits;
do {
  if ((hFile = _lopen(data.FileName, OF_READ)) == HFILE_ERROR) break;
  if (_hread(hFile, &bfh, sizeof(bfh)) != sizeof(bfh)) break;
  if (bfh.bfType != 'BM') break;
  if (_hread(hFile, &bih, sizeof(bih)) != sizeof(bih)) break;
  nClrUsed =
    (bih.biClrUsed) ? bih.biClrUsed : 1 << bih.biBitCount;
  nSize =
    sizeof(BITMAPINFOHEADER) + nClrUsed * sizeof(RGBQUAD);
  lpbi = (LPBITMAPINFO) GlobalAllocPtr(GHND, nSize);
  if (!lpbi) break;
  MoveMemory(lpbi, &bih, sizeof(bih));
  nSize = nClrUsed * sizeof(RGBQUAD);
  if (_hread(hFile, &lpbi->bmiColors, nSize) != nSize) break;
```

```
  if (_llseek(hFile, bfh.bfOffBits, 0) == HFILE_ERROR) break;
  nSize = bfh.bfSize-bfh.bfOffBits;
  if ((bits = GlobalAllocPtr(GHND, nSize)) == NULL) break;
  if (_hread(hFile, bits, nSize) != nSize) break;
  hDC = GetDC(hWnd);
  hBitmap = CreateDIBitmap(hDC, &(lpbi->bmiHeader), CBM_INIT,
                           bits, lpbi, DIB_RGB_COLORS);
  if (hBitmap) {
    LPLOGPALETTE lppal;
    DWORD nsize = sizeof(LOGPALETTE)
      + (nClrUsed-1) * sizeof(PALETTEENTRY);
    lppal = (LPLOGPALETTE)  GlobalAllocPtr(GHND, nSize);
    if (lppal) {
      lppal->palVersion = 0x0300;
      lppal->palNumEntries = (WORD) nClrUsed;
      MoveMemory(lppal->palPalEntry, lpbi->bmiColors,
      nClrUsed * sizeof(PALETTEENTRY));
      hPal = CreatePalette(lppal);
      (void) GlobalFreePtr(lppal);
    }
  }
}  while(FALSE);
if (hFile != HFILE_ERROR) _lclose(hFile);
HPALETTE oldPal = SelectPalette(hDC, hPal, FALSE);
RealizePalette(hDC);
HDC hMemDC = CreateCompatibleDC(hDC);
HBITMAP oldBitmap =(HBITMAP)SelectObject(hMemDC, hBitmap);
BitBlt(hDC, 0, 0, (WORD)bih.biWidth, (WORD)bih.biHeight,
  hMemDC, 0, 0, SRCCOPY);
SelectObject(hMemDC, oldBitmap);
DeleteDC(hMemDC);
SelectPalette(hDC, oldPal, FALSE);
ReleaseDC(hWnd, hDC);
if (bits) (void) GlobalFreePtr(bits);
if (lpbi) (void) GlobalFreePtr(lpbi);
```

Ovo izgleda pomalo zastrašujuće, zar ne? Sada pogledajte ekvialent koji koristi Borlandov VCL:

```
Image.LoadFromFile('winnt.bmp');
```

Koji biste radije koristili? Čak ne morate ni da znate šta ovi delovi koda znače, da biste doneli odluku. Lako se da primetiti da je VCL verzija kraća (samo malo!) i čitljivija.

Ovi primeri sumiraju sve ono čemu služe kosturi programa. Kosturi kriju detalje za koje nije potrebno da znate. Sve što je sadržano u listingu 5.1 se izvršava u pozadini VCL koda (iako na Pascalu, a ne na C++). Nije potrebno da znate svaki detalj o tome šte se događa u pozadini kada VCL radi posao, a verovarno i ne želite da znate.

Sve što Vam je potrebno je preuzimanje objekta koji sačinjava kostur i njegovo postavljanje u Vaše programe.

Dobar kostur koristi sve prednosti objektno orjentisanog programiranja, a neki od njih to rade bolje nego drugi. Borlandova biblioteka Windows objekata - Object Windows Library (postoji i u C++, i u Pascal verziji) i biblioteka vizuelnih komponenti (VCL) su odlični primeri objektno orijentisanog programiranja. Ove biblioteke pružaju odgovarajuće apstrakcije koje su Vam potrebne da se izdignete iz zbrke i pređete na ozbiljno programiranje.

U čemu je štos?

Pomalo ste skeptični, zar ne? Dobro. Dovoljno ste pametni da shvatite koliko je sve ovo jednostavnije za korišćenje, da ne biste odustali. Istina je, da ste u pravu. Možda ćete pomisliti da aplikacija pisana korišćenjem kostura može biti veća i sporija od svoje kopije pisane na jeziku niskog nivoa. Ovo je delimično tačno. Aplikacije pisane korišćenjem kostura ne moraju biti sporije, naprotiv. U objektno orijentisanim jezicima postoje neka opšta nasleđa, ali je sigurno da se većina delova ne može uočiti u tipičnim Windows programima.

Osnovna mana Windows programa pisanih u Delphiju je što su Delphi programi veći u odnosu na programe pisane na jezicima kao što je C. Na primer, recimo da imate jednostavan Windows program pisan u jeziku C dužine 75 KB. Ekvivalent ovog programa pisan u Delphiju može imati dužinu i do 250 KB. Ovo može izgledati kao značajna razlika, ali primer prikazuje najgoru situaciju. Razlika u veličini između aplikacija pisanih na jeziku C i aplikacija pisanih na Delphiju korišćenjem kostura je veća kada se radi o malim programima. Kako Vaši programi postaju veći i sofisticiraniji, razlika u veličini je manje primetna.

Jedan od razloga za razliku u veličini programa leži u razlici između proceduralnih jezika i objektno orijentisanih jezika. Objektno orijentisani jezici (na primer, C++ i Object Pascal) nose dodatno opterećenje kao što je upravljanje izuzecima, informacije potrebne za rad programa, kao i druge OOP dodatke. Po mom mišljenju, razlika u veličini koda je prihvatljiva zbog mogućnosti koje pruža Object Pascal.

Pre nego što me proglasite za zagovornika rasipača koda, dozvolite mi da kažem da sam zabrinut, kao i svaka druga osoba, ukoliko dođe do rasipanja koda. Verujem da svi mi pišemo najkraći kod koji nam dozvoljava alat koji koristimo. Takođe sam realan i shvatam da je vreme izbacivanja programa na tržište pokretačka sila u softverskoj industriji danas. Voljan sam da menjam nešto duži kod za snagu koju mi nude Object Pascal i VCL. Objasniću na drugi način. Nisam zainteresovan da provedem mesece pišući Windows program koji se nakon prevođenja svodi na 100 KB izvršnog koda, kada sve to mogu postići koristeći Delphi dva dana i završiti sa 400 KB izvršnog koda. Dužina rezultujućeg izvršnog programa je beznačajna u odnosu na vreme razvoja koje se time štedi.

Kosturi programa uče objektno orijentisano programiranje i dizajniranje

Ukoliko ste prestali da ozbiljno shvatate ovu ludu igru zvanu Windows programiranje, eventualno ćete prestati da zavirujete u izvorni kod Vašeg omiljenog programskog kostura. Pre, ili kasnije, ćete saznati kako profesionalci rade stvari. VCL izvorni kod je pravo mesto da dobijete ovu vrstu informacija.

Za vikend, kada pokupite svo lišće, kada sredite i ofarbate kuću, kada operete veš, kada su deca kod babe, a Vi mislite da dobro poznajete Delphi, provedite neko vreme pregledajući VCL izvorni kod. (Paketi Delphi Professional i Client/Server se isporučuju sa VCL izvornim kodom.) U početku Vam može izgledati zastrašujuće, ali nakon određenog vremena ćete videti šta su dizajneri uradili. Nemojte se plašiti. Pokušaj shvatanja stvari koje su van granica Vašeg znanja uključuje Object Pascal. Ostavite komplikovanije stvari za kasnije.

Uočite kako VCL dizajneri koriste privatne, zaštićene i javne pristupe u okviru klasa. Uočite kako stvari koje treba sakriti od korisnika nisu dostupne javnom pogledu. Proučavanje VCL izvornog koda Vas može puno naučiti o jeziku Object Pascal i objektno orijentisanom dizajnu.

Biblioteka vizuelnih komponenti (VCL)

Verovatno ste primetili da ova knjiga u svom naslovu sadrži i naziv "Delphi 4". Očigledno je da je Delphi prisutan već neko vreme. Kada je 1995. bio predstavljen Delphi 1, trenutno je postao hit. Delphi pruža brz razvoj aplikacija (RAD - Rapid Application Development) korišćenjem stvari koje se zovu *komponente*. Komponente su objekti koji se mogu postaviti na formu i kojima se može manipulisati preko karakteristika, metoda i događaja. To je vizuelno programiranje, ukoliko želite da ga tako nazovem.

Koncept programiranja baziranog na formama je prvi uveo Microsoft Visual Basic. Za razliku od Visual Basic-a, Delphi koristi derivat Pascala kao osnovni jezik za programiranje. Ovaj novi jezik, nazvan Object Pascal uvodi OOP u Pascal. Delphi i Object Pascal predstavljaju brak između objektno orijentisanog programiranja i programiranja baziranog na formama. Kao dodatak, Delphi može da proizvede samostalan izvršni kod - prave programe. Programi ne zahtevaju DLL datoteke za rad; programi su prevedeni, a ne interpretirani; programi se izvršavaju desetine puta brže od programa pisanih na Visual Basic-u. Svet programera je impresioniran.

Delphi nije samo izbacio Object Pascal i pustio Vas da se koprcate. Takođe je predstavio biblioteku vizuelnih komponenti. Kao što sam napomenuo VCL je kostur aplikacije za Windows programiranje u jeziku Object Pascal. Najzapaženija mogućnost VCL-a je činjenica da je VCL dizajniran korišćenjem koncepta karakterisika, metoda i događaja - modela vizuelnih komponenti. Pogledajmo model vizuelnih komponenti detaljnije.

Komponente

Kao što je bilo opisano u lekciji dana 1 "Početak sa Delphijem", VCL komponente su objekti koji izvršavaju određeni zadatak vezan za programiranje. VCL komponente se nalaze u okviru Object Pascal klasa. Počev od ovog dela knjige, upoznavaćete VCL komponente u svakoj lekciji. Neću trošiti puno vremena za objašnjavanje svakog detalja o komponentama u ovom trenutku, pošto ćete preko primera videti kako komponente rade, do kraja knjige. Detaljnije ću objasniti komponente u lekciji dana 7 "VCL komponente".

Karakteristike, metode i događaji

U lekciji dana 1, ukratko sam predstavio karakteristike, metode i modele događaja. Ova tri elementa čine javni interfejs komponenti u okviru VCL-a (deo komponenti koje korisnik vidi). Pogledajmo ove elemente još jednom.

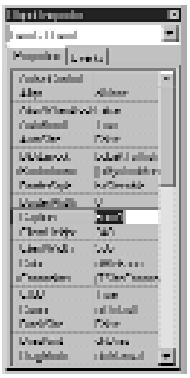
Karakteristike (properties)

Karakteristike *(properties)* su elementi komponenti koji kontrolišu način rada komponente. Većina komponenata ima zajedničke karakteristike. Na primer, sve vizuelne komponemte imaju karakteristike `Top` (gore) i `Left` (levo). Ove dve karakteristike kontrolišu gde će komponenta biti postavljena u toku dizajniranja, odnosno u toku rada programa. Sve komponente imaju karakteristiku `Owner` (vlasnik), koju VCL koristi za praćenje komponenti potomaka za određenu formu pretka, odnosno komponentu koja je poseduje.

Karakteristike i Object Inspector Slika uvek vredi hiljadu reči, stoga pokrenite ponovo Delphi da biste videli karakteristike na delu. Kada pokrenete Delphi, pozdraviće Vas prazna forma i Object Inspector.

Ukoliko ste Delphi opcije konfigurisali tako da snimite radnu površinu nakon zatvaranja Delphija, možda će se pojaviti poslednji projekt na kojem ste radili, sledeći put kada budete startovali Delphi. U tom slučaju odaberite opciju File➥New Application u okviru glavnog menija da biste dobili praznu formu.

Prozor Object Inspector će izgledati približno kao na slici 5.1. (Prilikom startovanja Delphija , veličina prozora Object Inspector zavisi od trenutne rezolucije ekrana, pa Vaš prozor Object Inspector može biti duži, ili kraći u odnosu na Object Inspector prikazan na slici 5.1.) Ukoliko je potrebno, kliknite na jezičak Properties u okviru prozora Object Inspector, kako bi bile prikazane karakteristike forme. Karakteristike komponenti su poređane po abecednom redu.



Slika 5.1
Prozor Object Inspector

Ukoliko postoji više karakteristika koje istovremeno mogu biti prikazane, Object Inspector prikazuje traku za pomeranje, tako da možete videti dodatne karakteristike. Prozor Object Inspector možete pomerati, odnosno možete menjati njegovu veličinu. Moj Object Inspector ima onoliku dužinu koliko mi ekran dozvoljava, kako bih mogao da vidim maksimalan broj karakteristika istovremeno. Pomerajte kursor duž karakteristika sve dok ne naiđete na karakteristiku `Left`, a zatim kliknite na nju. Promenite vrednost karakteristike `Left` (bilo koji broj između 0 i 600 je dobar), zatim pritisnite taster Enter na tastaturi. Uočite kako se forma pomera, dok menjate vrednosti.

Ovo ilustruje važan aspekt karakteristika: karakteristike su više nego jednostavna polja klasa. Svaka karakteristika ima priključeno odgovarajuće polje podataka, ali karakteristika sama za sebe nije polje podataka klase. Promena karakteristike obično dovodi do izvršavanja koda u pozadini.

NOVI TERMIN Karakteristike su obično povezane sa *metodama pristupa* koje se izvršavaju kada se menja karakteristika.

Promena vrednosti karakteristike. Karakteristike se mogu menjati u *toku dizajniranja* (kada dizajnirate Vašu formu) i u toku rada programa (kada program izvršava kod koji ste napisali). U oba slučaja, ukoliko karakteristika ima metod pristupa, ovaj metod pristupa će biti pozvan i izvršen prilikom izmene karakteristike. Primer izmene karakteristike u toku dizajniranja ste mogli da vidite kada ste menjali karakteristiku `Left` i posmatrali pomeranje forme na ekranu.

Ovo je jedna od jakih strana VCL-a, i predstavlja način kako se VCL koristi u Delphiju. Trenutno možete videti na ekranu rezultat izmene dizajna. Ne prikazuju sve karakteristike vidljive promene na formi u toku dizajniranja, pa se izmene ne mogu dogoditi kod svakog slučaja. Ipak, kada je to moguće rezultat nove vrednosti karakteristike se trenutno prikazuje na formi.

Da biste menjali karakteristiku u toku rada programa, jednostavno dodelite vrednost karakteristici. Kada dodeljujete vrednost, VCL radi u pozadini i poziva metode za

pristup date karakteristike. Da biste promenili karakteristiku `Left` u toku rada, koristite kod:

```
Left := 200;
```

U slučaju karakteristike `Left` (kao i kod karakteristike `Top`), VCL pomera i ponovo iscrtava formu. (Za Vas, Windows API programere, koji možete da shvatite šta se u ovom slučaju dešava, navodim da promena karakteristike koja definiše poziciju na ekranu, poziva Windows API funkcije `SetWindowPos` i `InvalidateRect`.)

Specifikatori pristupa karakteristici (property *access specifiers*). Karakteristike imaju dva specifikatora pristupa, koji se mogu koristiti prilikom čitanja, odnosno prisupa karakteristici. Postoji specifikator za čitanje (*read specifier*) i specifikator za pisanje (*write specifier*).

Suvišno je reći da specifikatori pristupa pridružuju metode za čitanje i pisanje karakteristikama. Kada se karakteristika čita, odnosno zapisuje, metode pridružene karakteristici se pozivaju automatski. Kada dodelite vrednost, kao u prethodnom slučaju, pristupate specifikatoru za pisanje. Efektivno, VCL proverava da li postoji metod pristupa za specifikator pisanja. Ukoliko postoji, poziva se metod pristupa. Ukoliko metod pristupa ne postoji, VCL dodeljuje novu vrednost polju podataka koja je pridružena karakteristici.

Kada karakteristiku uzimate kao referencu (koristite karakteristiku sa desne strane znaka jednakosti), pristupate specifikatoru čitanja:

```
X := Left;
```

U ovom slučaju, VCL poziva specifikator čitanja da bi očitao vrednost karaktertistike `Left`. U većini slučajeva specifikator čitanja radi tek nešto više od samog vraćanja trenutne vrednosti karakteristike.

Atributi karakterisike Karakteristike karakteristika (izvinite, ali ne mogu da odolim) su definisane preko zapisivača komponente. Karakteristika može biti definisana tako da se može samo čitati. Karakteristika samo za čitanje (read-only) se može samo čitati - njena vrednost se može pregledati - ali se ne može zapisati. Drugim rečima, možete preuzeti vrednost karakteristike, ali je ne možete menjati. U ređim slučajevima, karakteristika može biti definisana kao karakteristika koja se može samo zapisati (karakteristika se može samo zapisati, ali se ne može čitati, što u većini slučajeva nije korisno). Očigledno da je ovo suprotno od karakteristike koja se može samo čitati.

Na kraju, neke karakteristike mogu biti definisane da budu vidljive samo u toku rada programa (runtime-only). Pošto se karakteristike koje su vidljive samo u toku rada programa ne pojavljuju u toku dizajniranja, Object Inspector ih ne prikazuje. Karakteristike vidljive samo u toku rada, mogu istovremeno biti deklarisane i kao karakteristike koje se mogu samo čitati, što znači da im je moguće pristupiti samo u toku rada programa i da se mogu samo čitati (ne i zapisivati).

Tipovi karakteristika Neke karakteristike koriste slučajeve VCL klasa kao podlogu za polja podataka. Da bi ovo ilustrovali, postavimo memo komponentu na praznu formu:

1. Pređite na paletu Delphijevih komponenti, odaberite jezičak Standard, a zatim kliknite na dugme Memo. (Oblačić za savet će se pojaviti kada kursorom pređete preko dugmeta Memo.)
2. Sada se prebacite na formu i kliknite na mesto gde želite da postavite gornji levi ugao memo polja. Ubrzo nakon što postavite memo komponentu na formu, Object Inspector će se prebaciti na pokazivanje karakteristika komponente koju ste upravo postavili na formu, što je u ovom slučaju komponenta `TMemo`.
3. Pronađite karakteristiku `Lines` i kliknite na nju. Uočite da vrednost karakteristike sadrži tekst `(TStrings)`, a takođe postoji i malo dugme sa tri tačke na desnoj strani vrednosti karakteristike.

Dugme sa tri tačke saopštava da tekuća karakteristika može biti editovana korišćenjem editora karakteristike. Za niz stringova, na primer, okvir za dijalog će biti prikazan, pa možete upisati string. U slučaju karakteristike `Font`, klik mišem na dugme sa tri tačke poziva okvir za dijalog Choose Font. Tačan tip editora karakteristike zavisi od same karakteristike, iako određene karakteristike mogu deliti zajedničke editore. Editor karaktertistika možete pozvati klikom na dugme sa tri tačke, odnosno dvostrukim klikom na vrednost karakteristke.

Karakteristika `Lines` memo komponente je slučaj klase `TStrings`. Kada dva puta kliknete mišem na kolonu Value, biće prikazan string editor u koji zatim možete upisati string, koji će biti prikazan u memo komponenti u toku rada aplikacije. Ukoliko ne želite da se prikaže bilo koji string u memo komponenti, potrebno je da obrišete podatke u editoru karakteristika.

Karakteristika `Font` je još jedan primer karakteristike koja je slučaj klase VCL. *Font* uključuje stvari poput: tipa, boje, veličine fonta, itd. Pronađite karakteristike `Font` u okviru prozora Object Inspector. (Nije važno da li ste odabrali memo komponentu u okviru forme.)

Uočite da pored reči *Font* postoji znak plus. Ovo označava da postoje posebne karakteristike koje možete editovati u okviru date karakteristike. Ukoliko kliknete dva puta mišem na naziv karakteristike, uočićete da prozor Object Inspector širi karakteristiku i otkriva njene posebne elemente. Sada možete zasebno editovati svaki element karakteristike `Font`. U slučaju karakteristike `Font`, karakteristike karakteristika mogu biti editovane pozivanjem editora karakteristika. Možete koristiti bilo koju metodu koja daje iste rezultate.

Neke karakteristike su skupovi. (Skupovi su bili obrađeni u lekciji dana 3, "Klase i objektno orijentisano programiranje".) Karakteristika `Style` u okviru objekta `Font` je dobar primer skupa. Uočite da karakteristika `Style` ima znak plus na početku. Ukoliko dva puta kliknete mišem na karakteristiku `Style`, uočićete da se nod

`Style` proširuje kako bi otkrio sadržaj skupa. U ovom slučaju skup se sastoji od različitih stilova koji su karakteristični za fontove: podebljano (bold), kurziv (italic), podvučeno (underline) i precrtano (stikeout). Dvostrukim klikom miša na stil, možete uključiti, odnosno isključiti stil. Skup može biti prazan, odnosno može sadržati jednu, ili više dozvoljenih vrednosti.

Neke karakterisrtike mogu biti specifikacije (*enumerations*) - liste mogućih opcija.

Specifikacija je lista mogućih opcija za određenu karakteristiku. Kada kliknete mišem na karakteristiku sa specifikacijom, pojaviće se dugme sa strelicom na dole na desnoj strani vrednosti karakteristike. Da biste videli opcije specifikacije, kliknite na dugme sa strelicom na dole, kako bi lista opcija bila prikazana. Alternativno, možete dva puta kliknuti mišem na kolonu Value. Kada dva puta kliknete mišem na vrednost karakteristike, Object Inspector će se kretati kroz opcije. Karakteristika `Cursor` daje dobar primer karakteristike sa specifikacijom. Pronađite karakteristiku kursor i kliknite mišem na dugme sa strelicom da bi se pojavila lista potencijalnih kursora koju možete odabrati.

Specifikacije i skupovi se razlikuju po tome što se kod karakteristike sa specifikacijom mogu odabrati samo opcije koje postoje (samo jedan kursor može biti aktivan). Skup može sadržati više vrednosti, odnosno ne mora sadržati ni jednu vrednost. (Stil fonta može sadržati vrednosti: bold, underline, italic, odnosno može da bude bez vrednosti.) Sve dok Delphi radi, a prikazana je prazna forma, možete provesti neko vreme ispitujući različite komponente i njihove karakteristike. Samo napred, čekaću Vas.

Pravilo kuće: karakteristike

- Karakteristike mogu biti polja klasa i može im se pristupiti kao klasnim poljima.
- Karakteristike nisu klasna polja. One su posebna kategorija klasnih elemenata.
- Karakteristike obično pozivaju metode za pristup kada se upisuju u avedene metode (dodeljuju se vrednosti), ali ne uvek. Zavisi od načina na koji je određena komponenta zapisana.
- Objavljene karakteristike obično imaju generičke vrednosti. Generička vrednost je ona vrednost koja se inicijalno pojavljuje u prozoru Object Inspector kada se komponenta prvi put aktivira, a to je vrednost koja ćese koristiti ukoliko se karakteristici ne dodeli neka druga vrednost.
- Karakteristike mogu biti dizajnirane kao: karakteristike za čitanje/pisanje, karakteristike koje se mogu samo čitati, odnosno karakteristike koje se mogu samo pisati.
- Karakteristike vidljive samo u toku rada programa se ne pojavljuju u prozoru Object Inspector i mogu biti menjane samo u toku rada programa.

- Karakteristike mogu uključiti:
- jednostavne tipove podataka
- stringove
- nizove
- skupove
- specifikacije
- objekte VCL klasa

Metode

Metode su bile obrađene u lekciji dana 3, "Klase i objektno orijentisano programiranje", stoga ih neću posebno objašnjavati u ovom poglavlju. *Metode* u VCL komponentama su funkcije i procedure koje mogu biti pozvane da bi komponenta izvršila određenu aktivnost. Na primer, sve vizuelne komponente imaju metodu pod nazivom `Show` koja prikazuje komponentu, a takođe i metodu pod nazivom `Hide`, koja sakriva komponentu. Na primer:

```
MyWindow.Show;
{ do some stuff, then later... }
MyWindow.Hide;
```

Metode u VCL-u mogu biti deklarisane kao javne, zaštićene, odnosno privatne. Javnim metodama korisnici mogu pristupiti preko komponenti. U ovom primeru i `Show` i `Hide` metode su javne. Zaštićenim metodama se ne može pristupiti preko komponenti koje ih koriste, ali se može pristupiti preko klasa (komponenti), izdvojenih iz komponente. Naravno, privatnim metodama se može pristupiti samo iz njihovih klasa.

Neke metode uzimaju parametre i vraćaju vrednosti, druge ne. U zavisnosti od toga kako je metoda napisana korišćenjem zapisivača komponenti, zavisi da li će metoda imati, odnosno da li neće imati vrednost koju vraća. Na primer, metoda `GetTextBuf` menja tekst u okviru komponente `TEdit`. Ova metoda se može koristiti za preuzimanje teksta iz kontrole za editovanje, kao što se može videti na primeru:

```
var
  Buff     : array [0..255] of Char;
  NumChars : Integer;
begin
  NumChars := EditControl.GetTextBuf(Buff, SizeOf(Buff));
end;
```

Kao što možete primetiti, ova metoda preuzima dva parametra i vraća celobrojnu vrednost. Kada se metoda pozove, sadržaj kontole za editovanje se upisuje u promenljivu `Buff`, a vrednost koja se vraća predstavlja broj karaktera koji se menja u okviru kontrole Edit.

Za sada, ovo je sve što Vam je potrebno da biste mogli da koristite metode. Metode će detaljnije biti obrađene u lekciji dana 20, "Kreiranje komponenti".

Pravila kuće: Metode

- Metode mogu biti privatne, zaštićene, odnosno javne.
- Metode se pozivaju korišćenjem operatora tačka.
- Metode mogu prihvatati parametre i mogu vraćati vrednosti
- Samo javne metode mogu pozvati komponente koje ih koriste.

Događaji

NOVI TERMIN Za Windows-e je rečeno da je to operativni sistem kojim se *upravlja događajima*. Sistem kojim se upravlja događajima, znači da programom upravljaju događaji koji se javljaju u okruženju Windows-a. Događaji uključuju pomeranje miša, klik na miša i pritiske na tastere tastature.

Programeri koji prelaze sa DOS-a, odnosno sa mainframe programskih okruženja, će možda imati teškoća sa konceptom programa kojima upravljaju događaji. Windows program neprestano uvlači Windows u događaje. Događaji u Windows-ima uključuju aktiviranje menija, klik na dugme, pomeranje prozora, osvežavanje izgleda prozora, aktiviranje prozora, itd.

Windows obaveštava program o nastanku događaja šaljući mu Windows poruku. Postoji više od 200 mogućih poruka koje Windows može poslati aplikaciji. To je veoma mnogo poruka. Srećom, ne morate da poznajete svaku od njih da bi programirali na Delphiju; postoji samo nekoliko desetina poruka koje se često koriste.

VCL događaji U VCL-u događaj predstavlja bilo šta što se sa komponentom može dogoditi, a da korisnik treba da bude obavešten o tome. Svaka komponenta je dizajnirana da odgovori na određene događaje. To obično predstavlja Windows događaj, ali može da znači isto tako i neke druge stvari. Na primer, komponenta dugme je dizajnirana da odgovori na klik mišem, kao što možete očekivati. U slučaju nevizuelne kontole, kao što je komponenta baze podataka, moguće je odgovoriti na događaje koji ne pripadaju Windows događajima, kao što je situacija kada korisnik stigne do kraja tabele.

NOVI TERMIN **Upravljanje događajima (handling events)** Kada odgovorite na događaj komponente, može se reći da *upravljate* događajem.

Događajima se upravlja preko metoda koji se zovu upravljači događajima (event handlers). U toku prva tri dana u lekcijama knjige ste često koristili upravljače događajima.

Tipični Windows program većinu vremena provodi besposlen, čekajući da se pojavi neki događaj. VCL čini upravljanje događajima neverovatno jednostavnim. Događaji za koje je komponenta dizajnirana su prikazani na kartici Events prozora Object Inspector. Nazivi događaja opisuju događaj na koji komponenta odgovara. Na primer, događaj koji upravlja klikom na taster miša se naziva `OnClick`.

Nije potrebno da upravljate svakim događajem koji komponenta definiše. U suštini, to retko radite. Ukoliko ne odgovorite na određeni događaj, poruka događaja će biti ili povučena, ili će se njome upravljati na generički način, kako je opisano u VCL-u, odnosno samoj komponenti. Možete upravljati samo onim događajima koji Vas interesuju, a ostale ignorisati.

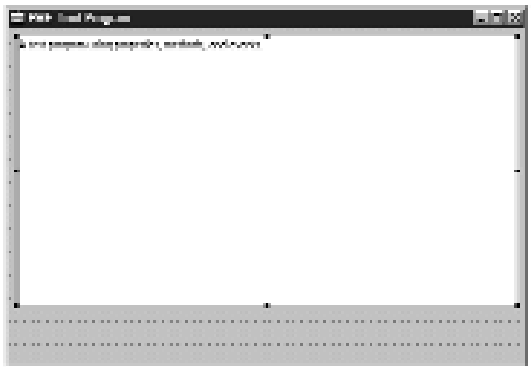
Ovi koncepti će imati više smisla ukoliko ih isprobamo u praksi. Za početak, otvorite novu aplikaciju. Odaberite opciju File➥New Application u okviru glavnog menija. Ukoliko Vas upitaju da li želite da snimite tekući projekt, kliknite mišem na dugme No. Ponovo ćete dobiti praznu formu. Prvo, podesite glavnu formu:

1. Promenite karakteristiku `Name` u `PMEForm` (PME za properties, methods, events - karakteristike, metode, događaji).
2. Promenite karakteristiku `Caption` u `PME Test Program`.

Zatim treba da dodate memo komponentu na formu:

1. Odaberite jezičak Standard u okviru palete komponenti, a zatim kliknite mišem na dugme Memo.
2. Kliknite mišen na formu i postavite memo komponentu.
3. Promenite karakteristiku `Name` u `Memo`. Uverite se da je memo komponenta odabrana, tako da se ne dogodi da greškom promenite naziv forme umesto naziva memo komponenti.
4. Dva puta kliknite na karakteristiku `Lines` u koloni Value. Biće prikazan editor string liste.
5. Obrišite reč `Memo`, pa ukucajte A `test program using properties, methods and events`. (Test program koji koristi karakteristike, metode i događaje.) Kliknite mišem na dugme OK, da biste zatvorili editor string liste.
6. Promenite veličinu memo komponente, tako da zauzme veći deo forme. Ostavite mesta za dugme u dnu forme.

Vaša forma bi trebalo da liči na formu sa slike 5.2.



Slika 5.2
Forma sa dodatom memo komponentom

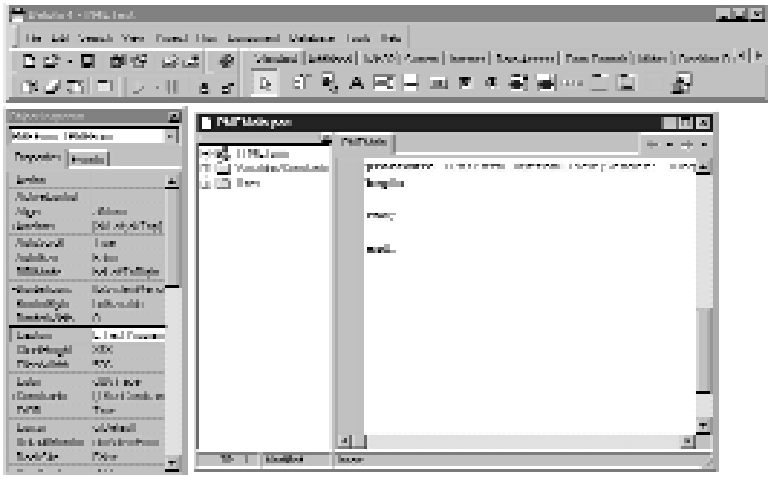
Sada možete dodati dugme na formu:

1. Odaberite jezičak Standard na paleti komponenti, a zatim kliknite na komponeti Button.
2. Kliknite na formu ispod memo komponente, da biste postavili dugme na formu.
3. Promenite karakteristriku `Name` za dugme u `Button`.
4. Promenite karakteristiku `Caption` u `Show/Hide`.
5. Centrirajte dugme po horizontali u okviru forme.

> **SAVET** Komponente možete centrirati vizuelno, ali egzaktniji metod koji možete koristiti je paleta za poravnavanje (Alignment palette). Odaberite opciju ViewÐAlignment Palette u okviru glavnog menija, a zatim kliknite na dugme Center horizontally u okviru prozora palete za poravnavanje da biste komponentu horizontalno centrirali u formi.

Ovo dugme ćete koristiti da sakrijete, odnosno prikažete memo komponentu. Potrebno je da napišete nešto koda kako bi dugme radilo kada se na njega klikne mišem. Uverite se da ste odabrali komponentu dugme, a zatim kliknite na jezičak Events u okviru prozora Object Inspector.

Biće prikazana lista događaja za koje je komponenta dugme predviđena. Prvi događaj bi trebao da bude događaj `OnClick`. Dva puta kliknite mišem na kolonu Value događaja `OnClick`. Ono što će se desiti, predstavlja veliku stvar u vizuelnom programiranju. Editor koda se aktivira i prikazuje procerduru `OnClick` koja je spremna da prihvati unos Vašeg koda. Slika 5.3 prikazuje editor koda u kom se nalazi upravljač događajem `OnClick`.



Slika 5.3
Delphijev editor koda sa prikazanim upravljačem događaja OnClick

NAPOMENA Vaš editor koda možda neće izgledati isto kao na slici 5.3. Iz kod editora sam uklonio Module Explorer da bi video više koda. Jedna od odličnih stvari u Delphijevom okruženju je činjenica da je okruženje potpuno prilagodljivo. Ukoliko Vam se ne sviđa generički izgled uvek ga možete izmeniti.

Dodavanje koda za upravljanje događajima Uočite da je upravljač događajem već postavljen i da sve što treba da uradite je upisivanje koda. Ukoliko dobro pogledate upravljač događaja, videćete da je to, ustvari procedura pod nazivom `ButtonClick`, da je element klase `TPMEForm` i da preuzima pointer od `TObject` pod nazivom `Sender` kao parametar. (Malo pre sam obradio parametar `Sender`.) Sve što je ostalo da se uradi je kucanje koda koji pokazuje, odnosno sakriva komponentu memo svaki put kada se klikne na dugme. Deo koda sam pozajmio iz prethodnog poglavlja u kome su obrađene metode. Izmenite metodu `ButtonClick` tako da dobijete:

```
procedure TPMEForm.ButtonClick(Sender: TObject);
const
  IsVisible : Boolean = False;
begin
  IsVisible := not IsVisible;
  if IsVisible then
    Memo.Hide
  else
    Memo.Show;
end;
```

Ovaj kod prvo deklariše konstantu sa tipom pod nazivom `IsVisible`.

NOVI TERMIN Konstanta sa tipom (*typed constant*), kada se koristi na ovaj način, predstavlja promenljivu koja zadržava svoju vrednost između dva poziva metoda.

Prva linija koda u okviru metoda menja vrednost `Boolean` promenljive na `True`, odnosno `False`, primenjujući logičko `NOT` (ne) na trenutnu vrednost promenljive. Evo kako radi:

1. Inicijalno statička promenljiva je definisana kao `False`. Kada se prvi put pozove upravljač događajem, promenljivoj se dodeljuje vrednost `NOT False`, što je, ustvari, `True`.
2. Kada se sledeći put pozove upravljač događaja, promenljivoj se dodeljuje `NOT True`, itd.
3. Svaki put kada se pozove metod, konstanta `IsVisible` sadrži vrednost koja je suprotna u odnosu na vrednost iz prethodnog poziva metode. Nakon toga, komanda `if/else` poziva ili `Show`, ili `Hide`, u zavisnosti od vrednosti konstante `IsVisible`.

To bi bilo sve, ali da li to i radi? Hajde da otkrijemo. Kliknite na dugme Run u okviru trake sa alatima. Nakon prevođenja, program se startuje i prikazuje. Ovo je trenutak istine. Kliknite na dugme, a memo komponenta će nestati. Kliknite na dugme ponovo, pa će se memo komponenta ponovo pojaviti. Radi!

Nakon što ste se na trenutak igrali, zatvorite program (koristite dugme za zatvaranje programa u gornjem desnom uglu trake za naslov) i vratili ste se u editor koda.

Pre nego što nastavite, snimite projekt. Odaberite opciju File➥Save All u okviru glavnog menija. Prva stvar za koju ćete biti upitani je naziv junita (izvorne datoteke). Upišite `PMEMain`, a zatim kliknite na dugme OK. Zatim ćete biti upitani za naziv datoteke projekta. Upišite `PMETest`, a zatim pritisnite taster Enter, ili kliknite na dugme OK.

Korišćenje karakteristike `Visible` Kompletan rad sa `Bulovim` promenljivama može biti pomalo dosadan. Prisetite se dela knjige kada smo obradili karakterisr tike. Zar ne bi bilo dobro kada bi memo komponenta imala karakteristiku koja bi nam saopštavala da li je komponenta trenutno vidljiva? Zar postoji takva stvar? Naravno da postoji. Ona se naziva, kao što možete pretpostaviti `Visible` (vidljivo). Hajde da je iskoristimo. Ponovo editujte upravljač događaja i izmenite ga da izgleda ovako:

```
procedure TPMEForm.ButtonClick(Sender: TObject);
begin
  if Memo.Visible then
    Memo.Hide
  else
    Memo.Show;
end;
```

Ponovo kliknite na dugme `Run`. Program je startovan i, napred - nazad, dugme radi upravo ono za šta je predviđeno. Kako to? Koristili ste karakteristike, metode i događaje u istom primeru.

Da li ste "odlepili"? Sačekajte, pošto će se još mnogo toga dogoditi. Pa, skinite tu ružnu zelenu boju sa Vašeg lica... Vaš šef će misliti da radite!

Parametar `Sender` Kao što ste mogli da vidite, metoda `ButtonClick` uzima pointer na `TObject` pod nazivom `Sender` (znali Vi to, ili ne, sve klasne promenljive u Delphiju su pointeri). Svaki upravljač događajem će imati najmanje jedan parametar `Sender`. U zavisnosti od događaja kojim se upravlja, upravljač događajem može imati jedan, odnosno više parametara. Na primer, upravljač događajem `OnMouseDown` izgleda ovako:

```
procedure TPMEForm.FormMouseDown(Sender: TObject; Button:
TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
end;
```

Evo Vas kako prikupljate informacije o dugmetu koje je pritisnuto, koji taster tastature je pritisnut u trenutku kada je korisnik kliknuo na taster miša i x, y koordinate kursora u trenutku kada je korisnik kliknuo na taster miša. Upravljač događajem sadrži sve informacije koje su Vam potrebne da radite sa određenim događajem za koji je upravljač događajem napravljen.

Šta je ustvari `Sender`? Sender je pointer na komponentu koja šalje poruku upravljaču poruka. U ovom primeru, parametar `Sender` je dodatni teret pošto znate da je dugme `Show/Hide` pošiljalac (`sender`). `Sender` postoji kako bi Vam omogućio da postavite više komponenti koje mogu koristiti isti upravljač događajem.

Da bi ovo ilustrovali, kreirajmo novo dugme, i označimo jedno dugme sa Show (pokaži), a drugo sa Hide (sakrij):

1. Ukoliko je editor koda na vrhu, pritisnite funkcijski taster F12, da biste se vratili na editor forme.
2. Kliknite na dugme Show/Hide, da biste ga odabrali. Promenite i `Name` i `Caption` karakteristiku u `Show`.
3. Dodajte novo dugme na formu sa desne strane dugmeta Show. Uredite dugmad ukoliko želite da date pristojan izgled formi.
4. Promenite karakteristiku `Name` dugmeta u `Hide`. Karakteristika `Caption` će se takođe promeniti u `Hide` (treba da pritisnete taster Enter pre nego što se `Caption` karakteristika promeni).
5. Kliknite na dugme Show, a zatim kliknite na jezičak Events u okviru prozora Object Inspector. Uočite da događaj `OnClick` sada ima naziv `ShowClick`. Izmenite ga da ponovo pokazuje `ButtonClick`. (Inicijalni naziv upravljača događajem je i generički naziv. Možete ga promeniti u naziv koji želite.)
6. Kliknite na Hide dugme i pronađite događaj `OnClick` u prozoru Object Inspector (već je odabran). Pored vrednosti se nalazi dugme sa strelicom na

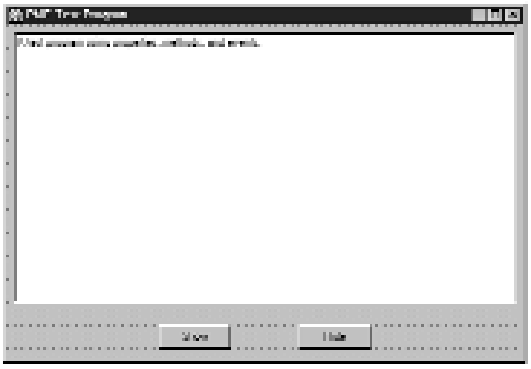
dole. Kliknite mišem na dugme sa strelicom i odaberite `ButtonClick` sa liste (na listi postoji samo jedan element).

7. Dva puta kliknite mišem na vrednost `ButtonClick`. Biće Vam prikazan editor koda sa kursorom koji se nalazi na metodi `ButtonClick`. Izmenite kod kao na sledećem listingu:

```
procedure TPMEForm.ButtonClick(Sender: TObject)

begin
  if Sender = Hide
    then Memo.Hide
  else
    Memo.Show;
end;
```

8. Pecite na 250 stepeni jedan sat dok ne dobije zlatno braon boju. (Samo proveravam.)

Vaša forma bi trebalo da izgleda približno kao forma na slici 5.4. Prevedite i startujte program. Kliknite na svaki taster da biste se uverili da funkcionišu onako kako je najavljeno.



Slika 5.4
Forma sa svim komponentama na broju

Ono što ste upravo uradili je kreiranje jednog metoda za upravljanje događajem koji upravlja događajem `OnClick` koristeći oba dugmeta. Koristili ste parametar `Sender` da odredite koje dugme šalje događaj `OnClick`, a zatim ste sakrili, odnosno prikazali memo komponentu u zavisnosti od potrebe. Mogli ste da kreirate odvojene upravljače `OnClick` za svako dugme, ali sa ovim metodom kod je kompaktniji. Pored toga, ovo je bila dobra ilustracija korišćenja pointera `Sender`.

Uočite da sam upoređivao promenljivu `Sender` sa karakteristikom `Name` komponente. Pošto su obe pointeri, poređenje se vrši kako bi se utvrdilo da li obe promenljive sadrže istu adresu.

Korak 6 u prethodnoj vežbi ilustruje važan detalj: nakon što ste kreirali upravljač događajem `OnClick` za određenu komponentu, možete prikačiti na isti upravljač događajem događaj `OnClick` bilo koje komponente u okviru forme. Na ovaj način imate mogućnost da koristite isti upravljač događajem za više komponenti. Događaji će biti detaljnije obrađeni kako napredujete čitajući knjigu.

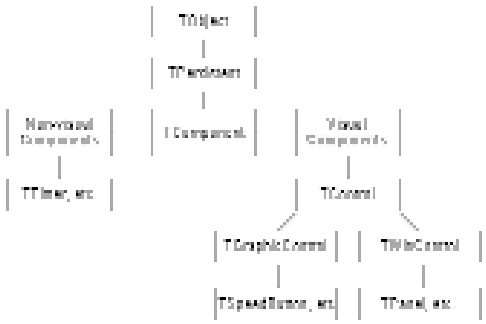
Pravila kuće: Događaji

- Možete odgovoriti na bilo koji događaj komponente ukoliko je to potrebno.
- Od Vas se ne zahteva da odgovorite na sve događaje koje komponenta definiše.
- Događajima se upravlja preko metoda za upravljanje događajima koji se zovu upravljači događajima (event handlers).
- Više komponeti može deliti zajednički upravljač događajem.
- Nazivi upravljača događajima koje generiše Delphi su generički nazivi i programeri ih mogu menjati.
- Budite sigurni da nazive upravljača događajima menjate samo u prozoru Object Inspector.
- Parametar Sender upravljača događajem možete koristiti da bi utvrdili koja komponenta generiše događaj.
- Dvostrukim klikom miša na naziv upravljača događajem u okviru prozora Object Inspector biće prikazan editor koda u koji možete upisati deo koda za upravljanje događajem.
- Svaki upravljač događajem sadrži parametre koji su porebni za ispravno upravljanje događajem.

Otkrivanje VCL-a

Biblioteka vizuelnih komponenti (VCL) je dobro dizajniran kostur. Kao što je to slučaj kod većine dobrih kostura, VCL do maksimuma koristi nasleđivanje. Veliki deo VCL kostura je sastavljen od klasa koje predstavljaju komponente. Ostale VCL klase nisu u vezi sa komponentama. Ove klase izvršavaju zadatke spremanja, ponašaju se kao pomoćne klase i pružaju neke pomoćne servise.

Hijerarhija VCL klasa koja se bavi komponentama je kompleksna. Srećom, ne treba da znate detaljno strukturu VCL-a, da bi počeli programiranje na Delphiju. Na vrhu VCL lanca možete pronaći `TObject`. Slika 5.5 prikazuje neke od osnovih, fundamentalnih klasa i klase koje su izdvojene iz navedenih klasa.



Slika 5.5
Hijerarhija VCL klasa

`TObject` je pradeda svih VCL klasa komponenti. (Zapamtite da su sve klase u jeziku Object Pascal izdvojene iz klase `TObject`.) Ispod klase `TObject` možete videti `TPersistent` klasu. Ova klasa upravlja mogućnostima komponenti da same sebe snime u datoteke i memoriju, a takođe upravlja nekim detaljima koje nije potrebno da upoznate. Zahvalan sam (a i Vi bi takođe trebali da budete) da nije potrebno veliko poznavanje klase `TPersistent`, kako bi se pisala većina aplikacija u Delphiju.

Klasa `TComponent` služi kao direktnija fundamentalna klasa vezana za komponente. Ova klasa pruža kompletnu funkcionalnost koju osnovne komponente zahtevaju. Nevizuelne komponente su izdvojene direktno iz klase `TComponent`. Vizuelne komponente su izdvojene iz klase `TContol`, koja je, kao što to možete videtri na slici 5.5 izdvojena iz klase `TComponent`. Klasa `TControl` pruža dodatnu funkcionalnost koju zahtevaju vizuelne komponente. Individualne komponente su izdvojene iz klasa `TGraphicControl` i `TWinControl`.

Kada postavite komponentu na formu, Delphi kreira pointer na komponentu u deklaraciji klase forme, tako da možete pristupiti komponenti u okviru koda. Delphi koristi karakteristiku `Name` komponente, da bi dodelio naziv promenljivoj pointeru. Kada ste kreirali u prethodnom delu knjige primer aplikacije, postavili ste memo komponentu na formu. U tom trenutku Delphi je kreirao `TMemo` promenljivu i dodelio joj naziv `Memo`.

Slično se dogodilo i prilikom kreiranja dugmeta u okviru forme. Delphi je kreirao `TButton` promenljivu koja predstavlja dugme. Pre nego što se sve ovo dogodilo, Delphi je izdvojio novu klasu iz klase `TForm` i, naravno, kreirao slučaj ove klase da predstavlja formu.

Malo razumevanja VCL klasa i komponenti je očigledno i neophodno pre početka rada sa VCL-om. Naravno, ne mogu Vam dati pregled svake VCL klase posebno, pošto ne bih postigao skoro ništa. Pogledajmo neke klase koje se najčešće koriste.

Klase forme i aplikacije

Klase formi i aplikacija predstavljaju forme i objekte `Application` u VCL-u. Ove klase su izdvojene iz klase `TComponent` i same predstavljaju klasu. Izdvojene su da bi ih razlikovali od kontrola koje se postavljaju na formu.

Klasa `TApplication`

Klasa `TApplication` enkapsulira osnovne operacije Windows programa. Klasa `TApplication` vodi računa o dužnostima kao što su upravljanje ikonom aplikacije, omogućavanje pomoći za određeni kontekst, a takođe i obavlja poslove fundamentalnog upravljanja porukama. Svaka Delphi aplikacija ima pointer na objekt `TApplication` koji se naziva `Application`. Klasu `TApplication` primarno koristite za izvršavanje okvira za poruke, upravljanje programom za pomoć u određenom kontekstu i postavljate tekst za savet na dugmad i statusnu traku. Klasa `TApplication` izgleda pomalo čudno u VCL-u, pošto neke od njenih karakteristika (`Icon`, `HelpFile` i `Title`) mogu biti setovane preko kartice Application u okviru za dijalog Project Options.

Klasa `TForm`

Klasa `TForm` enkapsulira forme u VCL. Forme se koriste za glavni prozor, okvire za dijalog, sekundarne prozore i skoro sve tipove prozora koje možete da zamislite. Klasa `TForm` je "šljakerska" klasa u okviru VCL-a. Sadrži blizu 60 karakteristika, 45 metoda i 20 događaja. Forme su detaljno obrađene u lekciji dana 4.

Klase komponenti

Ova grupa klasa sadrži širok spektar klasa koje dalje mogu biti podeljene u odvojene kategorije, što sam i uradio u narednim poglavljima.

Klase standardnih komponenti

Standardne komponente enkapsuliraju većinu uobičajenih Windows kontrola. Klasa standardnih komponenti uključuje: `TButton`, `TEdit`, `TListBox`, `TMemo`, `TMainMenu`, `TScrollBar`, `TPopupMenu`, `TCheckBox`, `TRadioButton`, `TRadioGroup`, `TGroupBox`, `TPanel`, `TActionList`.

Većina ovih klasa enkapsulira Windows kontrole, tako da ih neću u ovom trenutku objašnjavati. Klasa `TMainManu` enkapsulira glavni meni aplikacije. U toku dizajniranja, dvostrukim klikom miša na ikonu komponente `MainMenu` prikazuje dizajner menija.

`TMainMenu` klasa sadrži karakteristike koje kontrolišu da li je opcija menija siva, da li je odabrana, redni broj kontekst pomoći, tekst saveta opcije i slično. Svaka stavka menija ima jedan događaj `OnClick`, tako da možete priključiti upravljač događajem

na stavku menija koja je odabrana. O menijima i dizajneru menija ćete saznati više u sutrašnjoj lekciji.

Komponenta `TPanel` Još jedna standardna komponenta od interesa je `TPanel`.

NOVI TERMIN Pano (*panel*) predstavlja pravougaonu oblast na formi koja obično sadrži svoje komponente, koje se mogu uzeti kao zasebna jedinica.

Komponenta Panel je komponenta kontejner. Uobičajeno je da ova komponenta može sadržati druge komponente. Panoi imaju karakteristike koje kontriolišu koji tip ivice može imati pano; bilo da je pano izdignut (raised), ulegnut (sunken), odnosno ravan (flat); kao i debljinu okvira. Kombinacija ovih karakterisrtika se može koristiti za kreiranje 3D panoa.

4 **Komponenta `TActionList`** Komponenta `TActionList` je novina u Delphiju 4. Ova komponenta se može koristiti za lakše dodavanje komandi koje aktiviraju komponentu, odnosno grupu komponenti. Na primer, aplikacija koja koristi Clipboard može imati elemente za isecanje, kopiranje i lepljenje (cut, copy, paste) u okviru menija, na traci za alate, odnosno u okviru menija sadržaja. Ukoliko postoji podatak na Clipboard-u, dugme za lepljenje i stavke menija mogu biti aktivirane. Ukoliko ne postoje podaci na Clipboard-u, dugme i stavke menija mogu biti neaktivne. Sve kontrole (dugmad na traci za alate i stavke menija) mogu biti aktivirane, odnosno neaktivne u isto vreme korišćenjem komponente `TActionList`.

Komponente na kartici Additional Delphi sadrži još jednu grupu komponenti koje sam ubacio u standardne kontrole. Ove kontrole se mogu pronaći na kartici Additional u okviru palete komponenti. Klase koje predstavljaju ove komponente uključuju: `TBitBtn`, `TSpeedButton`, `TMaskEdit`, `TStringGrid`, `TDrawGrid`, `TImage`, `TShape`, `TBevel`, `TScrollBox`, `TCheckListBox`, `TSplitter`, `TStaticText`, `TChart`. Klasa `TBitBtn` predstavlja dugme sa slikom.

Klasa `TSpeedButton` je takođe dugme sa slikom, ali ova komponenta nije pravo dugme. Ona predstavlja zamenu za dugme. Ovo omogućava postavljanje velikog broja dugmadi prečica (speed buttons), koji ne koriste Windows resurse za svako dugme zasebno.

Komponenta `TImage` Vam omogućava da postavite sliku na formu koju možete odabrati sa datoteke koja se nalazi na disku. Možete koristiti komponentu `TBevel` da kreirate okvire i linije koje su izdignute, odnosno spuštene.

Komponente `TBevel` se mogu koristiti za razdvajanje formi na oblasti koje pružaju estetski zadovoljavajuću formu. Klase `TStringGrid` i `TDrawGrid` Vam daju utisak predstavljanja informacija u mrežastom obliku.

Win32 korisničke klase za kontrolu

VCL ima klase komponenti koje enkapsuliraju većinu korisničkih kontrola 32-bitnih Windows-a. Ove klase uključuju: `TListView`, `TTreeView`, `TTrackBar`, `TProgressBar`, `TTabControl`, `TPageControl`, `TRichEdit`, `TImageList`, `TStatusBar`, `TAnimate`, `TDateTimePicker`, `TToolBar`, `TCollBar` i nekoliko drugih. Neke od ovih kontrola po samoj prirodi su komplikovane, pa su i VCL klase koje ih predstavljaju isto tako komplikovane. Verujte mi kada Vam kažem da VCL puno pojednostavljuje zamršeni rad sa ovim uobičajenim kontrolama. Trebalo bi da neko vreme vežbate rad sa navedenim klasama, pre nego što ih potpuno razumete. U lekciji dana 13 "Iza osnova Delphija", obrađene su komponente: `TToolBar`, `TCoolBar`, `TImageList` i `TStatusBar`.

Klase komponenti baze podataka

VCL sadrži domaćina (host) za komponente baza podataka koje uključuju i vizuelne i nevizuelne klase. Nevizuelne komponente baza podataka uključuju: `TDataSource`, `TDatabase`, `TTable` i `TQuery`. Ove klase enkapsuliraju operacije sa bazom podataka koje se odvijaju u pozadini.

Klase vizuelnih komponenti baze podataka su deo VCL operacija sa bazama podataka koje korisnici mogu videti i mogu sa njima raditi. Na primer, komponenta `TDBGrid` se koristi da prikaže bazu podataka u obliku tabele (mreže). Na ovaj način `TDBGrid` komponenta se ponaša kao interfejs između korisnika i baze podataka. Koristeći komponentu `TDBGrid`, korisnik može pregledati i editovati tabelu baze podataka koja se nalazi na disku.

Komponenta `TDBNavigator` obezbeđuje dugmad koja omogućavaju korisniku pomeranje kroz tabelu baze podataka. Ova klasa uključuje dugmad za sledeći slog, prethodni slog, prvi slog, poslednji slog, prekid editovanja, prihvatanje editovanja i vraćanje na stanje pre editovanja (undo).

Ostale klase komponenti vezane za rad sa podacima povezuju standardne Windows kontrole sa poljiima baze podataka. Ove klase uključuju `TDBText`, `TDBEdit`,`TDBListBox` i `TDBImage` između ostalog.

Grupa komponenti koja je obično povezana sa programiranjem baza podataka su komponente QuickReport. Ova grupa komponeti čini pisanje izveštaja jednostavnim, naročito ako je izvor Vaših podataka baza podataka. Komponente QuickReport za baze podataka će biti obrađene u lekcijama dana 16, 17 i 18.

Klase uobičajenih dijaloga

Da ne bi brinuli, Windows ima uobičajene okvire za dijalog za operacije poput otvaranja datoteka, snimanja datoteka, izbora fonta i izbora boje. VCL enkapsuklira ove uobičajene okivire za dijalog u klase koje predstavljaju određeni tip. Klase su:

TOpenDialog, TSaveDialog, TOpenPictureDialog, TSavePictureDialog, TFontDialog, TColorDialog, TPrintDialog i TPrinterSetupDialog. VCL, takođe, dodaje klase TFindDialog i TReplaceDialog ovoj grupi komponenti. Sve komponente u okviru grupe su nevizuelne, pa nemaju vizuelni interfejs u toku dizajniranja. Okviri za dijalog su vidljivi, naravno, samo u toku rada programa.

Klase sistemskih komponenti

Kartica System na paleti komponenti sadrži kombinaciju vizuelnih i nevizuelnih komponenti. Klasa TTimer se koristi da predstavi sistemski tajmer Windows-a. Njen jedini događaj je OnTimer, koji se poziva svaki put kada tajmer da signal (okine). Interval tajmera se podešava kroz karakteristiku Interval. TTimer je nevizuelna komponenta.

U ovu grupu klasa je ubačena i klasa TMediaPlayer. Ova klasa Vam omogućava da izvršavate multimedijalne datoteke, kao što su wave audio, AVI video i MIDI audio. Multimedijalne datoteke mogu biti puštene, zaustavljene, stavljene na pauzu, odnosno pozicionirane na određeni deo datoteke; isto tako je moguće primeniti još mnogo drugih operacija. Ova klasa ima mnogo karakteristika i događaja koji uveliko pojednostavljuju kompleksan svet Windows Media Control Interface (MCI - multimedijalni kontriolni interfejs).

Komponenta TPaintBox pruža prazan kanvas na kom možete crtati šta poželite. Ova komponenta ima mnogo potencijalnih upotreba. Grupa System uključuje OLE i klase za dinamičku razmenu podataka (DDE - Dynamic Data Excange).

Grupa komponenti Win 3.1

Nemojte praviti grešku i automatski izbaciti ovu grupu komponenti samo zato što jezičak na kartici to pokazuje. Ova grupa sadrži neke odlične komoponente. (Kartica Win 3.1 ima svoje korene u Delphiju 1.) U suštini, najviše volim komponente TTabSet i TNotebook. Ova grupa takođe uključuje nekoliko klasa komponenti koje omogućavaju da kreirate sopstvene okvire za dijalog File Open i File Save. Klase su TFileListBox, TDirectoryListBox, TDriveComboBox i TFilterComboBox.

Internet komponente

U zavisnosti od verzije Delphija koju imate (Standard, Professional, odnosno Client/Server), možete imati karticu Internet. Ova kartica sadrži komponente koje koriste Internet programiranje. Komponente uključuju HTML, FTP, SMTP, POP3 i HTTP komponente. Ova kartica takođe sadrži komponente koje se koriste za osnovno programiranje mreže preko Winsock API. Većina ovih komponenti su prirodne VCL komponente, ustvari, bar jedna od njih; THTML komponenta je ActiveX kontrola.

Komponente primeri

Kartica Samples sadrži komponente koje se mogu koristiti za učenje pisanja komponenti. Izvorni kod ovih komponenti je pridružen, tako da možete videti kako komponente rade. Komponente primeri uključuju: `TGauge`, `TColorButton`, `TSpinButton`, `TSpinEdit`, `TDirectoryOutline` i `TCalendar`.

ActiveX kontrole

Kartica ActiveX sadrži ActiveX kontrole koje možete koristiti u Vašim aplikacijama. Ove kontrole uključuju Chart FX od Software FX Inc., Visual Speller od Visual Components Inc., Formula One Spreadsheet, Formula One VtChart i Graph kontrolu od Bits Per Second, Ltd.

GDI klase

GDI (Graphics Device Interface - grafički interfejs za uređaje) klase imaju mnogo posla u Windows GUI aplikacijama. Ove klase enkapsuliraju korišćenje bitmapa, fontova, sadržaja uređaja (DC - device context), četkica i pera. Preko GDI objekata se prikazuje grafika i tekst u okviru prozora. GDI klase nisu dodeljene određenoj komponenti, ali mnoge komponente imaju slučajeve ovih klasa kao karakteristike. Na primer, edit kontrola ima karakteristiku pod nazivom `Font` koja je slučaj klase `TFont`.

Pojam sadržaj uređaja (*device context*) je dobro poznat tradicionalnim Windows programerima. VCL, pak, ovaj termin ne koristi često, pošto VCL enkapsulira Windows kontekst uređaje u `TCanvas` klasi. VCL koristi pojam kanvas (*canvas - platno*) da ukaže na kontekst Windows uređaja. Kanvas obezbeđuje površinu po kojoj možete crtati korišćenjem metoda poput `MoveTo`, `LineTo` i `TextOut`. Bitmape se mogu prikazati na kanvasu korišćenjem metoda `Draw`, ili `StretchDraw`. Koncept kanvasa koji možete koristiti daje više smisla arhaičnom terminu kontekst uređaja; zar ne mislite da je tako? Grafičke operacije će biti obrađene u lekciji dana 12, "Programiranje grafike i multimedije". Sledi lista najčešće korišćenih GDI klasa

- `TCanvas` klasa sadrži slučajeve drugih GDI klasa. Na primer, kada želite da uradite sekvencu `MoveTo/LineTo`, biće nacrtana linija sa tekućom bojom pera. Karakteristika Pen se koristi da odredi tekuću boju pera i predstavlja slučaj klase `TPen`. Klasa `TPen` ima karakteristike koje određuju koji tip linije treba nacrtati: debljinu linije, stil linije (puna, isprekidana, tačkasta, itd.), odnosno mod u kom će se crtati linija.
- Klasa `TBrush` predstavlja četkicu koja se koristi za popunjavanje određenim dezenom oblika koji se koriste za operacije sa kanvasom, kao što su `FillRect`, `Polygon` i `Ellipse`. Karakteristika `TBrush` uključuje `Collor`, `Style` i `Bitmap`. Karakteristika `Style` omogućava definisanje šrafiranih dezena za četkicu. Karakteristika `Bitmap` omogućava Vam da definišete bitmapu za popunjavanje dezena.

- `TBitmap` enkapsulira operacije sa bitmapama u okviru VCL-a. Karakteristike uključuju: `Palette`, `Height`, `Width` i `TransparentCollor`. Metode uključuju `LoadFromFile`, `LoadFromResourceID` i `SaveToFile`. Klasu `TBitmap` koriste druge komponente klasa kao što su `TImage`, `TBitBtn` i `TSpeedButton`, a kao dodatak i klasa `TCalvas`. Slučaj klase `TBitmap` se takođe može koristiti kao neekranska bitmapa. Ove vrste bitmapa se često koriste u aplikacijama koje intenzivno rade sa grafikom, kako bi se umanjilo treperenje i na taj način poboljšala grafika.

- Klasa `TFont` upravlja operacijama sa fontovima. Karakteristike uključuju: `Color`, `Height` i `Style` (podebljano, kurziv, normalno, itd.). Klasu `TFont` mogu koristiti sve klase komponenti koje prikazuju tekst.

Kao dodatak nabrojanim GDI klasama, postoje klase koje mogu raditi kao pomoćne klase, odnosno mogu proširiti osnovnu klasu i na taj način obezbediti dodatnu funkcionalnost. U toku rada sa Delphijem, naučićete više o ovim klasama i kako ih možete koristiti. Slika 5.6 prikazuje hijerarhiju VCL klasa koje enkapsuliraju GDI operacije.

Slika 5.6
VCL GDI hijerarhija klasa

Pomoćne klase

VCL sadrži više pomoćnih klasa koje možete koristiti u Vašim aplikacijama. *Pomoćne klase* pojednostavljuju određene zadatke programiranja na Windows-ima. Na primer, klasa `TIniFile` olakšava čitanje i pisanje Windows datoteka za konfiguraciju (`.INI datoteke`). Uobičajeno je da se `.INI datoteke` sve manje koriste, a da se podaci upisuju u Registry datoteku. Kao dodatak Registry operacijama, VCL sadrži klase `TRegistry` i `TRegkeyInfo`.

Klasa `TStringList` omogućava korišćenje nizova stringova. Klasu `TStringList` koristi većina klasa komponenti za upisivanje stringova. Na primer, TMemo klasa koristi objekt klase `TStringList` za svoju karakteristiku `Lines`. Klasa `TStringList` ima mogućnost snimanja liste stringova u datoteku, odnosno čitanja stringova iz datoteke korišćenjem metoda `LoadForomFile` i `SaveToFile`. Klasa `TStringList` se takođe može koristiti za čitanje i zapisivanje tekst datoteka.

Još jedna pomoćna VCL klasa je klasa `TList`. Ova klasa Vam omogućava da kreirate nizove bilo kog tipa objekta. Klasa `TList` čuva samo listu pointera. Osnovna prednost klase `TList` leži u pružanju nizova koji mogu dinamički rasti, odnosno smanjivati se nakon dodavanja, odnosno brisanja objekta.

VCL takođe uključuje skup klasa koji Vam omogućavaju čitanje i pisanje tokova (tok je, ustvari, blok podataka). Klase `TStream`, `TFileStream`, `TMemoryStream` i `TResourceStream` Vam omogućavaju da čitate i upisujete podatke u tokove. Klasa `TStream` predstavlja osnovnu klasu svih klasa tokova. Klasa `TFileStream` se koristi u radu sa datotekama na disku, `TMemoryStream` se koristi za manipulaciju podacima u memoriji, a `TResourceStream` se koristi za učitavanje resursa iz EXE i DLL datoteka. Ove klase imaju širu primenu koja se obezbeđuje korišćenjem određenih funkcionalnih osobina. Detaljnije informacije o ovim klasama možete pronaći u Delphijevom VCL programu za pomoć.

I ovo ne bi bilo sve...

Nema sumnje da u ovom delu nisu pokrivene sve VCL klase. Ipak, dotaknute su klase koje ćete najverovatnije koristiti u Vašim aplikacijama.

Vratite se unazad nekoliko strana i ponovo pogledajte listing 5.1. Listing pokazuje kako je jednostavno učitati i prikazati bitmape u okviru Delphija. Uradimo vežbu koja ovo dokazuje. Prvo, otvorite novu aplikaciju. Pred Vama će biti prazna forma. Pratite sledeće korake:

1. Promenite karakteristiku `Caption` forme u `Bitmap Test Program`.
2. Kliknite na karticu `Additional` u okviru palete komponenti, odaberite komponentu `Image`, a zatim postavite komponentu na formu.
3. Pronađite karakteristiku `Align` i promenite je u `alClient`. Komponenta slike popunjava klijent oblast forme.
4. Pronađite `Stretch` karakteristiku i promenite je u `True`.
5. Pronađite karakteristiku `Picture` i dva puta kliknite mišen na kolonu `Value`.
6. Okvir za dijalog Picture Editor će biti prikazan. Kliknite na dugme Load. Biće prikazan okvir za dijalog FileOpen.
7. Pređite u direktorijum `\Program Files\Common Files\Borland Shared Files\Images\Splash\256Color` i odaberite sliku koja će biti prikazana (ja volim sliku `HANDSSHAKE.BMP`). Kliknite na dugme OK.
8. Vratili ste se u okvir za dijalog Image Editor, a bitmapa koju ste odabrali je prikazana u prozoru za pregled slike. Kliknite na dugme OK. (Ukoliko želite da odaberete drugu bitmapu, kliknite ponovo na dugme Load.) Bitmapa sada zauzima klijent oblast forme.

9. Kliknite na dume Run. Kada se pokrene aplikacija možete menjati veličinu prozora, a bitmapa će uvek ispunjavati klijent oblast u okviru prozora.

Vidite kako je lako. Bilo bi mnogo jednostavnije da se niste mučili sa podešavanjem slike koja će popuniti klijent oblast forme. Slika 5.7 prikazuje program za testiranje bitmapa u toku rada.

Slika 5.7
Program za testiranje bitmapa u toku rada

Zaključak

Danas ste učili o kosturima programa i kako se VCL uklapa u ovu oblast. Obradili smo karakteristike, metode i događaje i stekli praktična iskustva kroz procese. Završili smo sa pregledom VCL klasa, koje ćete najverovatnije sretati u toku programiranja na Delphiju. Nisu pokrivene sve VCL klase, ali su zato ukratko objašnjene klase koje se najčešće koriste.

Kuda ova industrija ide? Izgleda da će budućnost pripasti komponentama. Dobra stvar vezana za VCL je ukoliko odlučite (ili budete primorani) da pređete na C++, još uvek možete koristiti VCL. Možda ćete biti iznenađeni, ukoliko saznate da se potupno isti VCL koji se koristi u Delphiju, takođe koristi i u Borlandovom C++Builder-u. Sve što naučite o VCL-u, možete trenutno primeniti u jeziku C++, kada za to dođe vreme. VCL je potpuno isti, pa ne treba ništa novo učiti.

Radionica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstirte razumevanje obrađenog materijala, kao i vežbe koje Vam omogućavaju da steknete iskustvo u korišćenju gradiva koje ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A, "Odgovori na kviz pitanja".

Pitanja i odgovori

P **Šta je to kostur programa?**

O Kostur programna se takođe naziva biblioteka klasa, a predstavlja skup klasa koji pojednostavljuju Windows programiranje. Dobar kostur programa implementira objektno orijentisani dizajn i objektno orijentisano programiranje, usvajajući objekno orijentisani pristup pisanja Windows aplikacija.

P **Da li je VCL kostur?**

O Da, VCL je kostur programa. VCL je napisan u jeziku Object Pascal i radi i na Delphiju i na C++Builder-u.

P **Čini se da je način rada sa komponentama najbolji pristup. Da li je to tačno?**

O U većini slučajeva jeste. VCL je odličan izbor za aplikacije koje koriste mnogo okvira za dijalog, kao i za aplikacije koje rade sa bazama podataka. Ne postoji skoro ništa što ne možete uraditi koristeći Delphi, pošto Delphi prosto blista u oblasti brzog razvoja aplikacija (RAD).

P **Da li su karakteristike samo polja podataka klase?**

O Ne. Karakteristike su posebna bića. Neke karakteristike jednostavno menjaju polja podataka u okviru klase. Druge karakteristike, ukoliko se menjaju, pozivaju metode koje izvršavaju posebne operacije nad karakteristikama. U ovim slučajevima, karakteristike se koriste za komplikovanije poslove od jednostavne promene polja klase.

P **Da li treba da odgovorim na svaki događaj koji komponenta definiše?**

O Ne. Možete odgovoriti samo na one događaje koji su potrebni za rad Vaše aplikacije, odnosno ne morate odgovoriti ni na jedan događaj.

P **Da li su komponente Win 3.1 zastarele?**

O Ni u kom slučaju! Ove komponente su veoma važne u Delphiju 1, pošto je Delphi 1 bio 16-bitno okruženje za programiranje (otuda naziv kartice Win 3.1). To ne znači da su ove komponente zastarele u Delphiju 4, naprotiv.

P **Sigurno postoji mnogo VCL klasa. Mislio sam da će programiranje na Delphiju biti lakše.**

O Programiranje na Delphiju je mnogo jednostavnije od programiranja pod Windows operativnim sistemom korišćenjem tradicionalnih alata. Windows programiranje, bez obzira koliko alat za programiranje bio dobar, zahteva veliko iskustlvo i veliko znanje. Postaćete ekspert ukoliko sve naučite.

Kviz

1. Da li su sve komponente vidljive u toku dizajniranja?
2. Da li je komponenta OpenDialog vizuelna, ili nevizuelna komponenta?
3. Koji je naziv VCL klase koja predstavlja Delphi formu?
4. Da li se sve verzije Delphija isporučuju sa istim skupom komponenti?
5. Da li su sve VCL klase bez razlike izdvojene iz klase TObject?
6. Navedite jednu nevizuelnu VCL komponentu.
7. Da li sve komponente dele neke zajedničke karakteristike?
8. Navedite dve uobičajene karakteristike koje sve vizuelne komponente dele.
9. Mogu li dve, odnosno više komponenti deliti isti upravljač događajima?
10. Koji je VCL termin za Windows kontekst uređaja? Koji je naziv VCL klase koja enkapsulira kontekst uređaja?

Vežbe

1. Napišite kratak sastav u kom ćete opisati po čemu se razlikuju karakteristike i klasna polja.
2. Kreirajte Delphi aplikaciju koja prikazuje bitmapu na glavnoj formi nakon pritiska na dugme.
3. Kreirajte Delphi aplikaciju koja prikazuje okvir sa porukom: `Hello, Bubba!` kada se klikne na glavnu formu.
4. Postavite nekoliko komponenti po Vašem izboru na formu. Kliknite na komponentu i proučite karakteristike komponenti koje se nalaze u prozoru Object Inspector. Ponovite korake za svaku komponentu forme.
5. Prebacite se na karticu Events i pogledajte sve događaje koji su vezani za komponentu na formi postavljenu u koraku 4.

Dan 6

Rad sa dizajnerom formi i dizajnerom menija

Kao što ste mogli saznati do sada, Delphi je okruženje za programiranje koje se oslanja na forme, što predstavlja model koji pruža najviše prednosti prilikom vizuelnog programiranja. U ovom poglavlju ćete upoznati:

- dizajner formi (Form Designer)
- dizajner menija (Menu Designer)

Da bi ilustrovali korišćenje dizajnera forme, napravićete aplikaciju koja liči na Windows Notepad program. Do sada ste stekli vredno iskustvo radeći sa dizajnerom formi. U drugom delu današnje lekcije ćete se detaljnije upoznati sa dizajnerom menija.

Današnja lekcija će Vam možda izgledati suviše elementarno, ukoliko ste intenzivno koristili Delphi. Čak i da je to slučaj, pregledajte je na brzinu, kako bi otkrili stvari koje Vam prethodno nisu bile poznate, odnosno da bi ponovo otkrili stvari koje ste zaboravili. Želim da se opkladim sa Vama da najmanje jedna stvar u ovom poglavlju predstavlja novinu za Vas.

Rad sa dizajnerom forme

Delphijev dizajner forme (Form Designer) je moćan vizuelni alat za programiranje. Omogućava Vam da postavite, odaberete, pomerite, promenite veličinu, odnosno poravnate komponente i još mnogo toga. Dizajner forme Vam isto tako omogućava

da podesite veličinu i poziciju same forme, dodate menije i kreirate specijalizovane okvire za dijalog - sve što Vam je potrebno da kreirate korisnički interfejs tipičnog Windows programa.

Upoznaćemo se sa svim mogućnostima dizajnera forme u narednim poglavljima. Kao što ste mogli da vidite, ohrabrivao sam Vas da zastanete i eksperimetišete svaki put kada Vas je interesovalo kako nešto radi. Ponekad par minuta igranja može da Vas nauči tehnikama koje ćete kasnije dugo koristiti.

Meni sadržaja dizajnera forme

Prilikom prvog startovanja Delphija, odnosno prilikom kreiranja novog projekta, pred Vama se nalazi prazna forma dizajnera forme. Dizajner forme, poput većine Delphi prozora, ima pridružen meni sadržaja. Tabela 6.1 prikazuje i opisuje svaku opciju menija sadržaja dizajner forme.

Tabela 6.1: Opcije menija sadržaja dizajnera forme

Opcija	Opis
Align to Grid	Poravnava odabrane komponente na osnovu mreže dizajner forme.
Bring to Front	Prebacuje odabrane komponente na vrh svih komponenti.
Send to Back	Prebacuje odabrane komponente u pozadinu svih komponenti.
Revert to Inherited	Za odabrane kontrole vraća originalno stanje, ukoliko radite sa formom koju ste nasledili iz skladišta objekata. (Nasleđivanje formi iz skladišta objekata je obrađeno u lekciji dana 8, "Kreiranje aplikacija u Delphiju".)
Align	Prikazuje okvir za dijalog Alignment (poravnavanje).
Size	Prikazuje okvir za dijalog Size.
Scale	Prikazuje okvir za dijalog Scale.
Tab Order	Prikazuje okvir za dijalog Edit Tab Order.
Creation Order	Prikazuje okvir za dijalog Creation Order.
Flip Children	Za verzije Windows-a koje nisu na engleskom jeziku, ova komanda menja redosled čitanja komponenti. Za englesku verziju Windows-a, ova opcija je neaktivna.
Add to Repository	Dodaje tekuću formu u skladište objekata (Object Repository). Forme koje su korisnici posebno prilagodili mogu biti sačuvane za kasniju upotrebu. (skladište objekata je obrađeno u lekciji dana 8.)
View as Text	Prikazuje opis forme kao tekst u okviru editora koda. Tekst verziju forme možete editovati ukoliko želite. Odaberite opciju View as Form u okviru menija sadržaja editora koda, da bi se vratili na formu. Da biste prelazili sa opcije View as Text na opciju View as Form, možete koristiti kombi naciju tastera Alt+F12.

NAPOMENA Delphi kreira datoteku forme (DFM) za svaku formu koju koristite i postavlja je u direktorijum Vašeg projekta. Datoteka forme je binarna resursna datoteka koju obični ljudi ne mogu čitati. Kada odaberete opciju View as Text u okviru menija sadržaja, Delphi konvertuje binarnu resursnu datoteku u čitljivu formu. Kada se vratite nazad u opciju View as Form, Delphi ponovo prevede formu u datoteku forme i implementira sve izmene koje ste uradili.

Većina opcija menija sadržaja je obrađena u narednim poglavljima. Ostale opcije su obrađene u drugim poglavljima koja Vas upoznaju sa određenim aspektima Delphija.

Postavljanje komponenti

Postavljanje komponenti na formu je jednostavno. Sve što treba da uradite je da prvo iz palete komponenti odaberete formu koju želite, a zatim da kliknete mišem na formu kako bi postavili komponente. Kada kliknete mišem na formu, gornji levi ugao komponente se nalazi na poziciji na koju ste kliknuli. Uočite da prilikom klika miša na dugme u okviru palete komponenti, dugme ostaje pritisnuto. Kada kliknete mišem na formu da biste postavili komponentu, dugme na paleti komponenti se vraća u početni položaj kako bi označilo da je operacija završena.

SAVET Kao što ste mogli da naučite u lekciji dana 4, "Otkrivanje Delphijevog okruženja", da biste postavili komponentu nekoliko puta, pritisnite i držite taster Shift, kada prvi put pritisnete dugme komponente na paleti komponente. Svaki put kada kliknete mišem na formu, biće dodata nova komponenta. Kliknite na dugme sa strelicom u okviru palete komponenti, kako biste zaustavili postavljanje komponenti.

Kod većine komponenti možete menjati veličinu. Možete postaviti komponentu na formu, a zatim joj promeniti veličinu, odnosno možete podešavati veličinu komponente u trenutku kada je postavljate na formu. Da biste promenili veličinu u toku postavljanja komponente, kliknite mišem na formu gde želite da se nalazi gornji levi ugao, a zatim povlačite kursor miša sve dok komponenta ne dobije željenu veličinu. Kada otpustite taster miša, komponenta će biti postavljena na formu sa definisanim dimenzijama.

NAPOMENA Na ovaj način se ne mogu podešavati dimenzije svih komponenti. Na primer, nevizuelne komponente su na formi predstavljene ikonom. Čak iako kliknete mišem na formu i uvećate je do željene veličine, veličina nevizuelne komponente će biti ignorisana. Još jedan primer predsta-vljaju komponente koje imaju jednu liniju za editovanje: Komponenta za editovanje može biti postavljena prevlačenjem, ali samo širina postavljene komponente će biti prihvaćena. Visina komponente će biti ignorisana, pošto zavisi od generičkih parametara visine kontrole za editovanje jedne linije.

UPOZORENJE Ukoliko se predomislite u toku postavljanja kontrole korišćenjem metoda za prevlačenje, možete pritisnuti taster Esc na tastaturi, pre nego što otpustite taster miša, kako bi poništili tekuću operaciju. Dugme komponente na paleti komponenti će ostati pritisnuto, pa ćete morati da kliknete na dugme sa strelicom, kako bi se vratili u mod za izbor komponente.

Postavljanje komponenti je dovoljno jednostavno, tako da ne morate provoditi mnogo vremena vežbajući. U lekciji jučerašnjeg dana ste stekli iskustvo sa postavljanjem komponenti, stoga ćemo preći na drugu oblast.

Mreža dizajnera forme (Form Designer grid)

Dizajner forme ima ugrađenu mrežu koja pomaže u dizajniranju forme. Generički, Delphi prikazuje mrežu. Dimenzije mreže su početno postavljene na osam piksela po horizontali i osam piksela po vertikali. Kada definišete da dizajner forme prikazuje mrežu, na svakom ukrštanju linija mreže će se nalaziti tačka. Komponente koje se postavljaju na formu će se prilagoditi najbližoj tački mreže. Pod pojmom prilagoditi (*snap to*), podrazumevam da se gornji levi ugao komponente automatski pomera na najbližu tačku mreže. Ovo je prednost, pošto često želite da grupa kontrola bude poravnata bilo po levoj, denoj, gornjoj, ili donjoj ivici. Ukoliko je aktivna opcija Snap to Grid (prilagođavanje mreži), potrebno je da priđete dovoljno blizu određenog mesta, pa da dizajner forme automatski postavi Vašu komponentu na najbližu tačku mreže. Na ovaj način štedite vreme potrebno za podešavanje svake komponente na određeno mesto u okviru forme.

Karakteristike mreže mogu biti menjane korišćenjem kartice Preferences u okviru za dijalog Environment Options. (Okvir za dijalog Enviroment Options će detaljnije biti obrađen u lekciji dana 9, "Projekti, editor koda i ispitivač koda".) Od ovog trenutka možete menjati dimenzije mreže, odnosno isključiti opciju prilagođavanja mreže (Snap to Grid). Takođe možete uključiti, odnosno isključiti prikazivanje mreže. Kada je prikazivanje mreže isključeno, mreža i dalje ostaje aktivna (ukoliko je opcija Snap to Grid uključena), ali tačke koje označavaju mrežu neće biti iscrtane na formi.

Odabiranje komponenti

Nakon što ste postavili komponentu na formu, obično ćete je odabrati kako bi je izmenili. Komponentu možete odabrati da biste izveli neku od navedenih operacija:

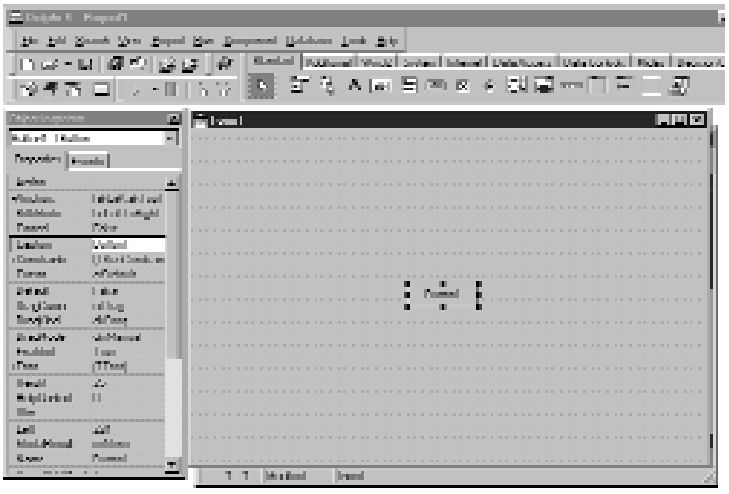
- Pomeranje komponente.
- Promena karakteristika komponente.
- Poravnavanje komponente.
- Promena dimenzija komponente.
- Isecanje ili kopiranje komponente na Clipboard.
- Postavite redosled komponente (pomerite je ispred svih komponenti, odnosno prebacite u pozadinu).
- Obrišete komponentu.

Izbor pojedinačnih komponenti

Da biste odabrali pojedinačnu komponentu, kliknite mišem na nju. Kada odaberete komponentu pojaviće se osam crnih kvadratića za podešavanje veličine komponente, postavljenih po ivici i na taj način označiti da je komponenta odabrana. (Način promene dimenzija komponente će biti obrađen za koji trenutak.) Slika 6.1 prikazuje formu sa odabranom komponentom tipa Button.

Nakon što odaberete komponentu, sadržaj prozora Object Inspector se menja, kako bi prikazao karakteristike i događaje za odabranu kontrolu. Da biste deaktivirali kontrolu, kliknite mišem na pozadinu forme, odnosno kliknite mišem na kontrolu, uz istovremeni pritisak na taster Shift. (Shift+klik na taster miša će detaljnije biti opisan u sledećem poglavlju.)

SAVET Svaka komponenta ima dodeljen generički upravljač događajima. Kada dva puta kliknete mišem na komponentu forme, editor koda prikazuje generički upravljač događajem za odabranu komponentu u koji možete upisati kod. U većini slučajeva generički upravljač događajem je OnClick. šta se tačno događa nakon dvostrukog klika mišem na komponentu, zavisi od načina na koji je komponenta dizajnirana. Na primer, u slučaju komponente Image, dvostruki klik miša će prikazati okvir za dijalog Picture Editor.



Slika 6.1
Forma sa odabranom komponentom tipa Button

Izbor grupe komponenti

Više komponenti možete odabrarti kako bi upravljali celom grupom. Ovo se postiže na jedan od tri načina:

- Shift+klik na taster miša, ukoliko koristite i tastaturu i miš.
- Prevlačenjem miša (preko komponenti).

4 Izborom opcije Edit➥Select All u okviru glavnog menija.

Da biste odabrali sve komponente forme, odaberite opciju Edit➥Select All u okviru glavnog menija.

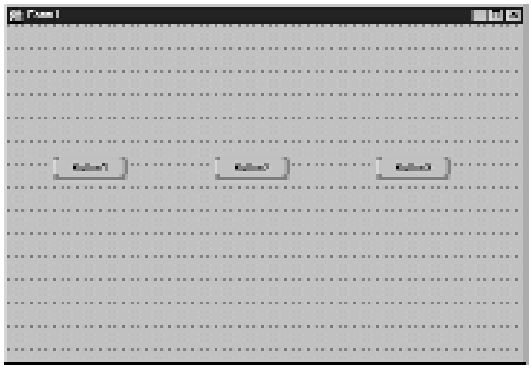
Izbor komponenti korišćenjem kombinacije Shift+klik na taster miša

Da biste koristili kombinackiju Shift+klik na taster miša, prvo odaberite jednu kontrolu. Zatim pritisnite i držite taster Shift na tastaturi i kliknite na kontrole koje želite da uključite u izbor. Svaka kontrola na koju kliknete će biti oivičena sa četiri siva kvadaratića koji ukazuju da je odabrana kontrola deo grupe koja je odabrana.

Kontrolu iz izbora možete ukloniti, ukoliko držeći taster Shift, ponovo kliknete na komponentu. Drugim rečima, kombinacija Shift+klik na taster miša naizmenično uključuje i isključuje komponentu iz izbora. Da bi ovo ilustrovali, počnite sa praznom formom, a zatim pratite sledeće korake:

1. Postavite tri dugmeta bilo gde u okviru forme. Dugmadima će automatski biti dodeljeni nazivi Button1, Button2 i Button3.
2. Kliknite na Button1. Crni pravougaonici za promenu veličine će se pojaviti oko komponente.
3. Pritisnite i držite Shift taster na tastaturi. Kliknite mišem na Button2. Dugme je dodato izboru. Sada se pojavljuju sivi kvadratići na uglovima dugmadi Button1 i Button2.
4. Shift+klik na taster miša primenite na dugme Button3. Sada su sva tri dugmeta odabrana.
5. Shift+klik na taster miša primenite na dugme Button2. Dugme Button2 se više ne nalazi u izboru (sivi kvadratići su nestali), dok su dugmad 1 i 3 još uvek odabrana.
6. Shift+klik na dugme miša primenite na dugme Button1. Sada je samo komponenta Button3 odabrana. Sivi kvadratići su zamenjeni crnim pravougaonicima.
7. Shift+klik na taster miša primenite na dugme Button1 i Button2. Ponovo su sva tri dugmeta odabrana.

Slika 6.2 prikazuje kako forma izgleda na kraju vežbe. Setite se da dugmad mogu biti postavljena bilo gde na formi. Neka Vam forma bude pri ruci, pošto će Vam trebati ponovo u sledećoj vežbi.



Slika 6.2
Forma na kojoj su sva tri dugmeta odabrana

NAPOMENA Ukoliko kliknete na komponentu koja je deo izbora, ništa se neće dogoditi. Da biste odabrali samo jednu kontrolu koja je trenutno deo odabrane grupe, prvo treba da kliknete na pozadinu forme, odnosno pritisnete taster Esc, kako biste uklonili izbor grupe. Zatim možete kliknuti mišem na kontrolu koju želite da odaberete.

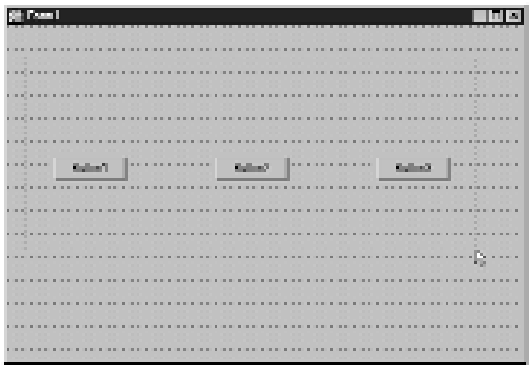
Izbor više komponenti prevlačenjem miša

Više kontrola možete odabrati prevlačenjem pravougonika oko kontrola koje želite da odaberete. *Pravougaonik* za ograničavanje je predstavljen kao pravougaonik sa isprekidanim linijama i u toku prevlačenja menja veličinu. U principu, ne morate da prevučete pravougaonik za ograničenje preko cele komponente. Kompnentu možete samo dodirnuti, da bi je uključili u izbor.

Uverite se da ste počeli izbor postavljajući kursor miša na pozadinu forme, a ne na komponentu. Pritisnite i držite levi taster miša i počnite sa prevlačenjem. U toku prevlačenja ćete moći da vidite pravougaonik za ograničenje. Okružite, ili dotaknite komponente koje želite da odaberete, a zatim otpustite taster miša. Komponente koje su bile unutar pravougaonika za ograničenje (ili su bile dotaknute) su uključene u izbor.

Kada ste odabrali grupu kontrola, možete korišćenjem kombinacije Shift+klik na taster miša, koja je objašnjenja u prethodnom poglavlju, dodati druge kontole u izbor, odnosno ukloniti kontrole iz izbora. Na primer, možda ćete poželeti da odaberete sve kontrole određene oblasti u okviru forme, izuzev jedne kontrole. Okružite kontrole, a zatim uklonite izbor sa kontrole koju želite da isključite iz izbora.

Vratite se na formu sa tri dugmeta koju ste ranije kreirali (ukoliko ste već uklonili formu, kreirajte novu i postavite tri dugmeta). Počnite od gornjeg levog ugla i prevlačite mišem na desno, da okružite dugmad. Otpustite taster miša i sve kontrole će biti odabrane. Slika 6.3 prikazuje formu i pravougaonik za ograničenje u toku prevlačenja.



Slika 6.3
Kontrole u trenutku izbora prevlačenjem miša

UPOZORENJE Da biste odabrali dve nepovezane grupe kontrola, možete koristiti kombinaciju Shift+prevlačenje mišem. Na primer, ukoliko imate dve odvojene grupe kontrola na različitim delovima forme, možete prevući mišem oko prve grupe, a zatim držeći Shift taster prevući miša preko druge grupe. Obe grupe će biti odabrane.

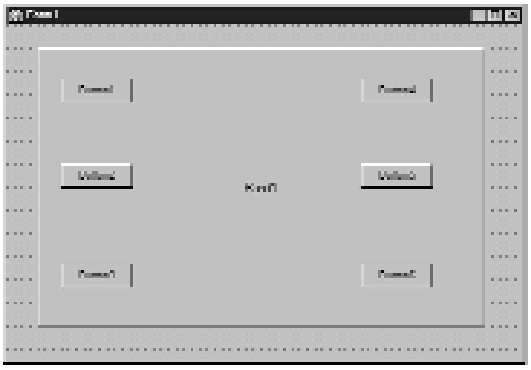
NAPOMENA Ne morate prevlačiti miša na dole i na desno. Prevlačenje možete obavljati u bilo kom smeru.

Izbor više elemenata: komponente u okviru komponenti

Često će Vam se dogoditi da imate komponente koje su postavljene u okviru drugih komponenti. Komponenta `Panel` se obično koristi kao kontejner za druge komponente. Da biste odabrali grupu komponenti u okviru panela, treba da držite pritisnut taster Ctrl, dok prevlačite mišem kako bi odabrali komponentu. (Pokušajte istu operaciju, a da ne pritisnete Ctrl taster, da biste videli šta će se dogoditi!) U slučaju da se pitate, odgovor je: "Da", možete koristiti kombinaciju Ctrl+Shift+prevlačenje miša. (Pretpostavljam da su dizajneri u Borlandu otkrili način rada koji koristi i taster Alt.)

Da bi ovo ilustrovali, počnite sa praznom formom. Zatim pratite sledeće korake:

1. Odaberite komponentu `Panel` u okviru palete komponenti i postavite je na formu, koristeći metod prevlačenja. Proširite je tako da zauzme veći deo forme.
2. Sada odaberite komponentu `Button` i postavite šest dugmadi na formu. Vaša forma bi trebalo da izgleda slično kao i forma na slici 6.4.



Slika 6.4
Forma sa panelom i šest dugmadi

3. Prevucite pravougaonik za ograničenje oko dugmadi Button1, Button2 i Button3. Uočite da ste pomerili panel, što niste očekivali (i niste želeli). Vratite panel na mesto gde se nalazi.
4. Držite pritisnut taster Ctrl i prevucite pravougaonik preko dugmadi Button1, Button2 i Button3. Sada su dugmad odabrana.
5. Sada držite pritisnute tastere Ctrl i Shift i prevucite pravouganik za ograničenje oko dugmadi Button5 i Button6. Sada su sva dugmad odabrana, izuzev dugmeta Button4.

Koristeći kombinaciju Ctrl+prevlačenje miša na jedini način možete odabrati grupu komponenti koje se nalaze u nekoj drugoj komponenti, ukoliko koristite metod prevlačenja. Kombinacijom Shift+klik na taster miša, možete odabrati komponente koje se nalaze u okviru druge komponente, kao što ste to radili odabirajući komponente u okviru forme.

Pomeranje komponenti

Pomeranje komponenti je uobičejen i jednostavan zadatak. Da biste pomerili pojedinačnu komponentu, postavite kursor miša na komponentu i prevlačite mišem. Pravougaonik u toku prevlačenja komponente predstavlja pomeranje zajedno sa kursorom miša. Kada se pravougaonik nađe na mestu gde želite da postavite komponentu, otpustite taster miša i komponenta će biti pomerena.

Kada pomerate kontrole koristeći prevlačenje i puštanje, automatski se osvežavaju vrednosti karakteristika `Left` i `Top`. Ako zastanete na trenutak u toku pomeranja komponente, uočićete oblačić za savet koji se pojavljuje pored kursora miša. Oblačić za savet pokazuje novu poziciju levog gornjeg ugla komponente.

Sličan oblačić za savet se pojavljuje prilikom promene veličine komponente prevlačenjem. U slučaju da menjate veličinu komponente, oblačić za savet će pokazati novu širinu i visinu komponente. Slika 6.6 prikazuje oblačić za savet koji možete videti prilikom promene veličine komponente (u donjem desnom uglu).

NAPOMENA Najlakši način za pomeranje komponente je prevlačenje i puštanje. Ukoliko su Vam potrebne preciznije kontrole, možete izmeniti karakteristike `Left` i `Top` željene komponente u okviru prozora Object Inspector. Takođe možete koristiti različite opcije za poravnavanje, koje će biti obrađene u današnjoj lekciji u okviru poglavlja "Poravnavanje komponenti".

Ukoliko je uključena opcija Snap to Grid, pravougaonik za prevlačenje će se prilagoditi najbližoj tački mreže.

SAVET Ukoliko se predomislite u toku prevlačenja možete pritisnuti taster Esc, pre nego što otpustite taster miša i time prekinuti operaciju. Komponenta će se vratiti na početnu poziciju.

Prevlačenje grupe kontrola radi na istom principu. Nakon što ste odabrali grupu komponenti, postavite kursor miša na bilo koju komponentu i počnite sa prevlačenjem. Pravougaonik za prevlačenje će biti prikazan na svakoj kontroli u okviru grupe. Na ovaj način možete videti gde će svaka komponenta biti postavljena nakon što otpustite taster miša.

NAPOMENA Ukoliko komponente u grupi imaju različite kontrole roditelje, ne možete ih pomerati kao grupu. Na primer, pretpostavimo da ste odabrali obe komponente tipa `Button` u okviru glavne forme zajedno sa komponentom `SpeedButton` u okviru panela. Pošto ove dve komponente imaju različite kontrole roditelje, ne možete ih pomerati kao grupu.

SAVET Kada odaberete kontrole, možete ih pomerati piksel po piksel držeći pritisnut taster Ctrl uz istovremeno korišćenje kursorskih tastera na tastaturi. Ova tehnika radi i sa grupama kontrola i sa pojedinačnim kontrolama. Kada koristite ovu tehniku, prilagođavanje mreži više nije moguće.

Nakon što ste pomerili komponentu korišćenjem ovog metoda, komponenta se više ne nalazi na mreži - pomerena je za neku vrednost. Ukoliko nakon ovog koraka pomerate komponentu, biće zadržana razlika između tačke ukrštanja mreže i komponente.

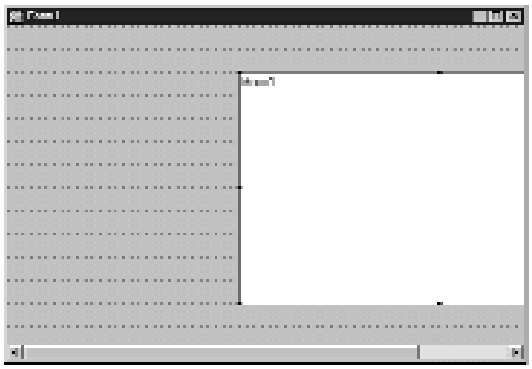
SAVET Ako pomerate kontrole korišćenjem metoda Ctrl+kursorski tasteri na tastaturi i poželite da ponovo poravnate komponentu u odnosu na mrežu, odaberite opciju Edit➥Align to Grid u okviru glavnog menija. Gornji levi ugao kontrole će se prilagoditi najbližoj tački mreže.

Kontrola ne može biti prevučena van forme roditelja. Ukoliko pokušate da prevučete komponentu preko leve, ili gornje ivice forme, uočićete da se komponenta skraćuje na ivici forme. Ukoliko, pak, prevlačite komponentu preko desne, ili donje ivice forme i pustite je, pojaviće se traka za pomeranje na formi, tako da ćete moći da pomerite traku za pomeranje forme, kako bi videli ostatak.

Karakteristike `Width` i `Height` (širina i visina) forme se ne menjaju. Ukoliko prevučete komponentu na vidljivi deo forme, trake za pomeranje će nestati. Ovo ponašanje je generičko i pojavljivaće se sve dok ne promenite karakteristiku forme `AutoScroll` u `False`. Slika 6.5 prikazuje `Memo` komponentu koja je delimično prevučena preko desne ivice forme. Uočite da se pojavila traka za pomeranje na dnu forme.

Sprečavanje komponenti da se pomeraju, odnosno da im se menja veličina

Komponente mogu biti zaključane na određenom mestu, tako da se ne mogu pomeriti. Zaključavanje komponenti je korisno ukoliko znate da je dizajn forme konačan i da ne želite da brinete da ćete slučajno pomeriti kontrole. Da biste zaključali kontrole forme, odaberite opciju Edit➥Lock Controls u okviru glavnog menija. Zaključane kontrole se ne mogu pomerati, odnosno ne može im se menjati veličina. Kada su kontrole zaključane, kvadratići za promenu veličine su sivi sa crnim okvirom. Da biste otključali kontrole, ponovo odaberite opciju Edit➥Lock Controls. Kontrole će moći ponovo da se pomeraju. Možete zaključati sve komponente u okviru forme koristeći ovu tehniku, odnosno ne morate zaključati ni jednu komponentu. Naravno, ne možete zaključati samo odabrane komponente.



Slika 6.5
Forma sa karakteristikom AutoScroll na delu.

##Ređanje, isecanje, kopiranje i lepljenje komponenti

Ponekad ćete postavljati komponente jednu na drugu u želji da ostvarite vizuelni efekat. Na primer, figure sa senkom se mogu kreirati postavljanjem bele figure preko crne (obe komponente treba da budu tipa `Shape`). Očigledno je da ne možete imati senku na vrhu figure, pa ćete kontrolom redosleda izdati komandu Delphiju koja kontrola ide gore, a koja dole. Uradimo jednostavnu vežbu, da bi ovo ilustrovali. U toku rada, sigurno ste saznali kako možete da koristite opcije za kopiranje i lepljenje komponenti. Počnite sa praznom formom (do sada ste sigurno naučili). Zatim sledite korake:

1. Kliknite na jezičak Additional u okviru palete komponenti i odaberite komponentu `Shape`. Kliknite na formu kako bi postavili komponentu. Beo kvadrat će se pojaviti na formi.
2. Promenite veličinu pravougaonika po želji (moj pravougaonik ima dimenzije 209x129 piksela).

3. Uverite se da je komponenta `Shape` odabrana. Odaberite opciju Edit➥Copy u okviru glavnog menija.

4. Odaberite opciju Edit➥Paste u okviru glavnog menija. Kopija pravougaonika će biti postavljena par piksela na dole i desno u odnosu na original. Obično tu i želite da se nalazi.

NAPOMENA Nakon operacije lepljenja, komponenta koju ste upravo zalepili će biti odabrana.

5. Dva puta kliknite mišem na karakteristiku `Brush` u okviru prozora Object Inspector i promenite karakteristiku `Color` u `clBlack`. Novi okvir je sada crn, ali se nalazi iznad originalnog okvira. Ovo nije dobro!

6. Kliknite na drugi taster miša (obično desni) i odaberite opciju Send to Back u okviru menija sadržaja (isto tako možete koristiti opciju Edit➥Send to Back u okviru glavnog menija). Crni okvir se pomerio iza belog okvira. Sada Vaš okvir ima senku. (Isti efekat ste mogli postići i na drugi način. Mogli ste da kliknete na beli okvir, a zatim koristeći opciju Bring to Front postavite beli okvir na crni.)

Ova vežba ilustruje dve mogućnosti dizajnera forme. Pokazuje kako možete da menjate redosled kontrola koje se nalaze na steku, odnosno kako možete koristiti opcije Copy i Paste (kopiraj i zalepi), da biste kopirali komponente. Karakteristike ori-ginalne komponente se u procesu kopiranja i lepljenja prenose na komponentu kopiju. Svaki put kada lepite komponentu na formu, komponenta se postavlja dole-desno u odnosu na prethodnu komponentu.

NAPOMENA Ukoliko je komponenta koja se koristi kao kontejner odabrana u trenutku izvršavanja operacije za lepljenje, komponenta koja se lepi će biti postavljena na formu kao potomak komponente kontejner. Na primer, želite da pomerite dugme sa glavne forme na pano. Možete odabrati dugme, a zatim opciju Edit➥Cut u okviru glavnog menija, kako bi uklonili dugme sa forme i postavili ga na Clipboard (tablu). Zatim možete odabrati pano i koristeći opciju Edit➥Paste u okviru glavnog menija zalepiti dugme na pano.

Ne bih želeo da detaljnije objašnjavam operaciju isecanja. Kada isečete komponentu, ona nestaje sa forme i prelazi na Clipboard. Kasnije možete da zalepite koponentu na formu, odnosno na drugu komponentu kao što je `Panel`.

SAVET Komponentu možete isto tako kopirati i zalepiti je u izvorni kod. Rezultat bi izgledao otprilike ovako:

```
object Edit1: TEdit
  Left = 24
  Top = 16
  Width = 457
  Height = 21
  TabOrder = 0
  Text = 'Edit1'

end
```

Ovo nije kod koji će biti preveden, ali ćete ovom tehnikom moći da vidite veličinu i poziciju komponente u okviru forme. Ova informacija postaje korisna u slučaju kada kreirate komponente u toku rada programa, umesto da kreirate komponente u toku dizajniranja. Možete postaviti privremenu komponentu na formu, uzeti njenu veličinu i poziciju korišćenjem opcija za kopiranje i lepljenje, a zatim obrisati komponentu. Zatim možete napisati kod koji će kreirati komponentu u toku rada programa, znajući da će se nalaziti na odgovarajućem mestu, odnosno da će biti odgovarajuće veličine.

Promena veličine komponente

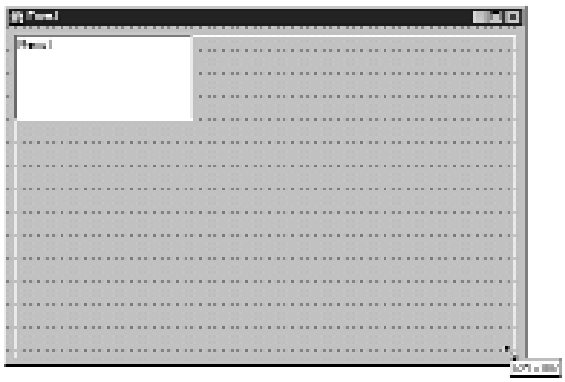
Kod nekih komponenata generičke dimenzije prilikom postavljanja na formu mogu biti prihvatljive. Dugmad predstavljaju dobar primer. Standardno dugme ima visinu 25 piksela, a širinu 75 piksela. U većini slučajeva, generička veličina dugmeta je upravo ono što Vam treba. Kod nekih komponenata, pak, generičke dimenzije ni približno ne odgovaraju. `Memo` komponenta, na primer, skoro uvek mora biti uvećana kako bi popunila formu na kojoj radi.

Promena veličine povlačenjem

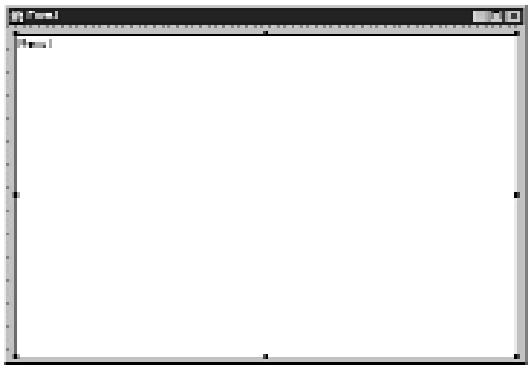
Kada odaberete kontrolu, osam crnih pravougaonika za promenu veličine se pojavljuju oko kontrole. Kada postavite kursor miša preko jednog od pravougaonika za promenu veličine, oblik kursora se menja u dvostruku strelicu koja se naziva kursor za promenu veličine *(sizing cursor)*. Kada ugledate kursor za promenu veličine, možete početi sa povlačenjem, kako biste promenili veličinu kontrole. Od izbora pravougaonika za promenu veličine zavisi i nova veličina komponente.

Pravougaonici za promenu veličine postavljeni u sredinu na gornjem i donjem delu komponente, menjaju vertikalnu dimenziju komponente (duža, ili kraća). Slično je i sa levim i desnim pravougaonikom za promenu veličine komponente, koji menjaju horizontalnu dimenziju (šire, ili uže). Ukoliko koristite pravougaonike na uglovima komponente, možete istovremeno menjati horizontalnu i vertikalnu veličinu. Kao što je to slučaj kod pomeranja komponente, pravougaonik za promenu veličine se pojavljuje u toku povlačenja. Kada dostignete željenu dimenziju, otpustite taster miša, pa će Vaša komponenta dobiti nove dimenzije. Slika 6.6 ilustruje promenu veličine memo komponente povlačenjem; slika 6.7 pokazuje formu nakon završetku operacije povlačenje.

Promena veličine važi samo za vizuelne komponente. Nevizuelne komponente se pojavljuju na formi kao ikone i njihove dimenzije se ne mogu menjati. Pravougaonici za promenu veličine se pojavljuju i kod nevizuelnih komponenti i mogu se povlačiti, ali će rezultat povlačenja biti ignorisan.



Slika 6.6
Memo komponenta u toku operacije promene veličine



Slika 6.7
Forma nakon promene veličine memo komponente

Grupama kontrola nije moguće menjati veličinu povlačenjem. Pravougaonici za promenu veličine (crni pravougaonici) su zamenjeni indikatorima izbora (sivi kvadratići) u slučaju da odaberete više od jedne komponente.

SAVET Da biste istovremeno promenili veličinu komponenti u okviru grupe, izmenite vrednosti `Width`, odnosno `Height` u okviru prozora Object Inspector, odnosno koristite okvir za dijalog Size (okvir za dijalog Size će biti obrađen u sledećem poglavlju). Sve odabrane komponente će dobiti nove vrednosti.

SAVET Da biste menjali veličinu kontrole, odnosno grupe kontrola piksel po piksel, držite pritisnut taster Shift i pritisnite bilo koji kursorski taster na tastaturi. Kursorski tasteri sa strelicama na gore i dole menjaju vertikalnu dimenziju kontrole, a desni i levi kursorski taster menjaju horizontalnu dimenziju. Na ovaj način samo karakteristike `Width` i `Height` će biti promenjene, dok će karakteristike `Top` i `Left` ostati nepromenjene.

Promena veličine korišćenjem okvira za dijalog Size

Još jedna opcija za promenu veličine komponente je okvir za dijalog Size. Okvir za dijalog možete otvoriti biranjem opcije Edit➥Size u okviru glavnog menija. Slika 6.8 prikazuje okvir za dijalog Size.

Slika 6.8
Okvir za dijalog Size.

Okvir za dijalog Size možete koristiti kada želite da grupi kontrola dodelite istu širinu, odnosno visinu. Recimo da imate šest komponenti za editovanje sa različitim širinama. Da bi Vaša forma bila bolje izbalansirana, možete poželeti da sve komponente imaju istu širinu. Prvo odaberite komponente, a zatim pozovite okvir za dijalog Size. U okviru dijaloga odaberite opciju Shrink to smallest (smanjivanje na najmanju), da bi sve komponente dobile širinu najkraće komponente za editovanje. Da bi sve komponente dobile širinu najduže komponente Edit u okviru grupe, koristite opciju Grow to largest. Takođe možete upisati željenu širinu u okvir Width, u kom slučaju treba da postavite visinu (Height) na No change (bez promene). Kada kliknete na dugme OK, komponente će dobiti istu širinu.

SAVET Okvir za dijalog Size takođe može biti pozvan iz menija sadržaja dizajnera forme.

Promena veličine korišćenjem okvira za dijalog Scale

Još jedan alat za promenu veličine komponente je okvir za dijalog Scale, a prikazan je na slici 6.9. Ovaj okvir za dijalog Vam omogućava da definišete procentualnu promenu veličine. Da biste dvostruko uvećali komponentu u polje Scaling factor (faktor promene veličine), upišite `200`. Da biste smanjili veličinu komponente na pola, upišite `50` u polje Scaling factor. Okvir za dijalog Scale je zgodan za brzu promenu veličine svih komponenti forme. Okvir za dijalog Scale možete pozvati opcijom Edit➥Scale u okviru glavnog menija, odnosno izborom opcije Scale u okviru menija sadržaja dizajnera forme.

Kontrole isto tako možete pomerati, odnosno menjati im veličinu koristeći opcije Alignment. Hajde da ih pogledamo u narednom poglavlju.

Slika 6.9
Okvir za dijalog Scale

NAPOMENA Zapamtite, komponente se uvek mogu pomerati promenom karakteristika `Left` i `Top`, odnosno može im se menjati veličina promenom karakteristika `Width` i `Height` u prozoru Object Inspector.

Poravnavanje komponenti

Bez obzira da li ste uključili opciju Snap to Grid, ponekad je potrebno da poravnate komponente nakon što ih postavite na formu. Poravnavanje komponenti može značiti poravnanje nekoliko komponenti duž zajedniičke ivice, centriranje komponenti u okviru forme, odnosno razmicanje komponenti. Postoje dva načina poravnavanja komponenti:

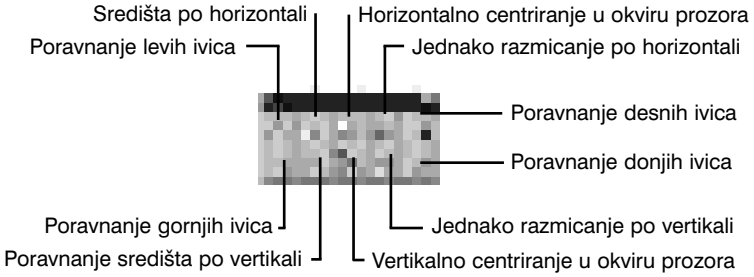
- Korišćenje palete Alignment i okvira za dijalog Alignment.
- Izmena karakteristike `Align`.

Poglavlje koje sledi objašnjava navedene metode.

NAPOMENA Možda ste uočili karakteristiku `Alignment` za neke komponente. Ova karakteristika se odnosi samo na način poravnanja teksta komponente (centriran, levo poravnat, desno poravnat) i nema nikakve veze sa poravnanjem komponente u okviru forme.

Paleta Alignment i okvir za dijalog Alignment

Često je potrebno pomeriti, odnosno promeniti veličinu komponente u odnosu na formu, ili u odnosu na drugu komponentu. Paleta Alignment sadrži nekoliko dugmadi koja Vam pomažu da obavite zadatak. Okvir za dijalog Alignment obavlja iste funkcije kao i paleta Alignment, ali ima različit format. Da biste prikazali paletu Alignment, odaberite opciju View➥Alignment Palette u okviru glavnog menija. Slika 6.10 prikazuje paletu Alignment i daje opis pojedinih dugmadi. Ukoliko zastanete kursorom u trenutku kada prelazite preko dugmeta koje se nalazi na paleti Alignment, pojaviće se oblačić za savet koji opisuje funkciju dugmeta.

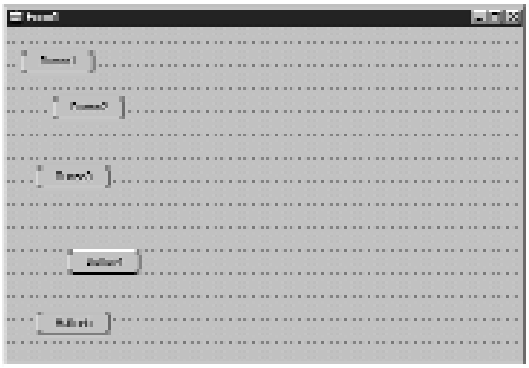


Slika 6.10
Paleta Alignment

SAVET Paleta Alignment Vam može uštedeti prilično posla. Nemojte provoditi suviše vremena pokušavajući da tačno poravnate kontrole. Postavite komponente na formu, a zatim koristite paletu Alignment da biste ih pozicionirali.

Dugme Align Left Edges se koristi za poravnanje komponenti po levoj ivici. Počnite sa praznom formom i pratite sledeće korake:

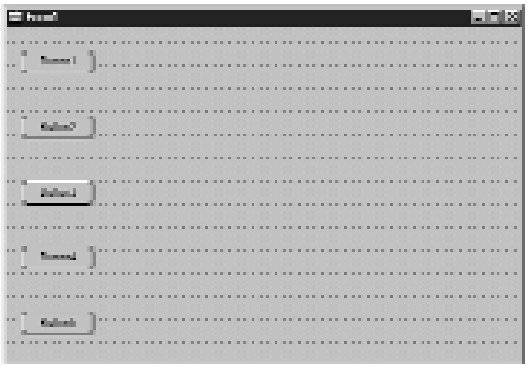
1. Postavite pet komponenti tipa Button vertikalno na formu, bez obzira gde se nalazi njihova leva ivica.
2. Odabertite dugmad prevlačeći pravougaonik za ograničenje. Indikatori izbori pokazuju da su sva dugmad odabrana. Forma bi trebalo da otprilike izgleda kao forma na slici 6.11.



Slika 6.11 Forma sa nasumično postavljenim dugmadima

3. Odaberite opciju View➥Alignment Palette u okviru glavnog menija. Biće prikazana paleta za poravnavanje. Pomerite paletu Alignment, ukoliko Vam zaklanja formu.
4. Kliknite na dugme Align Left Edges u okviru palete Alignment. Sva dugmad su poravnata.

Vidite kako je lako. Dok su Vam sva dugmad odabrana, pogledajmo još jednu opciju za poravnanje. Da biste jednako razmakli dugmad, možete koristiti opciju Space Equally Vertically. Pošto su dugmad već odabrana, treba samo da kliknete na dugme Space Equally Vertically u okviru palete Alignment i da se divite. Dugmad su savršeno razmaknuta. Forma će izgledati slično formi na slici 6.12.



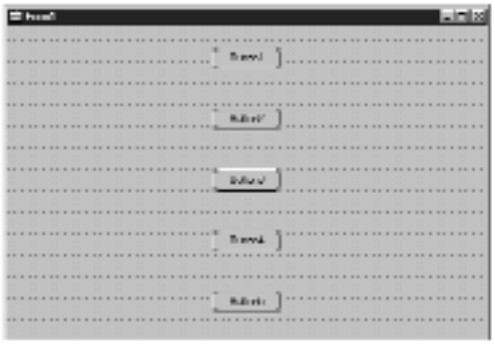
Slika 6.12
Forma sa dugmadima koja su poravnata i jednako razmaknuta

NAPOMENA Opcija za poravnanje Space Equally Vertically jednako razmiče komponente počev od prve komponente u koloni (gornje komponente) do poslednje komponente u koloni (donje komponente). Uverite se da ste ispravno postavili prvu i poslednju komponentu pre nego što odaberete opciju za jednako razmicanje po vertikali. Ovo se odnosi i na opciju za jednako razmicanje po horizontali (Space Equelly Horizontally).

Opcije Center Horizontally in Window (horilzontalno centriranje u okviru prozora) i Center Vertically in Winldow (vertikalno centriranje u okviru prozora), funkcionišu na način na koji asocira njihov naziv. Ove opcije su zgodne za centriranje pojedinačne kontrole u okviru forme, odnosno za centriranje grupe kontrola. Ukoliko su još uvek dugmad na Vašoj formi odabrana, kliknite na opciju Center Horizontally in Window, a zatim na opciju Center Vertically in Window. Dugmad će biti centrirana horizontalno i vertikalno u odnosu na formu.

NAPOMENA Kada odaberete grupu kontrola, a potom kliknete na bilo koje dugme za centiranje, kontrole će se ponašati kao grupa. Ukoliko odaberete bilo koju kontrolu i pokušate da je centrirate horizontalno i vertikalno u okviru forme, sve kontrole će se naći na gomili u sredini forme. Izborom grupe, a zatim centiranjem grupe dobićete rezultat koji ste želeli; kompletna grupa će biti centrirana.

Forma će dobiti izgled kao na slici 6.13.



Slika 6.13
Forma sa centriranim dugmadima

NAPOMENA Opcije za poravnavanje Center Horizontally in Window i Center Vertically in Window se mogu koristiti za poravnavanje komponenti koje se nalaze u okviru drugih komponenti, kao što je to slučaj sa dugmadima na panou. Komponente će biti centrirane horizontalno, odnosno vertikalno na komponenti roditelju, bez obzira da li je komponenta roditelj pano, forma ili neka druga komponenta kontejner.

Opcije Align Tops, Align Bottoms i Align Right Edges rade potpuno isto kao i opcija Align Left Edges, koju ste ranije koristili. Nema svrhe objašnjavati sve mogućnosti koje nastaju njihovim korišćenjem.

SAVET Prva komponenta koju odaberete će predstavljati sidro u slučaju da koristite opciju za poravnavanje ivica. Pogledajte sliku 6.4. Pretpostavimo da ste prvo odabrali komponentu Button3, a zatim koristeći kombinaciju Shift+klik na taster miša odabrali preostalu dugmad. Ukoliko odaberete opciju za poravnavanje levih ivica, komponenta Button3 će ostati na istom mestu, dok će se ostale poravnati levim ivicama sa komponentom Button3, pošto je ova komponenta definisana kao sidro.

Opcije za poravnavanje horizontalnih središta i poravnavanje vertikalnih središta se mogu koristiti za relativno centriranje komponenti (jedna u odnosu na drugu). Primer ćemo ilustrovati koristeći okvire. Otvorite novu formu (odnosno obrišite dugmad sa forme na kojoj radite). Zatim sledite korake:

1. Kliknite na karticu Additonal u okviru palete komponenti, a zatim odaberite komponentu `Shape`. Postavite komponentu blizu gornjeg levog ugla forme.
2. Promenite karakteristiku komponente Shape u `stCircle`.
3. Promenite karakteristike `Width` i `Height` u `150`.
4. Dva puta kliknite na karakteristiku `Brush` i promenite karakterisktiku `Collor` u `clBlack`.
5. Postavite još jednu komponentu `Shape` na formu.
6. Promenite karakteristiku druge komponente u `stCircle`. Sada imate dva kruga različiitih veličina na ekranu - beli krug i crni krug.
7. Kliknite na crni krug. Držite pritisnut taster Shift, a zatim kliknite na beli krug. Obe komponente su odabrane.
8. Odaberite opciju View➥Alignment Palette u okviru glavnog menija, ukoliko je to potrebno (možda je već prikazana). Pomerite paletu za poravnavanje tako da vidite oba kruga na formi. Posmatrajte krugove dok izvodite poslednja dva koraka.
9. Kliknite na dugme Align Vertical Centers u okviru palete za poravnavanje. Središta krugova su se poravnala po vertikali.
10. Kliknite na dugme Align Horizontal Centers u okviru palete za poravnavanje. Središta krugova su se poravnala po horizontali. Čestitam - napravili ste gumu!

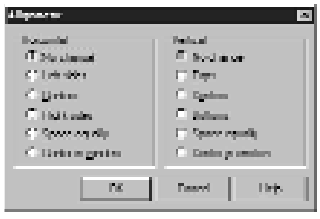
Da li ste uočili efekte prilikom izvršavanja poslednja dva koraka? Uočite da, pošto ste odabrali prvo crni krug, beli krug se pomerio kada ste kliknuli na dugme za poravnavanje, dok je crni krug ostao na svom mestu (pošto je komponenta sidro). Ove opcije za poravnavanje možete koristiti kako bi centrirali i postavili jednu na drugu proizvoljan broj kontrola. Ove opcije za poravnavanje nemaju efekta kada se koriste na jednoj jedinoj kontroli.

Kao što je to slučaj sa paletom komponenti, paleta za poravnavanje ima meni sadržaja. Postavite kursor miša preko palete za poravnavanje, a zatim kliknite na drugi taster miša. Na ekranu će biti prikazan meni sadržaja. Tabela 6.2 prikazuje spisak opcija menija sadržaja palete za poravnavanje i njihova objašnjenja.

Tabela 6.2: Opcije menija sadržaja palete za poravnavanje

Opcija menija	Opis
Stay on Top	Postavlja paletu za poravnavanje na vrh svih prozora. Ovo je korisno ukoliko često prelazite sa dizajnera forme na editor koda i obrnuto. Pošto je paleta za poravnavanje mali prozor, lako se može zagubiti.
Show Hints	Uključuje, odnosno isključuje savete (hints) za dugmad palete za porav navanje.
Hide	Sakriva paletu za poravnavanje. (Da bi sakrili paletu za poravnavanje, možete koristiti dugme za zatvaranje u okviru naslovne trake.) Da biste ponovo prikazali paletu za poravnavanje, odaberite opciju View➥Alignment Palette u okviru glavnog menija.
Help	Prikazuje pomoć sa otvorenom stranom na paleti za poravnavanje.

Okvir za dijalog Alignment izvršava potpuno iste aktivnosti kao i paleta za poravnavanje. Da biste odabrali okvir za dijalog Alignment odaberite opciju Edit➥Align u okviru glavnog menija, odnosno opciju Align u okviru menija sadržaja dizajnera forme. Slika 6.14 prikazuje okvir za dijalog Alignment.



Slika 6.14
Okvir za dijalog Alignment

U većini slučajeva, korišćenje palete za poravnavanje je jednostavno, ali sigurno ćete koristiti okvir za dijalog Alignment, ukoliko Vam se svidi.

Korišćenje karakteristike Align

Još jedan tip poravnanja se može postići korišćenjem karakteristike `Align`. Ova karakteristika kontrolliše način poravnavanja komponente sa roditeljem. Moguće vrednosti karakteristike `Align` i opise možete pronaći u tabeli 6.3.

Tabela 6.3: Moguće vrednosti karakteristike `Align`

Vrednost	Opis
`alBottom`	Komponente su poravnate u dnu prozora roditelja. Statusna traka je primer komponente poravnate na dnu glavne forme.
`alClient`	Komponenta je proširena tako da popunjava klijent oblast prozora roditelja. Ukoliko neka druga komponenta zauzima deo klijent oblasti, tekuća komponennta popunjava ostatki klijent oblasti. Primeri kompo nenti: `Memo`, `Image` i `RichEdit`.
`alLeft`	Komponenta je poravnata duž leve ivice prozora roditelja. Vertikalna traka sa alatima je primer komponente koja je levo poravnata.
`alNone`	Komponenta je postavljena i dizajnirana bez posebne veze sa prozorom roditeljem. Ovo je generička vrednost za većinu komponenti.
`alRight`	Komponenta je poravnata duž desne ivice prozora roditelja.
`alTop`	Komponenta je poravnata duž gornje ivice prozora roditelja. Traka sa alatima predstavlja primer ovog tipa poravnavanja komponente.

Primer pomaže da se objasni karakteristuka `Align`. Počnite sa praznom formom, a zatim izvršite sledeće korake:

1. Kliknite na jezičak Standard u okviru palete komponenti, a zatim odaberite komponentu `Panel`. Postavite pano bilo gde na formu.
2. Pronađite karakteristiku `Align` u prozoru Object Inspector (nalazi se na vrhu liste). Uočite da je ova karakteristika podešena na `alNone`. Promenite karakteristiku `Align` u `alTop`. Pano će biti poravnat duž gornje ivice forme i proširen tako da pokriva širinu forme.
3. Pokušajte da pomerite pano u sredinu forme. Pano će se ponovo vratiti na vrh.
4. Pokušajte da skratite širinu panoa. Uočite da pano zadržava svoju širinu.
5. Pokušajte da promenite visinu panoa. Uočite da visina panoa može biti promenjena (širina ne može).
6. Promenite karakteristiku `Align` u `alBottom`. Sada se pano zalepio na dno forme.
7. Promenite karakteristiku `Align` u `alRight`, a zatim u `alLeft`. Širina je ostala ista kao što je bila visina ranije. Kao efekat dobili ste rotaciju panoa. Još

jednom pokušaj da se pano pomeri, ili promeni njegova veličina po vertikali ne uspeva.

8. Promenite karakteristiku `Align` u `alClient`. Pano se proširuje i popunjava kompletnu klijent oblast. Veličina panoa se ne može menjati.

9. Promenite karakteristiku `Align` u `alNone`. Pano ponovo može biti pomeran, odnosno može mu se menjati veličina.

Kao što ste mogli da vidite, promena karakteristike `Align` na bilo koju vrednost osim `alNone`, efektivno lepi pano na jednu ivicu forme. U slučaju vrednosti `alClient`, pano se lepi na sve četiri ivice.

Postavljanje redosleda tabulatora (tab order)

NOVI TERMIN Redosled tabulatora *(tab order)* ukazuje na redosled po kom će komponente primati ulazni fokus, kada korisnik pritisne taster Tab na tastaturi.

Delphijeve forme automatski podržavaju navigaciju između komponenti korišćenjem tastera Tab. Ovo znači da se možete pomerati unapred od komponente do komponente korišćenjem tastera Tab, a unazad korišćenjem kombinacije tastera Shift+Tab.

NAPOMENA Postoje dva tipa vizuelnih komponenti. Prozorske komponente su komponente koje prihvataju fokus tastature, što znači da se na komponentu može kliknuti mišem, odnosno može joj se dodeliti redosled tabulatora. Kada komponenta ima fokus tastature, ili prikazuje poseban kusor (kao što je kod edit kontrole kursor oblika koji se naziva i-zrak), odnosno ima pravougaonik fokusa nacrtan negde na komponenti. Prozorske komponente uključuju: `Edit`, `Memo`, `ListBox`, `ComboBox` i `Button` komponente, kao i mnogo ostalih komponenti.

Neprozorske komponente su komponente koje ne prihvataju fokus tastature. To su komponente poput: `Image`, `SpeedButton`, `Label`, `Shape` i mnogih drugih neprozorskih komponenata.

Redosled tabulatora se odnosi samo na prozorske komponente. Neprozorske komponente su isključene iz redosleda tabulatora.

Redosled tabulatora je inicijalno baziran na redosledu postavljanja komponenti na formu prilikom dizajniranja forme. Redosled tabulatora možete menjati izmenom karakteristike `TabOrder` za svaku kontrolu u prozoru Object Inspector, ali ovaj metod je nezgodan pošto morate da promenite karakteristiku svake kontrole pojedinačno. Okvir za dijalog Edit Tab Order pruža jednostavniji način podešavanja redosleda tabulatora (pogledajte sliku 6.15).

Slika 6.15
Okvir za dijalog Edit Tab Order

Okvir za dijalog Edit Tab Order se poziva izborom opcije Edit➥Tab Order u okviru glavnog menija. Dijalog boks prikazuje sve prozorske komponente koje se nalaze na formi; neprozorske komponente nisu prikazane. Da bi promenili redosled tabulatora, kliknite na naziv komponente kojoj želite da promenite redosled tabulatora, a zatim kliknite na dugmad za gore i dole ukoliko je potrebno. Isto tako možete prevući komponentu na novu poziciju redosleda tabulatora. Nakon što ste definisali redosled tabulatora na način koji Vam odgovoara, kliknite na dugme OK, kako bi zapisali definisani redosled. Potvrdu novog redosleda možete proveriti pregledom karakteristike `TabOrder` svake kontrole.

NAPOMENA Redosled tabulatora počinje od 0. Prva komponenta u okviru redosleda tabulatora je 0, druga je 1, itd.

Kreiranje primera aplikacije

Da bi ilustrovali kako različite komponente rade zajedno, kreirajmo prototip aplikacije koja zamenjuje standardni tekst editor Windows-a, Notepad.

NAPOMENA Kreiranje tekst editora verovatno ne zvuči glamurozno. Da budem iskren i nije glamurozno. Ono čemu Vas može naučiti kreiranje tekst editora je susret sa brojnim problemima iz stvarnog života, koji mogu nastati prilikom programiranja u Delphiju. Možda nije glamurozno, ali će Vas naučiti više nego da kreirate Jednostavan Program v.1.0.

NOVI TERMIN *Prototip* je aplikacija koja ima izgled radne aplikacije, ali nedostaje joj puna funkcionalnost, pošto je obično u ranoj fazi izrade.

NAPOMENA Delphi je izuzetan alat za brzu izradu prototipa aplikacija. Možete imati glavne prozore i okvire za dijalog urađene i prikazane za mnogo manje vremena nego što bi Vam trebalo kada biste koristili tradicionalne programske alate za programiranje na Windows-ima. Ovo, naravno, ne znači da se Windows koristi samo za izradu prototipa. Delphi je sposoban da upravlja svim potrebama Vašeg 32-bitnog Windows programa.

Korak 1: Početak rada na novoj aplikaciji.

1. Odaberite opciju File➥New Application u okviru glavnog menija. Ukoliko budete upitani da li želite da snimite tekući projekt, odgovorite sa ne.
2. Forma je odabrana, pa promenite karakteristiku `Name` u `MainForm`.
3. Promenite karakteristiku `Caption` u `ScratchPad 1.0`.
4. Odaberite opciju Project➥Options u okviru glavnog menija. Kliknite na jezičak kartice Application i unesite `ScratchPad 1.0` za naslov aplikacije. Kliknite na dugme OK i zatvorite okvir za dijalog Project Options.

Korak 2:Dodavanje trake sa alatima.

Većina Windows aplikacija danas ima traku sa alatima. Kreiranje trake sa alatima zahteva nekoliko koraka. U ovom delu Vam neću objasniti sve što biste trebali da znate o trakama za alate koje se nalaze u Delphiju, pošto sam objašnjenje sačuvao za lekciju dana 13, "Iza osnova Delphija". Traku sa alatima u program ScratchPad ćete dodati u ovoj lekciji. Ono što možete da uradite u ovom trenutku je dodavanje trake sa alatima, koja se može iskoristiti za rezervisanje mesta za pravu traku sa alatima, koju ćete kreirati kasnije. Pratite korake:

1. Kliknite na jezičak Win32 u okviru palete komponmenti i odaberite komponentu ToolBar (treća sa desne strane).
2. Kliknite bilo gde na formu, da biste dodali traku sa alatima. Uočite da se traka sa alatima automatski poravnava na gornju ivicu forme.
3. Kliknite desnim tasterom miša na traku sa alatima i odaberite opciju New Button. Pojaviće se dugme na traci sa alatima.
4. Ponovite korak tri, da biste dodali drugo dugme.

To je sve što ćete raditi sa trakom sa alatima, za sada. Kao što sam rekao pravu traku sa alatima ćete praviti u lekciji dana 13.

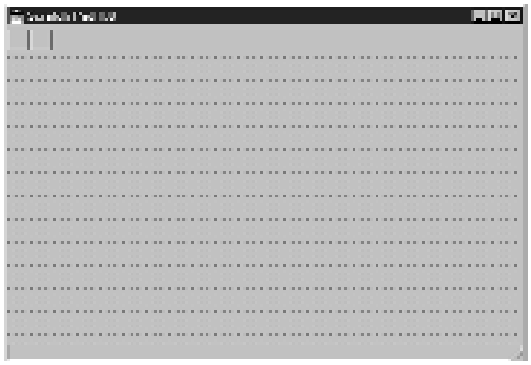
Korak 3: Dodavanje statusne linije.

Za sada se sve dobro odvija. Windows Notepad nema statusnu traku (a ni traku sa alatima, kad već spominjem trake), ali je možete postaviti u Vašu aplikaciju prateći sledeće korake:

1. Kliknite na jezičak Win32 u okviru palete komponenti i odaberite komponentu `StatusBar`.
2. Kliknite bilo gde na formu. Statusna traka se automatski postavlja na donju ivicu forme. Statusna traka ima generičku vrednost karakteristike `Align` definisanu kao `alBottom`.

3. Promenite karakteristiku `Name` u `StatusBar`.

Forma će dobiti izgled forme na slici 6.16.



Slika 6.16
Forma ScratchPad nakon prva tri koraka

Korak 4: Dodavanje memo komponente.

Potrebna Vam je komponenta u koju biste kucali tekst, pa možete koristiti memo komponentu, da biste dodali ovu mogućnost (verovali ili ne, skoro ste završili Vaš prototip):

1. Kliknite na jezičak Standard u okviru palete komponenti i odaberite komponentu `Memo`. Postavite komponentu bilo gde na formu.
2. Promenite karakteristiku `Name` u `Memo`.
3. Dva puta kliknite mišem na kolonu Value pored karakteristike `Lines`. Biće prikazan String List Editor. Obrišite reč `Memo`, a zatim kliknite na dugme OK.
4. Promenite karakteristiku `Scrollbar` u `ssVertical`. (Inicijalno, želite samo vertikalnu traku za pomeranje u okviru memo polja.)
5. Promenite karakteristiku `Name` karakteristike `Font` u `Fixedsys`. (Pošto je ova aplikacija plagijat Notepad-a, koristićete sistemski font.)
6. Promenite karakteristiku `Align` u `alClient`. Memo će se raširiti i ispuniti klijent oblast između trake sa alatima i statusne trake.

Zavalite se u stolicu i divite se vašem radu. Ovaj početak izgleda kao prava aplikacija! Ukoliko forma izgleda kao da je suviše velika, ili suviše mala, promenite joj veličinu hvatajući je za donji desni ugao. Pošto je program Vaš, učinite da izgleda onako kako želite.

SAVET Pritiskom na taster Esc odabraćete pretka kontrole koja je trenutno odabrana. Na primer, klijent oblast naše forme je pokrivena komponentama, što onemogućava izbor same forme. Da bi forma postala aktivna u prozoru Object Inspector, odaberite memo komponentu , a zatim

pritisnite taster Esc. Formu možete odabrati i na drugi način, korišćenjem kombo okvira za izbor komponente (Component Selector) u prozoru Object Inspector.

Uočite da se veličine svih kontrola automatski menjaju, kako bi održale vezu sa prozorom roditeljem - u ovom slučaju formom. Ovo je jedna od glavnih prednosti karakteristike `Align`. U ovom trenutku forma će izgledati slično formi na slici 6.17.

Pokretanje programa

Sada možete kliknuti na dugme Run, kako biste pokrenuli program. Tekst možete kucati u klijent oblasti prozora, a takođe možete pritiskati dugmad na traci za alate (iako dugmad nemaju funkciju u ovom trenutku). Ne zaboravite da je ovo još uvek prototip i da se trenutno može koristiti samo za demonstraciju rada. Sa programiranjem ćete nastaviti na kraju današnje lekcije.

Slika 6.17
Završen prototip

Bilo bi dobro da snimite projekt pošto ćete ga koristiti kasnije u ovoj lekciji. Odaberite opciju File➥Save All u okviru glavnog menija. Snimite izvorni kod iz junita glavne forme kao `SPMain`, a projekt kao `Scratch`.

Molim Vas pokažite mi menije!

Meniji su značajan deo u većini Windows aplikacija. Iako ponekad Windows programi nemaju menije, velika većina programa ih koristi. Delphi pojednostavljuje kreiranje menija opcijom za dizajniranje menija (Menu Designer). Dizajner menija ima sledeće mogućnosti:

- Može kreirati glavne menije i slobodne menije (menije sadržaja).
- Pruža trenutni pristup editoru koda kako bi mogli da kontrolišete događaj `OnClick` za svaku opciju menija.
- Može ubacivati menije koristeći šablone iz resursnih datoteka.

4 Može snimiti menije koje je definisao korisnik kao šablone (templates).

Svim komandama dizajnera menija možete pristupiti preko menija sadržaja dizajnera menija, odnosno koristeći prozor Object Inspector. Slika 6.18 prikazuje meni sadržaja dizajnera menija.

U većini slučajeva, opcije menija su dovoljno jasne, pa ih neću posebno objašnjavati. Bolje je da počnete da ih koristite; tako ćete ih brže shvatiti. Za početak, dodajmo glavni meni u aplikaciju `ScratchPad` koju ste kreirali u prethodnom delu lekcije. Zatim ćete kreirati meni sadržaja.

Slika 6.18
Meni sadržaja dizajnera menija

Kreiranje glavnog menija (main menu)

Dizajner menija Vam omogućava da brzo kreirate bilo kakav meni. Struktura menija za glavni meni se sastoji od komponente `MainMenu`, koja je predstavljena VCL klasom `TMainMenu`. Svaka opcija menija je komponenta `MenuItem` koja je enkapsulirana u klasi `TMenuItem`. Nemojte se brinuti o pojedinostima rada ovih klasa, odnosno kako su ove klase povezane, pošto dizajner menija omogućava da se bilo kakav meni jednostavno kreira. Koristeći ovaj kratak pregled, dodajmo glavni meni u aplikaciju `ScratchPad`.

Dodavanje glavnog menija na formu

Prva stvar koju treba da uradite je dodavanje meni komponente na formu.

NAPOMENA Do sada ste već stekli pristojno iskustvo u radu sa Delphijem. Od ovog trenutka skratiću objašnjenja nekih koraka koje bi trebalo da preduzmete kako bi izvršili određenu operaciju. Na primer, odsada ću Vam samo saopštiti: "Postavite komponentu `MainMenu` na formu", umesto: "Kliknite na jezičak Standard u okviru palete komponenti. Kliknite na dugme `MainMenu`, a zatim kliknite na formu kako bi postavili komponentu." Ne brinite, još uvek ću Vam objasniti dovoljno detaljno kada budemo radili nove operacije.

1. Otvorite projekt `ScratchPad` koji ste kreirali u prethodnom delu lekcije.

2. Postavite komponentu `MainMenu` na formu i promenite karakteristiku `Name` u `MainMenu`. Uočite da komponenta `MainMenu` ima tek nekoliko karakteristika i da nema događaja. Kompletan rad sa menijem se obavlja preko opcija menija.
3. Dva puta kliknite na ikonu `MainMenu`. Na ekranu će se prikazati dizajner menija.

Dizajner menija izgleda kao prazna forma bez tačaka mreže. Ukoliko želite, možete menjati veličinu dizajnera menija. Veličina prozora je bitna samo za Vašu udobnost; nema uticaja na funkcije menija u toku rada programa. Od ovog trenutka, dizajner menija Vas čeka da počnete sa kreiranjem menija. Nakon što ste kreirali svoj prvi meni, otkrićete da je kreiranje menija jednostavno i intuitivno.

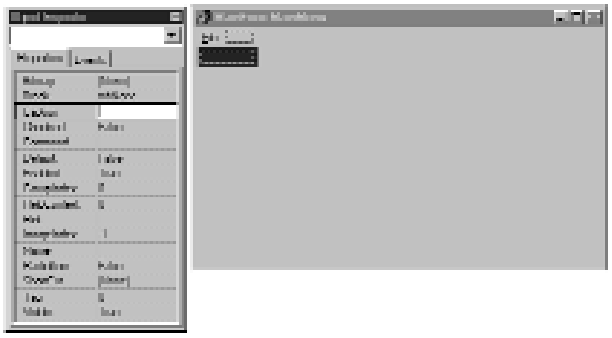
Ručno kreiranje menija

Iako postoji jednostavniji način kreiranja menija File, Vaš prvi meni ćete kreirati ručno. Dizajner menija uvek ima praznu opciju koja se ponaša kao mesto za postavljanje nove opcije menija koju ćete kreirati. Prilikom prvog startovanja dizajnera menija, prikazuje se prazna opcija.

1. Promenite karakteristiku `Name` u `FileMenu`.
2. Kliknite na karakteristiku `Caption` u okviru prozora Object Inspector, upišite `&File` i pritisnite taster Enter.

Ampersand (&) se koristi za kreiranje karaktera koji će biti podvučen u okviru opcije menija. Podvučeni karakter je akcelerator koji korisnik može pritisnuti u kombinaciji sa tasterom Alt, kako bi birao opcije menija koristeći tastaturu. Ampersand možete staviti bilo gde u okviru teksta opcije menija. Na primer, uobičajen tekst string za opciju Exit (izlaz iz programa) bi bio: `E&xit`, pa je u ovom slučaju, taster x akcelerator. Sve što treba da učinite je da postavite ampersand pre odgovarajućeg slova, a Windows će to iskoristiti.

Od ovog trenutka dogodiće se nekoliko stvari. Prvo, biće prikazan meni File u okviru dizajnera menija. Meni će se takođe pojaviti i na glavnoj formi koja se nalazi iza dizajnera menija. Naredna stvar će biti pojavljivanje praznog mesta ispod File menija koji ste upravo kreirali (treba da kliknete na File meni u okviru dizajnera menija da biste videli prazno mesto). Kao dodatak, novo prazno mesto za meni je kreirano sa desne strane menija File. Object Inspector prikazuje praznu komponentu `MenuItem`, koja čeka da unesete vrednosti karakteristika `Caption` i `Name`. Slika 6.19 prikazuje dizajner menija u ovom trenutku.



Slika 6.19
Dizajner menija i Object Inspector nakon kreiranja menija File

Nastavimo sa kreiranjem menija:

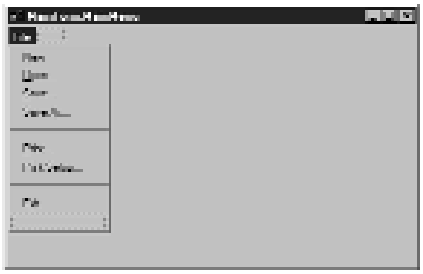
1. Promenite karakterisktiku `Name` nove opcije u `FileNew`.
2. Promenite karakteristiku `Caption` u `&New`, a zaktim pritisnite taster Enter. Ponovo se pojavila prazna opcija u dizajneru menija.
3. Ponovite korake jedan i dva da biste kreirali opcije menija: Open, Save i Save As. Ukoliko Vam treba pomoć oko postavljanja znaka &, pogledajte sliku 6.20. Nemojte brinuti ukoliko niste dobro uradili neki od koraka. Uvek se možete vratiti i popraviti grešku.

SAVET Standardizujte Vaše menije koliko god je to moguće. Budite sigurni da Vaši akceleratori (podvučeni karakteri) izgledaju potpunio isto kao i u drugim Windows programima. Takođe zapamtite da tri tačkice koje slede iza teksta opcije menija predstavljaju vizuelni znak korisniku da opcija menija poziva okvir za dijalog.

U ovom trenutku Vam je potrebna linija za odvajanje opcija menija (separator).

NOVI TERMIN Separator (linija za odvajanje opcija menija) je horizontalna linija u okviru menija koja razdvaja grupe opcija.

Dodavanje separatora je jednostavno u Delphijevom dizajneru menija. Sve što treba da uradite je da postavite znak minus (-) u karakteristiku `Caption`. Odaberite praznu opciju menija ispod opcije Save As, upišite u karakteristiku `Caption` znak minus, a zatim pritisnite taster Enter. Separator se nalazi na meniju. Nastavite sa dodavanjem opcija menija, sve dok ne dobijete meni koji je prikazan na slici 6.20. Ukoliko je potrebno da izmenite opciju menija, kliknite na željenu opciju i ukoliko je to potrebno, promenite karakteristike u prozoru Object Inspector.



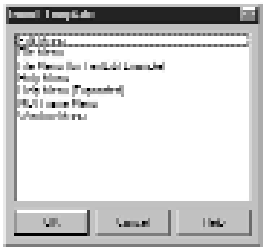
Slika 6.20
Dizajner menija sa završenim menijem File

NAPOMENA Dizajner menija uvek obezbeđuje praznu opciju menija na dnu svakog podmenija, odnosno na desnoj strani trake menija. Ove prazne opcije nećete moći obrisati, mada za to nema ni potrebe pošto se prazne opcije pojavljuju samo u dizajneru menija, a ne i u samom meniju u toku rada programa.

Nakon što ste završili sa kreiranjem menija File, potrebno je da kreirate meni Edit i meni Help.

Ubacivanje menija korišćenjem obrasca (template)

Došlo je vreme da primenite lakši pristup. Prvo kliknite na praznu opciju menija na desnoj strani menija File. Sada kliknite na drugi taster miša i odaberite opciju Insert From Template (ubaci iz obrasca). Okvir za dijalog Insert Template će se pojaviti kao što je prikazano na slici 6.21.



Slika 6.21
Okvir za dijalog Insert Template

Ovaj okvir za dijalog prikazuje listu obrazaca sa koje možete da birate obrazac koji Vam odgovara. Možete koristiti unapred definisane obrasce, odnosno kreirati sopstvene. U ovom slučaju zainteresovani ste samo da dodate meni Edit, stoga odaberite Edit Menu, a zatim kliknite na dugme OK. Kompletan meni Edit će u trenutku biti unet u dizajner menija. Ustvari, meni je malo opširniji. Ovim ćemo se pozabaviti nešto kasnije.

Iskoristite priliku i dodajte meni Help. Kliknite na praznu opciju desno od menija Edit. Odaberite ponovo opciju Insert From Template i ovaj put ubacite meni Help. (Nemojte odabrati meni Expanded Help.) U sledećem poglavlju ćete doterati oba menija. Uočite da se glavna forma osvežava i da sad prikazuje nove opcije menija.

NAPOMENA Možete ubaciti obrasce d kreirate slobodne menije, kao što kreirate stavke glavnog menija.

Da, ubacivanje obrazaca je stvarno jednostavno. Nakon određenog vremena provedenog uz Delphi, verovatno ćete kreirati svoje obrasce za brzo i jednostavno kreiranje menija. Još uvek treba da promenite karakteristiku Name u naziv koji ima smisla, ali i to je mnogo jednostavnije od kreiranja menija od samog početka.

NAPOMENA Opcija Insert From Resource (ubaci iz resursa) radi potpuno isto kao opcija Insert From Template, izuzev što očekuje resursnu skript datoteku (resursna skript datoteka ima nastavask RC), koja sadrži odgovarajuću definiciju menija. Meni resurs mora koristiti resursnu sintaksu menija begin/end i ne dozvoljava korišćenje zagrada. Kao primer, slede nepravilni meni resursi:

```
MENU_1 MENU
{
 POPUP "File"
 {
  MENUITEM "Open", 100
  MENUITEM "About", 101
 }
}
```

Meni resursi koji slede su dozvoljeni:

```
MENU_1 MENU
BEGIN
 POPUP "File"
 BEGIN
  MENUITEM "Open", 100
  MENUITEM "About", 101
 END
END
```

Prethodne definicije važe samo za menije koje se unose opcijom Insert From Resource, a ne i za meni resurse uopšte.

Brisanje opcija menija

Proces kreiranja Windows aplikacije je stvar koja živi i diše. Teško da ćete sve imati ispravno podešeno u prvom trenutku. Korisnici će zahtevati nove mogućnosti, šef će doći sa novim izmenama, a neke mogućnosti će čak biti izbačene. Često ćete biti u situaciji da menjate menije aplikacija kada dođe do ovih promena. Na primer, meni Edit koji je u prethodnom poglavlju ubačen, pomalo premašuje Vaše potrebe; postoji nekoliko opcija koje Vam još nisu potrebne. Nema problema - uvek ih možete obrisati:

1. Kliknite na meni Edit.
2. Kliknite na opciju pod nazivom Repeat <command>.

3. Pritisnite taster Delete na tastaturi, odnosno odaberite opciju Delete iz menija sadržaja dizajnera menija, kako biste obrisali opciju. Obrisana opcija nestaje, a preostale opcije se pomeraju na gore.
4. Obrišite takođe i opciju Paste Special.

Ovo je bilo jednostavno! Još niste završili sa menijem Edit, ali pre nego što krenemo dalje, želeo bih da Vam spomenem jednu veoma korisnu mogućnost dizajnera menija. Verovatno ste upoznati sa korišćenjem kombinacija Shift+klik na taster miša i Ctrl+klik na taster miša, kada birate opcije u drugim Windows programima. Ove tehnike se mogu koristiti u programu Windows Explorer za, recimo, odabiranje datoteka. Dizajner menija podržava kombinacije Shift+klik na taster miša i Ctrl+klik na taster miša, sa jednim izuzetkom - možete ih koristiti da odaberete više opcija menija, ali ne i da uklonite izbor. Kao i uvek, vežba će to ilustrovati bolje od objašnjenja:

1. Meni Edit bi još uvek trebao da bude prikazan. Ukoliko nije, kliknite na opciju menija Edit, da biste ga prikazali.
2. Kliknite na opciju menija pod nazivom Goto.
3. Držite pritisnut taster Shift, a zatim kliknite na opciju menija pod nazivom Object. Sve opcije menija između ova dva elementa su odabrane.
4. Pritisnite taster Delete na tastaturi, kako biste obrisali sve opcije istovremeno.
5. Pređite na meni Help i obrišite dve opcije u sredini. Ostaće samo opcije Contents i About.

Kao što ste mogli da vidite kombinacija Shift+klik se može koristiti za brzo brisanje neželjenih opcija menija. Sada imate skraćene menije, što je upravo ono što ste želeli da se pojavi u aplikaciji `ScratchPad`.

Ubacivanje novih opcija

Ubacivanje novih opcija menija je veoma jednostavno. Kliknite na opciju menija iznad koje želite da ubacite novu opciju, a zatim pritisnite taster Insert na tastaturi (odnosno odaberite opciju Insert u okviru menija sadržaja dizajnera menija). Ubačena je prazna stavka menija, pa možete da izmenite karakteristike `Name` i `Caption`, kao što ste to radili ranije. Hajde da unesemo opciju u meni Edit:

1. Kliknite na opciju Edit, da biste videli kompletan meni.
2. Kliknite na opciju menija Find.
3. Pritisnite taster Insert na tastaturi. Nova stavka menija se pojavila, dok su ostale stavke ispod nje pomerene na dole.
4. Promenite karakteristiku `Name` u `EditSelectAll`, a karakteristiku `Caption` u `Select &All`.

5. Kliknite na prazno mesto u dnu menija Edit. Dodajte separator menija (zapamtite, samo ubacite znak minus u karakteristiku `Caption`).
6. Kliknite na prazno mesto u okviru menija i ponovo dodajte novu stavku. Promenite karakteristiku `Name` u `EditWordWrap`, a karakteristiku `Caption` u `&Word Wrap`.

Pomeranje opcija menija

Ukoliko je potrebno, lako možete pomeriti opcije menija. Možete ih pomerati gore, ili dole u okviru podmenija u kom se nalaze, odnosno možete ih pomerati sa jednog na drugi podmeni. Postoje dva načina za pomeranje opcija menija. Prvi je korišćenjem opcija za isecanje i lepljenje. Isecanje i lepljenje radi kao što ste očekivali, pa nema potrebe da detaljnije objašnjavam.

Drugi način za pomeranje opcija menija je prevlačenje opcija na novu poziciju i puštanje. Pokušajmo. Potrebno je da opciju menija Select All pomerite ispod opcije Undo. Ovo je dovoljno jednostavno da bi se uradilo, stoga krenimo:

1. Kliknite na opciju Edit, kako bi bio prikazan meni Edit.
2. Kliknite na opciju Select All, a zatim je povlačite na gore sve dok ne bude osvetljen separator ispred opcije Undo.
3. Otpustite taster miša i opcija menija je pomerena.

Veoma jednostavno, zar ne? Da, za to služi Delphi.

Beč izmena karakteristika

Ponekad ćete poželeti da izmenite karakteristike nekoliko opcija menija istovremeno - ovo se zove beč izmena. Na primer, sigurno imate nekoliko opcija menija u aplikaciji `ScratchPad` koje još uvek niste implementirali. Još niste spremni da implementirate podršku za štampanje, odnosno da implementirate sistem za pomoć. Zbog toga treba da deaktivirate ove opcije menija:

1. Odaberite opciju Help➥Contents u okviru dizajnera menija.
2. Promenite karakterisktiku `Enabled` u `False`. Opcija menija je dobila sivu boju.
3. Kliknite na meni File.
4. Kliknite na opciju menija Print, držite pritisnut taster Shift, a zatim kliknite na opciju menija Print Setup. Obe opcije su odabrane.
5. U prozoru Object Inspector promenite karakteristiku `Enabled` u `False`. Obe opcije menija su deaktivirane.
6. Ponovite korake četiri i pet, kako bi deaktivirali opcije Find i Replace u okviru menija Edit.

Možete istovremeno izmeniti grupu opcija menija koristeći ovaj metod. Jednostavno odaberite opciju koju želite da izaberete, a zatim izmenite željenu karakteristiku. Sve opcije menija koje su odabrane će dobiti novu vrednost karakteristike.

Dodavanje bitmapa u opcije menija

Možete jednostavno dodavati bitmape u Vaše menije. Prvo kliknite na opciju menija za koju želite da dodate bitmapu. Zatim dvostrukim klikom miša na kolonu Value karakteristike `Bitmap` u okviru prozora Object Inspector, otvorite Picture Editor. Zatim odaberite bitmapu za opciju menija. Bitmapa može biti slika, odnosno grupa bitmapa (više slika). Ukoliko koristite više slika, treba da podesite karakteristiku `ImageIndex`, da biste upisali broj indeksa slike koju želite da prikažete u okviru opcije menija; broj indeksa predstavlja broj slike u okviru grupe slika.

Bitmape opcija menija se ne prikazuju u okviru dizajnera menija, niti u formi u toku dizajniranja. Da biste videli bitmape morate pokrenuti program.

Kreiranje podmenija

Ne postoji ništa posebno u kreiranju podmenija. *Podmeniji* su opcije menija koje, kada se na njih klikne, prikažu dodatne opcije. Podmeniji su označeni strelicom na desno pored teksta opcije menija. Podmenije možete kreirati birajući opciju CreateSubmenu u okviru menija sadržaja dizajnera menija, odnosno pritiskom na taster Ctrl i pritiskom na desni kursorski taster. Podmeni možete kreirati ukoliko dodate obrazac menija.

Dodavanje prečica (shortcuts)

Prečice za tastaturu možete jednostavno dodati opciji menija promenom karakteristike `ShortCut` u okviru prozora Object Inspector. Meni Edit koji ste prethodno ubacili ima već ugrađene prečice. Na primer, uobičajena prečica za isecanje (cut) je Ctrl+X. Ukoliko pogledate u meni Edit možete videti Ctrl+X ispisano pored opcije Cut (prečica je već dodeljena pošto ste učitali meni iz obrasca).

Kliknite na opciju menija Cut i uočićete da karakteristika `ShortCut` pokazuje Ctrl+X. Kliknite na kolonu Value pored karakteristike `ShortCut`, na desnoj strani kolone Value videćete dugme sa strelicom na dole. Kliknite na dugme da bi se prikazao spisak dostupnih prečica. Spisak koji možete videti sadrži skoro sve prečice tastature koje su Vam potrebne. Da biste podesili prečicu tastature za opciju menija, jednostavno odaberite prečicu sa liste.

Standardna prečica za opciju Select All je Ctrl+A, zato dodajte ovu prečicu za opciju menija Select All:

1. Odaberite opciju Edit➥Select All iz Vašeg menija u okviru dizajnera menija.
2. Kliknite na karakterisktiku `ShortCut` u prozoru Object Inspector.

3. Odaberite Ctrl+A sa spiska dostupnih prečica. Sada opcija Select All prikazuje Ctrl+A pored naziva opcije.

To bi bilo sve što treba da uradite; Delphi brine o svemu ostalom. Prečice funkcionišu bez pisanja ijedne linije koda.

Završni izgled

Završimo naš meni. Prvo ćete postaviti opciju menija Word Wrap uključenu. Ova opcija menija se koristi za prebacivanje reči u drugi red. Kada je prebacivanje reči uključeno (word wrapping), što predstavlja generičkui vrednost, opcija menija će biti označena. Kada se prebacivanje reči isključi, opcija menija neće biti označena. Kliknite na opciju menija Word Wrap i promenite karakteristiku `Checked` u `True`. Oznaka za potvrdu (check mark) će se pojaviti, kako bi mogli da vidite da je prebacivanje reči uključeno.

Još jedna promena Vam je ostala; promenite karakteristiku `Name` na svim opcijama menija koje ste bacili koristeći šablon. Ove opcije imaju dodeljene generičke nazive koje treba da promenite u nazive sa više smisla. Pratite sledeće korake:

1. Kliknite na opciju menija Edit➥Undo. Promenite karakteristiku `Name` od `Undo1` u `EditUndo`. Uočite da ste naziv opcije dodali reč Edit, na početak naziva i uklonili broj `1` sa kraja naziva.
2. Možete koristiti konvenciju za dodelu naziva kakvu želite, ali neka nazivi budu konzistentni. Ponovite proces za opcije Cut, Copy, Paste, Find i Replace.
3. Sada se prebacite na meni Help i izmenite karakteristiku `Name` opcije Contents u `HelpContents`, a u stavki menija About u `HelpAbout`.

Toliko o završavanju Vašeg menija. Prođite kroz opcije menija, kako biste ih ponovo proverili. Ukoliko pronađete grešku, ispravite je. Kada budete zadovoljni i budete smatrali da je sve u meniju ispravno urađeno, kliknite na dugme za zatvaranje i zatvorite dizajner menija.

Editoru koda možete direktno pristupiti iz dizajnera menija kliknuvši dva puta na opciju menija. Nakon dvostrukog klika mišem na opciju menija, biće prikazan editor koda i kursor će se nalaziti na događaju `OnClick` opcije menija za koju treba da upišete kod. U ovom slučaju vratiće se u glavnu formu i odatle editovati kod.

Pisanje koda

Uredu, sada imate mnogo stavki menija, a nemate kod koji će ih naterati da rade. Trebaće puno posla da se sve ovo dovede u red, zar ne? U suštini je jednostavno. Većina koda koji Vam je potreban je već deo klase `TMemo`. Sve što treba da uradite je

nastavlja se

da pozovete odgovarajuće `TMemo` metode u upravljače menija. Treba da uradite još nekoliko poslova, ali veći deo koda koji treba da unesete ste već videli ranije.

Dodavanje komponenti u formu

Pre nego što napišete kod, treba da dodate uobičajene komponente `OpenDialog` i `SaveDialog` na formu:

1. Postavite komponentu `OpenDialog` na formu.
2. Promenite karakteristiku `Name` u `OpenDialog`.
3. Postavite komponentu `SaveDialog` na formu.
4. Promenite karakteristiku `Name` u `SaveDialog`.
5. Uredite ikone `MainMenu`, `OpenDialog` i `SaveDialog`.

Pisanje koda za opcije menija

Uredu, ovo je bilo dovoljno jednostavno. Vratimo se sada na pisanje koda za opcije menija. Počećete sa opcijom File➥Exit (ona je najlakša). Uverite se da je dizajner menija zatvoren, kako ne bi pomešali dizajner menija i dizajner forme.

1. Odaberite File➥Exit u okviru glavnog menija. Editor koda će se pojaviti na vrhu svih prozora i biće prikazan upravljač događajem `FileExitClick`.
2. Kursor se nalazi na svom mestu i spreman je za rad. Upišite narednu liniju koda (uvek uvlačim opciju za dva prazna mesta):

```
Close;
```

U koraku dva ste koristili funkciju `Close` za zatvaranje forme. Ovo fino radi pošto je aplikaciju ustvari predstavlja glavna forma. Ukoliko želite da prekinete rad aplikacije bilo gde u programu, treba da koristite sledeće:

```
Application.Terminate;
```

Ovaj kod Vam omogućava da se prekine rad aplikacije bez obzira koja je forma trenutno otvorena.

To bi bilo sve. Zar Vam nisam rekao da je ova opcija najlakša? Uradimo još nešto; zatim ću Vam dozvoliti da slobodno završite ostatak menija na svoju ruku.

1. Odaberite opciju Edit➥Cut u okviru glavnog menija. Editor koda se pojavljuje i prikazuje upravljač događajem, `EditCutClick`.
2. U liniju pored kursora upište sledeći kod:

```
Memo.CutToClipBoard;
```

I to bilo sve za određenu opciju menija. Možda niste u potpunosti shvatili, ali VCL radi većinu stvari u pozadini. Kompletna ideja kostura programa je da sve detalje niskog nivoa prebaci sa programera na sistem. Život je lep.

Dodavanje završnog izgleda

Jedan od najinteresantnijih aspekata programa kao što je Delphi, predstavlja činjenica da ćete retko videti u celini Vaš program. Delphi obično pokazuje samo deo koda koji je potreban za određeni događaj, pa obično vidite Vaše programe u delovima. Listing 6.1 prikazuje junit glavne forme programa `ScratchPad` u ovom trenutku. Deklaraciju klasa je kompletno generisao Delphi. Pogledajte primere na kojima ste upravo radili koristeći pisani kod i uočite izmene za preostale opcije menija. Kopirajte kod za svaki upravljač događajem `OnClick` iz listinga 6.1. (Linije sa komentarima objašnjavaju ulogu koda. Ne morate ih pisati ukoliko to ne želite.)

Upravljači događajima se pojavljuju u izvornom kodu redosledom kojim ste ih kreirali. Nemojte se zabrinuti ukoliko upravljači događajima u Vašem izvornom kodu ne budu poređani istim redosledom kao što je to urađeno u listingu 6.1. Redosled pojave funkcija nema uticaja na prevodioca.

Listing 6.1: `SPMAIN.PAS`

```
unit SPMain;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
  Dialogs,
  Menus, StdCtrls, ComCtrls, ToolWin;

type
  TMainForm = class(TForm)
    StatusBar1: TStatusBar;
    ToolBar1: TToolBar;
    ToolButton1: TToolButton;
    ToolButton2: TToolButton;
    Memo: TMemo;
    MainMenu: TMainMenu;
    FileMenu: TMenuItem;
    FileNew: TMenuItem;
    FileOpen: TMenuItem;
    FileSave: TMenuItem;
    FileSaveAs: TMenuItem;
    N1: TMenuItem;
    FilePrint: TMenuItem;
    FilePrintSetup: TMenuItem;
    N2: TMenuItem;
    FileExit: TMenuItem;
```

nastavlja se

Listing 6.1: SPMAIN.PAS *nastavak*

```
    Edit1: TMenuItem;
    EditReplace: TMenuItem;
    EditFind: TMenuItem;
    N4: TMenuItem;
    EditPaste: TMenuItem;
    EditCopy: TMenuItem;
    EditCut: TMenuItem;
    N5: TMenuItem;
    EditUndo: TMenuItem;
    Help1: TMenuItem;
    HelpAbout: TMenuItem;
    HelpContents: TMenuItem;
    EditSelectAll: TMenuItem;
    N3: TMenuItem;
    EditWordWrap: TMenuItem;
    OpenDialog: TOpenDialog;
    SaveDialog: TSaveDialog;
    procedure FileExitClick(Sender: TObject);
    procedure EditCutClick(Sender: TObject);
    procedure EditCopyClick(Sender: TObject);
    procedure EditPasteClick(Sender: TObject);
    procedure FileNewClick(Sender: TObject);
    procedure FileSaveClick(Sender: TObject);
    procedure FileOpenClick(Sender: TObject);
    procedure FileSaveAsClick(Sender: TObject);
    procedure EditUndoClick(Sender: TObject);
    procedure EditSelectAllClick(Sender: TObject);
    procedure EditWordWrapClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FileExitClick(Sender: TObject);
begin
  { All done. Close the form. }
  Close;
end;

procedure TMainForm.EditCutClick(Sender: TObject);
```

```
begin
  { Call TMemo.CutToClipboard. }
  Memo.CutToClipboard;
end;

procedure TMainForm.EditCopyClick(Sender: TObject);
begin
  { Call TMemo.CopyToClipboard. }
  Memo.CopyToClipboard;
end;

procedure TMainForm.EditPasteClick(Sender: TObject);
begin
  { Call TMemo.PasteFromClipboard. }
  Memo.PasteFromClipboard;
end;

procedure TMainForm.FileNewClick(Sender: TObject);
var
  Res : Integer;
begin
  { Open a file. First check to see if the }
  { current file needs to be saved. }
  if Memo.Modified then begin
    { Display a message box. }
    Res := Application.MessageBox(
      'The current file has changed. Save changes?',
      'ScratchPad Message', MB_YESNOCANCEL);
    { If Yes was clicked then save the current file. }
     if Res = IDYES then
       FileSaveClick(Sender);
    { If No was clicked then do nothing. }
    if Res = IDCANCEL then
      Exit;
  end;
  { Delete the strings in the memo, if any. }
  if Memo.Lines.Count > 0 then
    Memo.Clear;
  { Set the FileName property of the Save Dialog to a }
  { blank string. This lets us know that the file has }
  { not yet been saved. }
  SaveDialog.FileName := '';
end;

procedure TMainForm.FileOpenClick(Sender: TObject);
var
  Res : Integer;
begin
  { Open a file. First check to see if the current file needs }
  { to be saved. Same logic as in FileNewClick above. }
```

Listing 6.1: SPMAIN.PAS

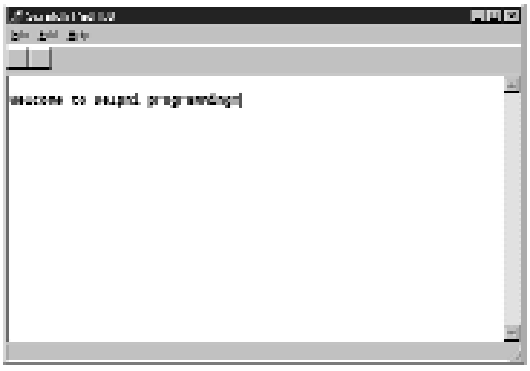
```
    if Memo.Modified then begin
      Res := Application.MessageBox(
        'The current file has changed. Save changes?',
        'ScratchPad Message', MB_YESNOCANCEL);
      if Res = IDYES then
        FileSaveClick(Sender);
      if Res = IDCANCEL then
        Exit;
    end;
    { Execute the File Open dialog. If OK was pressed then }
    { open the file using the LoadFromFile method. First   }
    { clear the FileName property. }
    OpenDialog.FileName := '';
    if OpenDialog.Execute then begin
      if Memo.Lines.Count > 0 then
        Memo.Clear;
      Memo.Lines.LoadFromFile(OpenDialog.FileName);
      SaveDialog.FileName := OpenDialog.FileName;
    end;
  end;

  procedure TMainForm.FileSaveClick(Sender: TObject);
  begin
    { If a filename has already been provided then there is }
    { no need to bring up the File Save dialog. Just save the }
    { file using SaveToFile. }
    if SaveDialog.FileName <> '' then begin
      Memo.Lines.SaveToFile(SaveDialog.FileName);
      { Set Modified to False since we've just saved. }
      Memo.Modified := False;
    { If no filename was set then do a SaveAs. }
    end else FileSaveAsClick(Sender);

  end;

  procedure TMainForm.FileSaveAsClick(Sender: TObject);
  begin
    { Display the File Save dialog to save the file. }
    { Set Modified to False since we just saved.     }
    SaveDialog.Title := 'Save As';
    if SaveDialog.Execute then begin
      Memo.Lines.SaveToFile(SaveDialog.FileName);
      Memo.Modified := False;
    end;
  end;

  procedure TMainForm.EditUndoClick(Sender: TObject);
  begin
```

```
  { TMemo doesn't have an Undo method so we have to send }
  { a Windows WM_UNDO message to the memo component.     }
  SendMessage(Memo.Handle, WM_UNDO, 0, 0);
end;

procedure TMainForm.EditSelectAllClick(Sender: TObject);
begin
  { Just call TMemo.SelectAll. }
  Memo.SelectAll;
end;

procedure TMainForm.EditWordWrapClick(Sender: TObject);
begin
  { Toggle the TMemo.WordWrap property. Set the Checked }
  { property of the menu item to the same value as WordWrap. }
  Memo.WordWrap := not Memo.WordWrap;
  EditWordWrap.Checked := Memo.WordWrap;
  { If WordWrap is on then we only need the vertical scroll }
  { bar. If it's off, then we need both scroll bars.        }
  if Memo.WordWrap then
    Memo.ScrollBars := ssVertical
  else
    Memo.ScrollBars := ssBoth;
end;

end.
```

A sada, trenutak na koji ste čekali...

Nakon što ste kreirali upravljače događajima za opcije menija, spremni ste da pokrenete program. Kliknite mišem na dugme Run i program će biti preveden i pokrenut. Ukoliko prevodilac prijavi grešku, pažljivo uporedite izvorni kod sa listingom 6.1. Uradite potrebne izmene, a zatim kliknite ponovo na dugme Run. Možda ćete morati da prođete kroz ovaj proces nekoliko puta kako bi se Vaš program u potpunosti preveo, a zatim pokrenuo. U svakom slučaju, nadam se, program će raditi (obećavam!).

Kada pokrenete program, primetićete da, iako nije u potpunosti završen, radi potpuno isto kao Windows Notepad. Iako imate još nekoliko stvari da uradite pre završetka programa, imate dovoljno dobru osnovu - naročito kada razmotrite koliko ste vremena utrošili do sada. Slika 6.22 pokazuje kako radi program ScratchPad.



Slika 6.22
Program ScratchPad u toku rada

Slobodni meniji (meniji sadržaja)

Još nije završena priča o menijima. U Delphiju možete kreirati slobodne menije na isti način kao što kreirate glavni meni. Dobra strana Delphija je da se može dodeliti slobodni meni bilo kojoj komponenti korišćenjem karakteristike `PopupMenu`. Kada postavite kursor preko komponente i kliknete drugim tasterom miša, automatski će biti prikazan slobodni meni. Pisanje upravljača događajima za slobodne menije, je potpuno isto kao i pisanje upravljača događajem za glavni meni.

Uobičajeno za programe koji rade sa tekstom je postavljanje operacija za isecanje, kopiranje i lepljenje na meni sadržaja. Ovu mogućnost ćete dodati u `ScratchPad`. Da biste kreirali slobodni meni, izvešćete malu prevaru i kopirati deo glavnog menija. Izvršite sledeće korake:

1. Odaberite komponentu `PopupMenu` u okviru palete komponenti i postavite je na formu.
2. Promenite karakteristiku `Name` u `MemoPopup`.
3. Dva puta kliknite na ikonu `PopupMenu` da biste pokrenuli dizajner menija.
4. Kliknite na drugi taster miša kako biste pozvali meni sadržaja dizajnera menija. Odaberite opciju Select Menu u okviru menija sadržaja. Biće prikazan okvir za dijalog koji prikazuje menije dostupne u okviru Vaše aplikacije. Odaberite MainMenu i kliknite na dugme OK.
5. Kliknite na meni Edit. Kliknite na opciju menija Cut, držite pritisnut taster Shift, a zatim kliknite na opciju menija Paste. Sada su osvetljene opcije Cut, Copy i Paste.
6. Da biste kopirali odabrane opcije na Clipboard, odaberite opciju Edit➥Copy u okviru glavnog menija Delphija (nemojte odabrati opciju Edit➥Copy u okviru menija koji kreirate u dizajneru menija), ili pritisnite tastere Ctrl+C.

7. Zatim ponovo odaberite opciju Select Menu u okviru menija sadržaja dizajnera menija. Ovaj put odaberite MemoPopup, a zatim kliknite na dugme OK. Dizajner menija prikazuje prazan slobodni meni.
8. Odaberite opciju Edit➥Paste u okviru glavnog menija, odnosno pritisnite tastere Ctrl+V. Opcije menija Cut, Copy i Paste će biti ubačene u slobodni meni.

Uredu, još par stvari i biće gotovo. Treba da promenite karakteristiku `Name` za nove stavke menija:

1. Za opciju menija Cut, promenite karakteristiku `Name` u `PopupCut`.
2. Za opciju menija Copy, promenite karakteristiku `Name` u `PopupCopy`.
3. Za opciju menija Paste, promenite karakteristiku `Name` u `PopupPaste`.

Poslednji korak je pisanje upravljača događajima za opcije slobodnog menija. Hmmm... Već ste napisali kod za opcije Cut,Copy i Paste glavnog menija. Bilo bi sramota da duplirate ove delove koda (iako postoji samo jedna linija za svaku opciju). Da li možete da koristite iste upravljače događajem koje ste koristili ranije? Naravno da možete. Potrebno je da pratite sledeće korake:

1. Kliknite na opciju Cut slobodnog menija.
2. Kliknite na karticu Events u prozoru Object Inspector.
3. Kliknite na dugme sa strelicom u okviru kolone `Value` koja se nalazi pored događaja `OnClick` (jedini događaj na listi). Lista upravljača događaja koji su do sada kreirani će biti prikazani.
4. Odaberite upravljač događajem `EditCutClick` sa spiska. Sada će nakon izbora opcije Cut slobodnog menija biti pozvan upravljač događajem za opciju Edit➥Cut glavnog menija programa. Nije potrebno nikakvo dupliranje koda.
5. Ponovite korake od jedan do četiri za opcije Copy i Paste slobodnog menija. Na kraju, zatvorite dizajner menija.
6. Na glavnoj formi kliknite na komponentu `Memo`. Promenite karakteristiku `PopupMenu` u `MemoPopup` (birajući sa liste).

Na ovaj način možete dodati skoro svaki događaj bilo kom upravljaču događaja. Pokrenite program, kako bi testirali novi meni sadržaja. Naravno da radi!

Kreiranje i snimanje obrazaca menija

Delphi Vam pruža nekoliko obrazaca menija koje možete ubaciti u Vaše glavne menije i slobodne menije. Takođe možete kreirati i snimiti obrasce koje ste sami osmislili, kako biste ih kasnije koristili u svojim programima. Prvo startujte dizajner menija i kreirajte meni.

NAPOMENA Kada kreirate menije koje ćete koristiti kao obrasce, prethodno morate imati glavni meni, odnosno slobodni meni na formi, kako biste startovali dizajner menija. Možete koristiti privremenu praznu formu, ukoliko želite. Otvorite praznu formu, postavite komponentu `MainMenu` na formu, a zatim dvostrukim klikom miša na ikonu meni komponente pozovite dizajner menija. Nakon završetka kreiranja obrasca menija, uklonite formu bez snimanja.

Kada kreirate meni, odaberite opciju Save As Template u okviru menija sadržaja dizajnera menija. Biće prikazan okvir za dijalog Save Template (snimi obrazac). Dodelite meniju naziv koji ima smisla, a zatim kliknite na dugme OK, pa će meni biti snimljen kao obrazac. Da biste dodali meni, odaberite opciju Insert From Template u okviru menija sadržaja dizajnera menija, kao što ste to učinili ranije. Svi meniji koje ste kreirali će se pojaviti zajedno sa obrascima Delphijevih menija.

Da biste uklonili menije koje ste prethodno dodali, odaberite opciju Delete Templates u okviru menija sadržaja dizajnera menija. Biće prikazan okvir za dijalog Delete Templates, pa ćete moći da odaberete obrazac koji želite da obrišete. Odabrani obrasci menija će biti obrisani klikom miša na dugme OK. Ukoliko ne želite da obrišete ni jedan obrazac, kliknite mišem na dugme Cancel.

Zaključak

Čestitamo! Upravo ste obradili gomilu Delphijevih mogućnosti za vizuelno programiranje. Nadam se da je bilo poučno i da ste uživali. Dizajner forme je veoma moćan alat koji Vam omogućava da vizuelno programiranje primenjujete koliko god je to moguće. Ukoliko niste morali da kreirate prozore i dijaloge koristeći tradicionalne alate za Windows programiranje, možda nećete u potpunosti ceniti ovu prednost. Verujte mi, prednost je značajna. Dizajner menija je takođe moćan alat, a naročito mogućnost uvoza menija koja kreiranje menija pretvara u zabavu i značajno olakšava tu vrstu posla. Dizajner menija Vam omogućava da menjate postojeće menije u trenutku.

Radionica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstite razumevanje obrađenog materijala, kao i vežbe koje Vam omogućavaju da steknete iskustvo u korišćenju gradiva koje ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A, "Odgovori na kviz pitanja".

Pitanja i odgovori

P Često koristim paletu za poravnavanje i svaki put kada pređem sa editora koda na dizajner forme zagubim paletu za poravnavanje. Šta mogu da uradim u vezi toga?

O Pronađite paletu za poravnavanje (tu je negde!), a zatim kliknite drugim tasterom miša da biste dobili meni sadržaja palete za poravnavanje. Odaberite opciju Stay on Top u okviru menija sadržaja. Od ovog trenutka će palete za poravnavanje uvek biti na vrhu, pa ćete moći da je nađete.

P **Pokušavam da odaberem grupu komponenti na panelu prevlačeći pravougaonik za izbor oko komponenti, ali nastavljam da pomeram pano. Šta nije u redu?**

O Dok prevlačite miša, birajući komponente u okviru panoa, treba da držite pritisnut taster Ctrl.

P **Pomerao sam komponente po formi nekoliko puta, pa je sada redosled tabulatora pomešan. Šta treba da uradim da bih popravio redosled?**

O Odaberite opciju Tab Order u okviru menija sadržaja dizajnera forme. Organizujte redosled tabulatora na način koji želite. Nakon klika mišem na dugme OK, biće implementiran novi redosled tabulatora.

P **Obrasci menija koji su unapred dati su dobri, ali imaju suviše opcija koje mi nisu potrebne. Šta mogu da uradim po tom pitanju?**

O Možete uraditi dve stvari. Prvo, možete uvesti meni, a zatim obrisati opcije koje Vam nisu potrebne. Korišćenjem klika na miša, odnosno kombinacije Shift+klik na taster miša, možete se otarasiti nepoželjnih opcija menija za samo nekoliko sekundi. Brisanje opcija iz menija koje ste uvezli iz obrasca, nema nikakvih naknadnih posledica. Druga mogućnost je da nakon što ste korišćenjem kombinacija klik na taster miša i Shift+klik na taster miša izmenili meni, snimite meni kao novi obrazac i na taj način dobijete meni onakakv kakav ste želeli. Na ovaj način možete zadržati originalni Delphijev obrazac i dodati prilagođen obrazac listi.

P **Da li mogu snimiti sopstvene menije kao obrasce?**

O Da. Prvo kreirajte meni, a zatim odaberite opciju Save As Template u okviru menija sadržaja dizajnera menija. Dodelite mu naziv, kliknite na dugme OK i Vaš obrazac će biti snimljen. Da biste ponovo koristili ovaj meni, ubacite ga korišćenjem opcije Insert From Template.

Kviz

1. Kada koristite kombinaciju Ctrl+prevlačenje mišem za odabiranje komponenti?
2. Koji značaj ima izbor prve komponente kada poravnavate grupu komponenti?
3. Koji je najbrži način za izbor grupe komponenti?
4. Kako možete da postignete da sve komponente u okviru grupe imaju širinu najšire komponente?

5. Šta će se dogoditi kada kliknete mišem dva puta na komponentu u okviru forme?
6. Šta radi opcija `alClient` karakterisktike `Align`?
7. Šta znače tri tačke iza naziva opcije menija?
8. Koja su dva načina za pomeranje opcija menija?
9. Kako možete dodati akceleratore menija opcijama menija?
10. Kako možete u početku deaktivirati opciju menija?

Vežbe

1. Postavite pet komponenti Edit na formu i organizujte ih tako da su poređane vertikalno sa poravnatom levom ivicom.
2. Isključite opciju Snap to Grid (odaberite opciju Tools➥Environment Options u okviru glavnog menija). Postavite pet kontrola po izboru na formu, a zatim poravnajte njihove desne ivice.
3. Postavite komponentu `ListBox` na praznu formu i izmenite je tako da zauzme kompletnu klijent oblast forme.
4. Dodajte okvir About programu `ScratchPad`. Koristite paletu za poravnavanje, kako bi brzo poravnali tekst natpise.
5. Dodajte opciju Undo i separator menija u meni sadržaja programa `ScratchPad`.
6. Otvorite novu aplikaciju, postavite šest komponenti za editovanje na formu u slučajnom poretku. Sada uredite redosled tabulatora, tako da pritiskom na taster Tab prelazite sa komponte na komponentu od vrha do dna. Pokrenite program, kako biste testirali redosled tabulatora.
7. Dodajte prečicu tastature Ctrl+S opciji menija File➥Save u okviru programa `ScratchPad`.
8. Otvorite projekt Picture Viewer koji ste kreirali u lekciji dana 4. Uklonite sve nepotrebne opcije menija.

Dan 7

VCL komponente

Kao što ste mogli do sada da naučite, Delphi duguje svoju snagu komponentama. Koristeći dizajner forme možete postaviti komponentu na formu i menjati njene karakteristike u toku dizajniranja. U nekim slučajevima to je sve što će Vam biti potrebno. Ukoliko je neophodno, komponentama možete manipulisati i u toku rada programa, menjajući njihove karakteristike i pozivajući njihove metode. Dalje, svaka komponenta je dizajnirana da odgovori na određeni događaj. Karakteristike, metode i događaji su bili obrađeni u lekciji dana 5, "Model vizuelnih komponenti", pa ih neću ponavljati.

Danas ćete saznati nešto više o komponentama. Naučićete o često korišćenim komponentama i dodatno, o klasama biblioteke vizuelnih komponenti (VCL - Visual Component Library) koje predstavljaju komponente. U toku čitanja ove lekcije, slobodno eksperimentišite. Ukoliko pročitate nešto što biste želeli da ispitate u bilo kom smislu, ispitajte. Učenje na iskustvu je vredno bilo čega što budete radili, zato se nemojte plašiti da eksperimentišete.

Pregled komponenti

Pogledajmo ponovo stvari koje ste već naučili o komponentama. Ipak, pre nego što pređemo na stvar, voleo bih da Vam na trenutak objasnim razliku između VCL komponente i Windows kontrole. Windows kontrola uključuje komponente kao što su edit kontrole, okviri lista, kombo okviri, statičke kontrole (natpisi) i dugmad; da ne spominjemo sve Win32 kontrole. Windows kontrole, po svojoj prirodi nemaju kara-kteristike, metode i događaje. Umesto njih se koriste poruke koje saopštavaju

kontroli šta treba da uradi, odnosno da prime informaciju od kontrole. Kada bi rekli da je rad sa kontrolama na ovom nivou dosadan i nezgrapan, to bi bilo prilično blago rečeno.

VCL komponente su klasa koja enkapsulira Windows kontrole (sve VCL komponente ne enkapsuliraju kontrole, naprotiv). VCL komponente kao efekat dodaju karakteristike, metode i događaje Windows kontrolama, da bi rad sa njima bio lakši. Može se reći da VCL ima nov pristup u radu sa Windows kontrolama. Takođe se može reći da su sve VCL komponente kontrole, ali nisu sve kontrole komponente. Na primer, VCL Edit komponenta je kontrola, ali standardna Windows edit kontrola nije VCL komponenta. VCL komponente rade sa Windows kontrolama, kako bi podigli na viši nivo rad sa kontrolama.

Na osnovu prethodnog objašnjenja, ubuduće ću se koristiti terminima *kontrola* i *komponenta* naizmenično kada obrađujem VCL komponente. (Ali nikad neću nazivati Windows kontrole komponentama!)

Vizuelne komponente

Vizuelne komponente uključuju komponente poput edit kontrola, dugmadi, okvira lista, natpisa, itd. Većina komponenti koje ćete koristiti u Delphi aplikacijama su vizuelne komponente. Vizuelne komponente Vam, koliko je to moguće, u toku dizajniranja pokazuju kako će komponente izgledati u toku rada programa.

NOVI TERMIN Neke komponente su vizuelne, dok su ostale nevizuelne komponente. *Vizuelna komponenta*, kako joj samo ime kaže, je komponenta koju korisnik može videti u toku dizajniranja.

Nevizuelne komponente

NOVI TERMIN *Nevizuelna komponenta* je komponenta koju korisnik ne može videti u toku dizajniranja.

Nevizuelna komponenta radi u pozadini kako bi izvršila određeni programski zadatak. Primeri podrazumevaju sistemske tajmere, komponente baze podataka i liste slika. Uobičajeni okviri za dijalog kao što su File Open, File Save, Font i slično se mogu takođe smatrati nevizuelnim komponentama. (Ove komponente su nevizuelne, pošto se ne prikazuju u toku dizajniranja. U toku rada postaju vidljive u trenutku kada se pozovu.) Uobičajene dijalog komponente će biti obrađene u nastavku lekcije u poglavlju: "Uobičajeni okviri za dijalog".

Kada postavite nevizuelnu komponentu na formu, Delphi prikazuje ikonu koja predstavlja komponentu na formi. Ova ikona se koristi za pristup komponenti u toku dizajniranja, kako bi se menjale karakteristike komponente, ali se ikona ne pojavljuje u toku rada programa. Nevizuelne komponente imaju karakteristike, metode i događaje, kao i svaka vizuelna komponenta.

Pogledajmo neke zajedničke karakteristike koje komponente dele.

Karakteristika Name

Karakteristika Name ima vitalnu ulogu kod komponenti. U lekciji dana 5, "Model vizuelnih komponenti", u poglavlju "Otkrivanje VCL-a", obrađen je deo dešavanja sa komponentom prilikom postavljanja komponente na formu. Nakon što postavite komponentu na formu, Delphi prelazi da radi u pozadini, dok čeka Vaš sledeći korak. Jedna od stvari koje Delphi radi je kreiranje pointera na komponentu i dodeljivanje karakterisike Name, koja predstavlja naziv promenljive. Na primer, recimo da postavite Edit komponentu na formu i promenite karakteristiku u MyEdit. Od tog trenutka, Delphi upisuje u deklaraciju klase za tekuću formu (u odeljku published):

```
MyEdit: TEdit;
```

Kada se pokrene aplikacija, Delphi kreira slučaj klase TEdit i dodeljuje je komponenti MyEdit. Možete koristiti ovaj pointer da pristupite komponenti u toku rada programa. Da biste dodelili tekst edit kontroli, treba da koristite:

```
MyEdit.Text := 'Jenna Lynn';
```

Delphi takođe koristi karakteristiku Name kada kreira nazive upravljača događajima. Recimo da želite da odgovorite na događaj OnChange komponente Edit. Normalno, kliknućete mišem na kolonu Value u prozoru Object Inspector, pored događaja OnChange, kako bi Delphi kreirao upravljač događajem za željeni događaj. Delphi kreira generički naziv funkcije baziran na karakteristici Name trenutno aktivne komponente, kako bi se događajem se moglo upravljati. U ovom slučaju, Delphi bi kreirao funkciju pod nazivom MyEditChange.

Možete promeniti karakteristiku Name bilo kada, ukoliko to učinite samo preko prozora Object Inspector. Kada promenite karakteristiku Name tekuće komponente u toku dizajniranja, Delphi prolazi kroz kompletan kod koji nije prethodno generisan i menja naziv pointera i svih funkcija za upravljanje događajima.

Delphi će promeniti sve kodove koje je generisao da bi izmenio karakteristiku Name tekuće komponente. Delphi neće menjati kod koji ste Vi pisali. Drugim rečima, Delphi će voditi računa o izmeni koda koji je napisao, a na Vama je da održavate kod koji ste Vi napisali. U suštini, treba da podesite karakteristiku Name kada inicijalno postavite komponentu na formu i nakon toga je ostavite na miru. Ne postoji problem ukoliko želite da promenite naziv kasnije, ali to će Vam doneti više posla.

Nastavimo sa primerom; ako promenite karakteristiku Name edit kontrole, sa MyEdit na FirstName, Delphi će promeniti naziv pointera u FirstName, a naziv upravljača događajem u FirstNameChange. Sve se ovo radi automatski; Vi ne treba da radite ništa, osim da promenite karakteristiku Name i verujete Delphiju da će odraditi svoj deo posla.

Nemojte nikad menjati karakteristiku `Name` u toku rada programa. Nikad manuelno nemojte menjati naziv komponente (naziv koji Delphi dodeljuje pointeru komponente), odnosno naziv upravljača događajem u editoru koda. Ukoliko izvršite bilo koju od ovih aktivnosti, Delphi će izgubiti trag komponente i rezultati neće biti dobri, najblaže rečeno. Možda ćete čak, izgubiti mogućnost da učitate Vašu formu. Jedini siguran način da promenite naziv komponente je korišćenje karakteristike `Name` u okviru Object Inspectora.

Delphi dodeljuje generičku vrednost karakteristici `Name` za sve komponente koje su postavljene na formu. Ukoliko, primera radi, postavite `Edit` komponentu, Delphi će karakteristici `Name` dodeliti naziv `Edit1`. Ukoliko postavite drugu `Edit` komponentu na formu, Delphi će joj dodeliti naziv `Edit2`, itd. Vašim komponentama bi trebalo da dajete nazive koji imaju razumljivo značenje kako bi kasnije izbegli zbrku i dodatni posao.

Možete da ostavite generičke nazive komponentama koje nikada nećete pozvati u okviru svog koda. Na primer, ukoliko imate nekoliko komponenti Label koje sadrže statički (nepromenljiv) tekst, možete ostaviti generičke nazive pošto ovim komponentama nećete pristupati u toku rada programa.

Važne uobičajene karakteristike

Sve komponente imaju kao zajedničke, određene karakteristike. Na primer, sve vizuelne komponente imaju karakteristike `Left` i `Top` koje određuju mesto komponente na formi. Karakteristike kao što su `Left`, `Top`, `Height`, `Width` (levo, gore, visina, širina) imaju nazive koji sami sebe objašnjavaju, pa ih neću posebno obrađivati. Neke od uobičajenih karakteristika, pak, zahtevaju bolje objašnjenje.

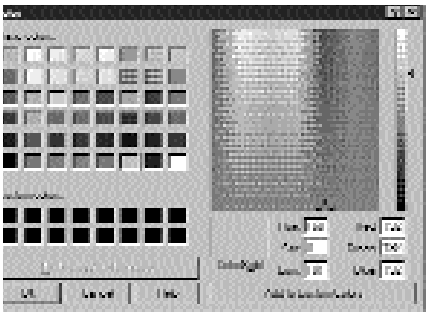
Karakteristika `Align` (poravnavanje)

U lekciji dana 6, "Rad sa dizajnerom forme i dizajnerom menija", obrađene su karakteristike `Align` i `Alignment`, pa ih ponovo neću objašnjavati. Za kompletne informacije o karakteristici `Align`, možete pogledati lekciju dana 6. Tada sam nagovestio da se karakteristika Align ne pojavljuje kod svih komponenti u toku dizajniranja. Edit kontrola koja zauzima jednu liniju, na primer, može imati samo standardnu visinu, tako da karakteristika `Align` nema smisla za ovaj tip komponente. Kako stičete iskustvo u radu sa Delphijem (i u zavisnosti od tipa aplikacije koju pišete), verovatno se često oslanjate na karakteristiku `Align`.

Karakteristika `Color` (boja)

Karaktertistika `Color` postavlja boju pozadine komponente. (Boja teksta se podešava preko karakteristike `Font`.) Naravno, karakteristika `Color` je jednostavna za korišćenje, ali postoji nekoliko napomena koje treba spomenuti.

Način na koji je karakterisktika `Color` predstavljena u prozoru Object Inspector je pomalo jedinstvena. Ukoliko kliknete mišem na kolonu Value, primetićete dugme sa strelicom koje označava da možete odabrati boju sa liste. Ovo je samo deo, ali pored liste postoji i druga opcija. Ukoliko dva puta kliknete mišem na kolonu Value, okvir za dijalog `Color` će biti prikazan. Ovaj okvir za dijalog Vam omogućava da odaberete boju iz grupe unapred definisanih, odnosno da kreirate sopstvenu boju klikom na dugme Define `Custom Colors` (definisanje proizvoljnih boja). Slika 7.1 prikazuje okvir za dijalog Color nakon klika na dugme `Define Custom Colors`.



Slika 7.1
Okvir za dijalog Color

NAPOMENA Ovo je isti okvir za dijalog `Color` koji će biti proikazan kada implementirate komponentu `ColorDialog` u Vašu aplikaciju.

Ukoliko odaberete boju iz okvira za dijalog `Color`, primetićete da se karakteristika `Color` menja u heksadecimalni string. Ovaj string predstavlja vrednosti crvene, zelene i plave komponente (RGB - red green blue), koje čine boju. Ukoliko znate tačnu RGB vrednost boje, možete je upisati (nije valjda!).

Veći deo vremena ćete provoditi birajući boje sa liste boja koja Vam je ponuđena. Kada kliknete na dugme sa strelicom, da bi prikazali listu potencijalnih vrednosti, videćete funkcionalno podeljene dve grupe vrednosti. Prva grupa počinje sa `clBlack`, a završava se sa `clWhite`. Ovo su Delphijeve unapred definisane boje; ova lista predstavlja najčešće korišćene boje. Da biste odabrali jednu od boja sa liste, kliknite na boju koju želite da odaberete. Ukoliko ne možete da pronađete boju koja udovoljava Vašim zahtevima, možete pozvati okvir za dijalog `Color` koji je bio objašnjen nešto ranije.

Druga grupa boja na listi počinje sa `clScrollBar`. Ova grupa boja predstavlja Windows sistemske boje. Ako koristite boje sa liste, Vaša aplikacija će automatski podesiti Vaše boje prema šemi boja u okviru Windows operativnog sistema. Ukoliko želite da Vaša aplikacija prati šemu boja koje je korisnik odabrao za svoj sistem, treba da odaberete boje sa liste Windows sistemskih boja, umesto sa liste Delphijevih boja.

Korišćenje boja bi trebalo da bude pažljivo razmotrena. Ispravno korišćenje boja pruža estetski ugodno okruženje za korisnika. Zloupotreba boja može učiniti aplikaciju odbojnom tako da je korisnici nerado upotrebljavaju. Boje su kao magnet

za nove programere. Uobičajena je želja da se baci što više boja na formu, pošto je zabavno i jednostavno, ali nemojte da se zabavljate na račun svojih korisnika.

Karakteristika `Cursor`

Karakteristika `Cursor` kontroliše kursor koji se prikazuje kada korisnik pomera miša preko komponenata. Windows automatski menja kursor za neke komponente. Na primer, Windows menja kursor na izgled I-zraka (I-beam) kada se kursor pomeri preko `Edit`, `Memo`, odnosno `RichEdit` komponente.

Da bi Windows upravljao kursorom, stavite karakteristiku `Cursor` na vrednost `crDefult`. Ukoliko imate posebne prozore (komponente), možete definisati drugi tip kursora. U trenutku kada se miš pomera preko komponente, Windows će promeniti izgled kursora u izgled koji ste unapred definisali karakteristikom `Cursor`.

Često je potrebno da promenite izgled kursora u toku rada programa. Dug proces, na primer, može biti naznačen korisniku prikazivanjem kursora u obliku peščanog sata. Kada resetujete kursor, treba da budete sigurni koji izgled je kursor prethodno imao. Naredni isečak koda ilustruje ovaj koncept:

```
var
  OldCursor : TCursor;
begin
  OldCursor := Screen.Cursor;
  Screen.Cursor := crHourGlass;
  { do some stuff which takes a long time }
  Screen.Cursor := OldCursor;
end;
```

Na ovaj način se osigurava da kursor koji je za aplikaciju podešen na posebnu vrednost, bude regularno vraćen nakon izmene.

Naredna karakteristika kursora, `DragCursor`, se koristi za promenu izgleda kursora, kada se kursor nađe na kompomnenti koja podržava prevlačenje i spuštanje (drag-and-drop). Kao što je slučaj sa bojama, trebali bi da budete obazrivi sa korišćenjem kursora. Koristite kursore koje ste posebno definisali ukoliko je to potrebno, ali nemojte preterivati.

Karakteristika `Enabled`

Komponente mogu biti aktivirane (enabled), odnosno deaktivirane (disabled) korišćenjem karakteristike `Enabled`. Kada je komponenta deaktivirana, nije u mogućnosti da primi fokus (klik mišem na komponentu nema efekta), i obično korisniku vizuelno daje do znanja da je deaktivirana. Kod dugmadi, na primer, tekst dugmeta i bitmapa su sivi. Karakteristika `Enabled` ima `Bulovu` vrednost: podesite je na `True` ukoliko želite da aktivirate komponentu, odnosno podesite je na `False`, ukoliko želite da deaktivirate komponentu. Aktiviranje i deaktiviranje prozora (zapamtite da su prozorske komponente takođe prozori) je mogućnost samih Windows-a.

NAPOMENA Neke komponente pokazuju svoje stanje kada su deaktivirane u toku dizajniranja, ali većina to ne čini. Komponenta `BitBtn` je jedna od komponenti koja pokazuje stanje u toku dizajniranja, ukoliko je deaktivirana.

Karakteristika `Enabled` se odnosi uglavnom na prozorske komponente, ali se isto tako može odnositi i na neprozorske komponente. Komponenta `SpeedButton` je primer neprozorske komponente koja može biti deaktivirana.

NAPOMENA Izmena karakteristike `Enabled` za komponentu Panel ima dublji smisao. Pano se obično koristi kao kontejner za druge kontrole. Budući da pano postaje roditelj kontrolama koje su postavljene na ovu komponentu, ukoliko deaktivirate pano, komponente neće biti prikazane kao da su deaktivirane, ali neće funkcionisati pošto im je roditelj (pano) deaktiviran.

Iako komponente mogu biti deaktivirane u toku dizajniranja, aktiviranje i deaktiviranje komponenti je aktivnost koja se obično radi u toku rada programa. Opcije menija, na primer, mogu biti aktivirane, odnosno deaktivirane, u zavisnosti od konteksta koji aktiviraju. Isto važi i za dugmad. Postoji niz razloga zašto možete poželeti da deaktivirate druge tipove kontrola.

Da biste deaktivirali komponentu u toku rada programa, treba da karakteristici `Enabled` dodelite vrednost `False`, a da biste aktivirali komponentu treba da karakteristici `Enabled` dodalite vrednost `True`. Naredni isečak koda aktivira, odnosno deaktivira opciju menija na osnovu uslova:

```
if CanSave then
  FileSave.Enabled := True
else
  FileSave.Enabled := False;
```

Ovaj proces se često naziva aktiviranje komande (*command enabling*) i važan je deo Windows programa koji treba da izgleda profesionalno.

SAVET Komponenta `TActionList` se može koristiti da aktivira, odnosno deaktivira komponentu, odnosno grupu komponenti. Komponenta `TActionList` je detaljnije obrađena u lekciji dana 13, "Iza osnova Delphija", u poglavlju "Aktiviranje komande".

Karakteristika `Font`

Karakteristika `Font` je glavna karakteristrika i zahteva da bude spomenuta u ovom delu knjige, ali nema puno toga što bi trebalo o njoj da se kaže. Karakteristika `Font` je slučaj klase `TFont` koja ima sopstvene karakteristike. Podešavanje karakteristika Font možete izvesti ukoliko dva puta kliknete mišem na naziv fonta u okviru prozora Object Inspector (što će učiniti da se nod `Font` proširi i prikaže karakteristike fonta), odnosno koristeći okvir za dijalog `Font`. (Okvir za dijalog `Font` je detaljnije objašnjen u današnjoj lekciji u okviru poglavlja "Okvir za dijalog `Font`".) Slika 7.2 prikazuje prozor Object Inspector sa nodom karakterisike `Font` koji je proširen i otkriva karakteristike klase TFont.

Slika 7.2
Prozor Object Inspector prikazuje karakteristiku `TFont`

Karakteristika `Color` podešava boju fonta, a karakteristika `Name` omogućava da se odabere tip fonta.

Karakteristike `Height` i `Size` klase `TFont` zaslužuju posebnu pažnju:

- Karakteristika `Height` se koristi da definiše visinu fonta u pikselima.
- Karakteristika `Size` se koristi da definiše visinu fonta u tačkama (points).

Kada promenite jednu od ove dve karakteristike, druga će se automatski promeniti. Karakteristika `Height` je obično definisana kao negativan broj. Pogledajte opciju Delphija za pomoć u toku rada (online help), da biste videli objašnjenje koje se nalazi u delu opisa klase `TFont`.

Karakteristika `Pitch` nije posebno korisna. Objasniću je za trenutak, ali ću pre toga ukratko objasniti fontove. Fontovi mogu biti proporcionalni, ili neproporcionalni (proporcionalno razmaknuti, ili jednako razmaknuti):

- Većina fontova je proporcionalna (*proportionally spaced*), što znači da svako slovo zauzima onoliko mesta koliko mu je potrebno; na primer, veliko slovo M zauzima mnogo više mesta nego malo slovo i. Pogledajte slova u ovoj knjizi i videćete šta želim da kažem. Primeri proporcionalnih fontova su: Times New Roman, Arial i Bookman.
- Neproporcionalni fontovi (*fixed space*) imaju drugi način prikazivanja (obično se nazivaju fontovi sa fiksnim razmakom). Svi karakteri zauzimaju isti prostor. Ovo je zgodno za prozore poput editora koda (na primer, Delphijev editor koda), odnosno bilo koji drugi prozor gde je potreban font sa fiksnim razmakom. Courier New je, možda, najčešće korišćeni font sa fiksnim razmakom, a takođe se koristi i font Fixedsys kao osnovni font u nekim Windows aplikacijama. Fontovi sa fiksnim razmakom (*neproporcionalni fontovi*) se teže čitaju, pa se u normalnim situacijama ne koriste za duge blokove teksta, nego samo za pisanje programa.

Teoretski, karakteristika `Pitch` se može koristiti da primora proporcionalni font da se ponaša kao neproporcionalni i obrnuto. Problem nastaje kada Windows treba da izvrši zamenu fontova i da utvrdi da li je ta konverzija bila uspešna. Drugim rečima,

nikada nećete znati šta ste ustvari dobili. Mnogo je bolje da odabete font koji Vam je potreban, nego da se oslonite na karakteristiku `Pitch`.

Na kraju, karakteristika `Style` klase `TFont` se može koristiti da uključuje i isključuje stilove fonta: podebljano, kurziv, podvučeno i precrtano. Ovi stilovi se međusobno ne isključuju, pa možete mešati stilove onako kako želite.

Iako možete koristiti prozor Object Inspector da menjate karakteristike fonta, okvir za dijalog Font (poziva se kada kliknete na dugme sa tri tačkice pored karakteristike Font) je korisniji, pošto Vam prikazuje uzorak fonta prilikom izbora. Da biste jednostavno promenili karakteristiku Style, odnosno karakteristiku Size koristite prozor Object Inspector. Ukoliko tražite pravi font, okvir za dijalog Font, je bolji izbor.

Karakteristika `Hint`

Karakteristika Hint se koristi za postavljanje saveta vezanog za komponentu. Tekst saveta ima dva dela. Prvi deo se ponekad naziva kratak savet (*short hint*). Ovo je savet koji se prikazuje kada korisnik postavi kursor miša na komponentu i zastane. Prozor koji se pojavljuje da bi prikazao tekst sa savetom se naziva oblačić za savet (*tooltip*).

Drugi deo teksta saveta se ponekad naziva dugi savet (*long hint*). Dugi savet je opcioni tekst za savet koji se prikazuje u okviru statusne trake, kada korisnik postavi kursor miša na komponentu. Tekst dugog i kratkog saveta se razdvaja znakom uspravne crte (đ - engl. pipe). Na primer, da bi definisali i dugi i kratki tekst saveta za dugme prečicu (speed button) File Open, treba da upišete sledeći tekst u karakteristiku `Hint`:

```
File Open¦Open a file for editing
```

Da bi se kratak savet prikazao, morate karakteristiku `ShowHint` objekta aplikacije podesiti na `True` (generička vrednost), a takođe i karakteristiku komponente `ShowHint`. Prikazivanje dugog saveta u okviru statusne trake zahteva nešto više posla, pa će ovaj deo biti obrađen u sutrašnjoj lekciji.

Možete definisati posebno tekst kratkog saveta, tekst dugog saveta, odnosno možete definisati oba teksta zajedno. Da bi saopštili Delphiju koji tekst saveta koristite, možete upotrebiti znak vertikalne crte (pipe). Ukoliko ne koristite znak vertikalne crte, kratki savet i dugi savet će kori-stiti isti tekst.

Pošto ni za dugi savet, ni za kratki savet ne postoji ograničenje dužine, trebalo bi da znate prilikom kreiranja kako će se koristiti svaki od ovih saveta. Kratki saveti bi uglavnom trebali da budu ograničeni na manje od 30 karaktera. Dugi saveti mogu biti duži, ali zapamtite da dugi saveti mogu biti skraćeni prilikom prikazivanja na statusnoj traci.

Karakteristike ParentColor, ParentCtl3D, ParentFont i ParentShowHint

Karakteristike `ParentColor`, `ParentCtl3D`, `ParentFont` i `ParentShowHint` rade na potpuno isti način, pa ću ih istovremeno obraditi. Kada su ove karakteristike podešene na `True`, komponenta preuzima karakteristike `Color`, `Ctl3D`, `Font` i `ShowHint` od komponente roditelja. Na primer, za većinu komponenti karakteristika `ParentFont` je generički podešena na `True`. Ovo znači da će komponenta naslediti `Font` koji njena komponenta roditelj trenutno koristi. Da bi bolje razumeli ove karakteristike, uradite vežbu:

1. Kreirajte praznu formu. Podesite karakteristiku `Size` u okviru karakteristike `Font` na 16.
2. Postavite komponentu `Label` na formu. Uočite da natpis automatski koristi font veličine 16 tačaka.
3. Postavite komponentu `Button` na formu. I ova komponenta koristi font veličine 16 tačaka.

Karakteristiku `ParentFont` možete promeniti u `False`, ali pošto su komponente već postavljene, kasno je za izmenu, pa ćete font morati da menjate manuelno, da bi dobili željeni font na komponenti.

Karakteristika Tag

Karakteristika `Tag` je promenljiva dužine 4 bajta, koja je definisana za Vašu internu upotrebu. Karakteristiku `Tag` možete koristiti da biste upisali podatke koji će biti potrebni Vašoj komponenti. Podaci upisani u karakteristiku `Tag` mogu biti pointer na drugu klasu, indeksna vrednost, odnosno bilo koja druga mogućnost. Korišćenje karakteristike `Tag` će najverovatnije biti razmotreno kod tehnika naprednog programiranja.

Ostale uobičajene karakteristike

Tabela 7.1 prikazuje ostale uobičajene karakteristike koje se često koriste. Ove karakteristike ne zahtevaju podrobnije objašnjenje, pa su prikazane samo kao refe-renca. Prikazane karakteristike nećete pronaći u svim komponentama.

Tabela 7.1: Dodatne karakteristike komponente

Karakteristika	Opis
`BorderStyle`	Može biti `bsSingle`, ili `bsNone`. Koristite `bsNone` kada želite da se komponenta utopi u pozadinu.
`BoundsRect`	Pravougaonik čitave komponente (nije ograničen samo na klijent oblast).
`Caption`	Postavlja natpis komponente. Mnoge komponente nemaju natpise, pa za njih karakteristika `Caption` nije predstavljena.
`ClientHeight`	Sadrži visinu klijent oblasti komponente.
`ClientRect`	Sadrži pravougaonik za klijent oblast komponente.
`ClientWidth`	Sadrži širinu klijent oblasti komponente.
`Contraints`	Postavlja veličinu granica komponente (maksimalna širina i visina, minimalna visina i širina). Važnije za forme nego za ostale komponente.
`Ctl3D`	Ukazuje da li će kontrola biti nacrtana sa 3D okvi rom. Ukoliko je karakteristika `BorderStyle` podešena na `bsNone`, ova karakteristika nema efekta.
`Height`	Podešava visinu komponente.
`HelpContext`	Karakteristika `HelpContext` se koristi da dodeli indeksni broj u okviru datoteke za pomoć određenoj komponenti.
`Left`	Podešava X koordinatu komponente.
`Parent`	Pointer na roditelja komponente.
`PopupMenu`	Određuje slobodni meni koji će biti prikazan kada korisnik klikne na drugi taster miša.
`TabOrder`	Za prozorske komponente. Definiše mesto komponente u redosledu tab ulatora.
`TabStop`	Za prozorske komponente. Označava da li će se komponenta naći u redosledu tabulatora. Podešavanje ove karakteristike na vrednost `False` uklanja komponentu iz redosleda tabulatora.
`Top`	Podešava Y koordinatu komponente.
`Visible`	Kada se čita, pokazuje da li je komponenta trenutno vidljiva; kada se u karakteristiku upisuje podatak, ili sakriva, ili pokazuje komponentu.
`Width`	Podešava širinu komponente.

Primarne metode komponenti

Postoji više od dvadeset metoda koje su zajedničke za većinu komponenti. Prozorske komponente imaju više od četrdeset zajedničkih metoda koje se mogu koristiti. Interesantno je da se većina ovih metoda ne nalazi u široj upotrebi. Veći deo

funkcionalnosti komponenti je ostvaren preko karakteristika. Na primer, da bi sakrili komponentu možete saopštiti metodi `Hide` da sakrije komponentu, odnosno možete podesiti karakteristiku `Visible` na vrednost `False`. Dodatno, komponente imaju metode koje su posebno definisane za njihovu svrhu; a ove metode ćete najčešće koristiti kada radite sa određenom komponentom.

Postoji nekoliko metoda koje vredi spomenuti, pa su navedene u tabeli 7.2. Uočite da neke od ovih metoda nisu dostupne svim kontrolama. Ovo nisu često korišćene metode koje su dostupne svakoj komponenti, ali su metode koje se najčešće koriste kod komponenti, uopšteno gledano. Takođe, ovaj spisak komponenti se više koncentriše na komponente koje predstavljaju kontrole (komponente koje se postavljaju na forme), umesto na komponente koje predstavljaju forme. Metode koje su bliske formama su bile obrađene u lekciji dana 4, "Istraživanje Delphijevog okruženja".

Tabela 7.2: Uobičajene metode komponenti

Metoda	Opis
`Broadcast`	Koristi se da pošalje poruku svim prozorskim komponentama potomcima.
`ClientToScreen`	Konvertuje koordinate klijent prozora u koordinate ekrana.
`ContaintsControl`	Vraća vrednost `True` ukoliko je definisana komponenta potomak komponente, odnosno forme.
`HandleAllocated`	Vraća vrednost `True` ukoliko je karakteristika `Handle` za komponentu bila kreirana. Samo čitanje karakteristike `Handle` automatski kreira upravljač, ukoliko već nije kreiran, pa se meto da `HandleAllocated`može koristiti za proveru postojanja upravl jača, a da se upravljač ne kreira.
`Hide`	Sakriva komponentu. Komponenta je još uvek dostupna za kasnije prikazivanje.
`Invalidate`	Zahteva da komponenta bude ponovo iscrtana. Komponenta će biti iscrtana, kako bi olakšala rad Windows-ima.
`Perform`	Šalje poruku direktno komponenti, umesto da je prenosi preko Windows sistema za poruke.
`Refresh`	Zahteva da komponenta bude trenutno ponovo iscrtana i briše komponentu koja se nalazila pre ponovnog iscrtavanja.
`Repaint`	Zahteva da komponenta bude trenutno ponovo iscrtana. Pozadina kom ponente se ne briše pre ponovnog iscrtavanja.
`SetBounds`	Omogućava Vam da istovremeno definišete karakteristike `Top`, `Left`, `Width` i `Height`. Na ovaj način se štedi vreme u odnosu na zasebnopodešavanje karakteristika.
`SetFocus`	Dovodi komponentu u fokus i čini je aktivnom. Važi samo za prozorske komponente.
`Update`	Nameće trenutno ponovno iscrtavanje kontrole. Obično možete koristiti metode `Refresh`, ili `Repaint`, da biste ponovo isctrali komponente.

Preći ćemo na one događaje na koje komponente najčešće mogu odgovoriti.

Uobičajeni događaji

Kao što je slučaj kod karakteristika i metoda, na neke događaje se može odgovoriti češće. Komponente pokrivaju širok spektar mogućih Windows kontrola, pa će svaka komponenta imati posebne potrebe. Događaji koji su specifični samo za forme nisu obrađeni, pošto ih možete naći u lekciji dana 4. Najčešće korišćeni događaji su prikazani u tabeli 7.3.

Tabela 7.3: Događaji komponenata kojima se najčešće upravlja

Događaj	Opis
`OnChange`	Ovaj događaj se pokreće kada se kontola menja na bilo koji način. Tačna implementacija zavisi od komponente.
`OnClick`	Šalje se kada korisnik klikne na bilo koji taster miša.
`OnDblClick`	Ovaj događaj se pojavljuje kada korisnik dva puta klikne na komponentu (dvostruki klik mišem).
`OnEnter`	Ovaj događaj se pojavljuje kada prozorska komponenta primi fokus (kada je aktivirana).
`OnExit`	Ovaj događaj se pojavljuje kada prozorska komponenta izgubi fokus kao rezultat prelaska na drugu kontrolu. Ne dešava se kada korisnik pređe na drugu formu, odnosno na drugu aplikaciju, a ne zatvori tekuću formu, odnosno aplikaciju.
`OnKeyDown`	Ovaj događaj se aktivira kada korisnik pritisne taster, dok je kontrola u fokusu. Tasteri koji se mogu koristiti su svi alfanumerički tasteri na tastaturi kao i kursorski tasteri, `Home`, `End`, `Ctrl` taster, itd.
`OnKeyPress`	Ovaj događaj se aktivira kada korisnik pritisne taster, ali samo ako je pritisnuti taster alfanumerički, `Tab`, brisanje unazad (backspace), `Enter`, ili `Esc`
`OnKeyUp`	Ovaj događaj se aktivira samo kada je taster otpušten.
`OnMouseDown`	Ovaj događaj je aktiviran kada je taster miša pritisnut u trenutku kada se kursor miša nalazi na komponenti. Parametri koji se prosleđuju upravljaču događajem pružaju informaciju koji je taster miša pritisnut, da li su pritisnuti posebni tasteri (`Alt`, `Shift`, `Ctrl`) i `X`,`Y` koordinatu kursora miša u trenutku kada se događaj odigrao.
`OnMouseMove`	Ovaj događaj se pojavljuje svaki put kada se miš pomeri preko kontrole.
`OnMouseUp`	Ovaj događaj je aktiviran kada se otpusti taster miša u trenutku prelas ka kursora preko kontrole. Taster miša prethodno mora biti pritisnut u trenutku kada se kursor nalazi na kontroli.
`OnPaint`	Ovaj događaj se šalje svaki put kada je komponentu potrebno ponovo iscrtati. Možete odgovoriti na ovaj događaj da biste, ukoliko komponenta to zahteva, iscrtali komponentu na poseban način.

Postupanje sa događajima miša

Događaji miša imaju neke osobine kojih bi trebalo da se pazite. Ukoliko odgovarate samo na klik mišem na komponentu, poželećete da sve ostane jednostavno i da odgovorite samo na događaj `OnClick`. Ukoliko morate da koristite događaje `OnMouseDown` i `OnMouseUp`, trebalo bi da pazite na činjenicu da će događaj `OnClick` biti poslat zajedno sa događajima `OnMouseDown` i `OnMouseUp`. Na primer, jednostruki klik mišem će kao rezultat dati sledeće događaje (događaji će biti poređani ovim redom):

```
OnMouseDown
OnClick
OnMouseUp
```

Slično, kada korisnik dva puta klikne mišem (dvostruki klik mišem), kao rezultat aplikacija će primiti više događaja nego što mislite. Kada se na komponentu dva puta klikne mišem, aktiviraju se sledeći događaji:

```
OnMouseDown
OnClick
OnDblClick
OnMouseUp
```

Cilj koji sam želeo da postignem je da Vas upozorim da vodite računa prilikom odgovaranja na događaje jednostrukog i dvostrukog klika mišem na komponentu. Budite svesni da prilikom dvostrukog klika mišem dobijate četiri događaja.

Prilikom pritiska na taster tastature takođe se odvija više događaja. Pritisak na taster u okviru edit kontrole, na primer, će kao rezultat dati događaje: `OnKeyDown`, `OnKeyPress`, `OnChange` i `OnKeyUp`.

Izvorni kod u okviru knjige (idite na `http://www.mcp.com/info` i upišite `0-672-31286-7`) sadrži program pod nazivom `EventTst`, koji ilustruje činjenicu da se više događaja aktivira prilikom pritiska tastera tastature i pritiska tastera miša. Pokrenite ovaj program i videćete koliko mnogo događaja će biti aktivirano nakon određene aktivnosti korisnika.

Uskoro ćete moći da vidite VCL komponente detaljnije. Prvo bih želeo da Vam predstavim klasu koju koriste određene VCL komponente - `TStrings`.

Klasa `TStrings`

Klasa `TStrings` je VCL klasa koja administrira listom stringova. Nekoliko VCL komponenti koriste slučajeve klase `TStrings`, da bi upravljali podacima klase (obično tekstom). Na primer, u lekciji dana 6 ste koristili klasu `TStrings`, kada ste kreirali aplikaciju `ScratchPad`. "Ne mogu se setiti da sam koristio klasu `TStrings`." - reći ćete. Ustvari jeste, ali niste bili svesni da ste koristili klasu. Da li se sećate kada ste snimali i učitavali datoteke? Koristili ste nešto poput:

```
Memo.Lines.SaveToFile(SaveDialog.FileName);
```

Karakteristika `Lines` klase `TMemo` je slučaj klase `TStrings`. Metoda `SaveToFile` klase `TStrings` uzima stringove i snima ih u datoteku na disku. Možete koristiti istu tehniku, da biste učitali okvir za listu iz datoteke koja se nalazi na disku, odnosno da bi snimili sadržaj okvira za listu na disk. U slučaju klase `TListBox`, karakteristika koja sadrži elemente okvira za listu ima naziv `Items`. Na primer, probajte ovu vežbu:

1. Kreirajte novu aplikaciju i postavite komponentu `ListBox` na formu. Promenite veličinu okvira za listu po želji.
2. Promenite karakteristiku `Name` okvira za listu u `ListBox`.
3. Dva puta kliknite mišem na pozadinu forme (ne na okvir za listu). Editor koda će prikazati funkciju `FormCreate`.
4. Izmenite funkciju `FormCreate` tako da dobije izgled:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  WinDir   : array [0..255] of Char;
  FileName : string;
begin
  GetWindowsDirectory(WinDir, SizeOf(WinDir));
  FileName := WinDir + '\win.ini';
  ListBox.Items.LoadFromFile(FileName);
end;
```

5. Kliknite na dugme Run da biste preveli i pokrenuli program.

Nakon pokretanja programa, okvir za listu će sadržati Vašu datoteku `WIN.INI`. Korišćenjem ove metode na lak način se može učitati u okvir za listu bilo koja ASCII tekst datoteka. Komponenta `ComboBox` takođe ima karakteristiku `Items` koja radi na potpuno isti način.

Možete dodati, obrisati, ubaciti, ili pomeriti stavku u okviru za listu, kombo okviru, odnosno memo polju pozivom metoda `Add`, `Append`, `Delete`, `Insert` i `Move` klase `TStrings`.

Kako će se metoda `Add` ponašati, zavisi od vrednosti karakteristike `Sorted`. Ukoliko je karakteristika `Sorted` podešena na vrednost `True`, metoda `Add` će ubaciti string na mesto gde bi string trebao da se nalazi na listi sortiranih stavki. Ukoliko je karakteristika `Sorted` definisana kao `False`, novi string će biti dodat na kraj liste.

Komponenta može biti oslobođena svog sadržaja pozivom metode `Clear`. Zasebnom stringu se može pristupiti korišćenjem operatora za indeks niza. Na primer, da biste preuzeli prvi string sa liste stringova, možete koristiti:

```
Edit.Text := ListBox.Items[0];
```

NAPOMENA Stringovi u okviru klase `TStrings` su sadržani u karakteristici `Strings`. Karakteristika `Strings` je deklarisana kao generička karakteristika niza za klasu `TStrings`, stoga nije potrebno da pose-bno ukazujete na ovu karakteristiku kada pronalazite određeni string (mada možete ukoliko to želite). Na osnovu toga sledeće dve linije koda su potpuno iste sa gledišta prevodioca:

```
Edit.Text := ListBox.Items[0];
Edit.Text := ListBox.Items.Strings[0];
```

Svaki string u okviru niza `TStrings` sadrži sam string i četiri bajta dodatnog prostora. Dodatnom prostoru se može pristupiti preko karakteristike `Objects`, a možete ga koristiti na način koji Vam odgovara. Na primer, možemo reći da, ukoliko kreirate okvir za listu koji prikazuje bitmape, string možete zapisati na uobičajen način, a pointer na objekt `TBitmap` možete zapisati u niz `Objects`.

SAVET Možda će biti situacija kada želite da administrirate listom stringova koji nisu povezani sa komponentom. Klasa `TStringList` se može koristiti upravo za tu svrhu. Ova klasa radi isto kao i klasa `TStrings`, ali se može koristiti van komponente. Klasa `TStringList` je posebno pogodna za čitanje, manipulaciju i zapisivanje tekst datoteka.

NAPOMENA U stvarnosti, klasa `TStrings` pripada tipu koji se može nazvati fundamentalna apstraktna klasa. Fundamentalna apstrakna klasa se nikada ne koristi direktno. Ona služi samo kao osnovna klasa iz koje se izdvajaju druge klase. Karakteristika `Lines` je ustvari slučaj klase `TMemoStrings`, pre nego slučaj klase `TStrings`, kao što sam Vam objasnio u ovoj lekciji. Ovo može biti zbunjujuće pošto je karakteristika `Lines` deklarisana kao pointer klase `TStrings`, a ustvari je slučaj klase `TMemoStrings`. Deklaracija i kreiranje karakteristike `Lines` izgleda ovako:

```
  Lines : TStrings;
{ ...later }
  Lines := TMemoStrings.Create;
```

Ovo je razlog zašto se karakteristika `Lines` može smatrati slučajem klase `TStrings`, mada to ustvari nije. Nisam mislio da Vas navedem na pogrešan put, ali mislim da je bolje da ove stvari razdvojim nakon objašnjenja klase `TStrings` umesto da Vas zbunjujem ovim informacijama u toku trajanja diskusije.

NOVI TERMIN Fundamentalna apstraktna klasa (*abstract base class*) je klasa koja se ne može koristiti direktno. Klasa potomak mora biti kreirana korišćenjem fundamentalne apstrakne klase, a slučaj klase potomka se koristi umesto potomka fundamentalne apstraktne klase.

Standardne Windows komponente za kontrolu

Vratimo se u doba `Jure`, kada je postojalo nešto što se zvalo Windows 3.0. Windows 3.0 je nudio opcije kao što su edit kontrole (u jednoj, ili više linija), okvire sa listama, kombo okvire, dugmad, polja za potvrdu, radio dugmad i statičke kontrole.

Ove kontrole su morale biti prilično dobro dizajnirane pošto preovladavaju i u Windows programima današnjice - imajući u vidu sve nove Win32 kontrole.

Neću objašnjavati svaku Windows kontrolu i njoj odgovarajuću VCL komponentu. Postoji nekoliko stvari koje bi trebalo da znate u vezi standardnih komponenti, a koje su objašnjene u sledećim poglavljima.

Pozvaću se na komponente na jedan od dva načina: nazivom komponente, ili nazivom VCL klase koja definiše komponentu. Možda ću napisati: "Komponenta `Label` se koristi za ...", odnosno možda ću napisati: "Klasa `TLabel` se koristi za ...". U oba slučaja mislim na istu komponentu.

`Edit` kontrole

Delphi sadrži četiri edit kontrole. Komponente `Edit`, `Memo` i `MaskEdit` se baziraju na standardnim Windows edit kontrolama. Kompomnenta `RichEdit` je bazirana na Win32 proširenoj edit kontroli koja ne pripada standardnim Windows kontrolama. Ipak ću u ovom delu obraditi kontrolu `RichEdit`, pošto sadrži mnogo mogućnosti koje su zajedničke sa ostalim edit kontrolama.

Komponenta `Edit`

Komponenta `Edit` enkapsulira osnovnu jednolinijsku kontrolu. Ova komponenta nema karakteristiku `Align`, odnosno `Alignment`. Karakteristika `Alignment` ne postoji pošto tekst u jednolinijskoj edit kontroli može biti samo poravnat na levo. Komponenta `Edit` nema `Align` karakteristiku pošto ne može (odnosno preciznije ne bi mogla) da se proširi da ispuni klijent oblast prozora.

Ukoliko je potrebno da tekst u okviru edit komponente bude desno poravnat, odnosno centriran, koristite komponentu `Memo`, ali postavite njenu visinu na visinu standardne edit komponente. Zatim, ukoliko je to potrebno, podesite karakteristiku `Alignment`.

Neka Vaše forme budu standardizovane kad god je to moguće. Iako možete podesiti visinu `Edit` komponente po želji, možete zbuniti korisnike ukoliko visina komponente bude veća od standardne Windows edit kontrole (može izgledati korisniku kao edit kontrola sa više linija).

Komponenta `MaskEdit`

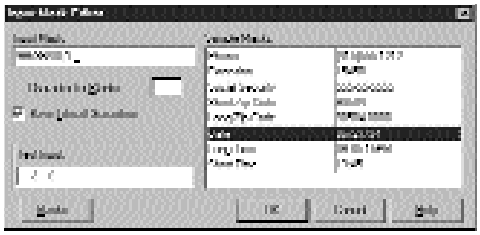
Komponenta `MaskEdit` je `Edit` komponenta kojoj je dodat filter za unos, odnosno maska. Komponenta `MaskEdit` ne predstavlja zasebnu Windows kontrolu, nego samo VCL proširenje standardne edit kontrole. Maska se koristi da bi primorala korisnika da unese broj u određenom opsegu, odnosno određeni broj karaktera. Maska dodatno može sadržati posebne karaktere koji se mogu postaviti gernerički u edit kontrolu. Na primer, datum se obično upisuje u formatu:

```
03/21/98
```

Maska za editovanje koja se koristi za datum može sadržati kose crte na određenim mestima, tako da korisnik može uneti samo brojeve. Maska za editovanje može odrediti da je moguće uneti samo brojeve, kako bi se izbegla mogućnost da korisnik unese karakter koji nije broj.

NAPOMENA Komponenta `DateTimePicker` (nalazi se na kartici Win32) omogućava da se odabere datum, ili vreme iz specijalizovane edit komponente. Ukoliko je karakteristika Kind podešena na `dtkDate`, komponenta prikazuje kalendar iz kog korisnik može izabrati datum. Kada je karakteristika `Kind` podešena na `dtkTime`, komponenta `DateTimePicker` prikazuje edit kontrolu sa više polja koja omogućava korisniku da podesi sate, minute, sekunde i AM (pre podne - ante meridiem) ili PM (posle podne - post meridiem). Komponenta `DateTimePicker` je bolja od komponente `MaskEdit` za unos datuma i vremena.

Karakteristika `EditMask` kontroliše masku koja se koristi. Kada kliknete na dugme sa tri tačke u okviru kolone Value za karakteristiku `EditMask`, biće prikazan Input `MaskEditor`. Ovaj okvir za dijalog Vam omogućava da izaberete jednu od unapred definisanih maski, odnosno da kreirate sopstvenu. Možete odabratri unapred definisane maske za nekoliko država. Slika 7.3 prikazuje okvir za dijalog `Input MaskEditor` sa prikazanoim setom unapred definisanih maski za Sjedinjene Države.



Slika 7.3
Okvir za dijalog Input Mask Editor

Za dodatne informacije o kreiranju Vaših posebnih maski, pogledajte Delphijev program za pomoć u toku rada.

Komponenta Memo

Komponenta `Memo` enkapsulira edit kontrolu sa više linija. Karakteristika `Lines` je najznačajnija karakteristika komponente `Memo`. Kao što sam napomenuo u prethodnomn delu knjige, kada je bila obrađena klasa `TStrings`, karakteristika `Lines` Vam omogućava da snimite sadržaj komponente `Memo` na disk, učitate u kompone-ntu `Memo` tekst iz datoteke, odnosno pristupite svakoj zasebnoj liniji u okviru memo komponente.

Karakteristika `ScrollBars` je jedinstvena za komponentu `Memo`. Ova karakteristika Vam omogućava da definišete da li Vaša komponenta ima horizontalnu traku za pomeranje, vertikalnu traku za pomeranje, odnosno da li ima obe trake za pomeranje. Karakteristike `ScrollBars` ste koristili u lekciji dana 6, kada ste pisali aplikaciju `ScratchPad`. Komponenta `Memo` je veoma upotrebljiva komponenta koju ćete verovatno veoma često koristiti.

Komponenta RichEdit

Komponenta RichEdit je najveća i najbolja od svih edit komponenti; bazirana je na Win32 proširenoj edit kontroli. Komponenta RichEdit Vam omogućava da menjate font, koristite nazubljivanje teksta, podesite font na tip koji je podebljan, kurziv, ili podvučen i još mnogo toga. U osnovi, komponenta RichEdit je mali tekst procesor u džepnom pakovanju. Komponenta RichEdit ima iznenađujuće malo karakteristika u toku dizajniranja naspram broja karakteristrika koje komponenta Memo poseduje.

Glavne karakteristike u toku rada programa uključuju SelAttributes i Paragraph. Komponenta RichEdit je kompleksna, ali jednostavna za korišćenje ukoliko uzmemo u obzir njenu kompleksnost. Pogledajte Delphijev program za pomoć u toku rada za detaljniji pregled komponente RichEdit.

Uobičajene karakteristike edit kontrole

Tabela 7.4 prikazuje karakteristike koje su specifične za komponente bazirane na edit kontrolama.

Tabela 7.4: Karakteristike za edit kontrole

Opcija	Odnosi se na	Opis
		Karakteristike
AutoSelectEdit,	MaskEdit	Kada je podešeno na True, tekst u okviru edit kontrole će automatski biti označen kada korisnik pritiskom na taster Tab pređe na kontrolu. Generička vrednost: True.
AutoSizeEdit,	MaskEdit	Kada je podešeno na True, edit kontrola će automatski promeniti veličinu kada se fontedit kontrole promeni. U suprotnom, edit kontrola ne menja veličinu prilikom promene veličine fonta. Generička vrednost: True.
CharCaseEdit,	MaskEdit	Određuje da li će edit kontrola prikazivati velika slova (ecUpperCase), mala slova(ecLowerCase) ili normalan tekst (ecNormal). Generička vrednost: ecNormal.
HideScrollBars	RichEdit	Kada je podešena na True, trake za pomeranje će biti prikazane, ukoliko je to potrebno, u protivnom će biti skrivene. Kada je podešena na False, trake za pomeran je će biti prikazane na osnovu vrednosti karakteristike ScrollBars.

nastavlja se

Tabela 7.4: Karakteristike za edit kontrole *nastavak*

Opcija	Odnosi se na	Opis
`HideSelection`	`Edit, Memo, RichEdit`	Kada je podešeno na `True`, tekst koji je označen neće biti prikazan kao označen pri likom prelaska korisnika na narednu kon trolu pritiskom na taster Tab. Generička vrednost: `False`.
`Lines`	`Memo, RichEdit`	Tekst koji se sadrži u komponenti. `Lines` je slučaj klase `TStrings`.
`MaxLength`	sve	Definiše maksimalan broj karaktera koje će komponenta moći da sadrži. Ukoliko je vrednost `0`, može se uneti neograničena dužina teksta (ograničena samo zahtevima sistema). Kada je podešena na bilo koju vrednost različitu od nule, broj karaktera je ograničen zadatom vrednosti. Generička vrednost: `0`.
`OEMConvert`	`Edit, Memo`	Podesite ovu karakteristiku na `True`, kada tekst koji se unosi sadrži naziv datoteke. Generička vrednost: `False`.
`PasswordChar`	`Edit, MaskEdit`	Kada je ova karakteristika podešena na vrednost različitu od ASCII `#0`, tekst koji unosite će biti prikazan korišćenjem defin isanih karaktera. Tekst u okviru edit kont role neće biti izmenjen. Većina kontrola lozinki koristi zvezdicu (`*` - asterisk) za sakrivanje lozinke. Generička vrednost: `#0`.
`PlainText`	`RichEdit`	Kada je podešena na `True`, RTF (Rich Text Format) datoteke će biti prikazane kao običan tekst bez formatiranja karaktera i paragrafa. Kada je podešena na `False` RTF datoteke će biti prikazane u punom for matu. Generička vrednost: `False`.
`ReadOnly`	sve	Kada je podešena na `True`, komponenta će prikazivati tekst koji se ne može menjati. Korisnik može označiti tekst i kopirati ga na Clipboard. Generička vrednost: `False`.
`ScrollBars`	`Memo, RichEdit`	Određuje koje trake za pomeranje će prikazati. Izbori su: `ssNone`, `ssBoth`, `ssHorizontal`, `ssVertical`. Generička vrednost: `ssNone`.

Opcija	Odnosi se na	Opis
`Text`	`Edit`, `MaskEdit`	Sadrži tekst u okviru komponente.
`WantReturns`	`Memo`, `RichEdit`	Kada je podešena na `True`, komponenta zadržava karakter za kraj reda i novu liniju koju korisnik unosi u edit kontrolu pritiskom na taster Enter. Kada je podešena na `False`, karakteri za kraj reda i novu lini ju ulaze u formu, ali se ne postavljaju u okviru edit kontrole. Ukoliko imate formu sa jednim dugmetom i kontrola `WantReturns` je podešena na `False`, pritisak na taster Enter će zatvoriti formu. Generička vrednost: `True`.
`WantTabs`	`Memo`, `RichEdit`	Kada je podešena na `True`, kada korisnik pritisne taster Tab u edit kontrolu se upisuje karakter tab. Kada je postavljen na `False`, tab karakteri prelaze na formu, što omogućava izlazak iz edit kontrole pri tiskom na taster Tab. Generička vrednost: `False`.
`WordWrap`	`Memo`, `RichEdit`	Kada je podešena na `True`, tekst koji je unet će se prebaciti na novu liniju kada dostigne desnu ivicu edit kontrole. Kada je podešena na `False`, edit kontrola se automatski pomera, kako se novi tekst unosi. Generička vrednost: `True`.
		Karakteristike u toku rada
`Modified`	sve	Označava da li je sadržaj edit kontrole izmenjen nakon što je poslednji put karak teristika `Modified` podešena. Nakon sni manja sadržaja komponente `Memo`, ili `RichEdit` u datoteku, treba da promenite `Modified` u False.
`SelLength`	sve	Sadrži dužinu teksta koji je trenutno odabran u okviru edit kontrole.
`SelStart`	sve	Sadrži početnu poziciju odabranog teksta u okviru edit kontrole. Prvi karakter u okviru edit kontrole je 0.
`SelText`	sve	Sadrži trenutno odabran tekst u okviru edit kontrole.

Edit kontrole imaju mnogo zajedničkih metoda kojih ima suviše da bi ovde bile prikazane. Metode `CutToClipboard`, `CopyToClipboard`, `PastFromClipboard` i `Clear`, rade sa Windows Cipboard-om i manipulišu tekstom. `GetSelTextBuff` i `GetTextBuff` metode preuzimaju odabrani tekst u okviru komponente, odnosno preuzimaju kompletan tekst u okviru komponente. Pogledajte Delphijev program za pomoć u toku rada, koristeći ključne reči `TEdit`, `TMaskEdit`, `TMemo` i `TRichEdit` za kompletan spisak metoda koje su pridružene svakoj od navedenih komponeti.

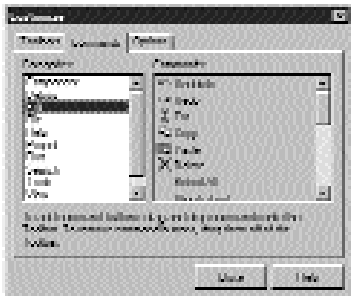
Događaji edit komponenti za koje ste najviše zainteresovani zavise od tipa edit kontrole koju koristite. U većini slučajeva najčešće se koriste događaji `OnEnter`, `OnExit`, `OnChange`, `OnKeyDown` (ili `OnKeyPress`) i `OnKeyUp`.

Komponente `ListBox` i `ComboBox`

Komponente `ListBox` i `ComboBox` se veoma često koriste. Komponenta `ListBox` predstavlja standardni Windows okvir za listu, koja predstavlja listu za izbor opcija koje korisnik može odabrati. Ukoliko okvir za listu sadrži više opcija nego što se može prikazati na prozoru liste, trake za pomeranje teksta omogućavaju pristup ostalim opcijama u okviru za liste.

NOVI TERMIN Neki okviri za liste su tip okvira za liste koje definiše korisnik. Kod okvira za listu koju definiše korisnik, programer preuzima odgovornost za unos opcija okvira za listu.

Ukoliko je potrebno možete kreirati okvir za listu koju definiše korisnik. Ova vrsta kontrole se veoma retko koristi pa je možda i nećete primenjivati. U lekciji dana 4 sam objasnio prilagođavanje Delphijeve trake sa alatima. Kao deo objašnjenja, bio je korišćen okvir za dijalog ToolBar Editor. Okvir za dijalog ToolBar Editor sadrži dva okvira za liste (pogledati sliku 7.4).



Slika 7.4
Okvir za dijalog ToolBar Editor sa okvirom za listu Commands koji je korisnički definisan okvir za listu

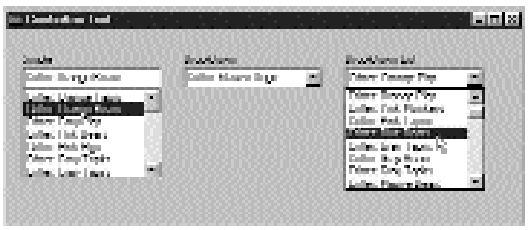
Okvir za listu na levoj strani je običan okvir za listu; prikazuje grupe dugmadi koje možete odabrati. Okvir za listu na desnoj strani je korisnički definisan okvir za listu; pokazuje dugmad koja će se pojaviti na traci sa alatima, kao i tekst opisa funkcije koju dugme izvršava.

Kombo okviri su specijalizovani okviri za liste. Ustvari, kombo okvir je kombinacija okvira za listu i edit kontrole. Korisnik može odabrati opciju sa liste, odnosno upisati vrednost u deo za editovanje. Kada korisnik odabere opciju sa liste, opcija se postavlja u edit kontrolu. Postoje tri različita tipa kombo okvira. Tip kombo okvira je određen karakteristikom `Style`. Tabela 7.5 prikazuje tipove kombo okvira i opis svakog od njih.

Tabela 7.5: Tipovi kombo okvira

Opcija	Opis
Simple	Jednostavan stil kombo okvira nije ništa drugo do edit kontrola postavl na na vrh liste. Korisnik može da odabere opciju sa liste, odnosno da upiše tekst u deo za editovanje.
Drop-down	Sličan jednostavnom stilu, s tom razlikom što deo koji predstavlja okvir za listu inicijalno nije prikazan. Postoji dugme sa strelicom koje korisnik može pritisnuti kako bi video listu i odabrao opciju. Korisnik kao i u prethodnom slučaju može upisati tekst u deo za editovanje.
Drop-down list	Ovo je najrestriktivniji tip kombo okvira. Kao i kod prethodnog stila, lista inicijalno nije prikazana. Korisnik može kliknuti na dugme sa strelicom na dole, kako bi lista bila prikazana, a zatim može odabati opciju sa liste, ali ne može uneti tekst u deo za editovanje. Ovaj stil možete koris titi kada želite da korisnik bira samo unapred definisane opcije.

Izvorni kod u okviru knjige, sadrži program pod nazivom `ComboTst` koji ilustruje različite tipove kombo okvira. Slika 7.5 prikazuje rad test programa. Pokrenite program i isprobajte kombo okvire, kako biste videli kako koji tip radi.



Slika 7.5
Program `ComboTst`

Tabela 7.6 prikazuje listu karakteristika koje su uobičajene za okvire sa listom i kombo okvire.

Tabela 7.6: Karakteristike za edit kontrole

Karakteristika	Odnosi se na	Opis
		Karakteristike
`Columns`	`ListBox`	Sadrži broj kolona okvira za listu. Možete kreirati više kolona, ukoliko upišete vrednost veću od `1`. Generička vrednost: `0`.
`ExtendedSelection`	`ListBox`	Određuje da li je dozvoljeno imati proširen izbor. Proširen izbor omogućava korisniku da odabere opcije korišćenjem kombinacije Shift+klik na taster miša i Ctrl+klik na taster miša. Ukoliko je `MultiSelect` podešen na `False`, ova karak teristika nema efekta. Generička vrednost: `True`.
`IntegralHeight`	`ListBox`	Kada je vrednost karakteristike `True`, visina okvira za listu će biti podešena tako da se ne prikazuje deo linije teksta. Kada je `False`, okvir za listu može prikazati deo linije teksta. Generička vrednost: `False`.
`ItemHeight`	oba	Koristi se za korisnički definisane okvire za listu i kombo okvire. Podešava visinu opcije u okviru kont role. Generička vrednost: 13.
`Items`	oba	Slučaj klase `TStrings` koji sadrži listu opcija okvi ra za listu. (Pogledajte poglavlje o klasi `TStrings` u prethodnom delu današnje lekcije za opis dostupnih karakteristika i metoda.)
`MaxLenght`	`ComboBox`	Maksimalan broj karaktera koje korisnik može upisati u deo za editovanje kombo okvira. Ista je kao `MaxLength` kod edit kontrola. Generička vred nost:`0` (bez ograničenja).
`MultiSelect`	`ListBox`	Ukoliko je vrednost `True`, iz okvira za listu možete odabrati više opcija. Generička vrednost: `False`.
`Sorted`	oba	Kada je podešeno na `True`, opcije okvira za listu su sortirane u rastućem redosledu. Kada je podešeno na `False`, opcije nisu sortirane. Generička vrednost: `False`.
`Style`	`ComboBox`	Stil kombo okvira. Mogući stilovi: csSimple, `csDropDown`, `csDropDownList`, `lbOwnerDrawFixed` `csOwnerDrawVariable`. (Pogledati tabelu 7.5 za opis tri osnovna stila.) Generička vrednost: `csDropDown`.

Karakteristika	Odnosi se na	Opis
	ListBox	Stilovi za okvire listi su: lbStandard, lbOwnerDrawFixed i csOwnerDrawVariable. Generička vred nost: lbStandard.
TabWidth	ListBox	Okviri za liste mogu koristiti tabulatore. Ova karakter istika podešava poziciju tabulatora u pikselima. Generička vrednost: 0.
Text	ComboBox	Sadrži tekst u edit delu kombo okvira.
		Karakteristike u toku rada
ItemIndex	ListBox	Sadrži indeks trenutno odabranih opcija, gde je 0 prva opcija na listi. Vraća -1 ukoliko nije odabrana ni jedna opcija. Kada se upisuje u karakteristiku, vraća indeks opcije.
SelCount	ListBox	Sadrži broj opcija odabranih u okviru za listu koji može prihvatiti izbor više opcija.
Selected	ListBox	Vraća vrednost True, ukoliko je definisana opcija odabrana, odnosno False ukoliko nije odabrana.
SelLengt	ComboBox	Sadrži dužinu teksta koji je trenutno odabran u okviru edit kontrole koja je deo kombo okvira.
SelStart	ComboBox	Sadrži početnu poziciju odabranog teksta u edit kon troli. Prvi karakter u okviru edit kontrole je 0.
SelText	ComboBox	Sadrži trenutno odabrani tekst u okviru edit kontrole.
TopIndex	ListBox	Vraća opciju okvira za listu koja je na početku liste. Može se koristiti da podesi određenu opciju kao prvu.

Kao što je to slučaj sa edit komponentama koje ste učili u prethodnom delu današnje lekcije, postoji veoma malo metoda za ListBox i ComboBox. Metoda Clear briše kontrole od svih podataka. Metode ItemAtPos vraćaju vrednost opcije okvira za listu na određenu X i Y koordinatu. Metoda SelectAll odabira tekst koji se nalazi u delu kontrole za editovanje kombo okvira.

Jednostavni i najčešće korišćeni događaji koji rade sa kombo okvirima i okvirima za liste su OnChange i OnClick. Ove događaje možete koristiti za utvrđivanje da li je odabrana opcija okvira za listu.

Klik mišem na deo za editovanje kombo okvira, odnosno na dugme za spuštanje liste, ne daje kao rezultat slanje događaja OnClick. Događaj OnClick će se aktivirati samo kada se klikne na deo kombo okvira koji je predstavljen listom.

Događaj OnChange se može koristiti da registruje promene dela za editovanje kombo okvira, a koristi se isto kao kod svake druge edit kontrole. Događaj OnDropDown se koristi da registruje pritisak dugmeta za spuštanje liste kombo

okvira. Događaji `OnMeasureItem` i `OnDrawItem` se koriste sa korisnički definisanim okvirima za listu i korisnički definisanim kombo okvirima.

Tipovi VCL dugmadi

VCL sadrži nekoliko tipova dugmadi koje možete da koristite u Vašim aplikacijama. Iako nisu svi bazirani na standardnim Windows kontrolama za dugmad, ipak ću ih sve objasniti. Pre nego što pređemo na pojedine komponente dugmadi, obradićemo neke osnovne pojmove.

Kada podešavate karakteristiku `Caption` dugmeta, koristite znak & kao što ste to činili podešavajući karakteristiku `Caption` opcija menija. Karakter nakon znaka & će biti podvučen i koristiće se kao prečica za dugme.

Karakteristike dugmadi

Komponente Button imaju samo četiri karakteristike koje treba spomenuti:

- `ModalResult`
- `Default`
- `Cansel`
- `Enabled`

Karakteristika `ModalResult`. Karakteristika `ModalResult` se koristi za zatvaranje formi koje su prikazane kao modalne. Generička vrednost karakteristike je `mrNone` (što je 0). Ove vrednosti možete koristiti za dugmad koje se koriste kao regularna dugmad na formi, a ne zatvaraju formu. Ukoliko koristite bilo koju vrednost različitu od 0 za karakteristkriku `ModalResult`, klik mišem na dugme će zatvoriti formu i vratiti vrednost karakteristike `ModalResult`. Na primer, ako postavite dugme na formu i podesite karakteristiku `ModalResult` na `mrOk`, klikom miša na dugme forma će biti zatvorena, a vrednost koja se vraća iz karakteristike `ShowModal` će biti `mrOk` (1). Na osnovu datog, možete učiniti nešto što liči na naredni isečak koda:

```
var
  Res : Integer;
begin
  Res := MyForm.ShowModal;
  if Res = mrOK then
    DoSomething;
  if Res = mrCancel then
    Exit;
end;
```

Tabela 7.7 prikazuje konstante `ModalResult` koje definiše VCL.

Tabela 7.7: VCL `ModalResult` konstante

Konstanta	Vrednostmr
`None`	`0`
`mrOk`	`1`
`mrCancel`	`2`
`mrAbort`	`3`
`mrRetry`	`4`
`mrIgnore`	`5`
`mrYes`	`6`
`mrNo`	`7`
`mrAll`	`8`
`mrNoToAll`	`9`
`mrYesToAll`	`10`

Za Vašu dugmad ne treba da koristite već unapred definisane konstante `ModalResult`; možete koristiti vrednost koju želite. Na primer, ukoliko imate okvir za dijalog koji ste sami kreirali i koji može da se zatvori korišćenjem nekoliko dugmadi; svakom dugmetu možete dodeliti drugu vrednost konstante `ModalResult` (na primer: `100`, `150` i `200`), pa ćete znati kojim dugmetom je zatvoren okvir za dijalog. Bilo koja vrednost različita od nule je dozvoljena, a maksimalna dozvoljena vrednost je ujedno i maksimalna dozvoljena vrednost tipa `Integer`.

Izvorni kod u okviru knjige sadrži program pod nazivom `ButtnTst` koji demonstrira korišćenje konstante `ModalResult`. Program Vam omogućava da izvršite formu koja sadrži nekoliko dugmadi. Nakon klika mišem na dugme, vrednost konstante `ModalResult` će biti prikazana na glavnoj formi.

Karakteristika `Default`. Karakteristika `Default` je još jedna iz grupe karakteristika dugmadi. Windows ima standardni mehanizam za rad sa okvirima za dijalog. Jedna od mogućnosti ovog mehanizma izgleda ovako: Ukoliko kontrola koja nije dugme ima fokus tastature, a korisnik pritisne taster Enter na tastaturi, okvir za dijalog će se ponašati kao da je korisnik kliknuo na dugme koje je definisano kao glavno (*default*).

Dugme difinisano kao glavno, je ono dugme koje ima stil podešen na `BS_DEFPUSH-BUTTON` (najčešće je to dugme OK). Ova mogućnost je godinama zadavala muke programerima i predstavljala je prokletstvo za osoblje koje je unosilo podatke. Karakteristika `Default` se koristi da podesi dugme kao glavno na formi. Generička vrednost za ovu karakteristiku je `False`. Da bi definisali dugme kao glavno, podesite njegovu karakteristiku `Default` na `True`. Ukoliko ne podesite karakteristiku `Default` bilo kog dugmeta na `True`, forma se neće zatvoriti nakon pritiska korisnika na taster Enter.

NAPOMENA Kada korisnik zatvara formu pritiskom na taster Enter, upravljač događajem `OnClick` glavnog dugmeta (ukoliko postoji) će biti pozvan pre zatvaranja forme.

Karakteristika `Cancel` radi sa tasterom Esc na način na koji karakteristika Default radi sa tasterom Enter. Kada korisnik pritisne taster Esc da bi zatvorio formu, vrednost koju vraća metoda `ShowModal` će biti vrednost konstante `ModalResult` za dugme čija je `Cancel` karakteristika podešena na `True`. Ukoliko dugme nema podešenu karakteristiku `Cancel` na `True`, biće vraćeno `mrCancel`, ukoliko je korisnik koristio taster Esc, kao način da se zatvori forma (`mrCancel`) je jednako 2; videti tabelu 7.7.

NAPOMENA Zatvaranje forme klikom miša na sistemsko dugme za zatvaranje, odnosno pritiskom na tastere Alt+F4 kao rezultat daje `mrCancel` koje vraća metoda `ShowModal`, kao što ste i mogli da očekujete. Pritiskom na taster Esc rezultat koji se upisuje u karakteristriku `ModalResult` će biti vrednost dugmeta čija je karakteristika `Cancel` podešena na `True`. Upravljač događajem `OnClick` za dugme `Cancel` će biti pozvan pošto se forma zatvara. Ukoliko korisnik koristi sistemsko dugme za zatvaranje forme, odnosno tastere Alt+F4, ne poziva se ni jedan upravljač događajem `OnClick`. Budite sigurni da će predvideti sve načine na koje korisnik može koristiti (ili zloupotrebiti) Vašu formu.

NAPOMENA Možete imati više dugmadi čija je karakteristika `Default` podešena na `True`. Slično, možete imati više dugmadi čija je karakteristika `Cancel` podešena na `True`. Ipak, kada korisnik pritisne taster Enter, prvo dugme u redosledu tabulatora koje ima karakteristiku Default, podešenu na `True`, će biti pozvano. Slično, kada korisnik pritisne taster Esc za zatvaranje forme, vrednost `ModalResult` konstante će se promeniti na osnovu prvog dugmeta u redosledu tabulatora čija je karakteristika `Cancel` podešena na `True`.

Karakteristika `Enabled`. Karakteristika `Enabled` je bila obrađena u prethodnom delu knjige, kada su bile obrađene komponente uopšte. Ova karakteristika se obično koristi sa dugmadima da aktivira, odnosno deaktivira dugme u zavisnosti od trenutnog stanja programa, odnosno od određene forme. Kada je dugme neaktivno (karakteristika `Enabled` je podešena na `False`), tekst natpisa postaje siv, a dugme ne funkcioniše. U slučaju kada dugmad imaju bitmape (`BitBtn` i `SpeedButton`), automatski će i bitmapa postati siva.

Komponente dugmad imaju samo jedan interesantan metod: metod `Click` koji simulira klik tastera miša. Kada pozovete metodu `Click` na dugme, događaj `OnClick` dugmeta će se izvršiti kao da je korisnik kliknuo na miša. Što se događaja tiče, tipično je da se samo događaj `OnClick` koristi.

Sada ćemo obraditi različite komponente tipa dugme koje Delphi sadrži.

Komponenta `Button`. Standardna komponenta `Button` je poput glumca Denija De Vita: nije lepa, ali sigurno može mnogo toga da uradi. Gotovo se ništa ne može dodati što se odnosi na standardnu komponentu `Button`. Generička karakteristika `Height` (visina) je 25 piksela, a generička karakteristika `Width` (širina) je 75

piksela. Karakteristično je da ćete postaviti dugme na formu da odgovori na događaj `OnClick`; i to bi bilo sve.

Komponenta `BitBtn`. Komponenta `BitBtn` je odličan primer kako se komponenta može proširiti da pruži dodatnu funkcionalnost. U ovom slučaju standardna komponenta `Button` je proširena da omogući prikazivanje bitmape na licu dugmeta.

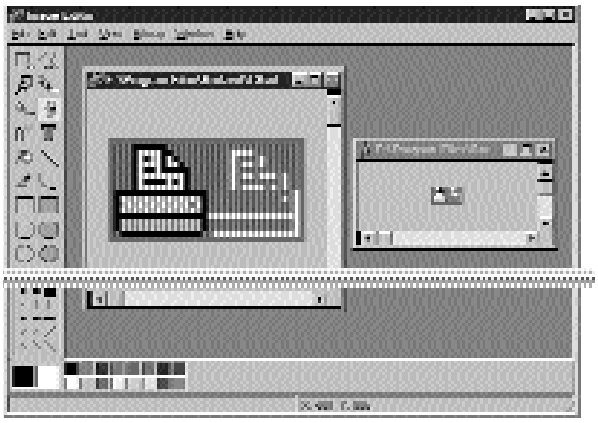
Komponenta `BitBtn` ima nekoliko korakteristika koje su dodate u odnosu na komponentu `Button`. Sve ove karakteristike rade zajedno, da bi omogućile da se prikažu zajedno bitmapa i tekst na licu dugmeta. Dodatne karakteristike su objašnjene u narednim poglavljima.

Karakteristika `Glyph`. Karakteristika glyph predstavlja bitmapu na dugmetu. Vrednost karakteristike `Glyph` je slika, odnosno grafika.

NOVI TERMIN Grafika (*glyph*) je slika koja je obično u formi Windows bitmapirane datoteke (BMP).

Grafika se sastoji od jedne, ili više bitmapa koje predstavljaju četiri moguća stanja dugmeta, a ona mogu biti: podignuto, pritisnuto, deaktivirano, ili stalno pritisnuto. Ukoliko kreirate Vašu dugmad, verovatno ćete mu postaviti samo jednu grafiku, koju će zatim komponenta `BitBtn` modifikovati kako bi prikazala preostala tri moguća stanja. Bitmapa će biti pomerena na dole i desno kada se na dugme klikne mišem, odnosno posiveti kada je dugme deaktivirano. Dugme koje je stalno pritisnuto će izgledati isto i u gornjem i u donjem položaju; naravno dugme će promeniti izgled lica, da bi odalo utisak pritisnutog dugmeta.

Ukoliko definišete više grafika, one moraju biti istih dimenzija i moraju se nalaziti na istoj traci bitmape. Bitmapa koja se isporučuje sa Delphijem, sadrži dve grafike. Slika 7.6 prikazuje bitmapu dugmeta za štampanje koja se nalazi u okviru Delphija (`print.bmp`) u aktuelnoj veličini i uvećano, kako bi se videli detalji. Uočite da ove dve grafike zauzimaju istu širinu na bitmapi.



Slika 7.6
Bitmapa
`PRINT.BMP`

SAVET Piksel u donjem levom uglu bitmape definiše boju koja će biti korišćena kao transparentna. Bilo koji piksel na bitmapi koji ima definisanu boju će biti transparentan kada se grafika prikaže na dugmetu. Prilikom definisanja Vaših bitmapa morate ovo imati na umu. Ukoliko ne želite da Vam bitmapa bude transparentna, piksel u donjem levom uglu mora imati boju koja se ne nalazi u okviru bitmape. Ukoliko ne želite da donji levi piksel definiše transparentnu boju, možete podesiti karakteristiku TransparentMode kao `tmFixed`, a zatim karakteristiku `TransparentColor` u bilo koju boju po Vašem izboru.

Da biste definisali grafiku za komponentu `BitBtn`, dva puta kliknite na kolonu Value u okviru prozora Object Inspector, pored karakteristike `Glyph`. Biće prikazan prozor Picture Editor i moći ćete da odaberete bitmapu koju ćete koristiti kao grafiku.

NAPOMENA Standardne grafike za dugmad koja se nalaze u okviru Delphija imaju dimenzije 15x15 piksela. Ova veličina se dobro uklapa u standardnu visinu dugmeta koja iznosi 25 piksela. Vaše grafike mogu biti bilo koje veličine, ali komponente `BitBtn` neće promeniti veličinu dugmeta u skladu sa veličinom bitmape. Ukoliko koristite veće grafike, na osnovu njih morate prilagoditi i veličinu dugmeta.

Karakteristika `Kind`. Karakteristika `Kind` je veoma dobra mogućnost komponente `BitBtn` zato što Vam omogućava da odaberete neki od unapred definisanih tipova dugmadi. Generička vrednost za karakteristiku `Kind` je `bkCustom`, što znači da ćete samostalno obezbediti grafiku i podesiti sve ostale karakteristike dugmeta. Izbor bilo kog unapred određenog tipa dugmeta kao rezultat može dati pet događaja:

- Karakteristika `Glyph` se automatski podešava za odabrani tip dugmeta.
- Karakteristike `Cancel`, ili `Default` će biti izmenjene u odnosu na tip dugmeta koji je odabran.
- Karakteristika `Caption` će biti izmenjena u zavisnosti od odabranog tipa dugmeta.
- Karakteristika `ModalResult` će biti podešena u zavisnosti od tipa dugmeta koje je odabrano.
- Dugme na formi će biti izmenjeno kako bi prikazalo sve ove promene.

Na primer, ukoliko u karakteristiku `Kind` upišete `bkOK`, ovo dugme će postati dugme OK. Grafika će biti predstavljena znakom za odabrano (`check mark`) zelene boje, karakteristika `Cancel` će dobiti vrednost `False`, karakteristika Default će dobiti vrednost `True`, karakteristika `ModalResult` će biti podešena na `mrOk`, karakteristika `Caption` će dobiti vrednost OK i izmenjene vrednosti će biti prikazane u okviru forme. Možete uvek preskočiti karakteristike koje je karakteristika `Kind` promenila, ali nije baš uvek običaj da se tako nešto radi. Slika 7.7 prikazuje program `Button` Test koji se nalazi u okviru izvornog koda ove knjige; na formi su prikazani razni tipovi komponente `BitBtn`. Forma sadrži svu unapred definisanu dugmad i još jedno dugme koje je posebno definisano (`custom button`).

Karakteristika `Layout`. Karakteristrika `Layout` određuje mesto gde će se relativno u odnosu na tekst nalaziti grafika dugmeta. Generička vrednost je `blGlyphLeft`. Isto tako možete odabrati drugo mesto za postavljanje grafike na dugmetu: desno od teksta, iznad teksta, ili ispod teksta.

Slika 7.7
Unapred definisani tipovi komande `BitBtn`

Karakteristika `Margin`. Karakteristika `Margin` definiše marginu između grafike i kraja dugmeta (čija je ivica definisana karakteristikom `Layout`). Generička vrednost je -1, što centrira grafiku i tekst u okviru prozora. Za definisanje apsolutne margine (u pikselima), možete uneti bilo koju pozitivnu vrednost.

Karakteristika `NumGlyphs`. Karakteristika `NumGlyphs` definiše broj grafika koji se nalazi u okviru trake za određeno dugme. Možete definisati između jedne i četiri grafike, kao što sam napomenuo. Grafike se moraju pojaviti na bitmap traci sledećim redosledom: gore, neaktivno, pritisnuto i stalno pritisnuto.

Karakteristika `Spacing`. Karakteristika `Spacing` kontroliše rastojanje u pikselima između grafike i teksta dugmeta. Generička vrednost je četiri piksela.

Komponenta `SpeedButton`

Komponenta `SpeedButton` je dizajnirana da se koristi sa komponentom Panel za pravljenje traka sa alatima. Razlikuje se od komponenti `Button` i `BitBtn` po tome što nije prozorska komponenta. Ovo znači da dugme prečica ne može primiti ulazni fokus i ne može biti postavljeno u redosled tabulatora.

Sa druge strane, komponenta `SpeedButton` ima nekoliko mogućnosti koje su zajedničke sa komponentom `BitBtn`. Način na koji karakteristika `Glyph` upravlja komponentom `SpeedButton` je, u suštini, isti kao i u slučaju komponente `BitBtn`, pa Vam neću ponovo objašnjavati. Postoji nekoliko bitnih razlika, pa ćemo ih ukratko obraditi.

Generička vrednost dugmadi prečica je kvadrat dimenzija 25x25 piksela. Vaša dugmad prečice mogu biti bilo koje veličine i mogu sadržati tekst, iako dugmad prečice obično nemaju tekst. Neke karakteristike se odnose samo na dugmad prečice, čega bi trebali da budete svesni; karakteristike su obrađene u narednim poglavljima.

NAPOMENA Metod kreiranja traka za alate Delphija 1.0 uključuje korišćenje komponente Panel na koju mogu da se postave razne komponente (uglavnom komponente `SpeedButton`). Delphi 4 sadrži komponentu `Toolbar`, čiji je osnovni cilj kreiranje traka sa alatima. Komponenta `Toolbar` ima dodatne prednosti, ali je nešto komplikovanija za korišćenje.

`GroupIndex`

Dugmad za prečicu mogu biti grupisana tako da se ponašaju kao radio dugmad (radio dugmad će biti obrađena kasnije u današnjoj lekciji u poglavlju,"Radio dugmad i polja za potvrdu"). Kada je pritisnuto jedno dugme u okviru grupe, ono ostaje u tom položaju, a prethodno pritisnuto dugme iskače. (Vraća se u prvobitni položaj.) Da biste grupisali dugmad za prečicu, jednostavno dodelite iste vrednosti karakteristici `GroupIndex` svim dugmadima u okviru grupe. (Generička vrednost `0` označava da dugme nije element ni jedne grupe.) Da biste bolje upoznali ovu mogućnost, pogledajte narednu vežbu:

1. Kreirajte praznu formu i postavite pet dugmadi prečica na formu. (Neću se mučiti dodavanjem grafika na dugmad u ovom jednostavnom primeru, ali ukoliko želite, možete dodati grafike.)
2. Odaberite svu dugmad i promenite vrednost karakteristike `GroupIndex` u `1`. Karakteristika `GroupIndex` za svu dugmad će biti promenjena u `1`.
3. Opciono: promenite karakteristiku `Down` bilo kog dugmeta u `True`.
4. Kliknite na dugme Run, kako bi preveli i pokrenuli program.

Kada pokrenete program, kliknite na nekoliko dugmadi. Uočićete da samo jedno dugme u okviru grupe može biti pritisnuto. Kao što možete videti, ukoliko dodelite vrednost različitu od nule karakteristici `GroupIndex`, dugme za prečicu menja ponašanje. Dugme za prečicu čija je karakteristika `GroupIndex 0`, se vraća u početni položaj kada kliknete mišem, dok se dugme za prečicu koje pripada grupi ne vraća u početni položaj već ostaje pritisnuto kada kliknete mišem na dugme.

`AllowAllUp`

Po pravilu, jedno dugme u okviru grupe mora biti pritisnuto sve vreme. Možete promeniti ovo ponašanje podešavanjem karakteristike `AllowAllUp` na `True`. Ukoliko izmenite karakteristiku `AllowAllUp` za jedno dugme, sva ostala dugmad u okviru grupe će automatski promeniti karakteristiku u `True`. Sada možete da odaberete bilo koje dugme u okviru grupe, odnosno da ne odaberete ni jedno.

SAVET Ponekad želite da se dugme prečica ponaša kao taster. Taster se koristi da uključi, odnosno isključi neku opciju i nije element grupe dugmadi. Da biste pretvorili dugme prečicu u taster, dodelite vrednost različitu od nule karakteristici `GroupIndex`, a karakteristici `AllowAllUp` dodelite vrednost `True`. Uverite se da karakteristika `GroupIndex` nema vrednost koju koristi bilo koja druga komponenta u okviru forme. Kada korisnik klikne mišem na dugme, ono ostaje pritisnuto. Kada ponovo klikne mišem, dugme se vraća u početni položaj.

Down

Karakteristika `Down`, kada se čita, vraća `True` ukoliko je dugme pritisnuto, a `False` ukoliko dugme nije pritisnuto. Kada se upisuju podaci u karakteristiku `Down`, u zavisnosti od upisane vrednosti, dugme će biti pritisnuto, odnosno neće biti pritisnuto. Upisivanje vrednosti u karakteristiku `Down` nema efekta, ukoliko dugme prečica nije deo grupe.

Radio dugmad i polja za potvrdu (radio buttons, check boxes)

Iako su radio dugmad i polja za potvrdu specijalizovana dugmad, ona su ipak, u suštini, dugmad. Ne bih želeo da dugo objašnjavam ove komponente, pošto je njihova implementacija veoma jasna. Obe komponente imaju karakteristiku pod nazivom `Checked` koja može da se koristi za proveru stanja (da li je potvrđeno, ili ne), odnosno može se čitati da bi se trenutno stanje očitalo.

Radio dugmad se obično koriste u grupi. Radio dugmad obično predstavljaju grupu opcija od kojih samo jedna može biti odabrana (kao grupa dugmadi prečica, o kojima ste već učili). Korišćenje pojedinačnih radio dugmadi nije preporučljivo, pošto može zbuniti vaše korisnike. Ukoliko ste u iskušenju da koristite samo jedno radio dugme, umesto njega koristite polje za potvrdu - za to, u suštini, i postoji polje za potvrdu.

Sva dugmad postavljena na jednu formu će automatski biti smatrana elementima iste grupe. Ukoliko imate više od jedne grupe radio dugmadi, a njima treba da se operiše nezavisno, koristite komponentu `RadioGroup`. Ova komponenta Vam omogućava da brzo postavite grupu radio dugmadi sa natpisima i 3D okvirom oko grupe. Da biste bolje shvatrili ovaj koncept, uradite sledeću vežbu:

1. Kreirajte praznu formu, ili iskoristite formu koju ste kreirali u prethodnoj vežbi. Postavite komponentu `RadioGroup` na formu (možete je pronaći na kartici Standard).
2. Pronađite karakteristiku `Items` i dva puta kliknite na kolonu `Value`.
3. Biće prikazan editor liste stringova. Upišite sledeći tekst u string list editor:

```
Redtailed Hawk
Peregrine Falcon
Gyrfalcon
Northern Goshawk
```

4. Kliknite na dugme OK kako bi zatvorili editor string liste. Okvir grupe je popunjen radio dugmadima koje sadrže tekst koji ste kucali.
5. Promenite karakteristiku `Caption` okvira radio grupe u: `Apprentice Falconers Can Legally Possess`.
6. Kliknite na dugme Run, da biste preveli i pokrenuli program.

Kada kliknete na bilo koje radio dugme, prethodno odabrano dugme će iskočiti kao što se može i očekivati. Ukoliko koristite komponentu `RadioGroup`, možete postaviti više grupa radio dugmadi na formu. Kao što je to slučaj sa okvirom za liste i kombo okvirom, komponenta `RadioGroup` ima karakteristiku `ItemIndex` koju možete čitati u toku rada programa, da biste odredili koja opcija grupe je odabrana. Karakteristiku ItemIndex možete isto tako podesiti, kako biste odabrali određeno radio dugme. Možda ste primetili da nijedno radio dugme nije odabrano prilikom startovanja aplikacije. Promenite karakteristiku `ItemIndex` u `0` u okviru prozora Object Inspector, a zatim ponovo pokrenite program. Ovaj put je prvo radio dugme odabrano.

Usput - ako živite u Sjedinjenim Državama, odgovor na kviz pitanje je Redtailed Hawk. (Zadovoljavajući odgovor bi bio i American Kestrel, ali nije na listi.)

Takođe možete koristiti komponentu `GroupBox` za postavljanje radio dugmadi. Komponenta `GroupBox` nije toliko zgodna koliko komponenta `RadioGroup`, ali je mnogo fleksibilnija. U komponentu `GroupBox` možete postaviti bilo koji tip kontrole. Nakon što se postave u okvir grupe, kontrole i okvir grupe mogu biti zajedno pomerane u toku dizajniranja.

Komponenta `CheckBox` se koristi da omogući korisnicima uključivanje, odnosno isključivanje opcije, a može se koristiti i za obaveštavanje korisnika da li je trenutna opcija uključena, ili isključena. Polje za potvrdu može imati maksimalno tri stanja, u zavisnosti od tipa: uključeno, isključeno, ili sivo. Ukoliko je karakteristika `AllowGrayed` polja za potvrdu definisana kao `False`, polje za potvrdu može biti samo odabrano, ili neodabrano. Kada je karakteristika `AllowGrayed` podešena na `True`, polje za potvrdu može imati jedno od tri stanja. Sivo, odnosno neodređeno stanje se definiše programski.

Drugim rečima, na Vama je da se odlučite šta sivo stanje znači u Vašoj aplikaciji. Ukoliko je karakteristika `AllowGrayed` defimnisana kao `False` (generički), možete koristiti karakteristiku `Checked` da odredite da li je polje za potvrdu odabrano, ili nije. Ukoliko je karakteristika `AllowGrayed` definisana kao `True`, morate koristiti karakteristiku `State`, da biste utvrdili (odnosno podesili) stanje polja za potvrdu. Karakteristika `State` može vratiti: `cbChecked`, `cbUnchecked`, ili `cbGrayed`.

Ponekad možete poželeti da koristite polje za potvrdu, da biste ukazali da li su neke mogućnosti uključene, ili isključene, ali ne želite da korisnik ima mogućnost promene stanja klikom miša na polje za potvrdu. U tom slučaju potrebno je da polje za potvrdu bude neaktivno, ali da se ponaša normalno. Da bi polje za potvrdu moglo samo da se čita, a da ne postane sivo, postavite polje za potvrdu na pano, a karakteristiku panoa `Enabled` postavite na `False`.

Komponenta `Label`

Komponenta `Label` se koristi da prikaže tekst na formi. Ponekad se natpis definiše u toku dizajniranja i nikad se ne menja. U drugim slučajevima natpis je izmenljiv i menja se u toku rada programa na osnovu zahteva. Promena naziva u toku rada

programa se može vršiti promenom karakteristike `Caption`. Komponenta `Label` nema posebnih metoda, ili događaja; dostupne su joj metode i događaji dostupni većini ostalih komponenti. Tabela 7.8 prikazuje karakteristike koje su bitne za komponentu `Label`.

Tabela 7.8: Karakteristike za komponentu `Label`

Karakteristika	Opis
`AutoSize`	Kada je podešena na `True`, natpis menja veličinu u skladu sa tekstom koji je definisan u karakteristici `Caption`. Kada je podešena na `False`, tekst je odsečen po denoj strani natpisa. Generička vrednost: `True`.
`FocusControl`	Natpis nije prozorska komponenta, pa ne može primiti ulazni fokus i ne može se ubaciti u redosled tabulatora. Ipak, ponekad se natpis može koristiti kao tekst za kontrolu, kao što je edit kontrola. U ovakvim slučajevima možete dodeliti prečicu za natpis (koristeći znak &), a zatim promeniti karakteristiku `FocusControl` u naziv kontrole koja bi trebalo da primi fokus kada korisnik pritisne taster za prečicu.
`ShowAccelChar`	Podesite ovu karakteristiku na `True`, ako želite da se znak & prikaže na natpisu, umesto da služi kao oznaka prečice tastature. Generička vrednost: `True`.
`Transparent`	Kada je ova karakteristika podešena na `True`, karakteristika `Color` se ignoriše i natpis postaje providan (transparentan). Na primer, ovo je korisno za postavljanje natpisa na bitmapiranu pozadinu. Generička vrednost: `False`.
`WordWrap`	Kada je podešena na `True`, tekst u natpisu će preći u drugu liniju, kadase dostigne desna ivica natpisa. Generička vrednost: `False`.

Komponenta `StaticText` (nalazi se na kartici `Additional`) je drugi tip komponente `Label`. Ova komponenta se razlikuje od standardne komponente `Label`, po tome što je prozorska kontrola (ima držač prozora). Komponenta `StaticText` je zgodna kada koristite natpis sa edit komponentom i želite da dodelite prečicu sa tastature komponenti.

Komponenta `ScrollBar` (traka za pomeranje teksta)

Komponenta `ScrollBar` predstavlja samostalnu traku za pomeranje teksta. Ova komponenta je samostalna, u smislu da nije prikačena na edit kontrolu, okvir za listu, formu, odnosno bilo koju drugu komponentu. Do sada nisam pronašao razlog za često korišćenje trake za pomeranje. Određeni tipovi aplikacija koriste trake za pomeranje veoma često, ali za svakodnevne, jednostavne aplikacije, ova komponenta se veoma retko koristi.

Karakteristike koje podešavaju osobine trake za pomeranje su: `Min`, `Max`, `LargeChange` i `SmallChange`. Pozicija trake za pomeranje se može podesiti, ili očitati korišćenjem karakteristike `Position`. Karakteristika `Kind` omogućava da definišete horizontalne, ili vertikalne trake za pomeranje.

Komponenta Panel

Komponenta Panel je u Delphiju tip "šljakerske" komponente. Skoro da ne postoje ograničenja za način upotrebe panoa. Panoi mogu biti korišćeni za držanje dugmadi trake sa alatima, da prikazuju natpise kao što su naslovi formi, da prikazuju grafiku i da drže standardnu dugmad. Jedna od prednosti panoa je što komponente postavljene na pano postaju potomci panoa. U suštini, tamo gde se pomeri pano, pomeraju se i komponente. Ovo može biti velika pomoć u toku rada programa, odnosno u toku dizajniranja.

Većinu svoje snage komponenta Panel duguje karakteristici `Align`. Na primer, pretpostavimo da želite da prikažete naslov na gornjem delu forme. Pretpostavimo da želite da naslov bude centriran bez obzira na promenu veličine porozora. Podešavanjem karakteristike `Align` na `alTop` i karakteristike `Alignment` na `taCenter`, Vaš naslov će uvek biti centriran. Zar to nije jednostavno?

Pano može imati nekoliko izgleda. Izgledi se mogu menjati promenom karakteristika: `BevelInner`, `BeverOuter`, `BorderStyle` i `BorderWidth`, kao što možete videti na slici 7.8.

Slika 7.8
Program Panel Styles Example prikazuje različite stilove

Komponenta `Panel` je toliko svestrana da će Vam trebati vremena da otkrijete sve njene moguće upotrebe.

I to nije sve...

Na nesreću, nema dovoljno mesta da bi se obradile sve komponente koje se nalaze u okviru Delphija. U lekciji dana 4, upoznali ste komponentu Image, kada ste kreirali program `Picture Viewer`. Takođe ste mogli da u istoj lekciji upoznate komponentu `Bevel`, kada ste kreirali okvir za dijalog `About`, a komponentu `Shape` u lekciji

dana 6, kao deo vežbe za poravnavanje komponenti. Ovo predstavlja samo primere komponenti koje Vas čekaju. Treba da isprobate svaku komponentu, kako bi utvrdili koliko Vam može biti korisna.

Postoji još jedna grupa komponenti koje bih želeo da obradim, pre nego što krenemo dalje. To je grupa `Dialog`.

Uobičajeni okviri za dijalog

Windows sadrži skup uobičajenih okvira za dijalog, koji svaki Windows program može da koristi; navešćeno neke od njih:

- File Open (otvaranje datoteke)
- File Save (snimanje datoteke)
- File Open Picture (otvaranje datoteke slike)
- File Save Picture (snimanje datoteke slike)
- Font
- Color (boja)
- Print (štampanje)
- Printer Setup (podešavanje štampača)
- Find (pronalaženje teksta)
- Replace (pronalaženje i izmena teksta)

Uobičajeni okviri za dijalog se mogu pronaći na kartici Dialogs u okviru palete komponenti. Ove komponente se smatraju nevizuelnim, pošto nemaju vizuelni interfejs u toku dizajniranja. Sledeće poglavlje objašnjava svaki od ovih okvira za dijalog sa jednim izuzetkom - opis okvira za dijalog Print i Printer Setup sam ostavio za lekciju dana 13, kada objašnjavam štampanje.

Metoda `Execute`

Jedna od mogućnosti koja je zajednička svim okvirima za dijalog je metoda `Execute`, koja se koristi za kreiranje i prikazivanje okvira za dijalog. Okviri za dijalog se prikazuju modalno, izuzev okvira za dijalog Find i Replace, koji se prikazuju nemodalno. Metoda `Execute` vraća `True`, ukoliko je korisnik kliknuo na dugme OK, dva puta kliknuo na naziv datoteke (u slučaju dijaloga za datoteke), odnosno pritisnuo taster Enter na tastaturi. Metoda `Execute` vraća `False` ukoliko je korisnik kliknuo na dugme Cancel, pritisnuo taster Esc, ili zatvorio okvir za dijalog sistemskim dugmetom za zatvaranje forme. Uobičajeni okvir za dijalog se obično implementira na sledeći način:

```
if OpenDialog.Execute then begin
  { user pressed OK so use the filename }
  Memo.Lines.LoadFromFile(OpenDialog.FileName);
  { do some other stuff }
end;
```

Ovaj kod prikazuje okvir za dijalog File Open i prikazuje naziv datoteke koju je korisnik odabrao. Ukoliko korisnik klikne na dugme OK, kod u okviru bloka `if` se izvršava i učitava se datoteka u komponentu `Memo`. Ukoliko korisnik nije pritisnuo dugme OK, kod u okviru bloka `if` se ignoriše i nikakva operacija se neće izvršiti.

NAPOMENA Kod koji se koristi u prethodnom isečku je još jedan primer kratke sintakse jezika Object Pascal. Prva linija:

`if OpenDialog.Execute then begin`

je ekvivalent linije:

`if OpenDialog.Execute = True then begin`

Možete koristiti bilo koju metodu, ali je prva metoda bolja.

Okviri za dijalog File Open i File Save

Okviri za dijalog File Open i File Save imaju zajedničkih nekoliko karakteristika. File Open okvir za dijalog se koristi u slučaju kada želite da omogućite korisniku otvaranje datoteke u okviru aplikacije (videti sliku 7.9). Okvir za dijalog je enkapsuliran u komponenti OpenDialog. Okvir za dijalog File Save se koristi kada korisnik izda nalog za snimanje datoteke. Ovaj okvir se koristi i kao okvir za dijalog Save As (snimi kao...). Okvir za dijalog File Save je enkapsuliran u komponenti `SaveDialog`.

Slika 7.9
Karakterističan okvir za dijalog File Open

Okviri za dijalog koji se koriste za datoteke su prilično jednostavni za korišćenje u svom osnovnom obliku. Iako imaju nekoliko karakteristika, nema potrebe objašnjavati ih, pošto i bez njih možete normalno koristiti ove dijaloge. Sledeće poglavlje obrađuje krakteristike koje su specifične za okvire za dijalog sa datotekama.

NAPOMENA Komponente `OpenDialog` i `SaveDialog` samo preuzimaju naziv datoteke od korisnika. Na programeru je da napiše kod koji će uraditi nešto sa datotekom.

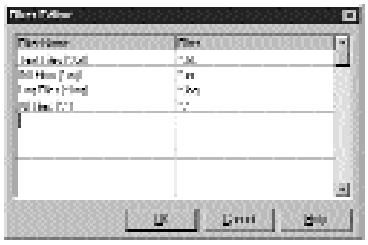
Karakteristika `DefaultExt`. Karakteristika `DefaultExt` se koristi da definiše generički nastavak naziva datoteke koji će okvir za dijalog koristiti. *Generički nastavak* predstavlja nastavak datoteke koji će automatski biti dodat nazivu datoteke, ukoliko korisnik ne upiše naziv nastavka.

Karakteristika `FileName`. Karakteristika `FileName` je najočiglednija karakteristika okvira za dijalog datoteka: sadrži naziv datoteke koji je korisnik odabrao. Ovu karakteristiku možete podesiti pre poziva okvira za dijalog, ukoliko želite da se naziv datoteke pojavi u delu za editovanje naziva datoteke okvira za dijalog, prilikom inicijalizacije okvira za dijalog. Nakon što korisnik klikne na dugme Ok, da bi zatvorio okvir za dijalog, ova karakteristika će sadržati kompletan put i naziv datoteke koja je odabrana.

Karakteristika `Files`. Karakteristika Files se može samo čitati; slučaj klase `TStrings` koji sadrži listu datoteka odabranih ukoliko je dozvoljen izbor više datoteka.

Karakteristika `Filter`. Karakteristika `Filter` sadrži listu tipova datoteka koje korisnik može da odabere. Tipovi datoteka su prikazani u kombo okviru sa natpisom File of type: (tip datoteke:) u okviru za dijalog. Možete definisati karakteristiku `Filter`, kako bi pokazali koje tipove datoteka može da obradi Vaša aplikacija. Na primer, program za jednostavno editovanje teksta može imati filter koji prikazuje sledeće tipove datoteka: `.TXT`, `.INI`, `.LOG`, da navedem samo nekoliko.

Filter može jednostavno biti podešen u toku dizajniranja korišćenjem okvira za dijalog Filter Editor. Da biste pozvali Filter Editor, dva puta kliknite mišem na kolonu Value pored karakteristike `Filter` u okviru prozora Object Inspector. Slika 7.10 prikazuje Filter Editor okvira za dijalog File Open koji je nešto ranije bio objašnjen.



Slika 7.10
Okvir za dijalog `Filter Editor`

Kolona `Filter Name` sadrži opis tipa datoteke. Kolona `Filter` je maska datoteke koja će se koristiti da prikazuje datoteke određenog tipa.

Iako možete da unesete string za filter direktno u kolonu Value prozora Object Inspector, jednostavnije je koristiti `Filter Editor`. Ukoliko koristite samo jedan filter, možete ga upisati direktno u kolonu Value karakteristike `Filter`. Opise i filter razdvajate vertikalnom crtom (pipe). Na primer, ukoliko želite da imate filter za sve tipove datoteka, upisaćete sledeće:

```
All Files (*.*)¦*.*
```

Karakteristika `FilterIndex`. Karakteristika `FilterIndex` se koristi da definiše filter koji će se koristiti prilikom pojavljivanja okvira za dijalog. Indeks ne počinje nulom kao što ste mogli očekivati, naprotiv. Prvi filter na listi ima indeks 1, drugi 2, itd. Za primer, pogledajte sliku 7.10. Ukoliko želite da se inicijalno prikaže filter All Files (sve datoteke), treba da u vrednost karakteristike `FilterIndex` upišete 4.

Karakteristika `InitialDir`. Karakteristika `InitialDir` se koristi da definiše početni direktorijum koji će okvir za dijalog prikazivati. Ukoliko za karakteristiku `InitialDir` nije definisana vrednost, biće korišćen tekući direktorijum (koji određuje Windows).

SAVET Program u okviru operativnog sistema Windows vodi računa o tome koji je direktorijum korisnik poslednji otvorio prilikom otvaranja, odnosno snimanja datoteke. Obično je ova informacija smeštena u bazu Registry. Pre nego što prikažete okvir za dijalog File Open, ili File Save, podesite karakteristiku `InitialDir` na prethodni direktorijum koji je korisnik koristio. Nakon što korisnik odabere datoteku, osvežite bazu Registry, kako bi pokazivala na novi direktorijum, ukoliko je to potrebno.

Karakteristika `Options`. Karakteristika `Options` kontroliše način na koji će se okviri za dijalog koristiti. Lista opcija je suviše duga, da bi ovde bila navedena, ali uobičajene opcije uključuju mogućnost kreiranja novog direktorijuma, da li će dugme Help biti prikazano u okviru za dijalog, da li je dozvoljeno korišćenje dugih imena datoteka, da li je dozvoljeno odabrati više datoteka, itd. Pogledajte Delphijev program za pomoć u toku rada, kako biste kompletirali informacije o komponentama `OpenDialog` i `SaveDialog`.

Karakteristika `Title`. Karakteristika `Title` se koristi da definiše, odnosno pročita naslov okvira za dijalog. Ukoliko naslov nije definisan, uobičajeni naslovi Open okvira za dijalog `OpenDialog` i Save okvira za dijalog `SaveDialog`, će biti iskorišteni.

SAVET Okvir za dijalog Save As nije ništa drugo do komponenta SaveDalog sa karakteristikom `Title` promenjenom u `Save As`.

Okviri za dijalog za datoteke nemaju događaje koji su im pridruženi.

SAVET Možete implementirati okvir za dijalog File Open (ili bilo koji drugi uobičajeni okvir za dijalog) u toku rada programa , a da ne postavite komponentu `OpenDialog` na formu. Da bi ovo postigli, kreirajte slučaj klase `TOpenDialog`, a zatim pozovite metodu `Execute`, kao što je navedeno u primeru:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  OpenDlg : TOpenDialog;
begin
  OpenDlg := TOpenDialog.Create(Self);
  if OpenDlg.Execute then begin
    { do something here }
```

```
  end;
  OpenDlg.Free;
end;
```

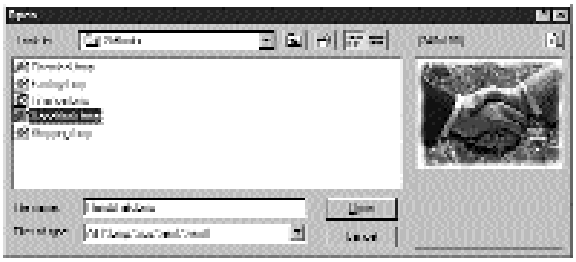
Ukoliko je neophodno, možete pre poziva metode `Execute` podesiti neke od karakteristika komponente.

Okviri za dijalog File Open Picture i File Save Picture (Otvori datoteku slike i Snimi datoteku slike).

Ova dva okvira za dijalog nisu ništa drugo do obični okviri za dijalog File Open i File Save sa dodatnom opcijom: prikazuju prozor za pregled i omogućavaju Vam da vidite sliku koju ste trenutno odabrali. Ovi okviri za dijalog imaju takođe i karakteristiku `Filter` koja je unapred podešena na uobičajene formate slika koje koristi Windows. U suprotnom, ponašaju se kao okviri za dijalog File Open i File Save. Slika 7.11 prikazuje okvir za dijalog File Open Picture u toku rada.

Okvir za dijalog `Color` (boja)

Okvir za dijalog `Color` omogućava korisniku da odabere boju. Kada korisnik pritisne dugme OK, karakteristika `Color` će sadržati informaciju o boji (pogledajte sliku 7.1, da biste videli okvir za dijalog Color). Okvir za dijalog `Color` kao i okviri za dijalog za datoteke, nemaju događaje na koje odgovaraju.

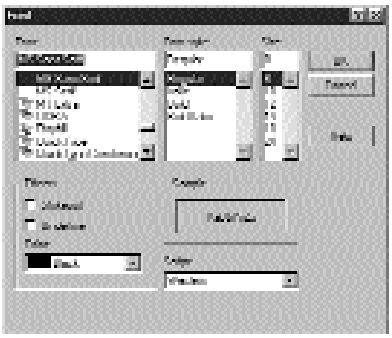


Slika 7.11
Okvir za dijalog File Open Picture

Okvir za dijalog `Font`

Okvir za dijalog `Font` omogućava korisniku da odabere `Font` sa liste dostupnih fontova u okviru svog sistema. Koristeći karakteristiku Device, možete odabrati da li želite da fontovi za ekran, fontovi za štampač, odnosno obe vrste fonta budu prikazane. Možete ograničiti maksimalnu i minimalnu veličinu fonta koju korisnik može odabrati menjajući karakteristike `MaxFontSize` i `MinFontSize`. Kao što je to slučaj okvira za dijalog za datoteke, karakteristika Options sadrži širok izbor opcija koje možete da koristite za kontrolu funkcija okvira za dijalog Font.

Ukoliko korisnik klikne na dugme OK, karakteristika Font će sadržati sve informacije koje su potrebne za implementaciju novog fonta. Slika 7.12 prikazuje okvir za dijalog Font sa generičkom konfiguracijom.



Slika 7.12
Okvir za dijalog Font

Okvir za dijalog `Font` ima jedan dodatni događaj u odnosu na uobičajene događaje za dijaloge. Događaj `OnApply` će se desiti kada korisnik klikne na dugme `Apply` okvira za dijalog `Font`. Dugme Apply nije prisutno na okviru za dijalog `Font`, sve dok ne kreirate odgovarajući (ukoliko nije prazan) upravljač događajem za događaj `OnApply`.

Okviri za dijalog Find i Replace

Okviri za dijalog Find i Replace pružaju korisniku mogućnost da unese tekst koji će tražiti i tekst koji će zameniti nađenim tekstom, kao i razne opcije za pretraživanje i zamenu. Okvir za dijalog Find je enkapsuliran u VCL komponentu FindDialog, a okvir za dijalog Replace je predstavljen komponentom `ReplaceDialog`. Okvir za dijalog Replace koji sadrži sve što se može naći u okviru za dijalog Find uz dodate mogućnosti za izmenu je prikazan na slici 7.13.

Slika 7.13
Okvir za dijalog `Replace`

Glavne karakteristike komponenti `FindDialog` i `ReplaceDialog` su: `FindText` (tekst koji treba pronaći), `ReplaceText` (tekst kojim se menja pronađeni tekst) i Options. Očigledno je da komponenta `FindDialog` nema karakteristiku `ReplaceText`. Karakteristika Options sadrži širok spektar informacija koje korisnik može podesiti u trenutku kada klikne na dugmad Find Next, Replace, odnosno Replace All.

Metoda `Execute` za komponente `FindDialog` i `ReplaceDialog` se pomalo razlikuje od drugih uobičajenih komponenti tipa `Dialog`. Osnovno je da su okviri za dijalog Find i Replace nemodalni okviri za dijalog. Nakon prikazivanja okvira za dijalog, vraća se metoda `Execute`.

Pošto je okvir za dijalog nemodalan, vrednost koja se vraća iz metode `Execute` nema smisla (uvek je `True`). Umesto toga, okviri za dijalog Find i Replace koriste događaje `OnFind` i `OnReplace` zajedno sa karakteristikom `Options`, da bi odredili šta se dešava sa okvirom za dijalog. Događaj `OnFind` se pojavljuje kada korisnik klikne mišem na dugme Find Next. Komponenta `ReplaceDialog` ima događaj `OnFind`, ali ima i događaj `OnReplace` koji se aktivira kada korisnik klikne na dugme Replace, ili Replace All. Ove događaje možete koristiti da odredite da li korisnik ima zahtev za aktivnost traženja, ili izmene. Vaši programi treba da očitavaju karakteristiku Options kako bi utvrdili da li korisnik ima nameru da izvrši operaciju traženja, ili izmene, kako bi je izveli.

Zaključak

Danas ste mogli da pregledate neke osnovne komponente koje sadrži Delphi, naučili ste o komponentama uopšte, a zatim naučili nešto o posebnim komponentama koje su bazirane na Windows kontrolama. Važno je da shvatite osnovne kontrole koje su dostupne u okviru Windows-a i Delphijeve komponente koje predstavljaju ove kontrole. Na kraju ste istražili neke od uobičajenih okvira za dijalog.

Radionica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstirte razumevanje obrađenog materijala, kao i vežbe koje Vam omogućavaju da steknete iskustvo u korišćenju gradiva koje ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A, "Odgovori na kviz pitanja".

Pitanja i odgovori

P **Ukoliko promenim karakteristiku `Name` komponente koristeći prozor Object Inspector, Delphi će automatski promeniti sve što se odnosi na tu komponentu u okviru mog koda. Da li je to tačno?**

O I da, i ne. Delphi će promeniti sve što se odnosi na naziv komponente u okviru koda koji je sam generisao, ali neće promeniti kod koji je korisnik pisao.

P **Komponenta `OpenDialog` je očigledno komponenta koja se može videti. Zašto se onda naziva nevizuelnom komponentom?**

O Pošto nije vidljiva u toku dizajniranja. Vidljiva je samo u toku rada programa, kada je pozovete koristeći metodu `Execute`.

P **Zašto je važno da karakteristiku `Name` menjam samo u okviru prozora Object Inspector?**

O Dok radite sa dizajnerom forme Delphi piše kod baziran na karakteristici `Name`. Ukoliko kasnije izmenite karakteristiku `Name`, bilo direktno da editujete izvorni kod datoteke, ili u toku rada programa, sve što je vezano za navedenu formu, ili komponentu će biti netačno i vodiće Vaš program ka krahu u toku rada, odnosno program će prijavljivati grešku u toku prevođenja.

P **Čini mi se da koristim karakteristike više nego metode kada radim sa komponentama u okviru koda. Da li je to pogrešno?**

O Nije pogrešno. U suštini, to je način na koji su VCL komponente dizajnirane. Dobro napisana komponenta maksimalno koristi karakteristike. Iz tog razloga se metode komponenata ne koriste često. Koristite metode kada je to neophodno, u suprotnom koristite karakteristike kako bi manipulisali Vašim komponentama u toku rada programa.

P **Odgovorio sam i na događaj `OnDblClick` i na događaj `OnClick` za komponentu. Svaki put kada dva puta kliknem mišem na komponentu, pozivaju se upravljači događajem `OnClick` i `OnDblClick`. Zašto?**

O Pošto ste dva puta kliknuli na komponentu, Windows generiše obe poruke; jednostruki klik mišem i dvostruki klik mišem. Ne možete ovo sprečiti, zato treba da pišete kod računajući na pojavu oba događaja.

P **Želim da koristim mogućnosti klase `TStrings` da sačuvam listu stringova koje moj program zahteva za rad. Prevodilac mi ne dozvoljavam da koristim objekt `TStrings`. Šta treba da radim?**

O Koristite objekt `TStringList`, umesto `TStrings`. Klasa `TStringList` postoji za ovu svrhu.

P **Potrebno je da mi edit kontrola bude poravnata na desno, ali ne postoji karakteristika `Alignment` za komponentu Edit. Da li mogu desno poravnati tekst u edit komponenti sa jednom linijom?**

O Ne. Ono što ipak možete uraditi je da koristite komponentu `Memo` i namestite je da se ponaša kao obična Edit komponenta. Uverite se da karakteristika komponente memo ima vrednost `False`, a karakteristika `Height` visinu standardne edit komponente (21 piksel) i karakteristiku `Alignment` podešenu na `taRightJustify`. Komponenta će se ponašati kao da je edit kontrola sa jednom linijom i biće desno poravnata.

P **Imam formu sa nekoliko dugmadi. Kada korisnik zatvori formu korišćenjem tastera Esc, vrednost `ModalResult` dugmeta se koristi kao da je vraćena vrednost iz `ShowModal`.**

- **Kada korisnik zatvori formu koristeći sistemsko dugme za zatvaranje, dobićete vraćenu vrednost `mrCancel`. Treba da budete spremni na oba slučaja kada se forma zatvori.**

Kviz

1. Da li možete da promenite karakteristiku `Name` u toku rada programa?
2. Koja karakteristika se koristi za aktiviranje i deaktiviranje kontrola?
3. Kako možete da u toku rada programa saopštite da je dugme deaktivirano?
4. Koja je razlika između dugog saveta i kratkog saveta?
5. Navedite tri od četiri metode koje se mogu koristiti za ponovno iscrtavanje kontrole.
6. Koliko tipova kombo okvira postoji?
7. Kako se karakteristika `ModalResult` koristi za komponente dugmad?
8. Koja se komponenta najviše koristi kao kontejner koji sadrži druge komponente?
9. Koja je povratna vrednost metoda `Execute` za komponentu `OpenDialog`, ukoliko korisnik klikne mišem na dugme OK kako bi zatvorio okvir za dijalog?
10. Kako da od komponente `SaveDialog` napravite okvir za dijalog Save As?

Vežbe

1. Kreirajte program koji sadrži dve edit komponente. Kada korisnik upisuje informacije u prvu komponentu, učinite da se i u drugoj edit kontroli pojavljuje istovremeno kada se unosi.
2. Kreirajte program sa okvirom za listu. Napišite kod koji učitava podatke za listu iz tekst datoteke, pre nego što aplikacija postane vidljiva.
3. Dodajte edit komponentu u program iz druge vežbe. Kada korisnik odabere opciju iz okvira za listu, neka se tekst opcije pojavi u edit kontroli.
4. Dodajte dugme u program iz druge i treće vežbe. Napišite kod tako da pritiskom na dugme bilo koji tekst u okviru edit kontrole bude dodat kao nova opcija okvira za listu.
5. Kreirajte program koji ima komponentu `RadioGroup` sa četiri opcije u grupi. Dodajte natpis koji menja tekst u zavisnosti od odabranog radio dugmeta.
6. Kreirajte program koji ima naslov na formi koji je centriran u vrhu forme bez obzira kako se menja veličina prozora programa.

7. Izmenite program iz šeste vežbe tako da font naslova može biti promenjen u bilo koji font koji je dostupan sistemu kada se klikne mišem na dugme.

8. Ponovo otvorite program `PictureViewer` kreiran u lekciji dana 4. Izmenite program tako da koristi okvire za dijalog File Open Picture i File Save Picture, umesto standardnih okvira za dijalog.

Dan 8

Kreiranje aplikacija u Delphiju

Delphi pruža niz alata koji će Vam pomoći u kreiranju formi, okvira za dijalog i aplikacija. Danas ćete naučiti:

- Object Repository (skladište objekata).
- Dialog Wizard (čarobnjak za dijalog).
- Application Wizard (čarobnjak za aplikacije).
- Dodavanje metoda i polja podataka u Vaš kod.
- Obrasci za komponente.
- Korišćenje resursa u Vašim Delphi aplikacijama.
- Paketi.

Za početnike, ću objasniti šta je skladište objekata (Object Repository), mesto gde Delphi smešta forme, aplikacije, odnosno ostale objekte za kasniju upotrebu.

Ukoliko budete pratili objašnjenje, srešćete se sa čarobnjacima. Čarobnjaci pružaju niz okvira za dijalog koji Vas, korak po korak, vode ka kreiranju procesa. Vi ćete upisivati podatke, a Delphi će kreirati formu, ili aplikaciju zasnovanu na podacima koje ste mu pružili. Čarobnjaci su veoma moćan alat za brz razvoj aplikacija. U ovoj lekciji ću Vam objasniti kako da koristite resurse u Vašim Delphi aplikacijama. Na kraju, ću Vam objasniti pakete u okviru Delphija.

Rad sa skladištem objekata (Object Repository)

Skladište objekata je alat sa kojim možete odabrati unapred definisane objekte i koristiti ih u svojim aplikacijama.

Skladište za objekte Vam omogućava da uradite sledeće:

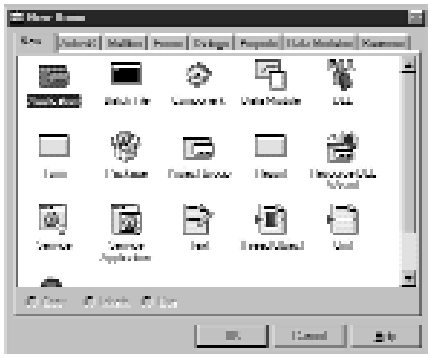
- Odabereta unapred definisanu aplikaciju, formu, ili okvir za dijalog i implementirate ga u Vašu aplikaciju.
- Dodate Vaše forme, okvire za dijalog i aplikacije u skladište objekata.
- Dodate druge objekte Vašim aplikacijama kao što su ASCII tekst datoteke i dodatne junite izvornog koda.
- Upravljate modulima podataka.
- Kreirate nove komponente.
- Kreirate nove pakete.
- Kreirate nove ActiveX kontrole za ActiveForms.
- Pozovete čarobnjake da Vam pomognu u pravljenju okvira za dijalog, ili aplikacija.

Ovo su samo neki primeri onoga što skladište za objekte pruža. Postoje i drugi objekti koje možete kreirati, a koji nisu navedeni.

Kartice i opcije skladišta objekata

Skladište objekata se automatski prikazuje kada odaberete opciju File➥New u okviru glavnog menija. Slika 8.1 prikazuje prozor skladišta objekata koji se pojavljuje kada odaberete opciju File➥New, a da nije otvoren projekat.

NAPOMENA Čudno, ma kako to zvučalo, skladište za objekte ima naslov New Items, a okvir za dijalog konfiguracije skladišta za objekte ima naslov Object Repository (skladište objekata). Kada bi rekli da ovo izgleda zbunjujuće, to bi bilo zbog nerazumevanja suštine ovog alata.



Slika 8.1
Prozor skladišta objekata

Skladište objekata ima nekoliko kartica, svaka od njih sadrži različite objekte koje možete ubaciti u Vaše aplikacije. Kao što ste mogli videti na slici 8.1, kartica New je inicijalno odabrana prilikom prikazivanja skladišta objekata. Tabela 8.1 prikazuje kartice skladišta objekata i opise opcija koje možete pronaći na svakoj strani.

Tabela 8.1: Kartice skladišta objekata

Kartica/jezičak kartice	Opis
New	Omoguva Vam da kreirate nove alikacije, forme, ili junite koje ćete koristiti u Vašim aplikacijama. Takođe Vam omogućava da kreirate napredne objekte kao što su paketi, DLL datoteke, komponente, NT servisne aplikacije, Web server aplikacije i module podataka.
ActiveX	Omogućava Vam da kreirate nove ActiveX kontrole, biblioteke tipova, COM objekte, ActiveForms i druge AktiveX objekte.
Multitier	Omogućava Vam da kreirate CORBA i MTS objekte i module podataka (samo verzija Client/Server).
Forms	Omogućava Vam da kreirate standardne forme od unapred definisanih formi, kao što su: okvir About, okvir sa dvostrukom listom, kartice sa jezičcima kartica, odnosno QuickReports.
Dialogs	Predstavlja izbore nekoliko osnovnih tipova okvira za dijalog, koje možete odabrati. Sadrži takođe i Dialog Wizard (čarobnjak za dijaloge).
Projects	Prikazuje kompletne projekte koje možete odabrati kako biste trenutnopostavili aplikaciju. Sadrži i Application Wizard (čarobnjak za aplikacije).
Data Modules	Omogućava Vam da odaberete module podataka za Vašu aplikaciju.
Business	Uključuje čarobnjake za forme baza podataka, Web aplikacije sa bazom podataka, izveštaje i grafikone i primer aplikacije Decision Cube.

NAPOMENA Ukoliko pozovete skladište objekata, a da je u tom trenutku otvoren projekt, u okviru skladišta objekata će se pojaviti i dodatna kartica. Kartica će nositi naziv Vašeg projekta. Klik mišem na karticu će pokazati sve objekte koji se trenutno nalaze u projektu. Ovo Vam omogućava da brzo preuzmete formu, odnosno drugi objekt, jednostavnim odabiranjem iz skadišta objekata.

Duž donje ivice svake strane ćete primetiti tri radio dugmeta. Ova dugmad sa natpisima Copy (kopiranje), Inherit (nasleđivanje) i Use (korišćenje), definišu kako će se implementirati izabrani objekt. U zavisnosti od odabranog objekta, neka radio dugmad (ili sva) mogu biti deaktivirana. Na primer, sva tri radio dugmeta mogu uvek biti siva, ukoliko se koristi kartica New. Razlog za ovakav prikaz leži u činjenici da je jedina opcija dostupna objektima na ovoj strani kopiranje, pa Delphi deaktivira sve opcije i primenjuje automatski opciju za kopiranje.

NAPOMENA Skladište objekata se ponekad naziva galerija (Gallery).

Dugme Copy

Kada odaberete radio dugme Copy, Delphi kreira kopiju odabranog objekta i postavlja ga u Vašu aplikaciju. Od ovog trenutka slobodno možete menjati objekt na bilo koji način. Original objekta u skladištu se ne menja kada menjate novi objekt u okviru Vaše aplikacije.

Da bi ovo ilustrovali, pretpostavimo da imate često korišćenu formu (formu u tradicionalnom smislu, a ne u Delphijevom), a koja je odštampana na papiru - na primer, plan rada. Pretpostavimo da želite da popunite formu planom rada. Nećete menjati originalnu formu, pošto bi u tom slučaju bila neupotrebljiva za kasniji rad. Postavićete originalnu formu u fotokopir, kopirati je, a zatim vratiti original na mesto gde će biti siguran.

Zatim, možete popuniti kopiju forme ukoliko je to potrebno. Pravljenje kopije objekta iz skladišta radi na potpuno isti način. Slobodno možete menjati kopiju na bilo koji način, a odabrati da original bude na sigurnom. Pravljenje kopija je najsigurniji metod za korišćenje objekata.

Dugme Inherit

Metod nasleđivanja je sličan kopiranju, ali ima jednu bitnu razliku: novi objekt je još uvek vezan za osnovni objekt. Ukoliko kreirate osnovni objekt, novokreirani će biti izmenjen tako da prikazuje promene koje ste uneli u glavni. Naravno, suprotna veza ne važi. Možete menjati novi objekt, a da to nema nikakvog efekta na originalni objekt.

Da bi ilustrovali ovaj tip korišćenja objekata, razmotrite sledeći scenario: često, menadžeri za informisanje kreiraju tabele u programima za rad sa tabelama (tabele za unakrsno izračunavanje - Spreadsheet) i koriste sadržaj tabela u programima za obradu teksta, da bi prikazali izveštaj.

Često koriste vezu podataka u tabeli za unakrsno izračunavanje kada prebacuju podatke iz Clipboard-a, ili uvoze tabelu u tekst procesor. Na ovaj način, kada se menjaju podaci u tabeli, dokument u tekst procesoru se automsatski menja, kako bi prikazao nove podatke. Na sličan način, izmene osnovne forme će se automatski reflektovati na sve forme nasleđene od osnovne forme. Opciju za nasleđivanje možete koristiti kada želite da nekoliko formi baziranih na običnoj formi bude izmenjeno u nekom trenutku. Sve promene u osnovnoj formi će se odraziti na sve nasleđene forme.

Dugme Use

Opcija za korišćenje nije uobičajena. Kada koristite objekat, otvarate ga direktno za editovanje. Odaberite ovu opciju kada želite da napravite izmene objekta koji ste snimili u skladištu. U poglavlju o opciji za nasleđivanje, naveo sam da će promene

koje su se desile glavnoj formi biti prikazane na svim formama koje je nasleđuju. Ukoliko želite da promenite osnovnu formu, otvorićete je u okviru skladišta objekata koristeći opciju Use.

Korišćenje skladišta objekata

Šta se tačno dešava sa objektom kada koristite opciju za izbor objekta iz skladišta objekata zavisi od nekoliko faktora. Faktori uključuju tip odabranog objekta, da li je projekt trenutno otvoren i korišćenje izbora (Copy, Inherit, ili Use).

Ako imate otvorenu aplikaciju i odaberete kreiranje nove aplikacije iz skladišta objekata, bićete upitani da li želite da snimite tekući projekt (ukoliko je to potrebno), pre nego što novi projekt bude prikazan.

SAVET Izbor opcije File➥New Application u okviru glavnog menija predstavlja prečicu za pokretanje nove aplikacije. Ovo je ekvivalent izbora opcije New u okviru glavnog menija, a zatim izbora objekta Application u okviru skladišta objekata. Slično, opcija New Form u okviru glavnog menija je prečica za ekvivalent koji koristi skladište objekata.

Kreiranje nove forme korišćenjem skladišta objekata se različito tretira u zavisnosti od toga da li je projekt trenutno otvoren. Ukoliko je projekt otvoren, nova forma se dodaje aplikaciji kao par forma/junit. Ukoliko projekt nije otvoren, nova forma i junit će biti kreirani kao zasebna forma. Forma kreirana van projekta mora biti dodata projektu, pre nego što se projekt počnete koristiti u toku rada. Ovu opciju možete koristiti kada kreirate novu baznu formu koju ćete dodati u skladište objekata.

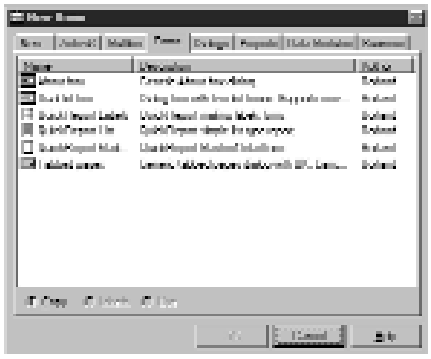
Ukoliko odaberete kreiranje novog junita, ili tekst datoteke, nova datoteka će biti kreirana u okviru editora koda (i ako je dodat novi junit, biće dodat tekućem projektu). Tekst datoteku možete kreirati iz nekoliko razloga. Na primer, pretpostavimo da želite da Vašoj aplikaciji dodate datoteku za konfiguraciju (`.ini` datoteka). Da biste kreirali novu datoteku za konfiguraciju, možete kreirati novu tekst datoteku u okviru skladišta objekata. Kreirajte novi junit svaki put kad želite da otvorite novu datoteku izvornog koda za Vašu aplikaciju, a da datoteka nije povezana sa formom (na primer, pridružena datoteka).

Izbor opcije za kreiranje nove DLL datoteke kao rezultat daje kreiranje novog projekta koji je podešen da kreira DLL datoteku. Kreiranje nove komponente, ili povezanog objekta, kao rezultat prikazuje okvir za dijalog koji postavlja pitanja vezana za dodatne informacije o projektu koji kreirate.

Pregledi skladišta objekata

Aktuelni prozor skladišta objekata je Win32 kontrola za pregled liste, slična desnoj strani Windows Explorer-a (gde su prikazane datoteke). Postoji nekoliko različitih izgleda koje možete odabrati: Large Icons, Small Icons, List, Details (Velike ikone, Male ikone, Lista, Detalji). Generički, pogled je podešen na Large Icons. Da biste

promenili izgled skladišta za objekte, kliknite desnim tasterom miša na skladište objekata i iz menija sadržaja, odaberite pogled koji želite. Slika 8.2 prikazuje skladište objekata sa odabranom karticom Forms i pogledom podešenim na Details.



Slika 8.2
Skladište objekata sa pogledom podešenim na Details

Meni sadržaja skladišta objekata takođe prikazuje nekoliko opcija za sortiranje. Podatke možete sortirati po nazivu objekta, opisu, datumu, ili autoru.

SAVET Kada je skladište objekata u pogledu podešeno na Details, možete kliknuti na zaglavlje kolone (Name, Description, Date, ili Author), kako bi trenutno sortirali po određenoj kategoriji.

Kreiranje novih objekata u skladištu objekata

Sigurno najosnovnija upotreba skladišta objekata je kreiranje novog objekta, korišćenjem originala iz skladišta. Da bi ovo ilustrovali možete kreirati jednostavnu aplikaciju sa glavnom formom, okvir za dijalog About i drugu formu. Pratite sledeće korake:

1. Uverite se da nije otvorena ni jedna druga aplikacija. Odaberite opciju File➥New u okviru glavnog menija. Biće prikazano skladište objekata.
2. Kliknite na ikonu Application, a zatim kliknite mišem na dugme OK, da biste kreirali novu aplikaaciju. Nova aplikacija je kreirana i prikazana je prazna forma.
3. Postavite dva dugmeta na formu. Promenite karakteristiku Caption jednog od dugmadi u `About...`, a karakteristiku `Caption` drugog dugmeta u `Display Form2`. Promenite karakteristike `Name` po želji.
4. Odaberite opciju File➥New u okviru glavnog meija. Ponovo će biti prikazano skladište objekata.
5. Kliknite mišem na karticu Forms u okviru skladišta objekata.
6. Odaberite objekt `About Box`. Uverite se da je odabrano radio dugme Copy, a zatim kliknite na dugme OK, da biste kreirali novu formu About Box. Biće prikazan okvir za dijalog About. Promenite, ukoliko je to potrebno, dodatne karakteristike.

7. Izmenite okvir za dijalog About po želji. (Upišite Vaše podatke, promenite ikonu, veličinu, poziciju, itd.)
8. Odaberite ponovo opciju File➥New u okviru glavnog menija. Po treći put će biti prikazano skladište objekata.
9. Kliknite na jezičak kartice Forms i odaberite objekt `Dual list box`. Kliknite mišem na dugme OK i zatvorite skladište objekata. Forma sa dve liste će biti prikazana. (želeo sam da odaberete ovu formu da biste videli kako izgleda.)
10. Upišite upravljače događajima za oba dugmeta koji prikazuju okvir za dijalog About i drugu formu. Nemojte zaboraviti da dodate junite za okvir About i drugu formu u klauzulu uses glavne forme.
11. Prevedite, pokrenite i testirajte program

Ovaj program ne radi ništa, ali ilustruje kako možete koristiti skladište objekata da biste brzo napisali aplikaciju. Kako vreme prolazi, dodavaćete Vaše objekte u skladište objekata i na taj način ćete postati produktivniji. Pogledajmo kako se to radi.

Dodavanje objekata u skladište objekata

Skladište objekata ne bi bilo ni približno tako efikasno ukoliko ne bi mogli da dodate Vaše objekte. Svoje objekte ćete moći da dodate i to bi trebalo da uradite. Dodavanje često korišćenih objekata u skladištu objekata Vas kao programera čini produktivnijim i naravno, vrednijim. Nema smisla uvek iz početka otkrivati točak.

Nakon što ste kreirali aplikaciju, formu, ili neki drugi objekat, snimite ga u skladište objekata, kako bi mogli da ga iskoristite kad god to poželite. Naravno, nećete želeti da snimite svaku formu koju ste kreirali, nego samo one forme koje najčešće koristite.

Možete kreirati objekat za posebne svrhe dodavajući ga u skladište, odnosno možete dodati objekat u skladište u toku normalnog razvoja aplikacije. (Pojam *objekat* je prilično širok, pa ću koristiti određeni primer da bi sve ovo dobilo smisao.) Recimo da ste kreirali okvire za dijalog About u toku kreiranja aplikacije. Iznenada ste odlučili da biste želeli da snimite ovaj okvir za dijalog, kako bi ga koristili u svim Vašim programima. Na kraju krajeva, sadrži naziv Vaše firme, logo i sve informacije koje su postavljene onako kako želite, pa bi bila šteta da ponovo kreirate isti okvir za dijalog za svaku aplikaciju koju koristite. Bez problema ga možete dodati u skladište objekata.

Da biste dodali formu u skladište objekata, prvo treba da je snimite (ukoliko ne snimite formu, pre nastavka rada ćete biti upitani da li želite da to učinite). Zatim, kliknite na desni taster miša bilo gde u okviru forme i odaberite opciju Add To Repository u okviru menija sadržaja dizajnera forme. Nakon toga će biti prikazan okvir za dijalog Add To Repository koji je prikazan na slici 8.3.

Slika 8.3
Okvir za dijalog Add To Repository

Okvir za listu Forms na levoj strani okvira za dijalog prikazuje trenutne forme kao i bilo koji drugi objekat u okviru aplikacije (kao što su moduli podataka). Prvo, odaberite formu koju želite i dodajte je u skladište objekata.

NAPOMENA Aktivna forma u dizajneru forme će već biti odabrana u okviru za listu Forms, koji se nalazi u okviru za dijalog Add To Repository.

A sad unesite naslov objekta. Ovaj naslov će se pojaviti ispod ikone objekta u skladištu. Polje za opis (Description) se koristi za upis dodatnih informacija o objektu. Ovaj opis se prikazuje kada je skladište objekata podešeno da prikazuje sve informacije o objektima (pogledajte sliku 8.2). Polje Author se koristi za upis imena autora objekta. Možete upisati Vaše ime, naziv firme, ili bilo koji drugi naziv za identifikaciju.

NAPOMENA Većina objekata koji se nalaze u skladištu objekata, i predstavljaju osnovne Delphijeve objekte, kao ime autora imaju upisano "Borland" (izuzeci su QuickReport i TeeChart).

Polje Page se koristi za određivanje kartice na kojoj će se nalaziti objekat. Možete odabrati jednu od postojećih kartica, ili samo upisati naziv nove kartice u ovo polje. Ukoliko kartica sa upisanim nazivom ne postoji, Delphi će kreirati novu karticu sa navedenim nazivom. U dnu okvira za dijalog je dugme sa natpisom Browse koje možete koristiti da odaberete iksonu koja će se koristiti za predstavljanje objekta.

SAVET Ikone možete odabrati iz direktorijuma `Borland Shared Files\Images\Icons`, odnosno direktorijuma `Delphi 4\Objrepos`. Ikone udirektorijumu `Delphi 4\Objrepos` koristi Delphi za elemente koje postavlja u skladište objekata.

Nakon što ste popunili sva polja i odabrali ikonu, kliknite na dugme OK, da biste dodali objekat u skladište. Objekat je dodat u skladište objekata i nalazi se na strani koju ste definisali. Kada to želite, objekat možete ponovo uzeti. Kao što ste mogli da vidite, dodavanje objekta u skladište je skoro isto tako jednostavno kao i njegovo preuzimanje.

UPOZORENJE Kada dodajete objekat u skladište objekata, Delphi u datoteku skladišta objekata upisuje element koji opisuje objekat. Ova informacija uključuje put i izvornu datoteku objekta koji se nalazi u skladištu. Ukoliko promenite direktorijum, ili obrišete datoteku sa izvornim kodom objekta, nećete moći da koristite objekt iz skladišta.

Dodavanje projekata u skladište objekata

Dodavanje projekata u skladište objekata se mnogo ne razlikuje od dodavanja zasebnih formi. Da biste dodali projekat u skladište objekata, odaberite opciju Project➥Add To Repository u okviru glavnog menija. Biće prikazan okvir za dijalog Add To Repository kao što je to bio slučaj kod dodavanja objekata u skladište; razlikuje se po tome što okvir za listu Forms nije prikazan. Popunite informacije o objektu (naziv, opis, ime autora, itd.), a zatim kliknite na dugme OK i projekat će biti dodat u skladište objekata.

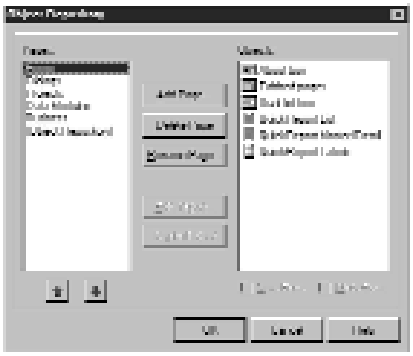
Nakon što se upoznate sa Delphijem, trebali bi da kreirate osnovu aplikacije (školjku), koja će sadržati opcije koje najčešće koristite. Prilikom startovanja nove standardne aplikacije, napravite kopiju školjke koja se nalazi u skladištu objekata. Na ovaj način ćete imati Vaše menije, trake sa alatima, okvir za dijalog koji sadrži opis programa (About box) i ostale standardne okvire za dijalog. Sve će to biti spremno za rad u par sekundi. Nakon što kreirate novu aplikaciju, možete je menjati kao što to radite sa bilo kojim projektom. Možete dodavati nove forme, brisati forme koje Vam ne odgovaraju itd.

Održavanje skladišta objekata

Karticama i objektima u okviru skladišta objekata možete administrirati korišćenjem okvira za dijalog koji konfiguriše skladište objekata.

Da biste videli okvir za dijalog koji konfiguriše skladište objekata, odaberite opciju Tools➥Repository u okviru glavnog menija, odnosno ukoliko imate već otvoreno skladište za objekte, odaberite opciju Properties u okviru menija sadržaja skladišta objekata. Okvir za dijalog će biti prikazan kao što je to prikazano na slici 8.4.

Ovaj okvir za dijalog Vam omogućava da obrišete objekte i kartice iz skladišta objekata, prebacite objekte sa jedne kartice na drugu, promenite redosled kartica skladišta objekata i slično. Lista kartica skladišta objekata je prikazana u okviru za listu, pod nazivom Pages na levoj strani okvira za dijalog. Kada odaberete jednu od kartica na listi Pages, okvir za listu na desnoj strani (sa natpisom Objects) prikazuje objekte koji se nalaze na označenoj kartici.



Slika 8.4
Okvir za dijalog koji konfiguriše skladište objekata

NAPOMENA Okvir za listu Pages ima dve važne opcije koje treba napomenuti. Prvo, uočite da kartica New, koja je uvek prva prikazana kada se pozove skladište objekata, nije prikazana. (Kartice ActiveX i Multitier, takođe nisu prikazane u okviru za listu Pages.) Kartica New je fiksno definisana i ne može se menjati. Uočite da postoji i opcija pod nazivom `[Object Repository]`. Ova opcija je spisak svih stavki koje se nalaze na svim karticama skladišta objekata.

Administriranje objektima

Pre nego što editujete, obrišete, ili promenite mesto objektu, morate ga prvo odabrati. Da biste odabrali objekat, kliknite na njega u okviru za listu Objects. Nakon što odaberete objekat, možete ga editovati ukloliko klinknete na dugme Edit Object. Editovanje objekta omogućava Vam da promenite naziv, opis objekta, informacije o autoru, kao i karticu na kojoj se objekat nalazi.

SAVET Da biste editovali objekat, dva puta kliknite mišem na okvir za listu Objects.

Objekat možete obrisati, tako što ga prvo odaberete, a zatim kliknete na dugme Delete Object. Pre nego što objekat bude uklonjen, biće zatražena potvrda za uklanjanje objekta sa kartice i iz skladišta.

NAPOMENA Kada obrišete objekat iz skladišta, objekat će biti uklonjen iz datoteke skladišta objekata i više se neće pojavljivati na bilo kojoj kartici skladišta podataka. Naravno, forma i datoteka izvornog koda koja opisuje objekat neće biti obrisana sa Vašeg hard diska.

Objekat može biti prebačen sa jedne strane na drugu jednostavnim prevlačenjem objekta sa okvira za listu Objects u okvir za listu Pages. Pustite objekat koji ste prevukli na karticu na kojoj želite da se ubuduće nalazi odabrani objekat; time je objekat pomeren.

Administriranje karticama

U prethodnom poglavlju ste radili editovanje, brisanje i pomeranje objekata sa kartice na karticu. Isto tako možete da dodate, obrišete, ili uklonite kartice skladišta objekata, koristeći okvir za dijalog koji se koristi za konfigurisanje skladišta objekata. Pre nego što obrišete stranicu, morate obrisati sve objekte na stranici. Nakon što je kartica ostala prazna, možete je ukloniti klikom miša na naziv kartice koja je prikazana u okviru za listu Pages. Naziv možete promeniti na potpuno isti način. Kada odaberete karticu i kliknete na dugme Rename Page (promena naziva kartice), okvir za dijalog će se pojaviti i tražiti od Vas da upišete novi naziv kartice.

Redosled kojim se kartice pojavljuju u okviru skladišta objekata, može biti izmenjen. Da biste promenili redosled kartica, kliknite na karticu koju želite da premestite, a zatim kliknite na dugme sa strelicom na gore, ili dole, koje se nalazi ispod okvira za listu Pages, kako biste pomerali karticu gore-dole po listi. Takođe, možete da prevučete karticu na novo mesto, ukoliko to želite.

Podešavanje generičkih formi i projekata

Okvir za dijalog koji se koristi za konfigurisanje skladišta objekata Vam omogućava da definišete tri generička objekta:

- Generičku formu koja se koristi kada odaberete opciju File➥New Form u okviru glavnog menija.
- Generičku formu koja se koristi kao glavna forma kada odaberete opciju File➥New Application u okviru glavnog menija.
- Generički projekat koji se koristi kada odaberete opciju File➥New Application u okviru glavnog menija.

Uočićete da se u zavisnosti od objekta koji ste odabrali, pojavljuju dva polja za potvrdu između okvira za listu Objects. Ukoliko odaberete formu, pojavljuju se polja za potvrdu New Form (nova forma) i Main Form (glavna forma). Ukoliko odaberete projekat, pojaviće se polje za potvrdu New Project (novi projekat).

Definisanje forme, ili projekta kao generičke forme, odnosno generičkog projekta, je jednostavno. Pretpostavimo da ste kreirali glavnu formu za koju želite da bude generička glavna forma prilikom kreiranja nove aplikacije. Odaberite formu iz okvira za liste Objects, a zatim kliknite na polje za potvrdu Main Form u dnu ekrana. Kada kliknete mišem na dugme OK, odabrana forma će postati generička. Slično će se dogoditi ukoliko želite da definišete generički projekat. Prvo pronađite projekat u okviru za dijalog koji se koristi za konfiguraciju skladišta objekata, kliknite na projekat koji želite da definišete kao generički, a zatim kliknite kako biste odabrali polje za potvrdu New Project. Od tog trenutka, kada odaberete opciju File➥New Application u okviru glavnog menija, projekat koji ste definisali kao generički će se pojaviti na ekranu.

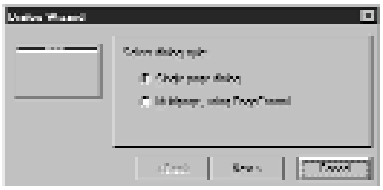
Ukoliko niste pažljivi, greškom možete da odaberete formu kao generičku formu nove aplikacije. Ukoliko se ovo dogodi, proverite svaku formu koja se nalazi u okviru za dijalog koji se koristi za konfiguraciju skladišta objekata. Jedna od formi će imati odabran okvir za potvrdu Main Form. Poništite polje za potvrdu i sve će se vratiti na normalu. Ovo isto važi i za generički projekat. Proverite karticu Projects, kako biste pronašli projekat kod kog je polje za potvrdu New Project odabrano.

Kreiranje formi i aplikacija korišćenjem čarobnjaka (Wizards)

Delphi ima dva ugrađena čarobnjaka koji su kreirani tako da Vas vode kroz proces kreiranja aplikacije. Čarobnjak za dijaloge (Dialog Wizard) Vam pomaže da kreirate okvir za dijalog, a čarobnjak za aplikacije (Application Wizard) Vam pomaže da kreirate osnovni izgled aplikacije. Čarobnjaci će biti obrađeni u narednim poglavljima.

Korišćenje čarobnjaka za dijaloge

Iskreno rečeno, čarobnjak za dijaloge nije od velike koristi, pošto okviri za dijaloge moraju biti izmenjeni u okviru dizajnera forme. Čarobnjak za dijalog se pokreće iz skladišta objekata. Prvo odaberite opciju File➥New u okviru glavnog menija, da biste prikazali skladište objekata, zatim se prebacite na karticu Dialogs, nakon čega treba da kliknete dva puta mišem na ikonu čarobnjaka za dijaloge (Dialog Wizard). Čarobnjak za dijaloge je prikazan na slici 8.5.



Slika 8.5
Čarobnjak za dijaloge

Možete odabrati kreiranje okvira za dijalog sa jednom karticom, odnosno okvir za dijalog sa više kartca. Ikona na levoj strani okvira za dijalog prikazuje kako će okvir za dijalog izgledati nakon svakog koraka. Ukoliko odaberete kreiranje okvira za dijalog sa jednom karticom, kada kliknete na dugme Next (sledeće), videćete sledeću stranu čarobnjaka za dijaloge (pogledati sliku 8.6).

Slika 8.6
Druga strana čarobnjaka za dijaloge

Ova strana Vam omogućava da odaberete dugmad okvira za dijalog (ukoliko ih želite) i da li dugmad treba da se nalaze na desnoj strani, odnosno i donjem delu okvira za dijalog. Ovo je poslednja strana čarobnjaka za dijaloge, ukoliko kreirate okvir za dijalog sa jednom karticom. Nakon što ste odabrali izgled dugmadi, kliknite na dugme Finish (završetak), nakon čega Delphi kreira okvir za dijalog po Vašoj želji.

Novi okvir za dijalog će biti prikazan na dizajneru forme, zajedno sa opcijama koje ste odabrali koristeći čarobnjaka za dijaloge. Karakteristika `BorderStyle` je podešena na `bsDialog`, što je uobičajeno za forme koje se koriste kao okviri za dijalog. Nakon što je čarobnjak za dijaloge kreirao osnovni okvir za dijalog, možete početi da radite kako bi dodali funkcionalne osobine Vašem okviru za dijalog.

Ukoliko odaberete kreiranje okvira za dijalog sa karticama, druga strana okvira za dijalog će dobiti izgled kao što je prikazano na slici 8.7. (Slika 8.7 prikazuje okvir za dijalog nakon što su dodati nazivi kartica.)

Ova strana ima višelinijsku edit kontrolu u koju možete da unesete nazive pojedinih jezičaka kartica, koje želite da vidite na okviru za dijalog. Unesite tekst za svaki jezičak kartice u zasebnu liniju kao što je to prikazano na slici 8.7. Kada kliknete na dugme Next, videćete poslednju stranu čarobnjaka za dijaloge, koja izgleda isto kao slika 8.6. Odaberite pozicije dugmadi, ukoliko želite, a zatim kliknite dugme Finish, kako bi Delphi kreirao okvir za dijalog sa karticama.

Slika 8.7
Čarobnjak za dijalog kreira okvir za dijaloge sa karticama

NAPOMENA Čarobnjak za dijaloge je najkorisniji prilikom kreiranja okvira za dijaloge sa karticama. Kada kreirate okvir za dijalog sa jednom karticom, jednostavnije je odabrati jedan od unapred definisanih okvira za dijalog iz skladišta objekata, umesto raditi dijalog korak po korak, koristeći čarobnjak za dijaloge.

Kreiranje aplikacija korišćenjem čarobnjaka za aplikacije (Application Wizard)

Čarobnjak za aplikacije je korisna alatka koja Vam može pomoći da brzo podesite školjku Vaše aplikacije. Da biste kreirali novu aplikaciju korišćenjem čarobnjaka za aplikacije, odaberite opciju File➥New u okviru glavnog menija. Kada se pojavi prozor skladišta za objekte, kliknite na jezičak kartice Project, a zatim dva puta kliknite mišem na ikonu Application Wizard (čarobnjak za aplikacije).

NAPOMENA Opcija New Application u okviru glavnog menija, kreira novu aplikaciju baziranu na trenutno definisanom generičkom projektu. Iako ste možda očekivali, ova opcija ne pokreće čarobnjaka za aplikacije.

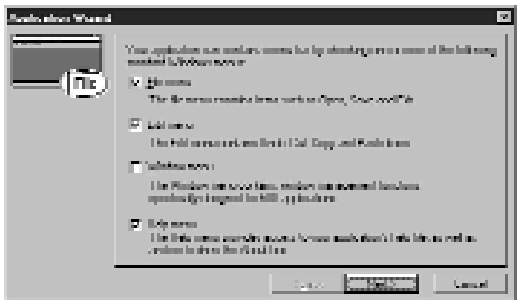
Prođimo kroz čarobnjaka za aplikacije stranu po stranu.

Strana 1: Izbor menija

Kada startujete čarobnjaka za aplikacije prva strana će izgledati kao na slici 8.8.

Ova strana Vam omogućava da odaberete opcije koje želite da sadrži glavni meni Vaše aplikacije. Možete odabrati meni File (datoteke), meni Edit, meni Window (prozori) i meni Help (pomoć). Odaberite svako polje pored opcije menija koje želite da uključite u Vašu aplikaciju.

SAVET Meni Window je obično rezervisan za MDI aplikaciju. Verovatno nećete postaviti meni Window na meni Vaše SDI aplikacije, izuzev ako nemate posebnu aplikaciju koja to zahteva.



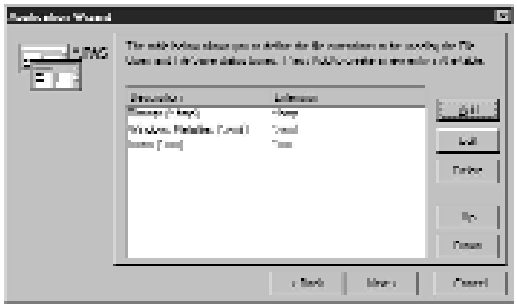
Slika 8.8
Strana jedan čarobnjaka za aplikacije

NAPOMENA Meniji koje dodaje čarobnjak za aplilkacije su odabrani prikazi opcija menija koje se najčešće koriste u Windows aplikacijama. Zapamtite da je čarobnjak za aplikacije tu sa željom da Vam pruži početno kreiranje Vaše aplikacije; da Vam da osnovnu strukturu. Na Vama je da iskoristite osnovnu strukturu i izmenite je tako da napravite aplikaciju koja radi.

Nakon što ste odabrali menije koje želite da koristite u Vašoj aplikaciji, kliknite na dugme Next kako biste prešli na sledeću stranu.

Strana 2: Podešavanje filtera okvira dijaloga za datoteke

Ukoliko odaberete meni File za Vašu aplikaciju, sledeća strana će izgledati kao na slici 8.9.



Slika 8.9
Podešavanje filtera dijaloga za datoteke

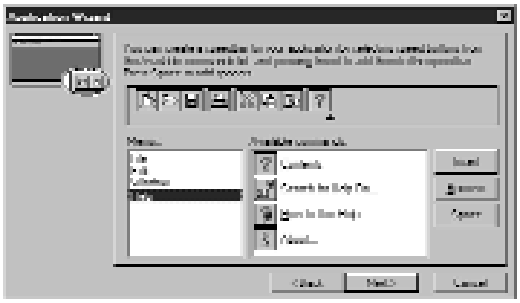
Ova strana Vam omogućava da podesite filtere koje će okviri za dijalog File Open i File Save u okviru Vaše aplikacije koristiti. (Slika 8.9 prikazuje okvir za dijalog nakon dodavanja filtera.) Kliknite mišem na dugme Add kako bi dodali nov filter. Okvir za dijalog koji će se prikazati će Vas upitati za opis i `filter`. Upišite filtere na isti način kao što definišete karakteristiku Filter za uobičajene komponente okvira dijaloga za datoteke. Unesite opis filtera, a zatim masku datoteke (`*.bmp`, na primer). Dugmad

Edit, Delete, Up i Down (editovanje, brisanje gore i dole) se mogu koristiti ukoliko je to neophodno za izmenu, brisanje, odnosno pomeranje filtera u okviru liste.

NAPOMENA Strane dva i tri će biti prikazane ukoliko odaberete menije na strani jedan čarobnjaka za aplikacije. Tačnije, strana dva će biti prikazana samo ukoliko odaberete meni File na strani jedan.

Strana 3: Postavljanje trake sa dugmadima prečica (speedbar)

Strana tri čarobnjaka za aplikacije Vam pomaže da definišete traku sa dugmadima prečicama - speedbar (takođe, ima naziv i traka sa alatima - toolbar) u okviru Vaše aplikacije. Ovo je možda najkorisnija opcija čarobnjaka za aplikacije (što ne znači da ostale opcije nisu korisne). Možete brzo postaviti Vašu traku sa dugmadima prečicama koristeći ovu stranu. Slika 8.10 prikazuje treću stranu čarobnjaka za aplikacije nakon kreiranja trake sa dugmadima prečicama.



Slika 8.10 Postavljanje trake sa dugmadima prečicama

Levi okvir za listu na trećoj strani, sa nazivom Menus (meniji), prikazuje četiri menija u okviru kojih možete birati dugmad. Kada odaberete jedan od menija, dugmad koja su dostupna za taj meni su prikazana na okviru za listu koja se nalazi desno od okvira za listu Menus (sa nazivom Available Commands - raspoložive komande). Da biste dodali dugme na traku sa dugmadima prečicama, kliknite na dugme u okviru za listu Available Commands, a zatim kliknite na dugme Insert. Dugme će biti dodato na uzorak trake sa dugmadima prečicama, koje se nalazi na gornjem delu strane.

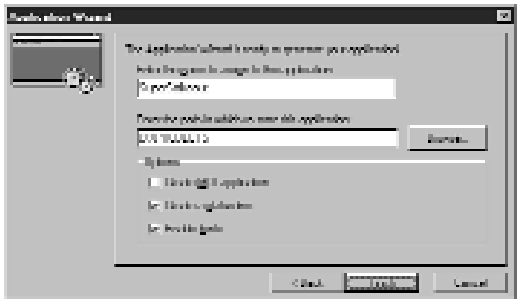
Dugme Space (razmak) se može koristiti za dodavanje separatora na traku sa dugmadima prečicama. Dodavanje separatora vizuelno razdvaja grupe dugmadi. Nastavite sa dodavanjem dugmadi i praznih mesta po želji, sve dok ne kompletirate traku za dugmad prečice. Ukoliko odlučite da uklonite dugme, kliknite na uzorak trake sa dugmadima prečicama, a zatim klinite mišem na dugme Remove (uklanjanje).

NAPOMENA Ukoliko niste želeli da dodate određenu grupu menija u Vašu aplikaciju, za tu grupu menija neće biti prikazana dugmad. Na primer, ako ne dodate meni Window, okvir za listu dostupnih komandi će biti prazan kada kliknete na opciju Window koja se nalazi u okviru za listu Menus.

SAVET Neke posebne aplikacije imaju traku sa dugmadima prečicama, ili nemaju meni. Da biste kreirali traku sa dugmadima prečicama u čarobnjaku za aplikacije, prvo morate kreirati meni. Da biste ovo zaobišli, saopštite čarobnjaku za aplikacije da želite meni, a zatim napravite traku za dugmad prečice. Nakon generisanja aplikacije, možete obrisati komponentu `MainMenu` iz aplikacije, kako bi uklonili meni.

Strana 4: Podešavanje završnih opcija

Četvrta i poslednja strana čarobnjaka za aplikacije omogućava Vam da definišete naziv programa, put koji određuje gde će program biti snimljen na hard disk i nekoliko završnih opcija. Slika 8.11 prikazuje poslednju stranu čarobnjaka za aplikacije.



Slika 8.11
Završne opcije čarobnjaka za aplikacije

Prvo polje na ovoj strani se koristi za definisanje naziva aplikacije. Ovo nije naziv koji će se prikazati u okviru za dijalog Project Options (opcije projekta), nego naziv datoteke koju će Delphi koristiti za snimanje projekta. Još uvek treba da definišete naziv aplikacije u okviru za dijalog opcije projekta. Drugo polje se koristi za definisanje direktorijuma u koji će se snimiti projekat. Ukoliko ne znate tačan put, kliknite na dugme Browse sa desne strane polja i odaberite put, koristeći okvir za dijalog za izbor direktorijuma (Select Directory).

SAVET Okvire za dijalog za izbor direktorijuma možete koristiti i za kreiranje i za izbor direktorijuma. Kliknite mišem na dugme Browse, da biste prikazali okvir za dijalog za izbor direktorijuma. Unesite putanju za direktorijum koji želite da kreirate, a zatim kliknite mišem na dugme OK, ili pritisnite taster Enter. Delphi će Vas upitati da li želite da kreirate novi direktorijum, ukoliko direktorijum koji ste upisali ne postoji.

Donja polovina poslednje strane Vam nudi tri dodatne opcije. Ukoliko kreirate MDI aplikaciju, kliknite na polje za potvrdu sa nazivom Create MDI Application (kreiranje MDI aplikacija). (MDI aplikacije su bile obrađene u lekciji dana 4, "Istraživanje Delphijevog okruženja".) Preostala dva polja za potvrdu Vam omogućavaju da dodate statusnu traku i tekst za savete Vašim komponentama.

Kada budete sigurni da ste odabrali sve opcije za Vašu novu aplikaciju, kliknite na dugme Finish. Delphi će kreirati aplikaciju baziranu na opcijama koje ste definisali. Delphi će napisati onoliko koda koliko je to moguće za aplikaciju. Ovo nije puno, ali

će bar neke osnovne stvari biti napisane. Na primer, ukoliko odaberete meni File, upravljač događajem `FileOpenClick` će biti prikazan i izgledaće ovako:

```
procedure TMainForm.FileOpen(Sender: TObject);
begin
  if OpenDialog.Execute then
  begin
    { Add code to open OpenDialog.FileName }
  end;
end;
```

Kod za izvršavanje okvira za dijalog File Open je na svom mestu; Vi samo treba da napišete kod koji radi sa vraćenim nazivima datoteka.

Nakon što čarobnjak za aplikacije kreira projekat, možete odabrati opciju Project➥Add to Repository kako biste snimili projekat za kasniju upotrebu. Ovo će Vam uštedeti trud kreiranja Vaših novih aplikacija korišćenjem čarobnjaka za aplikacije. Možda ćete poželeti da dodate okvir About (okvir za opis programa), pre nego što snimite projekat u skladište objekata.

Korišćenje čarobnjaka je brzo i jednostavno. Još uvek treba da napišete program, naravno, ali Delphi Vam daje odskočnu dasku kako bi Vas spasao maltretiranja oko kreiranja osnovnih elemenata aplikacije. Pošto je Delphi okruženje ljubazno prema brzom razvoju aplikacija (RAD - friendly), čarobnjaci još više pojednostavljuju proces brzog kreiranja aplikacija (RAD - Rapid Application Development). Delphijevi čarobnjaci su nešto kao RAD za RAD!

Delphi sadrži i druge čarobnjake, pored čarobnjaka za dijaloge i čarobnjaka za aplikacije. Na primer, čarobnjak baza podataka (biće obrađen u lekciji dana 17, "Kreiranje formi za baze podataka") se koristi da kreira forme za baze podataka, a čarobnjak za ActiveX kontrole (obrađen u lekciji dana 15, "COM i AktiveX") Vam pomaže da kreirate AktiveX kontrole. Ovo su specijalizovani čarobnjaci, pa ih nisam obradio u ovom poglavlju.

Dodavanje metoda i polja podataka u kod

Kao što ste mogli do sada da saznate, Delphi je odličan alat za kreiranje korisničkog interfejsa kao dela Windows aplikacije. Delphi kreira upravljače događajima umesto Vas, tako da možete da unosite kod koji će upravljati Vašom aplikacijom. Neće proteći mnogo vremena, a Vi ćete imati potrebu da dodajete komplikovaniji kod u Vaše aplikacije.

Delom to znači da ćete dodavati Vaša polja sa podacima i metode kodu koji je Delphi generisao. Na primer, jednostavna aplikacija može da sadrži dvadesetak upravljača događajima različitih tipova. Delphi kreira sve ove upravljače događajima umesto Vas; Vi jednostavno treba da popunite praznine radnim kodom. Da biste Vašu aplikaciju učinili pouzdanom, aplikacijom koja radi, možda ćete morati napisati još dvadesetak metoda samostalno.

Dodavanje Vaših metoda u polja podataka, pored koda koji je generisao Delphi, nije težak zadatak, ali trebali bi da znate pravila, u protivnom ćete upasti u neprilike.

Kako Delphi upravlja dekleracijama klasa

Kao što ste već saznali, kada kreirate novu formu u okviru dizajnera forme, Delphi automatski kreira datoteku izvornog koda junita. Kada Delphi kreira dekleraciju klase, u osnovi kreira dva odeljka. Prvi odeljak je deo sa dekleracijom klase kojom Delphi upravlja. Drugi odeljak je deo kojim Vi upravljate.

U lekciji dana 6, "Rad sa dizajnerom forme i dizajnerom menija", kreirali ste program ScratchPad. Ukoliko ste radili vežbe na kraju poglavlja, za program ste kreirali i okvir koji opisuje program, a takođe i dodali nekoliko dugmadi. Listing 8.1 sadrži dekleraciju klasa glavne forme, nakon što ste dodali ove izmene.

Zapamtite da se zasebne dekleracije komponenti pojavljuju na osnovu redosleda postavljanja komponenti na formu. Vaša deklereacija klasa bi trebalo da ima iste komponente kao što je to prikazano na listingu 8.1, ali sa nešto drugačijim redosledom.

Listing 8.1: Dekleracija klasa za glavnu formu programa `ScratchPad`

```
TMainForm = class(TForm)
    StatusBar1: TStatusBar;
    ToolBar1: TToolBar;
    ToolButton1: TToolButton;
    ToolButton2: TToolButton;
    Memo: TMemo;
    MainMenu: TMainMenu;
    FileMenu: TMenuItem;
    FileNew: TMenuItem;
    FileOpen: TMenuItem;
    FileSave: TMenuItem;
    FileSaveAs: TMenuItem;
    N1: TMenuItem;
    FilePrint: TMenuItem;
    FilePrintSetup: TMenuItem;
    N2: TMenuItem;
    FileExit: TMenuItem;
    Edit1: TMenuItem;
    EditReplace: TMenuItem;
    EditFind: TMenuItem;
    N4: TMenuItem;
    EditPaste: TMenuItem;
    EditCopy: TMenuItem;
    EditCut: TMenuItem;
    N5: TMenuItem;
    EditUndo: TMenuItem;
    Help1: TMenuItem;
    HelpAbout: TMenuItem;
```

```
    HelpContents: TMenuItem;
    EditSelectAll: TMenuItem;
    N3: TMenuItem;
    EditWordWrap: TMenuItem;
    OpenDialog: TOpenDialog;
    SaveDialog: TSaveDialog;
    MemoPopup: TPopupMenu;
    PopupCut: TMenuItem;
    PopupCopy: TMenuItem;
    PopupPaste: TMenuItem;
    procedure FileExitClick(Sender: TObject);
    procedure EditCutClick(Sender: TObject);
    procedure EditCopyClick(Sender: TObject);
    procedure EditPasteClick(Sender: TObject);
    procedure FileNewClick(Sender: TObject);
    procedure FileSaveClick(Sender: TObject);
    procedure FileOpenClick(Sender: TObject);
    procedure FileSaveAsClick(Sender: TObject);
    procedure EditUndoClick(Sender: TObject);
    procedure EditSelectAllClick(Sender: TObject);
    procedure EditWordWrapClick(Sender: TObject);
    procedure HelpAboutClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Veoma je važno da shvatite da je odeljak između prve linije dekleracije klasa i ključne reči `private` deo koji se može smatrati nedostupnim. Kao kada Vam kažu: "Nemojte ići ovuda." Ovaj odeljak treba da ostavite Delphiju da sam upravlja njime.

Postavljanje bilo kog koda u odeljak kojim upravlja Delphi vezan za dekleraciju klasa formi, može prouzrokovati probleme sa Vašim programom. U nekim slučajevima, možete dobiti kao odgovor grešku prevodioca. U drugim slučajevima, Vaš program se ne može popraviti (neuobičajeno, ali moguće). Steknite naviku da izbegavate ovaj odeljak dekleracije klasa kao da je kužan.

Bezbedno možete postaviti bilo koju klasu polja podataka, odnosno dekleracije metoda u odeljak `private`, odnosno `public` za dekleraciju klasa. Možete dodati odeljak `protected` i postaviti polja podataka, odnosno metode i u ovaj odeljak.

Nekoliko reči o statusnoj traci i savetima (status bar, hints)

U jednom trenutku ste dodali podršku za prikazivanje teksta saveta u okviru statusne trake programa `ScratchPad`. Pre nego što ovo uradite, potrebno je da ukratko saznate šta je to tekst saveta i kako se njime upravlja.

Kada je karakteristika objekta `Application`, `ShowHint` definisana kao `True` (generički), a kursor miša postavljen preko komponente čija je karakteristika `ShowHint` isto tako podešena na `True`, biće aktiviran događaj saveta. Karakteristika objekta `Application`, `Hint` će sadržati tekst saveta za kontrolu koja generiše događaj saveta. Aplikacija može koristiti događaj `OnHint` da bi prikazala savet na statusnoj traci.

Problem je što direktno ne možete pristupiti događaju OnHint objekta `Application`. Ono što možete uraditi je ponovno dodeljivanje vrednosti događaja `OnHint` da bi ukazivao na neku Vašu metodu. Zatim, ukoliko se događaj saveta dogodi, događaj biva preusmeren na jedan od upravljača događajem `OnHint`. Da biste ovo uradili, treba da napišete Vaš upravljač događajem za događaj `OnHint`. To će biti sledeće što ćemo uraditi.

Dodavanje metoda u Vaš kod

Da bi ilustrovali dodavanje metoda u aplikaciju, implementirajmo tekst saveta programa `ScratchPad` koji ste ranije napisali. Za početak, ponovo otvorite program `StrechPad`.

U ovoj grupi koraka ćete dodeliti tekst saveta svakom dugmetu trake sa alatima i pripremiti statusnu traku da prihvati savete. Zapamtite da su dugmad trake sa alatima koje ste postavili na traku sa alatima u lekciji dana 6, još uvek neaktivna, ali Vas to neće sprečiti da dodate savete. Izvršite sledeće korake:

1. Uverite se da je glavna forma `ScratchPad` vidljiva. Kliknite na prvo dugme trake sa alatima glavne forme.
2. Pronađite karakteristiku `Hint` u okviru prozora Object Inspector, a zatim pišite tekst saveta:

   ```
   Open➥Open an Existing File
   ```

3. Promenite karakteristiku `ShowHint` u `True`.
4. Ponovite korake dva i tri za ostalu dugmad na traci za alate dodajući tekst saveta koji želite.
5. Kliknite na statusnu traku koja je postavljena duž donje ivice glavne forme. Promenite karakteristiku `SimplePanel` u `True`. Ovo Vam omogućava da prikažete tekst string preko karakteristike `SimpleText`, koristeći kompletnu širinu statusne trake.

Uredu, sada je sve spremno za rad, pa je vreme da uradite ono zbog čega ste se vratili na ovaj program. Kreiraćete sopstveni upravljač događajem `OnHint`, a zatim metodu nazvati `MyOnHint`. Izvršimo ovaj proces korak po korak. Prvo, dodajte dekleraciju metode dekleraciji klase. Krenimo:

1. Pređite na editor koda i uverite se da je vidljiva datoteka `SPMain.pas`.
2. Pronađite dekleraciju klase za klasu `TScratchPad`, a zatim odeljak `private`. Dodajte sledeću liniju koda nakon ključne reči `private`:

```
procedure MyOnHint(Sender : TObject);
```

Da bi videli šta treba da uradite iz perspektive, poslednjih par linija iz klase bi trebalo da izgleda ovako:

```
private
  { Private declarations }
  procedure MyOnHint(Sender : TObject);
public
  { Public declarations }
end;
```

Uredu, za sada sve ide dobro. Sad ćete dodati dekleraciju metoda za Vašu novu metodu. Još dva koraka i bićete gotovi. Prvo treba da dodate aktuelnu metodu u odeljak Implementation. Nakon toga, treba da dodelite Vašu novu metodu događaju `OnHint` objekta `Application`. Sledite korake:

1. Pozicionirajte se na kraj odeljka Implementation.
2. Upišite sledeći kod (iznad poslednje ključne reči end junita):

```
procedure TMainForm.MyOnHint(Sender: TObject);
begin
  StatusBar.SimpleText := Application.Hint;
end;
```

3. Vratite se na prozor Object Inspector. Odaberite glavnu formu `ScratchPad` koristeći selektor objekata.
4. Prebacite se na karticu Events u okviru prozora Object Inspector i dva puta kliknite mišem na kolonu Value, pored događaja `OnCreate`. Editor koda će biti prikazan i spreman za upisivanje koda.
5. Unesite liniju koda tako da metoda `FormCreate` izgleda ovako:

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  Application.OnHint := MyOnHint;
end;
```

6. Prevedite i pokrenite program. Tekst dugog saveta koji ste upisali će se pojaviti na statusnoj traci kada postavite kursor miša preko trake sa alatima. Tekst kratkog saveta će biti prikazan u oblačiću za savet kada zastanete na dugmetu.

Korak dva podešava tekst saveta (za karakteristiku Hint objekta `Application`) za karakteristiku `SimpleText` komponente `StatusBar`. Korak pet uzima metodu koju ste kreirali u drugom koraku i dodeljuje je događaju `OnHint` klase

`Application`. Svaki put kada se dogodi događaj `OnHint`, metoda `MyOnHint` se poziva i tekst saveta se ispisuje na statusnoj traci.

U prethodnom primeru implementacije saveta za statusnu traku išao sam okolnim putem. Želeo sam da Vam pokažem kako da dodate metode u Vašu formu i kako da dodelite metode događajima. Postoji jednostavniji način da se implementira tekst statusne trake. Jednostavno podesite karakteristiku `AutoHint` na `True`. Još uvek Vam preostaje da definišete tekst saveta za svaku komponentu, ali sve ostalo je automatizovano. Karakteristika `AutoHint` je novina u Delphiju 4.

Dodavanje polja podataka klasa

Dodavanje polja podataka klasa, klasama generisanim u Delphiju radi na potpuno isti način. Sve što treba da uradite je da se uverite da ste dodali polje podataka u odeljak `private`, ili `public` dekleracije klasa, kao što ste to učinili ranije kada ste dodavali metode klasama. Takođe, možete postaviti dekleracije polja podataka u odeljak `protected`, ukoliko ste kreirali jednu od Vaših klasa.

Brisanje koda koje je Delphi generisao

Biće situacija kada je potrebno da obrišete kod koji je Delphi generisao u Vašoj aplikaciji. Na primer, možda ćete imati dugme na formi koje zbog promene u dizajnu više neće biti potrebno. Da bi obrisali dugme potrebno je da odaberete dugme na dizajneru forme i pritisnete taster Delete na tastaturi; i dugmeta više nema. Delphi je obrisao dugme, ali upravljač događajem OnClick dodeljen dugmetu je još uvek tu.

Delphi je registrovao da je dugme dodeljeno upravljaču događajem `OnClick` obrisano, ali ne briše upravljač događajem, pošto postoji mogućnost da druge komponente koriste isti upravljač događajem. Na Vama je da obrišete upravljač događajem, ukoliko želite da ga uklonite iz koda.

Brisanje upravljača događajem je jednostavan posao. Jednostavno uklonite kod iz upravljača događajem i snimite, ili prevedite projekat. Delphi će ukloniti sve prazne događaje na koje naiđe.

Neko bi rekao da ukoliko niste sigurni da li upravljač događajem koristi neka druga komponenta, upravljač događajem treba ostaviti u kodu. Po mom mišljenju, ovo je loše rešenje. Treba da preuzmete odgovornost za Vaš kod i da se oslobodite svih metoda koje ne koristite. Iako neiskorišćen, kod neće nikog povrediti, ali vodi ka uvećanju `.exe` datoteke. U nekim slučajevima neiskorišćen kod može degradirati peformanse programa. Budite revnosni u čišćenju Vaših programa od neefikasnog koda i koda koji se ne koristi.

Kreiranje obrazaca komponenti

NOVI TERMIN Obrazac komponente *(component template)* je komponenta, odnosno grupa komponenti koje ste izmenili po želji, a zatim snimili za kasniju upotrebu.

Obrazaci komponente Vam omogućavaju da kreirate, snimite i ponovo koristite grupe komponenti. U suštini obrazac komponente ne mora da bude grupa - može biti i jedna jedina komponenta. Kratak primer će Vam verovatno pomoći da vidite kako je korisno imati komponente obrasce. Ali pre toga, kratka lekcija o Windows edit kontroli.

Standardna Windows edit kontrola u jednoj liniji, kao sve Windows kontrole, ima određena, unapred definisana ponašanja. Jedno od ovih ponašanja upravlja korišćenjem tastera Enter. Ukoliko korisnik pritisne taster Enter kada je u edit kontroli, Windows će tražiti generičko dugme na prozoru. Ukoliko je generičko dugme pronađeno, Windows će istovremeno kliknuti na dugme.

Šta ovo znači za Vas? Recimo da imate nekoliko edit kontrola na formi i da generičko dugme kao što je dugme OK (ili bilo koje drugo dugme sa karakteristikom `Default`) podešeno na `True`. Kada pritisnete taster Enter, u trenutku kada je edit kontrola u fokusu, forma će biti zatvorena. Ukoliko na formi ne postoji generičko dugme, Windows-i će Vas obavestiti zvučnim signalom. Iako je to standardno ponašanje Windows-a, većina korisnika to smatra iritirajućim i zbunjujućim. Ono što većina korisnika više voli, naročito kada radi sa formom koja ima nekoliko edit polja, je da taster Enter menja fokus na sledeću kontrolu, umesto da zatvara formu.

Rešenje ovog problema je prilično jednostavno. Sve što treba da omogućite je upravljač događajem za događaj `OnKeyPress` i dodati kod koji izgleda ovako:

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if Key = Char(VK_RETURN) then begin
    Key := #0;
    PostMessage(Handle, WM_NEXTDLGCTL, 0, 0);
  end;
end;
```

Ovaj kod prvo proverava da li je pritisnut taster Enter (kod virtuelnog tastera `VK_RETURN`). Ukoliko je to tačno, podešava vrednost promenljive `Key` na `#0`. Ovo eliminiše zvučni signal koji Windows-i emituju kada se pritisne taster Enter na edit kontroli. Sledeća linija šalje formi Windows poruku `WM_NEXTDLGCTL`. Ova poruka postavlja fokus na sledeću kontrolu koja je na redosledu tabulatora. I to bi bilo sve.

Nakon što ste napisali kod za Vašu novu `Edit` komponentu, možete ga snimiti kao obrazac za komponentu. Kada to uradite kompletan kod je snimljen zajedno sa komponentom. Bilo koji kod obrasca koji kreirate postavlja se na karticu Templates

palete komponenti. Hajde da kreiramo obrazac komponente i da vidimo kako će raditi. Izvršite sledeće korake:

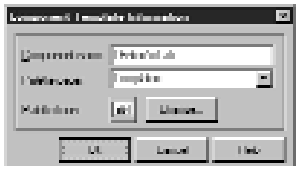
1. Postavite komponentu `Edit` na praznu formu. Promenite karakteristiku `Name` komponente u `EnterAsTab`, a zatim obrišite sadržaj karakteristike `Text`.

2. Pređite na karticu Events u okviru prozora Object Inspector i kreirajte upravljač događajem za događaj `OnKeyPress`. Upišite ovaj kod u upravljač događajem:

```
if Key = Char(VK_RETURN) then begin
  Key := #0;
  PostMessage(Handle, WM_NEXTDLGCTL, 0, 0);
end;
```

3. Uverite se da je `Edit` komponenta odabrana i odaberite opciju Component➥Create Component Template u okviru glavnog menija. Pojaviće se okvir za dijalog Component Template Information.

4. U polje Component Name upišite `TEnterAsTab`. Okvir za dijalog Component Template Information bi trebao da izgleda kao na slici 8.12.



Slika 8.12
Okvir za dijalog Component Template Information

5. Kliknite na dugme OK da snimite obrazac komponente.

Sada Vaša paleta komponenti ima jezičak kartice sa natpisom Templates. Pređite na karticu Templates (možda ćete morati da pomerite paletu komponenti da biste je pronašli), odaberite Vašu novu komponentu i postavite je na formu. Možete videti da je upravljač događajem `OnKeyPress` uključen u kod prilikom postavljanja komponente na formu.

SAVET Ukoliko imate nekoliko komponenti u okviru forme, kod za svaki upravljač događajem OnKeyPress će biti ponovljen prilikom postavljanja komponente `EnterAsTab` na formu. Da ne bi duplirali kod, na formu možete postaviti samo jednu komponentu `EnterAsTab`. Sve ostale komponente mogu biti standardne `Edit` komponente koje imaju događaj `OnKeyPress` zakačen na upravljač događajem `OnKeyPress` za komponentu `EnterAsTab`.

Jedna od najvećih prednosti obrasca komponente je da kod koji je napisan u svaki upravljač događajem komponente bude snimljen zajedno sa komponentom. Obrasci komponenti Vam omogućavaju da napravite zbirku komponenti koje su prilagođene Vašim potrebama: uobičajeni okviri za dijalog sa unapred definisanim filterima i naslovima, dugmad prečica sa već postavljenim grafikama (slikama), okviri za liste i

kombo okviri koji automatski učitavaju element iz datoteke, i bezbroj raznih drugih mogućnosti.

Iako koncept obrasca komponente radi sa jednom komponentom, više smisla će imati rad sa više komponenti. Ukoliko imate grupu komponenti koje treba da postavite na Vašu formu svaki put kada radite na novoj formi, za ove komponente možete kreirati obrazac. Nakon što kreirate obrazac komponente, ponovno korišćenje grupe komponenti je udaljeno od Vas samo jedan klik mišem.

Sigurno da postoje neke sličnosti između obrazaca komponenti i snimanja formi u skladište objekata. Treba da koristite obrasce komponenti za grupe komponenti koje obično koristite kao deo veće forme. Skladište objekata koristite kada želite da snimite kompletne forme koje ćete ponovo koristiti.

Korišćenje resursnih datoteka

Svaki Windows program koristi resurse. Resursi *(resources)* su oni elementi programa koji podržavaju program, ali nisu deo izvršnog koda.

Tipični resursi Windows programa su:

- akceleratori
- bitmape
- kursori
- okviri za dijalog
- ikone
- meniji
- tabele podataka
- tabele stringova
- informacija o verziji programa
- posebni resursi koje korisnik definiše (zvučne i video datoteke, na primer)

Informacije o verziji se mogu jednostavno dodati Vašim Delphi projektima korišćenjem kartice Version Info u okviru za dijalog opcije projekta (Project Options). Okvir dijaloga za opcije projekta će biti detaljnije obrađen u sutrašnjoj lekciji.

Resursi se uglavnom nalaze u resursnoj skript datoteci (*resource script file* - tekst datoteka sa nastavkom `.rc`). Resursna skript datoteka se prevodi korišćenjem resursnog prevodioca, a zatim povezuje sa izvršnom datotekom (`.exe`) aplikacije u toku faze povezivanja programa.

Smatra se da se resursi povezuju sa izvršnom datotekom. Neki resursi, kao što su bitmape, tabele stringova i zvučne datoteke mogu biti postavljeni u eksterne datoteke (`.bmp, .txt i .wav`), odnosno mogu biti ubačeni u izvršnu datoteku i nalaziti se u okviru ove datoteke. Moguće je birati način povezivanja ovih datoteka. Postavljanje resursa u izvršnu datoteku (`.exe`) ima dve glavne prednosti:

- Resursima se može pristupiti puno brže, pošto je potrebno manje vremena za pronalaženje resursa u okviru izvršne datoteke, nego za učitavanje resursa sa diska.
- Datoteka programa i resursi se mogu pronaći u istoj datoteci (`.exe`), pa nema potrebe za mnoštvom dodatnih datoteka za podršku.

Nezgodna strana ovakvog pristupa je što će Vaša izvršna datoteka biti nešto veća. Datoteka programa neće biti puno veća od kombinacije eksternih resursnih datoteka i izvršne datoteke, ali će dodatno povećanje datoteke, kao rezultet imati duže učitavanje programa.

Vaše potrebe će definisati da li želite da zadržite resurse u eksternoj datoteci, odnosno da li treba da ih povežete sa izvršnom datotekom. Najvažnija stvar je da možete koristiti oba načina (oba načina možete koristiti u istom programu).

Resursi u Delphiju

Tradicionalni Windows programi skoro uvek sadrže bar jedan okvir za dijalog i ikonu. Delphijeve aplikacije se, ipak, malo razlikuju. Prvo, ne postoje pravi okviri za dijalog u Delphi aplikacijama, znači da nema pravih resursa okvira za dijalog (Delphijeve forme su smeštene kao resursi, ali postoje kao `RCDATA` resursi, a ne kao resursi okvira za dijalog.

Delphijeve aplikacije nemaju tradicionalne resurse ikona. Delphi brine o kreiranju resursne datoteke za ikonu nakon što kreirate aplikaciju. Slično je prilikom izbora bitmape za dugmad prečice, komponente tipa `Image`, odnosno komponente tipa `BitBtn`. Delphi će uključiti bitmapiranu datoteku koju ste odabrali u resurs forme. Forma i svi njeni resursi će zatim biti dodati u datoteku programa prilikom kreiranja aplikacije. Sve se to manje više može obaviti automatski.

Postoje situacije kada želite da implementirate resurse van normalnih Delphi procesa. Na primer, ukoliko želite da pokrenete animaciju, treba da imate grupu bitmapa koje će se učitavati kao resursi i imati najveću moguću brzinu izvršavanja. U takvoj situaciji je potrebno da znate kako da povežete resursne datoteke sa datotekom Delphi programa.

Čin povezivanja resursne datoteke sa izvršnom datotekom je veoma jednostavan. Ustvari puno je teže kreirati resurse. Kreiranje osnovnih resursa kao što su bitmape, ikone i kursori, nije teško ukoliko imate dobar editor resursa, ali kreiranje 3D bitmapa i ikona profesionalnog kvaliteta je samo za sebe umetnost. Koliko ste

puta videli pristojan program sa užasnim sličicama na dugmadima? Ja sam ih video veoma mnogo. (Izvinite, malo sam skrenuo s teme.) Možete kreirati bitmape, ikone i kursore koristeći Delphi Image Editor. (Editor slika će biti obrađen u lekciji dana 11, "Delpijevi alati i opcije".)

Ukoliko želite da kreirate string resursa, resurse sa podacima korisnika, resurse zvučnih datoteka, ili bilo koji drugi poseban resurs, verovatno će Vam biti potreban editor resursa nekog drugog proizvođača.

NAPOMENA Ukoliko imate staru kopiju Borland Pascal-a, možete koristiti program Resource Workshop ovog programa, da bi editovali posebne resurse. Nakon kreiranja resursa, imaćete datoteku sa nastvkom `.rc` koju možete prevesti u `.res` datoteku koristeći Borlandov resursni prevodilac (Borland Resource Compiler - `BRCC32.EXE`). Borlandov resursni prevodilac se nalazi u okviru Delphija. Tehnički gledano, možete kreirati `.rc` datoteku koristeći bilo koji editor teksta, aprevesti ga koristeći resursni prevodilac, ali u stvarnosti je puno lakše koristiti resursni editor.

Prevođenje resursnih datoteka

Nakon što kreirate resursne datoteke potrebno je da ih prevedete koristeći resursni prevodilac. Ovo možete uraditi na jedan od sledeća dva načina:

- Prevesti resursnu datoteku manuelno iz komandne linije.
- Dodati opciju beč datoteke Vašoj projektnoj grupi.

Na ova dva načina dobijate datoteku `.res` koju možete povezati sa Vašom aplikacijom (ovo će biti obrađeno nešto kasnije). Grupe projekata će detaljnije biti obrađene u sutrašnjoj lekciji.

Prevođenje iz komandne linije

Da biste preveli resursnu datoteku iz komandne linije, otvorite DOS prozor u okviru Windows-a, a zatim upišite komandu poput ove:

```
brcc32 jjres.rc
```

Ovo podrazumeva da se direktorijum `Delphi 4➥Bin` nalazi u okviru komande Path datoteke `AUTOEXEC.BAT`. Ukoliko to nije slučaj, treba da upišete kompletnu putanju do datoteke `BRCC32.EXE`. Resursni prevodilac je veoma brz, pa možda nećete ni primetiti da je resursni skript preveden.

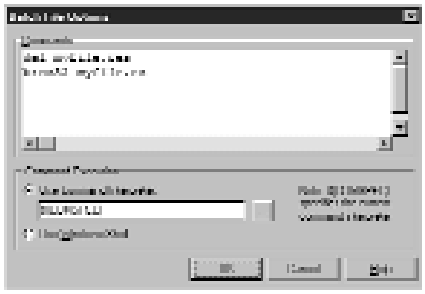
Korišćenje beč datoteke projekta

Dodavanje beč datoteke projekta u Vašu projektnu grupu je isto tako jednostavno kao prevođenje iz komandne linije, a dodatna prednost je što se time obezbeđujete da uvek imate prevedenu najnoviju verzuju resursne datoteke. Da biste dobili ideju kako beč datoteka projekta radi, izvršite sledeće korake:

1. Kreirajte novu aplikaciju. Ova aplikacija će se koristiti samo da Vam da uvid u proces.
2. Odaberite opciju View➥Project Manager da bi otvorili Delphijev Project Manager.
3. Kliknite mišem na dugme Add New Project na traci za alate prozora Project Manager. Biće prikazan prozor skladišta objekata.
4. Dva puta kliknite mišem na ikonu Batch File, da biste kreirali novu beč datoteku projekta. Beč datoteka projekta je dodata u Project Manager pod nazivom `Project2`.
5. Kliknite desnim tasterom miša na nod datoteke i odaberite opciju Save. Datoteku snimite pod nazivom `test.bat`.
6. Kliknite desnim tasterom miša ponovo na nod i odaberite opciju Edit➥Options. Biće prikazan okvir za dijalog Batch File Options.
7. Upišite tekst u memo polje Commands:

```
del myfile.res
brcc32 myfile.rc
```

Slika 8.13 prikazuje okvir za dijalog Batch File Options nakon poslednjeg koraka.



Slika 8.13
Okvir za dijalog Batch File Options

8. Kliknite na dugme OK kako biste zatvorili okvir za dijalog Batch File Options.

U prethodnoj vežbi ste kreirali beč datoteku koja će se izvršiti prilikom prevođenja grupe projekata. Komanda beč datoteke koju ste upisali u koraku sedam, briše datoteku pod nazivom `myfile.res`, a zatim poziva Delphijev resursni prevodilac da bi preveo datoteku `myfile.rc`. Prevođenje datoteke `myfile.rc` korišćenjem resursnog prevodioca će kao rezultat kreirati datoteku pod nazivom `myfile.res`.

Predvidljivo je da će sledeći projekat u grupi projekata koristiti datoteku `myfile.res`. Možda se pitate zašto sam prvo obrisao datoteku `myfile.res`. Datoteku sam obrisao, pošto sam tako siguran da će datoteka koja se nalazi na disku, biti datoteka koju je resursni prevodilac kreirao. Ukoliko resurasni prevodilac ne uspe da kreira datoteku, svaki sledeći projekat koji je koristi neće biti preveden, a

greška prilikom prevođenja će me upozoriti da nešto nije u redu sa kreiranjem resursne datoteke.

Izvorni kod knjige za današnju lekciju uključuje i projektnu grupu koja koristi beč datoteku projekta na potpuno isti način. Izvorni kod možete skinuti sa adrese:

`http://www.mcp.com/info`

Povezivanje resursnih datoteka sa izvršnom datotekom

Nakon što prevedete resursnu skript datoteku, morate povezati prevedenu resursnu datoteku sa izvršnom datotekom programa. Ovo ćete uraditi koristeći direktivu prevodioca `$R`. Na primer, da biste povezali binarnu resursnu datoteku koja se nalazi u datoteci `myfile.res`, blizu početka junita Vaše glavne forme upišite:

```
[ $R myfile.res ]
```

To bi bilo sve. Sve dok definisana datoteka postoji, Delphi će povezivati prevedene resurse za izvršnu datoteku u toku povezivanja.

Primer programa koji koristi resurse

Listing 8.2 sadrži junit glavne forme za program pod nazivom Jumping Jack. Ovaj program pokazuje jednostavnu animaciju sa zvučnim efektima. Glavna forma sadrži dva dugmeta, komponentu `Image` i komponentu `Label`. Program Jumping Jack ilustruje nekoliko aspekata korišćenja resursa u okviru Delphi aplikacija. Posebno Vam pokazuje kako da učitate bitmape koje su smeštene u resurse, kako da učitate i prikažete string resurse i kako da izvršavate audio datoteke koje se nalaze u resursima. Listing 8.3 je delimičan listing resursne datoteke koju koristi program Jumping Jack. Ispitajte listing, a zatim ću Vam objasniti šta program radi.

Skinite ovaj projekat sa adrese http://www.mcp.com/info da biste ispitali resursnu datoteku i grupu projekta.

Listing 8.2: `JJMain.pas`

```
unit JmpJackU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls, MMSystem;

{$R JJRES.RES}

const
  IDS_UP = 101;
  IDS_DOWN = 102;
```

nastavlja se

Listing 8.2: JJMain.pas *nastavak*

```
type
  TMainForm = class(TForm)
    Image: TImage;
    Label1: TLabel;
    Start: TButton;
    Stop: TButton;
    procedure FormCreate(Sender: TObject);
    procedure StartClick(Sender: TObject);
    procedure StopClick(Sender: TObject);
  private
    { Private declarations }
    Done : Boolean;
    procedure DrawImage(var Name: string);
  public
    { Public declarations }
  end;

var
  MainForm: TMainForm;

implementation

{$R *.DFM}

procedure TMainForm.FormCreate(Sender: TObject);
begin
  Image.Picture.Bitmap.
    LoadFromResourceName(HInstance, 'ID_BITMAP1');
end;

procedure TMainForm.StartClick(Sender: TObject);
var
  S       : string;
Listing 8.2. continued
  ResName : string;
  I       : Integer;
  Buff    : array [0..9] of Char;
begin
  { When the Start button is clicked the animation }
  { loop starts. The bitmap resources are named    }
  { ID_BITMAP1 through ID_BITMAP5 so we'll start   }
  { with a string called 'ID_BITMAP' and append    }
  { the last digit when needed. }
  S := 'ID_BITMAP';

  { A flag to let us know when we're done. }
  Done := False;
```

```
  { Start the loop and keep looping until the }
  { 'Stop' button is pressed. }

  while not Done do begin
    { Loop through the five bitmaps starting with }
    { 1 and ending with 5. }
    for I := 1 to 5 do begin
      { Append the value of 'I' to the end of the string }
      { to build a string containing the resource name.  }
      ResName := S + IntToStr(I);
      { Call a method to display the bitmap. }
      DrawImage(ResName);
    end;
    { Load the 'Up' string resource using the WinAPI }
    { function LoadString, display the string, and   }
    { tell Windows to repaint the Label. }
    LoadString(HInstance, IDS_UP, Buff, SizeOf(Buff));
    Label1.Caption := Buff;
    Label1.Refresh;
    { Play the 'up' sound using the WinAPI function }
    { PlaySound. Play it asynchronously. }
    PlaySound('ID_WAVEUP',
      HInstance, SND_ASYNC or SND_RESOURCE);
    { Pause for a moment at the top of the jump. }
    Sleep(200);
    { Repeat all of the above except in reverse. }
    for I := 5 downto 1 do begin
      ResName := S + IntToStr(I);
      DrawImage(ResName);
    end;
    PlaySound('ID_WAVEDOWN',
      HInstance, SND_ASYNC or SND_RESOURCE);
    LoadString(HInstance, IDS_DOWN, Buff, SizeOf(Buff));
    Label1.Caption := Buff;
    Label1.Refresh;
    Sleep(200);
  end;
end;

procedure TMainForm.StopClick(Sender: TObject);
begin
  { Stop button pressed, so tell the loop to }
  { stop executing. }
  Done := True;
end;

procedure TMainForm.DrawImage(var Name : string);
begin
  { Load the bitmap from a resource }
```

nastavlja se

Listing 8.2: JJMain.pas *nastavak*

```
    { using the name passed to us. }
    Image.Picture.Bitmap.
      LoadFromResourceName(HInstance, name);
    { Must pump the message loop so that Windows }
    { gets a chance to display the bitmap. }
    Application.ProcessMessages;
    { Take a short nap so the animation doesn't
    { go too fast. }
    Sleep(20);
  end;

  end.
```

Listing 8.3: JJRes.rc

```
STRINGTABLE
BEGIN
 101, "Up"
 102, "Down"
END

ID_WAVEUP   WAVE "up.wav"
ID_WAVEDOWN WAVE "down.wav"

ID_BITMAP1 BITMAP LOADONCALL MOVEABLE DISCARDABLE IMPURE
BEGIN
 '42 4D 76 02 00 00 00 00 00 00 76 00 00 00 28 00'
 '00 00 20 00 00 00 20 00 00 00 01 00 04 00 00 00'
 '00 00 00 02 00 00 00 00 00 00 00 00 00 00 00 00'
 remainder of bitmap resources follow
```

ANALIZA Dekleracija klase glavne forme deklariše polje podataka tipa `Boolean` pod nazivom Done koje se koristi da odredi kada treba zaustaviti animaciju. Metoda `DrawImage` se koristi da prikaže bitmapu u komponenti `Image`.

U listingu 8.2 treba da uočite da se koriste dve Windows API funkcije koje služe za učitavanje stringova i zvučnih resursnih datoteka. U metodi `StartClick` funkcija `LoadString` učitava string resurs u tekst bafer baziran na numeričkom identifikatoru stringa (pogledajte listing 8.3 da biste videli kako su string resursi kreirani). String je zatim dodeljen karakteristici `Caption` komponente `Label` u okviru forme.

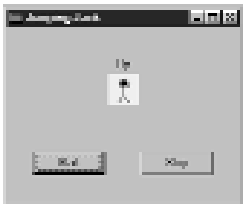
Funkcija `PlaySound` se koristi za puštanje zvučne datoteke koja je definisana kao resurs. Oznaka (flag) `SND_ASYNC` se koristi zajedno sa funkcijom `PlaySound` da bi saopštila Windows-ima da startuju datoteku zvuka i trenutno vrate kontrolu programu (puštanje zvuka je detaljnije obrađeno u lekciji dana 12, "Programiranje grafike i multimedije"). Ovo omogućava nastavak animacije i istovremeno puštanje datoteke zvuka. Oznaka (flag) `SND_RESOURCE` saopštava Windows-ima da je zvuk nalazi u resursima, a ne u datoteci na disku. Obe funkcije `LoadString` i `PlaySound`

koriste globalnu promenljivu `HInstance` da bi saopštili Windows-ima da pogleda u izvršnu datoteku kako bi pronašao resurse. Bitmapirani resursi se učitavaju korišćenjem VCL metode `LoadFromResourceName`.

Prvih pet linija listinga 8.3 ilustruju kako string tabela pretražuje skript datoteku. String tabele možete kreirati veoma jednostavno korišćenjem bilo kog editora teksta. Na osnovu toga kreira se resurs `WAVE` za obe zvučne datoteke koje su prethodno snimljene i nalaze se u direktorijumu projekta. Kada resursni prevodilac pronađe `WAVE` dekleraciju, čita zvučne datoteke i prevodi ih u binarnu resursnu datoteku.

NAPOMENA Kao što možete videti iz listinga 8.3, neke tipove resursa možete kreirati korišćenjem editora teksta. Ukoliko imate bitmape i zvučne zapise snimljene u eksterne datoteke, možete ih uključiti u `.RC` datoteke kao što je to prikazano u listingu 8.3, a zatim ih prevesti u binarnu resursnu datoteku korišćenjem resursnog prevodioca. Kasnije, binarna resursna datoteka se može priključiti izvršnoj datoteci Vaše aplikacije.

Listing 8.3 predstavlja samo deo listinga. Bitmape koje su kreirane tradicionalnim editorima resursa su obično prikazane u resursnim datotekama kao numerički podaci. Opisi resursa za bitmape mogu biti veoma dugi. Ostatak opisa resursa bitmape koji se koristi za bitmapu Jumping Jack, zahteva približno 200 linija resursnog koda, pa sam odlučio da ih ne prikažem sve. Slika 8.14 pokazuje sličicu Jumping Jack u trenutku rada programa.



Slika 8.14
Jumping Jack u toku rada

Kreiranje dodatnih resusa za Vaše programe ne iziskuje rad koji zahtevaju vrhunski programi, ali nije ni jednostavno. Potrebno je neko vreme da biste shvatili kako se sve uklapa. Možda nećete morati da dodajete posebne resurse u Vaše aplikacije. Ukoliko Vam to bude potrebno, imaćete dobru ideju odakle treba početi. U ovom poglavlju sam Vas pomalo iznenadio i zbunio, ali nemojte se brinuti. Vremenom će sve što je opisano dobiti neki smisao.

NAPOMENA Bitmape, ikone i kursori koji se mogu pronaći u drugim programima su obično zaštićeni autorskim pravima. Bez odobrenja nemojte uzimati resurse iz nekog drugog programa koji je zaštićen autorskim pravima. Ubuduće, podrazumevajte da su svi programi zaštićeni autorskim pravima, ukoliko izričito nije navedeno da su besplatni. Bitmape, ikone i kursore u okviru Delphija možete slobodno koristiti (nalaze se u direktorijumu `Common Files\Borland Shared Files\Images`) u Vašim aplikacijama, bez odobrenja firme Borland.

Korišćenje paketa

Nakon što ste završili Vašu aplikaciju možete je distribuirati na jedan od dva načina. (*Distribuiranje* predstavlja čin predaje Vaše aplikacije korisnicima.) Možete distribuirati Vašu aplikaciju korisnicima u okviru Vaše firme, odnosno možete je distribuirati javno. Bilo kako bilo, potrebno je da znate koje su Vam mogućnosti pružene. Uglavnom imate dva izbora: statičko povezivanje, ili dinamičko povezivanje korišćenjem paketa. Obradiću ove opcije kako bi mogli da budete informisani o načinu distribuiranja Vaše aplikacije. Počeću sa objašnjavanjem paketa, a zatim opcija za distribuiranje aplikacije.

Šta je paket?

Pre nego što predočim opcije koje su Vam dostupne, dobra je ideja da definišemo šta je to paket.

NOVI TERMIN *Paket* je deo prevedenog koda koji se nalazi u datoteci sa produžetkom BPL.

Objašnjenje Vam verovatno nije reklo dovoljno, pa ću Vam to malo razjasniti. Kada skinete omot sa paketa, on u osnovi ostaje DLL datoteka (dinamička datoteka za povezivanje) koja ima produžetak `.bpl` umesto tradicionalnog produžetka `.dll`. (Postoji još nešto što nije navedeno, ali ovakav opis je dovoljno blizak za ovu svrhu.) Postoje dva tipa paketa u Delphiju: paketi koji se koriste u toku rada programa i paketi koji se koriste u toku dizajniranja programa. Objasniću svaki od ova dva tipa, tako da bi mogli da shvatite kako paketi rade.

Paketi koji se koriste u toku rada programa (runtime packages)

Paketi koji se koriste u toku rada programa sadrže kod koji je potreban Vašoj aplikaciji u toku rada. Iako Delphi obezbeđuje veoma mnogo različitih paketa, primarni paket se naziva `VCL40.BPL`. Ovaj paket sadrži kompletan osnovni kod VCL-a u okviru jednog jedinog paketa. Ukoliko odaberete da koristite pakete u Vašim aplikacijama, Vaša aplikacija će učitati paket pod nazivom `VCL40.BPL` i po potrebi pozivati rutine iz ovog paketa. Ukoliko Vaša aplikacija koristi baze podataka, koristiće paket `VCLDB40.BPL` i pozivati rutine iz paketa ukoliko je potrebno. Pored dva navedena paketa Delphi sadrži i druge pakete koji ovde nisu spomenuti.

Pored VCL paketa, Vaša aplikacija može koristiti i druge pakete. Ukoliko koristite komponente nezavisnih proizvođača, odnosno komponente koje pišete sami, možete imati ovakav slučaj. Potrebno je da proverite dokumentaciju nezavisnih proizvođača komponenti kako bi otkrili koji paketi su potrebni za rad Vaše aplikacije. Malo sam počeo da objašnjavam stvari koje ćete učiti kasnije, ali ipak mi dozvolite da Vam objasnim nešto o pravljenju paketa, pa ću se vratiti distribuiranju aplikacija koje koriste pakete.

Dizajniranje paketa

Da bi objasnio dizajniranje paketa, možda bi dobra ideja bila da Vam prvo objasnim kako se dizajniraju komponente. Većinu komponenti kreiranih za Delphi možete pronaći u paketu koji se koristi u toku rada programa i paketu koji se koristi u toku dizajniranja. Paket koji se koristi u toku rada programa sadrži sav potreban kod koji je potreban za rad komponente. Paket za dizajniranje sadrži kod potreban za rad na formi u toku dizajniranja, uključujući editore karakteristika i editore komponenti.

Paket za dizajniranje sadrži listu zahteva (Requires list) koja saopštava Delphiju koji paketi su potrebni za rad aplikacije koju koristi paket. Paket za dizajniranje skoro uvek zahteva kod iz paketa koji se koristi za rad programa i verovatno kod iz jednog, ili više VCL paketa. Veoma je važno shvatiti da jedan paket (i paket koji se koristi u toku rada aplikacije i paket koji se koristi u toku kreiranja aplikacije) može sadržati kod za nekoliko komponenti. Nije neophodno imati nekoliko različitih paketa za svaku komponentu.

Pošto paket za dizajniranje sadrži samo kod koji je potreban za prikazivanje komponenti u toku dizajniranja, obično je ovaj paket mnogo manji od paketa koji se koristi u toku rada programa. Paketi koji se koriste u roku dizajniranja, Delphi koristi samo za dizajniranje - nisu potrebni za rad aplikacije.

Statičko povezivanje nasuprot dinamičkog povezivanja

Sada kada znate nešto o paketima, možete naučiti šta je to statičko povezivanje, odnosno dinamaičko povezivanje i po čemu se razlikuju.

Statičko povezivanje

Kada aplikacija koristi statičko povezivanje VCL komponenti, uopšte ne koristi pakete. Sav potreban kod za rad aplikacije se nalazi u okviru izvršne datoteke aplikacije. Vaša aplikacija je zaseban program i ne zahteva bilo koju dodatnu datoteku (pakete, ili DLL datoteke).

Postoje izuzeci za svako pravilo. Iskazi koji statički povezuju aplikaciju ne zahtevaju dodatnu DLL datoteku, mada to podrazumeva nekoliko izuzetaka. Prvi izuzetak je da aplikacija ne koristi baze podataka. Delphijeva aplikacija koja koristi baze podataka za svoj rad zahteva Borland Database Engine (BDE). BDE je, pre svega, zbirka DLL datoteka tako da aplikacija koja koristi baze podataka zahteva za svoj rad dodatne DLL datoteke, čak iako je statički povezana. Drugi izuzetak od pravila je da aplikacija ne koristi ni jednu ActiveX kontrolu. ActiveX kontrole su tip DLL datoteka, pa ukoliko Vaša aplikacija koristi ActiveX kontrole, više ne može da bude samostalna aplikacija.

Iako Vam Delphi daje mogućnost izbora za tip povezivanja, statičko povezivanje se podrazumeva. Statičko povezivanje ima dve osnovne prednosti u odnosu na dinamičko povezivanje:

- Prva prednost je što ne morate da brinete o isporučivanju bilo koje dodatne datoteke uz Vašu aplikaciju. Vaša aplikacija sadrži sav potreban kod za rad i nije potrebno dodavati nikakvu biblioteku koja se koristi u toku rada aplikacije.
- Druga prednost je u činjenici da aplikacija koja se statički povezuje ima manju veličinu od aplikacija koja koristi pakete. Nešto kasnije ću Vam objasniti zbog čega je to tako, kada Vam budem objašnjavao prednosti i mane dinamičkog povezivanja.

Statičko povezivanje ima jednu glavnu prednost, ali se ona pokazuje u aplikacijama koje sadrže puno korisničkih DLL datoteka. Prednost predstavlja činjenica da se VCL i RTL kod duplira u svakom modulu (takođe i u glavnoj aplikaciji), a isto tako i u svakoj DLL datoteci. Ovo znači da se kod duplira nepotrebno.

Na primer, pretpostavimo da svaki modul zahteva minimalno 200KB VCL osnovnog koda i RTL koda. Pretpostavimo da imate glavnu aplikaciju i 10 DLL datoteka za podršku. To znači da se koristi 2200KB koda (11 modula x 200KB po modulu), iako je potrebno samo 200KB koda. Aplikacija i DLL datoteke koje su statički povezane svaka za sebe i ne mogu međusobno deliti VCL i RTL kod.

Dinamičko povezivanje

Dinamičko povezivanje koristi scenario u kom aplikacija dinamički učitava kod svojih biblioteka u toku rada. U slučaju Delphi aplikacija, to znači da se bilo koji paket učitava u toku rada programa. Paketi potrebni za rad će sigurno uključiti jedan, ili više VCL paketa, a možda će zahtevati i pakete nezavisnih proizvođača.

Učitavanje paketa u Vašu aplikaciju je automatsko. Ne treba da pišete kod koji će učitati pakete, pošto Delphi o tome brine umesto Vas. Odabiranje dinamičkog povezivanja umesto statičkog povezivanja ne zahteva nikakvu promenu koda. Kada se radi o distribuiranju aplikacije, neke promene su potrebne, a one će biti objašnjene ubrzo.

Dinamičko povezivanje ima jednu prednost u odnosu na statičko povezivanje: nekoliko modula mogu deliti isti kod koji se nalazi u okviru jednog paketa. Prisetite se trenutka kada sam objašnjavao aplikaciju koja sadrži deset DLL datoteka za podršku. Korišćenjem dinamičkog povezivanja, aplikacija i DLL datoteke mogu deliti kod iz VCL paketa. Svaki modul će biti za najmanje 200KB umanjen, pošto se kompletan osnovni kod nalazi u DLL datoteci koja se koristi u toku rada programa. Ovo je prednost kada Vaši krajnji prizvodi sadrže nekoliko aplikacija, odnosno više DLL datoteka.

Dinamičko povezivanje ima i nekoliko problema. Prvi problem je da paketi i DLL datoteke koji su Vam potrebne za isporuku aplikacije mogu biti veoma veliki. Osnovni VCL paket, `VCL40.BPL` ima dužinu od 1,8MB. Vaša aplikacija će možda zahtevati druge pakete osim osnovnog VCL paketa. To znači da će Vaša aplikacija zahtevati minimalno 1,8MB DLL datoteka za svoj rad.

Drugi problem sa dinamičkim povezivanjem je suptilniji i problematičniji. Problem se može predstaviti u dve reči: promena verzije. Da bi objasnili problem, objasniću Vam mogući scenario. Recimo da kreirate aplikaciju korišćenjem Delphija 4.02 (podrazumevajući nekoliko revizija Delphija) i da koristite dinamičko povezivanje, što od Vas zahteva da isporučujete VCL pakete i RTL DLL datoteku. Vaš korisnik instalira Vašu aplikaciju na svojoj mašini i aplikacija radi savršeno.

U međuvremenu, ja sam napravio aplikaciju koristeći Delphi 4.0 (suviše mi je skupo da platim troškove isporuke nadogradnje) i takođe koristim dinamičko povezivanje. Vaš kupac kupuje moju aplikaciju i instalira je. Moj instalacioni program je ručni rad i ne ponaša se po pravilima, pa je obrisao pakete i DLL datoteke koje je Vaša aplikacija instalirala. Iznenada Vaša aplikacija neće više da radi, pošto su paketi koje sam isporučio kupcima stariji od Vaših paketa i nisu kompatibilni. Da li vidite problem?

U stvarnosti, firme koje rade komercijalni softver, kao što je to slučaj sa firmom Inprise, sprečavaju pojavu ovog problema dodeljujući svojim paketima i DLL datotekama različite nazive za svaku verziju proizvoda i dodavanjem informacije o verziji u pakete i DLL datoteke. (Dobar instalacioni program će proveriti broj verzije i instalirati pakete samo u slučaju da su noviji od paketa koji se nalaze u korisnikovom sistemu.) Ali paketi iz Borlanda ne predstavljaju pravi problem.

Pravi problem predstavljaju komponente drugih firmi koje nisu toliko pažljive u obavljanju svog posla. Ukoliko kupujete paket komponenti od firme Billy Bob Software, verujete da Billy Bob zna svoj posao kada je počeo da kreira pakete. To može, ali i ne mora biti dobra pretpostavka. Suočimo se sa tim, ekspanzijom Interneta, komponente su dostupne iz raznih izvora. U većini slučajeva ne znate šta dobijate, stoga budite pažljivi kada poručujete jeftine i besplatne komponente.

Pa, šta je bolje?

Mogu Vas čuti kako razmišljate: "Da li da koristim statičko povezivanje, ili dinamičko povezivanje?" Odgovor na ovo pitanje zavisi od tipa aplikacije koju pišete. U principu, ukoliko koristite jednu malu, odnosno aplikaciju srednje veličine, treba da koristite statičko povezivanje. Ukoliko pišete veoma velike aplikacije, odnosno aplikacije koje koriste veliki broj DLL datoteka, verovatno treba da koristite dinamičko povezivanje.

Jednostavna analiza primera će Vam pomoći da ovo sagledate. U lekciji dana 6, ste kreirali program `ScratchPad`. Ovaj program prevodi kod na približno 365KB (plus-minus par KB), kada koristi statičko povezivanje. Ukoliko povezujete program `ScratchPad` koristeći pakete, možete dobiti izvršnu datoteku veličine približno 21KB, *ali* treba da isporučite i 1,8MB paketa. Kao što možete videti dinamičko povezivanje u ovom slučaju nije dobro rešenje.

Korišćenje paketa koji se koriste u toku rada Vaših aplikacija

Ukoliko odaberete dinamičko povezivanje, potrebno je da promenite samo podešavanje opcija projekata. Šta Vam je za to potrebno:

1. Odaberite opciju Project➥Options u okviru glavnog menija kako bi se pojavio okvir za dijalog Project Options.
2. Kliknite na jezičak kartice Packages i odaberite opciju Build with runtime packages (Kreiraj sa paketima koji se koriste u toku rada programa), koja se nalazi blizu donje ivice kartice (možete ignorisati gornji deo kartice koji radi sa paketima koji se koriste u toku dizajniranja).
3. Kliknite na dugme OK kako biste zatvorili okvir za dijalog Project Options.
4. Ponovo prevedite i povežite projekat.

To bi bilo sve. Zapamtite da korišćenje dinamičkog povezivanja ne zahteva dodatne izmene Vašeg koda.

Isporučivanje aplikacija koje koriste pakete

Isporučivanje aplikacija koje koriste dinamičko povezivanje zahteva od Vas da znate koje pakete Vaša aplikacija koristi. Ukoliko ste pratili korake u prethodnom poglavlju, možete biti sigurni da Vam je kao minimum potrebna datoteka `VCL40.BPL`. Možda će Vam trebati drugi VCL paketi, što zavisi od komponenti koje Vaša aplikacija koristi.

Da biste ovo saznali, treba da pokrenete pomoćni program kao što je `TDUMP.EXE` i ispitate na koje se datoteke poziva Vaša aplikacija. Pomoćni program `TDUMP` možete pronaći u direktorijumu `➥Delphi 4➥Bin`. Da biste pokrenuli pomoćni program `TDUMP`, potrebno je da pređete u DOS prozor i pređete na direktorijum u kom se nalazi Vaša aplikacija. Zatim upišite sledeću komandnu liniju (predpostavljajući da se direktorijum `➥Delphi 4➥Bin` nalazi u Vašoj putanji, i da ne morate da upisujete putanju do programa `TDUMP`):

```
tdump myproject.exe
```

Budite spremni da pritisnete taster Pause, pošto će program `TDUMP` trenutno početi da ispisuje informacije. Negde između linija ćete pronaći ovaj deo:

```
Imports from Vcl40.bpl
    System::initialization() __fastcall
    System::Finalization() __fastcall
    System::RegisterModule(System::TLibModule*) __fastcall
```

Ovo se može ponoviti nekoliko puta. Treba da pazite na datoteke sa produžetkom `.BPL` i da beležite njihove nazive. Kada završite, imaćete listu paketa koje morate isporučiti sa Vašom aplikacijom.

NAPOMENA Ispis programa TDUMP se može preusmerti u tekst datoteku radi lakšeg pregleda. Na primer:

```
tdump myproject.exe > dump.txt
```

Sada možete otvoriti datoteku DUMP.TXT u Delphijevom editoru koda i pregledati sadržaj.

NAPOMENA Možete sebi uštedeti mnogo vremena i neprilika koristeći dobar instalacioni program. Program InstallShield Express se isporučuje sa Delphi verzijama Professional i Client/Server, pa tako već imate instalacioni program koji već možete da koristite. Ja, volim i instalacioni program Wise Install od firme Great Lakes Business Solutions. Dobar instalacioni program vodi računa o paketima koje Vaše aplikacije zahtevaju i automatski Vas uključuju u instalaciju. Ne preporučujem Vam pisanje zasebnih instalacionih programa ni pod kakvim okolnostima. Postoji suviše stavki koje mogu krenuti po zlu, a na koje morate obratiti pažnju kada pišete instalacioni program.

U većini slučajeva, verovatno nećete koristiti pakete koji se koriste u toku rada programa. U drugu ruku, ponekad su ovi paketi upravo ono što Vam treba.

Zaključak

Skladište objekata je odličan alat za ponovno korišćenje prethodno kreiranih formi, okvira za dijalog, projekata i drugih objekata. Mogućnost da dodate Vaše objekte u skladište predstavlja veliku dobit.

Čarobnjak za dijaloge i čarobnjak za aplikacije idu korak dalje i vode Vas kroz proces kreiranja. Čarobnjak za aplikacije je, u principu, veoma korisan alat. U sredini ove lekcije, naučili ste kako da dodate polja podataka i metode klasama koje je Delphi generisao.

Do kraja lekcije sam obradio različite tipove resursa za koje možete imati potrebu da ubacite u Vaše aplikacije i kako da ove resurse dodate u Delphi projekte. Na kraju ove lekcije sam obradio pakete. Paketi Vam daju fleksibilnost prilikom odlučivanja kako da isporučite Vaše aplikacije i isto tako čini instaliranje korisničkih komponenti jednostavnijim.

Radionica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstirte razumevanje obrađenog materijala, kao i vežbe koje Vam omogućavaju da steknete iskustvo u korišćenju gradiva koje ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A, "Odgovori na kviz pitanja".

Pitanja i odgovori

P **Kada da koristim opciju Use u okviru skladišta objekata?**

O Kada imate objekat postavljen u skladište objekata i želite da ga izmenite.

P **Da li postoji ograničenje u broju objekata koji mogu da se ubace u skladište objekata?**

O Tehnički gledano, u skladište objekata možete uneti koliko god želite objekata. Zapamtite da je svrha skladišta objekata pomoć za brzo pronalaženje i ponovno korišćenje Vaših formi, okvira za dijalog i drugih objekata. Ako postavite suviše objekata koje retko koristite u skladište objekata, počećete da gubite efiksnost, pošto će Vam trebati više vremena da pronađete objekat koji tražite. Takođe će skladištu objekata biti potrebno više vremena za očitavanje i prikazivanje svih objekata.

P **Imam veoma mnogo starih objekata u skladištu objekata i više ih ne koristim. Kako da ih se oslobodim?**

O Odaberite opciju Tools➥Repository u okviru glavnog menija. Pojaviće se okvir za dijalog za konfigurisanje skladišta objekata. Da biste uklonili objekat, prvo ga odaberite u okviru za listu Objects, a zatim kliknite na dugme Delete Object (brisanje objekta). Objekat će biti uklonjen iz skladišta objekata.

P **Imam objekat koji se nalazi u skladištiu objekata. Kada pokušam da koristim objekat, dobijem sledeću poruku: `Unable to find both a form and a source file`. (Nije moguće pronaći ni formu ni datoteku izvornog koda.) U čemu je problem?**

O Ili ste prebacili, ili obrisali izvorni kod programa, ili formu objekta. Skladište objekata zapisuje direktorijum u kom se objekat nalazi. Ukoliko premestite, ili obrišete objekat, skladište objekata ne može da pronađe objekat i prijavljuje grešku.

P **Da li mogu da dodajem objekte na karticu New u okviru skladišta objekata?**

O Ne. Kartica New skladišta objekata je nepromenljiva. Ne može se menjati, ili brisati. Vaše objekte možete postaviti na neku drugu stranu.

P **Dodao sam metodu u klasu glavne forme. Sada program neće da se prevede. U čemu je problem?**

O Verovatno ste greškom dodali dekleraciju metoda u odeljak za dekleraciju klasa kojim upravlja Delphi. Uverite se da je dekleracija Vašeg metoda, ili u `public`, ili u `private` odeljku dekleracije klasa (odnosno u odeljku `protected`, ukoliko postoji).

P **Imam resursni editor koji mi omogućava da dekompajliram resurse koji se nalaze u drugim programima. Ovo mi omogućava da "pozajmim" bitmape i druge resurse iz raznih programa. Da li to smem da radim?**

O Kratak odgovor je: "Ne". Treba da podrazumevate da su svi resursi u drugim programima materijal zaštićen autorskim pravima i da se ne može slobodno kopirati. Posavetujte se sa advokatom da biste dobili stručno mišljenje.

P **Imam veoma mnogo bitmapa i datoteka sa zvukom u okviru moje aplikacije. Da li sve ove resurse mogu postaviti u neku drugu datoteku izuzev izvršne datoteke?**

O Da, možete postaviti Vaše resurse u dinamički povezanu biblioteku (DLL).

Kviz

1. Kada koristite opciju Inherit prilikom odabiranja objekta u okviru skladišta objekata?
2. Koja je procedura za snimanje projekta u skladište objekata?
3. Šta se dešava sa nasleđenim formama kada promenite formu original?
4. Gde postavljate dekleracije korisničkih metoda u okviru dekleracije klasa forme?
5. Gde postavljate definiciju metoda (samu metodu) kada dodajete sopstvene metode u Delphi kod?
6. Kako možete odrediti ko je napisao određeni objekat koji se nalazi u skladištu objekata?
7. Gde dodajete i brišete kartice skladišta objekata?
8. Da li je lakše kreirati osnovnu aplikaciju od početka, ili korišćenjem čarobnjaka aplikacija?
9. Šta je bolje za male aplikacije: statičko povezivanje, ili dinamičko povezivanje korišćenjem paketa?
10. Da li možete da kreirate resursnu skript datoteku koja koristi tabelu stringova koristeći editor teksta?

Vežbe

1. Kreirajte novu formu. Dodajte nekoliko komponenti po Vašem izboru na formu. Snimite formu na karticu Forms u okviru skladišta objekata pod nazivom `BaseForm`.
2. Pokrenite novu aplikaciju. Odaberite opciju File➥New u okviru skladišta objekata. Pređite na karticu Forms. Kliknite na radio dugme Inherit. Odaberite objekat `BaseForm` koji ste kreirali u vežbi jedan i dodajte ga u aplikaciju. (Budite sigurni da ste koristili opciju Inherit.) Snimite projekat i zatvorite ga.

3. Otvorite objekat `BaseForm` koji ste kreirali u vežbi jedan. Obrišite sve komponente na formi i snimite formu.
4. Ponovo otvorite projekat koji ste kreirali u vežbi dva. Prikažite novu formu koju ste kreirali u ovoj vežbi. Uočite da su sve komponente nestale. (Zapamtite da ste nasledili ovaj objekat, pa su sve izmene koje ste načinili u osnovnoj formi primenjene i na nasleđenu formu.)
5. Odaberite opciju Tools➥Repository u okviru glavnog menija. Obrišite objekat `BaseForm` koji ste ranije kreirali.
6. Kreirajte projekat koristeći čarobnjak aplikacija. Ubacite sve opcije menija i definišite aplikaciju kao MDI.
7. Dodajte okvir za dijalog sa više kartica aplikaciji koju ste kreirali u vežbi šest. Koristite čarobnjak za dijaloge.
8. Koristite skladište objekata da biste dodali okvir za dijalog About u okviru programa koji ste kreirali u vežbi šest.
9. Kreirajte jednostavni program i prevedite ga i povežite. Pokrenite Windows Explorer da biste otkrili dužinu izvršne datoteke koju je Delphi kreirao. Sada promenite opcije projekta, tako da koristite pakete koji se koriste u toku rada programa. Ponovo napravite izvršnu datoteku. Proverite dužinu izvršne datoteke nakon ovog koraka. Koja je razlika u veličini datoteka?
10. Napišite sto puta na tabli: "Neću pozajmljivati bitmape iz drugih programa".

Dan 9

Projekti, editor koda (Code Editor) i prozor za ispitivanje koda (Code Explorer)

U današnjoj lekciji ćete naučiti više o Delphi-jevom okruženju i kako sve zajedno radi prilikom kreiranja programa za svakodnevnu upotrebu. Posebno ćete učiti o:

- Projektima i menadžeru projekta (Project Manager).
- Editoru koda (Code Editor).
- Prozoru za ispitivanje koda (Code Explorer).

Ovo će biti dug, ali i blagorodan dan.

Svakome treba projekt

U lekciji dana 4, "Istraživanje Delphi-jevog okruženja", predstavljeni su Vam Delphi projekti i otkrili ste nešto o načinu na koji projekti funkcionišu. U današnjoj lekciji ću detaljnije obraditi projekte. Projekti su smisao života u Delphi-ju. Ne možete kreirati program bez projekta. Projekti obezbeđuju da sve radi zajedno i omogućavaju kreiranje aplikacija koje rade. Sledeće poglavlje će Vam objasniti:

- Menadžer projekta (Project Manager).
- Projektne grupe (Project groups).
- Okvir za dijalog opcije projekta (Project Options).

Korišćenje menadžera projekta

U jednom trenutku, svakom projektu je potrebno administriranje. Možda morate dodati novi junit u projekt, odnosno morate ukloniti junit iz projekta. Možda je potrebno da dodate još neki tip datoteke projekta, kao što je binarna resursna datoteka. Junite, kao i ostale datoteke projekta možete dodavati, ili uklanjati korišćenjem menadžera projekta (Project Manager).

Projektne grupe

U lekciji dana 4, napisao sam da je projekat zbirka datoteka koje rade zajedno na kreiranju zasebnih izvršnih datoteka, odnosno DLL datoteka. Ovo je definicija projekta, kada se koristi Delphi-jevo okruženje. U stvarnom okruženju možete imati različite tipove projekata, kao što je, na primer, posao koji treba da završite.

Veliki projekti mogu uključiti jednu, ili više izvršnih i jednu, ili više DLL datoteka. Pošto se neki projekti sastoje od nekoliko izvršnih programa, Delphi Vam omogućava da grupišete nekoliko Delphi projekata i radite sa njima kao sa jednom celinom. Ova celina se naziva projektna grupa (*Project group*).

Zašto treba koristiti projektne grupe?

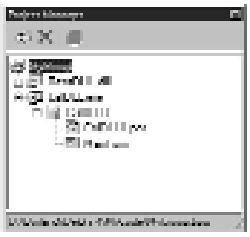
Možda se pitate koju prednost imaju projektne grupe. Projektne grupe Vam pružaju sledeće mogućnosti:

- Bolju kontrolu kompletnog softverskog projekta.
- Mogućnost da radite sa DLL datotekama i testirate izvršne datoteke koristeći istovremeno i izvršne i DLL datoteke.
- Mogućnost da napravite (prevedete i povežete) grupu projekata u isto vreme.
- Mogućnost da imate otvoreno nekoliko projekata istovremeno i jednostavno prelazite sa jednog projekta na drugi.
- Način da organizujete srodne projekte.

Projekt koji kreira samo jednu izvršnu datoteku, nema potrebu da bude u projektnoj grupi. Jedan projekt se teško može smatrati grupom, zar ne? U slučaju jednog jedinog projekta, koncept projektne grupe se ne može primeniti.

Ali, zamislite trenutak kada program sadrži jednu izvršnu datoteku (EXE) i jednu DLL datoteku za podršku. Obe datoteke se zajedno koriste (i EXE i DLL datoteka). Obično kada radite sa DLL datotekama želite da izvršna datoteka bude dostupna, kako biste trenutno mogli da testirate izmene koje ste napravili sa DLL datotekom. U ovom scenariju, projektna grupa ima smisla, pošto EXE i DLL datoteke svuda idu zajedno.

Kreirajte projektnu grupu koja sadrži ova dva zasebna projekta i snimite je. Kada želite da radite bilo na aplikaciji, ili DLL datoteci, možete otvoriti projektnu grupu umesto zasebnog projekta. Kada otvorite projektnu grupu, biće prikazani i EXE projekat i DLL projekat. Možete raditi, ili na DLL projektu, ili na EXE projektu u okviru editora koda i prebacivati se sa jednog na drugi projekt, kad poželite. Slika 9.1 prikazuje prozor menadžera projekta sa otvorenom projektnom grupom ovog tipa.



Slika 9.1
Prozor menadžera projekta pokazuje projektnu grupu

Još jedan razlog za kreiranje projektnih grupa je grupisanje srodnih projekata. Možda Vam sve ovo zvuči kao nešto što nema mnogo smisla, pa ću Vam razjasniti. U firmi TurboPower Software imamo proizvod pod nazivom Async Professional, koji predstavlja zbirku komponenti za serijsku komunikaciju. Ove komponente su raspoređene u tri glavne kategorije: osnovna serijska komunikacija, komunikacija korišćenjem telefaksa i TAPI. Za pokrivanje sve tri kategorije program Async Professional sadrži desetine programa primera.

Na osnovu ovog scenarija, mogli smo kreirati projektnu grupu svih naših primera vezanih za komunikaciju korišćenjem telefaksa, još jednu grupu za sve naše TAPI primere i treću grupu za sve naše primere vezane za osnovnu serijsku komunikaciju. Naši korisnici bi mogli da otvore projektnu grupu TAPI Primeri, kako bi imali sve TAPI primere u jednom paketu. Kompletna projektna grupa bi mogla da bude napravljena odjednom, time bi uštedeli vreme i trud otvaranja i kreiranja svakog zasebnog projekta. U ovom slučaju projekti ne rade zajedno, kao što je to slučaj sa DLL i EXE datotekama, ali su svi projekti međusobno povezani, pa koncept projektne grupe ima većeg smisla.

Aktivan projekt

U svakoj projektnoj grupi uvek postoji aktivni projekat. Aktivni projekat je u menadžeru projekta prikazan podebljanim (bold) slovima. Na slici 9.1, aktivni projekt je `TestDLL`. Aktivni projekat je onaj projekat koji će se napraviti (build) ukoliko odaberete opciju Make, ili Build u okviru menija Project, glavnog menija Delphi-ja. Ove meni opcije se menjaju svaki put kada se promeni aktivni projekat. Na primer, ukoliko je aktivan projekat pod nazivom `Project1`, opcije menija će nositi naziv `Make Project1` i `Build Project1`. Ukoliko je aktivan projekat `PictView`, ove dve opcije menija će nositi naziv `Make PictView` i `Build PictView`.

Aktivni projekat takođe ima značaja kada se dodaje novi junit, ili nova forma korišćenjem menadžera projekta. Kada kreirate novu formu koristeći menadžer projekta, ona će biti dodata aktivnom projektu bez obzira koji je nod menadžera projekta trenutno odabran. Aktivni projekat je onaj projekat u koji se nove forme, ili juniti dodaju ukoliko dodajete nove elemente korišćenjem glavnog menija Delphi-ja, odnosno korišćenjem Delphi-jeve trake sa alatima.

Projekat možete definisati kao aktivan na nekoliko različitih načina. Jedan od načina je da odaberete bilo koju opciju u okviru noda projekta koji želite da definišete kao aktivni projekt, a zatim kliknete mišem na dugme Activate Selected Project (aktiviranje odabranog projekta) u gornjem delu menadžera projekta. Drugi način je dvostruki klik mišem na nod samog projekta. Na kraju, možete odabrati opciju Activate iz menija sadržaja noda projekta da biste aktivirali odgovarajući projekat.

Prozor menadžera projekta

4 Menadžer projekta u okviru Delphi-ja 4 je nova opcija. Koncept menadžera projekta nije potpuna novost Delphi-ja 4, ali je trenutna implementacija mnogo bolja od prethodnog menadžera projekta. Menadžer projekta je glavni kontrolor za sve Vaše projekte i grupe programa. On Vam omogućava da dodajete datoteke u projekat, brišete datoteke iz projekta, pregledate junite, ili forme, dodajete projekte u grupu, menjate redosled projekata i još mnogo toga. Da biste prikazali menadžer projekta, odaberite opciju View➥Project Manager u okviru glavnog menija, odnosno pritisnite tastere Ctrl+Alt+F11. Prozor menadžera projekta sadrži kontrolu koja liči na stablo i prikazuje četiri nivoa. Pogledajmo nivoe:

- Projektna grupa.
- Projekti u okviru projektne grupe.
- Forme i ostale datoteke u okviru projekta.
- Pojedinačne forme i juniti u okviru noda forme.

Prirodno, svi zasebni nodovi mogu biti zatvoreni i raširenи, kao i svaka druga kontrola sa izgledom stabla. Nod menadžera projekta sadrži ikone koje ukazuju da li nod sadrži projekat, zasebnu datoteku, formu, odnosno par forma/junit. Pogledajte sliku 9.1 da biste videli različite ikone i nivoe koje menadžer projekta prikazuje.

NAPOMENA U prethodnim verzijama Delphi-ja, menadžer projekta je prikazivao putanju do bilo kog junita zajedno sa nazivom junita. Menadžer projekta Delphi-ja 4 ne prikazuje putanju i nazive datoteka na isti način. Da biste videli putanju i naziv datoteke, kliknite na junit koji se nalazi na prozoru menadžera projekta i statusna traka menadžera projekta će Vam pokazati kompletnu putanju i naziv junita (proverite na slici 9.1).

Meni sadržaja menadžera projekta

Većinu posla menadžera projekta možete obavljatri koristeći meni sadržaja. Postoje četiri odvojena menija sadržaja menadžera projekta. Naredna poglavlja opisuju svaki od ovih menija sadržaja.

Meni sadržaja projektne grupe

Meni sadržaja projektne grupe se može videti nakon klika desnog tastera miša na nod projektne grupe u vrhu stabla menadžera projekta. Tabela 9.1 prikazuje meni sadržaja projektne grupe zajedno sa opcijama koje se pojavljuju u okviru menija.

Tabela 9.1: Opcije menija sadržaja projektne grupe

Opcija	Opis
Add New Project	Otvara skladište objekata tako da možete odabrati novi element. Element može biti: aplikacija, DLL datoteka, forma, modul podataka, komponenta, odnosno bilo koji drugi objekt koji se nalazi u skladištu objekata.
Add Existing Project	Otvara projektnu datoteku sa diska i daje je u projektnu grupu.
Save Project Group	Snima projektnu grupu. Projektna grupa ima produžetak .bpg.
Save Project Group As	Snima projektnu grupu pod novim nazivom.
View Project Group Source	Prikazuje izvorni kod projektne grupe. Izvorni kod projektne grupe je posebna datoteka (datoteka za pravljenje projekta - makefile) koja sadrži reference na sve projekte u okviru projektne grupe.
Toolbar	Uključuje, odnosno isključuje traku sa alatima menadžera projekta.
Status Bar	Uključuje i isključuje statusnu traku menadžera projekta.
Dockable	Definiše da li se mogu usidriti drugi prozori na menadžer projekta.

Opcije menija Toolbar, Status Bar i Dockable se pojavljuju na svakom meniju sadržaja menadžera projekta. Više ih neću pominjati kada budem obrađivao druge menije sadržaja menadžera projekta.

Meni sadržaja projekta Meni sadržaja projekta se prikazuje kada kliknete desnim tasterom miša na nod projekta u okviru menadžera projekta. Tabela 9.2 prikazuje opcije menija sadržaja koje su specifične za meni sadržaja projekta.

Tabela 9.2: Opcije menija sadržaja projekta

Opcija	Opis
Add	Otvara okvir za dijalog Add to Project, tako da možete dodavati datoteke u projekat. Isto kao izbor opcije Project➥Add to Project u okviru glavnog menija, odnosno iz trake sa alatima u okviru Delphi-ja.
Remove File	Otvara okvir za dijalog Remove From Project, tako da možete ukloniti datoteke iz projekta. Isto kao izbor opcije Project➥Remove from Project u okviru glavnog menija, odnosno iz Delphi-jeve trake sa alatima.
Save	Snima projekte. Isto kao izbor opcije File➥Save u okviru glavnog meni ja Delphi-ja.
Options	Prikazuje okvir za dijalog opcije projekta (Project Options) za odabrani projekt. Isto kao izbor opcije Project➥Options u okviru glavnog menija
Delph-ija.	
Activate	Aktivira označeni projekat.
Compile	Prevodi odabrani projekt. Razlika između prevođenja i pravljenja pro jekta je opisana u lekciji dana 4.
Build	Pravi projekat.
View Source Source	Prikazuje izvorni kod projekta. Isto kao izbor opcije Project➥View u okviru Delphi-jevog glavnog menija.
Close	Zatvara projekt i sve njegove datoteke. Ukoliko je projekat deo sniml jene projektne grupe, ikona noda projekta će postati siva. Projekat je još uvek deo grupe, ali nije otvoren u okruženju. Ukoliko je projekat deo generičke projektne grupe, zatvara se i uklanja iz generičke grupe.
Remove Project	Uklanja projekat iz projektne grupe. Projekat nije obrisan sa hard diska, samo je uklonjen iz projektne grupe (isto kao klik miša na dugme Remove Selected Project u okviru trake sa alatima menadžera projekta).
Build Sooner	Pomera projekat na gore u okviru stabla projekta. Projekti se prave od vrha prema dnu u prozoru menadžera projekta.
Build Later	Pomera projekat na dole u okviru stabla projekta.

Meni sadržaja menadžera projekta je opširniji nego što je to navedeno u ovom poglavlju. Ukoliko je projekat beč datoteka, odnosno projekat sa paketima, meni sadržaja projekta će sadržati dodatne opcije. Razlike nisu značajne, pa neću objašnjavati dodatne opcije menija.

Meni sadržaja junita Meni sadržaja junita će biti prikazan kada kliknete desnim tasterom miša na nod junita u okviru sadržaja junita. Tabela 9.3 prikazuje opcije menija sadržaja junita.

Tabela 9.3: Opcije menija sadržaja junita

Opcija	Opis
Open	Prikazuje junit u okviru editora koda (važi za samostalne junite), odnos no dizajneru forme (ukoliko junit ima dodeljenu formu).
Remove From Project	Uklanja junit iz projekta. Menadžer projekta Vas ne pita da li želite da uklonite junit i ne postoji opcija za poništavanje akcije. Ukoliko greškom uklonite junit iz projekta, možete ga vratiti nazad.
Save	Snima junit. Isto kao izbor opcije File➥Save u okviru Delphi-jevog glavnog menija.
Save As	Otvara okvir za dijalog Save As, tako da možete da snimite junit sa novim nazivom. Potpuno isto kao izbor opcije File➥Save As i okviru Delphi-jevog glavnog menija.

Meni sadržaja datoteke Kada kliknete desnim tasterom miša na nod koji ne pripada projektnoj grupi, projektu, odnosno junitu (obično `.pas`, ili `.dfm` datoteka), biće prikazan meni sadržaja datoteke. Ovaj meni sadržaja ima samo jednu opciju. Opcija menija Open prikazuje odabrani nod, u editoru koda, ili u dizajneru forme u zavisnosti od tipa odabranog noda.

Traka sa alatima menadžera projekta i komande sa tastature

Kao dodatak meniju sadržaja menadžera projekta, radi lakšeg rada, menadžer projekta ima i traku sa alatima. Traka sa alatima menadžera projekta sadrži tri dugmeta:

- Dugme Add New Project prikazuje skladište objekata tako da možete dodati nov projekat u projektnu grupu. Isto kao klik mišem na opciju Add New Project u okviru menija sadržaja projekta.
- Dugme Remove Selected Project uklanja odabrani projekt iz projektne grupe. Ovo dugme možete koristiti samo za uklanjanje komplektnog projekta, a ne samo određene forme, ili datoteke u okviru projekta.
- Dugme Activate Selected Project aktivira odabrani projekt.

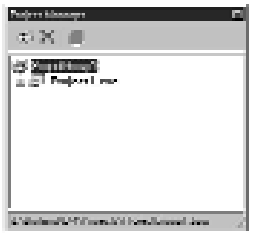
Komande sa tastature uključuju tastere Delete i Insert. Kada pritisnete taster Delete, uklanja se odabrani nod. Ukoliko je odabran nod projekta, odabrani projekt će biti uklonjen iz projektne grupe. Ukoliko je odabran nod junita, odabrani junit će biti uklonjen iz projekta u okviru kog se nalazi. Taster Insert se ponaša isto kao izbor opcije Add to Project u okviru menija sadržaja projekta.

Dugmad trake sa alatima menadžera projekta mogu biti velika, ili mala. Generički, dugmad trake sa alatima su mala. Možete menjati veličinu dugmadi trake sa alatima povlačeći donju ivicu trake sa alatima na gore (da bi smanjili dugmad), odnosno na dole (da bi povećali dugmad).

Kreiranje i korišćenje projektnih grupa

Projektne grupe su veliko olakšanje za kompleksne projekte, mada korišćenje projektnih grupa nije obavezno. Za svaki projekat ne morate koristiti projektne grupe. Menadžer projekta sadrži generičku projektnu grupu pod nazivom `ProjectGroup1`, koja se koristi kada ne otvorite, odnosno ne kreirate projektnu grupu. Pokušajte sledeće:

1. Odaberite opciju File➥Close All kako biste zatvorili sve otvorene projekte ili projektne grupe.
2. Odaberite opciju File➥New Application da biste kreirali novu aplikaciju.
3. Odaberite opciju View➥Project Manager da biste prikazali menadžer projekta. Menadžer projekta koji će biti prikazan nalazi se na slici 9.2.



Slika 9.2 Menadžer projekta prikazuje generičku projektnu grupu

Projektna grupa pod nazivom `ProjectGroup1` je privremena projektna grupa. Kada odaberete opciju Save All u okviru menija File bićete upitani da li želite da snimite projekat, ali nećete biti upitani da li želite da snimite projektnu grupu. Ukoliko želite da snimite projektnu grupu, morate je eksplicitno snimiti korišćenjem opcije Save Project Group, odnosno Save Project Group As u okviru menija sadržaja menadžera projekta.

Dodavanje junita

Dodavanje postojećih junita u Vaš projekt je jednostavno i postiže se klikom miša na dugme Add To Project u okviru trake sa alatima menadžera projekta, odnosno izborom opcije Add To Project u okviru menija sadržaja menadžera projekta.

U projekat ne možete dodati junit, ukoliko se u okviru projekta nalazi forma koja ima isti naziv. Na primer, ukoliko imate formu sa nazivom `MainForm` i pokušate da dodate junit iz drugog projekta koji takođe ima formu sa nazivom `MainForm`, Delphi će prijaviti grešku, čak iako su nazivi datoteka različiti.

Uklanjanje junita

Opciju Remove From Project možete koristiti za uklanjanje datoteka iz projekta. Kao alternativu, možete koristiti taster Delete na tastaturi, kojim se takođe uklanja junit iz projekta, ali prethodno morate odabrati junit koji želite da uklonite. Datoteke uklonjene iz projekta nisu obrisane sa Vašeg hard diska, nego su samo uklonjene iz procesa prevođenja i povezivanja projekta.

Budite oprezni kada uklanjate junite iz Vaših projekata. Morate voditi računa da ne uklonite junite na koje se pozivaju drugi juniti iz istog projekta. Ukoliko uklonite junite koji su potrebni Vašem projektu, kao rezultat ćete dobiti grešku prevodioca. Pre nego što uklonite junit, uverite se da se odabrani junit ne koristi u Vašem projektu. Ukoliko greškom uklonite junit koji je potreban Vašem projektu, možete ga ponovo vratiti koristeći opciju Add To Project, koja je objašnjena u prethodnom poglavlju.

Okvir za dijalog Remove From Project omogućava izbor više elemenata, tako da istovremeno možete ukloniti nekoliko junita iz projekta, ukoliko to želite.

Gledanje junita i formi

Da biste videli junit, formu, ili neku drugu datoteku, treba da kliknete na nod koji predstavlja formu, ili junit koji želite da vidite. Takođe možete odabrati opciju Open u okviru menija sadržaja menadžera projekta. Forma, ili junit će biti prikazani, ili u dizajneru forme, ili u editoru koda u zavisnosti od tipa noda koji ste odabrali.

Pravljenje projekata, ili projektnih grupa

Da biste napravili određeni projekt, to možete učiniti na jedan od navedenih načina:

- Desnim klikom miša na nod projekta u okviru menadžera projekta pozovite meni sadržaja i odaberite opciju Build.
- Odaberite opciju Project➥Build <*naziv projekta*> u okviru Delphi-jevog glavnog menija. Naziv ove opcije menija će se menjati u zavisnosti od naziva aktivnog projekta.
- Pritisnite tastere Ctrl+F9 na tastaturi kako biste preveli trenutno aktivni projekt.

Da biste napravili kompletnu projektnu grupu, odaberite opciju Project➥Build All Projects u okviru Delphi-jevog glavnog menija. Svi projekti u projektnoj grupi će biti napravljeni počev od prvog projekta u grupi (projekt na vrhu stabla menadžera projekta), pa do poslednjeg projekta u okviru grupe, prolazeći stablom na dole.

Razumevanje opcija projekta

Opcije projekta su mogućnosti koje se lako mogu ignorisati. Za nekoga su generičke opcije dovoljno dobre kada počinje da radi. Na kraju krajeva, ko ima vremena da brine o svim ovim opcijama prevodioca/programa za povezivanje kada se još uvek borite da naučite novine programskog okruženja? Ipak ćete od jednog trenutka početi da se interesujete šta ove opcije rade, pa je dobro da u međuvremenu dobijete neke informacije.

Ovo poglavlje obrađuje okvir za dijalog opcije projekta (Project Options). Okvir za dijalog možete pozvati biranjem opcije Project➥Options u okviru glavnog menija. Okvir za dijalog opcije projekta sadrži nekoliko kartica:

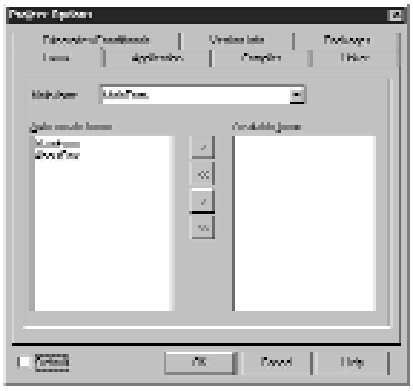
- Forms (forme)
- Application (aplikacija)
- Compiler (prevodilac)
- Linker (program za povezivanje)
- Directories/Conditionals (direktorijumi/uslovi)
- Version Info (informacije o verziji)
- Packages (paketi)

Neću posebno obrađivati svaku karticu okvira za dijalog opcije projekta. Obradiću samo najvažnije kartice, kako biste mogli da shvatite čemu svaka kartica služi. Počeću od najjednostavnijih, a to su kartice za forme i aplikaciju. Nakon toga ću preći na komplikovanije kartice.

U donjem delu svake kartice okvira za dijalog opcije projekta postoji polje za potvrdu pod nazivom Default. Ukoliko želite da tekuća podešavanja opcija postanu generička za sve nove projekte, označite polje za potvrdu Default. Kada kliknete dugme OK, trenutno odabrane opcije će postati generičke.

Kartica Forms (forme)

Kartica Forms okvira za dijalog opcije projekta, je mesto gde ćete moći da kontrolišete forme u Vašoj aplikaciji. Ovaj okvir za dijalog ste videli u lekciji dana 4, kada ste kreirali program PictureViewer. Slika 9.3 prikazuje karticu Forms okvira za dijalog opcije projekta za program `ScratchPad`.



Slika 9.3
Kartica Forms okvira za dijalog opcije projekta

U gornjem delu kartice za forme je kombo okvir za glavnu formu. Ovo je mesto gde saopštavate Delphiju koju formu treba da prikaže prilikom startovanja aplikacije. Generički će prva forma koju kreirate postati glavna forma. Ukoliko menjate Vaš projekt tako da različite forme postaju glavne, morate promeniti ovu opciju tako da nova forma postane glavna forma aplikacije. U sredini okvira za dijalog se nalaze dva okvira za listu. Okvir za listu na levoj strani je označen kao Auto-create forms (forme koje se automatski kreiraju); na desnoj strani je označen kao Available forms (dostupne forme). Pre nego što Vam objasnim kako da koristite ova dva okvira za listu, obradićemo automatsko kreiranje forme.

NOVI TERMIN *Automatsko kreiranje* znači da će Delphi konstruisati formu u toku startovanja aplikacije.

Svaki put kada kreirate formu, Delphi je postavlja na listu za automatsko kreiranje formi u okviru aplikacije. Forme koje se automatski kreiraju se prikazuju puno brže nego forme koje nisu automatski kreirane. Mana kod automatskog kreiranja formi je što Vaša aplikacija koristi više memorije nego što bi koristila kada forma nije automatski kreirana. Još jedna mana, iako možda nevažna, je da se Vaša aplikacija duže učitava ukoliko imate mnogo formi koje se automatski kreiraju.

NAPOMENA Prva forma na listi formi koje se automatski kreiraju je uvek glavna forma. Ukoliko promenite glavnu formu, nova forma koju odaberete će se pomeriti na vrh okvira za listu formi koje se automatski kreiraju. Još jedan način za podešavanje glavne forme je prevlačenje na vrh liste formi koje se automatski kreiraju.

Dobra strana automatskog kreiranja formi je što se automatski kreirana forma jednostavno prikazuje. Sve što treba da uradite je da pozovete funkciju `Show`, ili `ShowModal` za željenu formu:

```
AboutBox.ShowModal;
```

Ukoliko Vaše forme nisu automatski kreirane u Delphi-ju morate sami preuzeti odgovornost za njihovo kreiranje pre nego što ih budete koristili:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  About : TAboutBox;
begin
  About := TAboutBox.Create(Self);
  About.ShowModal;
  About.Free;
end;
```

Ovaj primer ne koristi pointer koji je Delphi generisao za okvir About. On kreira lokalni pointer, prikazuje formu, a zatim briše pointer, ubrzo pošto nestane potreba za formom. Kao što je to slučaj kod programiranja na jeziku Object Pascal, postoji nekoliko načina da bi se izvršio određeni zadatak. Pošto Delphi uvek kreira pointer na objekt forme, prethodni kod bi trebalo da napišete na ovaj način:

```
if not Assigned(AboutBox) then
  AboutBox := TAboutBox.Create(Self);
AboutBox.ShowModal;
```

Ovaj kod proverava da li je forma već kreirana. Ukoliko nije kreirana, objekt se kreira, a zatim se poziva metoda `ShowModal`. Odluka koji metod kreiranja forme ćete koristiti je Vaša, mada lično više volim pređašnji način, pošto se time procesom kreiranja forme upravlja lokalno.

Svaki put kada kreirate formu u okviru dizajnera forme, Delphi kreira pointer na formu. Ukoliko omogućite Delphiju da automatski kreira formu, ne morate da brinete da li će pointer biti važeći. Ukoliko ne želite da forma bude automatski kreirana, pointer na formu će imati vrednost `nil` (bez vrednosti), sve dok eksplicitno ne kreirate formu i inicijalizujete pointer. Ukoliko zaboravite i pokušate da koristite pointer pre nego što bude inicijalizovan, Windows će generisati grešku za pogrešan pristup.

Sada ćemo se vratiti nazad na okvir za dijalog opcije projekta. Okvir za listu koji sadrži automatski kreirane forme, prikazuje samo forme koje će se prilikom startovanja programa automatski kreirati. Ukoliko ne želite da se forma automatski kreira, prevucite je sa liste formi za automatsko kreiranje na listu dostupnih formi. Da biste prebacili nekoliko formi istovremeno, potrebno je da odaberete forme koje želite da prebacite (oba okvira za listu podržavaju višestruki izbor) i prevucite ih sve odjednom. Veoma jednostavno, zar ne?

Možete koristiti dugmad između dva okvira za listu da biste prebacili forme sa jedne liste na drugu, ali je obično lakše da koristite prevlačenje i puštanje.

Kartica Application (aplikacija)

Kartica aplikacije okvira za dijalog opcije projekta je veoma jednostavna (videti sliku 9.4).

Slika 9.4
Kartica aplikacije

Polje za naslov u okviru kartice se koristi za dodeljivanje naziva aplikaciji. Naslov je tekst koji će se pojaviti na traci otvorenih programa Windows operativnog sistema, kada minimizirate Vašu aplikaciju.

Naslov aplikacije i naslov glavne forme su dve odvojene stavke. Ukoliko želite da se pojavi naziv Vašeg programa kada je program minimiziran, morate definisati naslov aplikacije okvira za dijalog opcije projekta. Ukoliko ne obezbedite naziv aplikacije, koristiće se generički naziv projektne datoteke.

Polje Help file na kartici za aplikaciju koristite da bi definisali datoteku za pomoć koju će Vaša aplikacija koristiti. Ovo je datoteka za pomoć koju će Vaš program učitavati kada pritisnete taster F1 u toku rada aplikacije. Možete koristiti dugme Browse, kako biste pronašli datoteku za pomoć, ukoliko ne možete da se setite naziva i direktorijuma u kom se nalazi Vaša datoteka. Ukoliko nemate datoteku za pomoć, pritiskom na taster F1 u okviru Vaše aplikacije ništa se neće dogoditi.

Opcija Icon Vam omogućava da odaberete ikonu za Vašu aplikaciju. Ovo je ikona koja će biti prikazana u traci za aktivne programe Windows-a u toku rada Vaše aplikacije, odnosno kada je Vaša aplikacija minimizirana. Dodatno, ova ikona će biti prikazana na naslovnoj traci Vaše glavne forme ukoliko eksplicitno ne definišete ikonu za glavnu formu. Da biste odabrali ikonu, kliknite na dugme Load Icon (učitavanje ikone) i pronađite datoteku ikone (`.ico`), koristeći okvir za dijalog Application Icon.

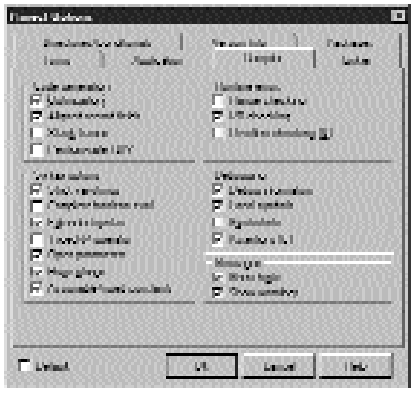
PoljeTarget file extension se koristi za definisanje nastavka naziva datoteke projekta, nakon kreiranja projekta. Na primer, ukoliko kreirate čuvar ekrana (screen saver), ovo polje bi trebali da promenite u `scr`, tako da Vaš čuvar ekrana bude kreiran sa produžetkom `.scr` umesto `.exe`. Apleti kontrolnog panoa (Control Panel) su još jedan primer. Ovo su posebni programi sa produžetkom `.cpl`. Produžetak

naziva datoteke se automatski postavlja na `.exe` kod izvršnih datoteka (aplikacije konzole i GUI aplikacije), odnosno na `.dll` za DLL datoteke, tako da kod normalnih projekata ne morate da upisujete vrednost u ovo polje.

Kartica Compiler (prevodilac)

Kartica za prevodilac okvira za dijalog opcije projekta je mesto gde podešavate opcije koje prevodilac koristi za pravljenje Vašeg projekta. Slika 9.5 prikazuje ovu stranu okvira za dijalog opcije projekta.

Katica prevodioca je podeljena na pet delova. Obradiću svaki od ovih delova i ispitati ih tako da možete da bolje razumete različite opcije na ovoj kartici.



Slika 9.5
Kartica prevodioca okvira za dijalog opcije projekta

Generisanje koda (Code generation)

Prevodilac može biti konfigurisan tako da optimizuje Vaš kod. Ukoliko je optimizacija isključena (polje za potvrdu Optimization nije potvrđeno), prevodilac ne pokušava da optimizuje Vaš kod. Ukoliko ovu opciju uključite, prevodilac će generisati najbrži mogući kod bez obzira na dužinu. U većini slučajeva ovu opciju treba da postavite na generičku vrednost. Ipak, ponekad je bolje da isključite optimizaciju kada debagirate Vašu aplikaciju. Optimizacija i debagiranje će biti detaljnije obrađeni u lekciji dana 10, "Debagiranje Vaših aplikacija".

Opcija Aligned Record Fields se koristi za kontrolisanje načina na koji su slogovi poravnati u memoriji. Kada je ova opcija uključena slogovi su poravnati na četvorobajtne delove. Kada je ova opcija isključena, slogovi su poravnati na jedan bajt.

Možda ćete poželeti da uključite opciju Stack Frames prilikom debagiranja. Kada završite debagiranje, možete isključiti ovu opciju kako bi prevodilac generisao manji i brži kod, ali vreme prevođenja će biti nešto duže ukoliko je opcija Stack Frames isključena.

Opcija Pentium-Safe FDIV omogućava da prevodilac generiše kod koji otkriva pogrešnu instrukciju za deljenje u pokretnom zarezu.

Opcije sintakse i greške u toku rada programa

Ova dva dela utiču na način generisanja koda za projekat. Delphi-jev sistem za pomoć povezan sa karticom Compiler objašnjava čemu služe sve ove opcije, pa ih neću dodatno objašnjavati. Da bi prikazali stranu za pomoć koja sadrži opcije prevodioca, kliknite na dugme Help u trenutku kada je prikazana kartica prevodioca, a zatim pritisnite taster F1 na tastaturi.

Debagiranje

Deo Debugging kartice za prevodilac koja se nalazi u okviru za dijalog opcije projekta, kontroliše kako prevodilac generiše kod za proces debagiranja. Kada je opcija za debagiranje aktivirana, Delphi će generisati informacije za debagiranje projekta. Ukoliko ne generišete informacije za debagiranje, nećete biti u mogućnosti da se zaustavite na tačkama prekida (breakpoints) i proverite promenljive u toku debagiranja. Objasniću to na drugi način; Vaš program ne možete debagirati, ukoliko ne saopštite Delphi-ju da generiše informacije za debagiranje.

Ukoliko promenite bilo koju opciju kartice prevodioca, treba ponovo da napravite aplikaciju (koristeći opciju Build) kako bi se promene aktivirale. Ovo osigurava da svi juniti budu prevedeni i povezani korišćenjem istih opcija prevodioca.

Poruke

Deo za poruke (messages) određuje da li želite da prevodilac izvesti o savetima i upozorenjima nakon prevođenja. Ja uvek ostavljam uključene opcije Show Hints i Show Warnings. Saveti i upozorenja ne bi trebali biti ignorisani na duge staze (na kratke staze možete ignorisati upozorenja pošto znate da je to posledica privremenog stanja Vašeg koda). Obično, upozorenja prevodioca trebaju biti poštovana. Naučite da tretirate savete i upozorenja kao greške. Kvalitetan kod se prevodi bez upozorenja.

Kartica Linker (program za povezivanje)

Kartica programa za povezivanje okvira za dijalog opcije projekta je mesto gde ćete podesiti opcije koje definišu način rada Vašeg programa za povezivanje. Sve dok veoma dobro ne budete upoznali Delphi, ovu stranicu možete da ostavite po strani i prihvatite generičke vrednosti. Slika 9.6 prikazuje karticu programa za povezivanje koja pripada okviru za dijalog opcije projekta. Opcije koje su dostupne na ovoj strani su objašnjene u narednom delu.

Slika 9.6
Opcije programa za povezivanje

Map dototeka

Opcije map datoteke kontrolišu da li je datoteka za mapiranje generisana i koliko će detalja biti uključeno u nju. Map datoteka je napredni alat za debagiranje i nećete je želeti koristiti sve dok bolje ne upoznate Delphi. Iz tog razloga neću detaljnije objašnjavati opcije map datoteke.

Opcije EXE i DLL

Deo sa opcijama EXE i DLL određuje tip izvršne datoteke koju će Delphi kreirati za projekt aplikacije. Ukoliko je polje za potvrdu Generate Console Application potvrđeno, Delphi će napraviti aplikaciju konzole koja je različita od GUI aplikacije.

Opcija Include TD32 Debug Info omogućava da linker poveže informacije za debagiranje sa EXE, ili DLL datotekom. (Oznaka TD32 označava 32-bitnu verziju starog Turbo Debugger-a. TD32 je napredni samostalni debager koji se isporučivao sa paketom Borland C++ i sa nekim verzijama Delphi-ja.) Neki alati za debagiranje koriste informacije za debagiranje u formatu TD32. Program Memory Sleuth firme TurboPower, na primer, zahteva da se u okviru izvršne datoteke nalazi TD32. Ovu opciju možete uključiti kada koristite programe poput Memory Sleuth.

Opcija Include Remote Debug Symbols generiše simbole za debagiranje koji su neophodni za udaljeno debagiranje Web broker aplikacije.

Zapis programa za povezivanje (Linker Output)

Deo Linker Output određuje koji tip prevedene binarne datoteke će kreirati program za povezivanje. Normalno, program za povezivanje kreira DCU datoteke (generičke datoteke za Delphi aplikacije). Možda ćete poželeti da kreirate objektne datoteke za C i C++ (OBJ), umesto DCU datoteka, kako bi omogućili da se Vaši Pascal juniti koriste zajedno sa C i C++ programima koji su kreirani u Borland C++Builder-u.

Veličina memorije

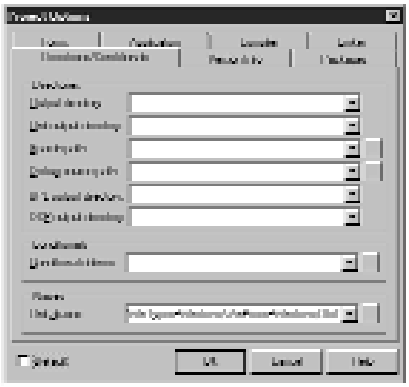
Odeljak za veličinu memorije mogu ignorisati svi osim najnaprednih korisnika. Generičke vrednosti su prihvatljive za sve aplikacije. U nekim slučajevima možete menjati baznu adresu kopije, ali je to veoma retko potrebno, možda samo ako kreirate DLL datoteku.

Opis

Polje za opis izvršne datoteke se koristi za definisanje stringa koji će biti dodat u aplikaciiju. Ovo polje se ponekad koristi da doda informacije o autorskim pravima u EXE i DLL datoteke. Uglavnom ćete koristiti polje u koje se upisuje informacija o verziji programa, kako biste upisali informaciju o autorskim pravima, umesto da koristite polje izvršne datoteke na ovoj kartici. Informacije o verziji su opisane u jednom od narednih poglavlja "Kartica za informaciju o verzijama".

Kartica direktorijumi/uslovi

Kartica za direktorijume/uslove (Directories/Conditionals) okvira za dijalog opcije projekta je mesto gde se podešavaju direktorijumi koje Vaš projekt koristi da pronađe datoteke biblioteka. Slika 9.7 prikazuje karticu direktorijumi/uslovi. Polja na ovoj kartici su opisana u narednim poglavljima.



Slika 9.7
Kartica direktorijumi/uslovi

Direktorijumi

Polja u ovom odeljku određuju direktorijum u kom Delphi može pronaći razne izvorne datoteke u toku prevođenja projekta. Takođe sadrži polja koja određuju direktorijum za kreiranje određene datoteke nakon prevođenja i povezivanja, odnosno pravljenja projekta.

Polje za direktorijum završne datoteke (Output Directory) se koristi za određivanje direktorijuma gde će se nalaziti izvršna datoteka, odnosno DLL datoteka. Direktorijum za kreiranje junita (Unit Output Directory) određuje gde će biti postavljena DCU datoteka koja se kreira prilikom prevođenja junita. Polje za putanju koja se pretražuje (Search Path) se koristi za definisanje direktorijuma gde se mogu pronaći sve dodatne biblioteke koje su potrebne za pravljenje projekta. Polje za putanju izvornih datoteka za debagiranje (Debug Source Path) se koristi za definisanje putanje do izvornih datoteka junita koje želite da debagirate, a ne nalaze se u direktorijumu projekta koji se trenutno koristi. Na primer, ukoliko želite da u toku debagiranja pristupite i DLL datoteci, treba da unesete putanju izvornog koda DLL datoteke u ovo polje. BPL izlazni direktorijum (BPL Output Directory) i DCP izlazni direktorijum (DCP Output Directory) su polja koja definišu gde će se snimati BPL i DCP datoteke u toku kreiranja paketa.

Uočite dugmad sa tri tačke koje se nalaze pored nekoliko polja na kartici direktorijumi/uslovi. Klikom miša na ovu dugmad biće prikazan editor koji Vam omogućava da dodate, uklonite, ili reorganizujete elemente određenog polja. Slika 9.8 prikazuje ove okvire za dijalog u toku editovanja polja za putanju pretraživanja.

Slika 9.8
Okvir za dijalog editor direktorijuma

Uslovi

Polje za definisanje uslova se koristi za kreiranje definicija koje želite da dodate nivou projekta. Na primer, pretpostavimo da imate kod u okviru projekta koji će biti preveden samo ako je definisan simbol `TRIALRUN` (probni rad). U ovom slučaju treba da dodate polju za definiciju uslova simbol `TRIALRUN`. Ukoliko treba da dodate više od jednog simbola, budite sigurni da je svaki simbol razdvojen znakom tačka - zarez.

Alijasi (Aliases)

Polje Unit Aliases se koristi za definisanje alijasa junita. Na primer, Delphi 1 je koristio junite pod nazivom `WinTypes.pas` i `WinProcs.pas` za sve što je vezano za Windows kod. Delphi 2, 3 i 4 koriste junit pod nazivom `Windows.pas`, umesto junita `WinTypes` i `WinProcs`. Polje za alijase junita će dodeliti alijase datotekama

`WinTypes` i `WinProcs` i uputiti ih na junit `Windows`. U ovom slučaju, dodela alijasa omogućava programima koji su pisani u Delphi-ju 1 da se prevedu u Delphiju 4 bez izmene liste `uses`.

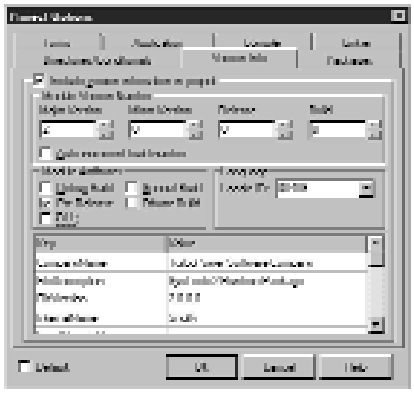
Kartica za podatke o verziji (Version Info)

Kartica za informaciju o verziji Vam omogućava da definišete informacije o verziji Vaše aplikacije. Informacije o verziji se upisuju u izvršne datoteke, DLL datoteke, odnosno ActiveX datoteke. Ova informacija se koristi prilikom instalacije programa, kako bi program za instalaciju utvrdio da li datoteka koja se instalira pripada novijoj, ili starijoj verziji programa.

Informacija o verziji ima i druge primene. Informaciju o verziji datoteke možete da pogledate iz programa Windows Explorer. Kliknite desnim tasterom miša na datoteku i odaberite opciju Properties u okviru menija sadržaja. Kada se pojavi okvir za dijalog Properties, kliknite na jezičak kartice Version da biste videli podatke o verziji datoteke. Slika 9.9 prikazuje okvir za dijalog Properties koji prikazuje informacije o verziji pomoćnog programa Database Desktop koji se nalazi u okviru Delphija.

Slika 9.9
Okvir za dijalog Properties prikazuje informaciju o verziji programa `DBD32.EXE`.

Slika 9.10 pokazuje karticu informacija o verziji okvira za dijalog opcije projekta. Na vrhu kartice se nalazi polje za potvrdu sa natpisom Include Version Information in Project (Uključivanje informacije o verziji u projekat). Ukoliko je ovo polje za potvrdu odabrano, informacija o verziji projekta će biti upisana u izvršnu datoteku. Ukoliko polje za potvrdu nije odabrano, informacija o verziji se ne upisuje u projekt, a ostatak kartice je deaktiviran.



Slika 9.10
Informacija o verziji Vašeg projekta se može upisati na karticu Version Info

Preostala polja kartice Version Info se koriste za definisanje raznih opcija o informacijama vezanim za verziju projekta. Glavni broj verzije (Major Version), pomoćni broj verzije (Minor Version), izdanje (Release) i napravljeno (Build) su polja koja zajedno formiraju broj verzije datoteke. Broj verzije datoteke na slici 9.10 je verzija 2.0, napravljeno 0. Ukoliko odaberete opciju za automatsko povećanje broja verzije koja je napravljena, broj u polju Build će se automatski povećavati, svaki put kada izvršite opciju za pravljenje projekta.

Odeljak za atribute modula (Module Attributes) se može koristiti za definisanje posebnih atributa koje želite da definišete uz datoteku. Odeljak za jezik (Language) Vam omogućava da odaberete lokalni identifikator za datoteke. Za detaljnije informacije o mogućim vrednostima polja Locale ID pogledajte opciju sistema za pomoć u toku rada Windows API u odeljku "Language Indentifiers and Locales" (Identifikatori jezika i lokali).

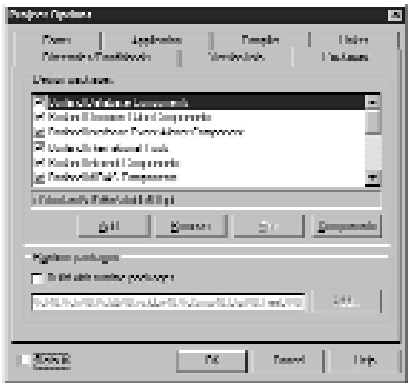
Tabela u dnu kartice Version Info se može koristiti za definisanje nekoliko različitih informacija. Ove informacije mogu uključiti naziv Vaše firme, opis datoteke, interni naziv datoteke, informacije o autorskim pravima i zaštitnom znaku, naziv proizvoda, verziju proizvoda, kao i dodatne komentare koje želite da priključite datoteci.

Možete upisati informacije u bilo koje polje, odnosno ne morate upisati informacije ni u jedno polje (polje `FileVersion` je skup polja koja se nalaze u odeljku za definisanje broja verzije modula). Takođe, možete dodati i posebna polja za definisanje informacija o verziji. Da biste dodali polje za informaciju o verziji, kliknite na tabelu u koju se upisuju informacije o verziji, a zatim koristeći kursorski taster, pomerite se na kraj tabele. Na kraju tabele pritisnite još jednom kursorski taster na dole i pojaviće se okvir za dijalog sa pitanjem koju dodatnu informaciju da upiše. Upišite naziv ključa i ključ će biti dodat informacijama o verziji koje su vezane za projekat. Dodavanje informacija o verziji projekta, nikad do sada nije bilo tako jednostavno!

Kartica paketa (Packages)

Kartica paketa se koristi za definisanje tipa povezivanja koji će Vaš projekt koristiti. Gornji deo kartice Vam omogućava da dodajete, odnosno uklanjate pakete za dizajniranje, ali sve to neće imati nikakve veze sa tekućim projektom. Jedina opcija koja ima veze sa tekućim projektom je polje za potvrdu Build with Runtime Packages (Kreiranje sa paketima koji se izvršavaju u toku rada programa).

Kada je ova opcija odabrana Vaša aplikacija će koristiti dinamičko povezivanje VCL komponenti i komponenti nezavisnih proizvođača. Ovo znači da će Vaša izvršna datoteka biti manja, ali ćete morati da isporučujete određene pakete uz Vašu aplikaciju. Kada je ovo polje za potvrdu isključeno, Vaša aplikacija koristi statičko povezivanje. *Statičko povezivanje* znači da se bilo koji kod vezan za VCL komponente, odnosno za komponente nezavisnih proizvođača, vezuje za izvršnu datoteku. Paketi su bili detaljnije obrađeni u jučerašnjoj lekciji, u poglavlju "Korišćenje paketa". Slika 9.11 prikazuje karticu paketa okvira za dijalog opcije projekta.



Slika 9.11
Kartica paketa

Delphi-jev editor koda

Nema sumnje da je Delphi po svojoj prirodi vizuelno orijentisan - što je jedna od velikih olakšica u programiranju Delphi-jem. Ipak, bilo koji program, značajan, ili beznačajan, sadrži veliki deo koda koji se mora ručno pisati. Nakon što unesete ulazno-izlazni deo Vaše aplikacije koji ste napisali korišćenjem impresivnog Delphi-jevog vizuelnog alata, treba da pređete na teži deo posla radeći sa Delphi-jevim editorom koda. Editor koda ima neke mogućnosti koje ćete naučiti da cenite kada ih budete otkrili.

U ovom poglavlju ćete naučiti:

- osnovne operacije editora
- specijalizovane opcije editora

- meni sadržaja editora koda
- izmene opcija editora

NAPOMENA Delphi-jev editor koda Vam omogućava da odaberete četiri tipa konfiguracije vezane za definiciju tastatutre: generički tip, tip klasičnog okruženja, kratki tip i ipsilon tip. Ostatak ovog poglavlja podrazumeva da je definicija tastature podešena na generički tip konfiguracije. Ukoliko ste se privikli na bilo koji drugi tip konfiguracije tastature, možete ignorisati reference za određene kombinacije tastera.

Osnovne operacije editora

Podrazumevam da već poznajete Delphi i da možete upisivati i brisati tekst, označavati tekst korišćenjem miša, isecati, kopirati, lepiti tekst i slično. Opise funkcija sa ovog nivoa neću obrađivati.

Ukoliko podrobnije pogledamo Delphi-jev editor koda, možemo videti da on pripada tipičnim editorima koda. Njegove mogućnosti označavanja sintakse, što olakšava ide-ntifikaciju ključnih reči, stringova, numeričkih konstanti i komentara, su veoma jednostavne. Nešto kasnije ćete moći da vidite kako se podešavaju karakteristike editora.

Editor koda je prozor sa karticama. Možete otvoriti koliko god želite prozora u okviru editora; svaki prozor će biti predstavljen jezičkom kartice na vrhu prozora editora, a na jezičku će se nalaziti naziv datoteke. Da biste prešli na izvornu datoteku, jednostavno kliknite na jezičak kartice koji pripada datoteci koju želite da pregledate. Ukoliko postoji više datoteka nego što se može prikazati u okviru prozora, pojaviće se dugmad za pomeranje, tako da ćete moći da pomerate jezičke kartica i na taj način tražite odgovarajuću izvornu datoteku.

Statusna traka u dnu editora koda Vam pruža statusne informacije (što je očigledno). Trenutni broj linije na kojoj se nalazi kursor, kao i pozicija kursora u okviru linije se nalaze na levom panou statusne trake. Ukoliko je datoteka menjana nakon poslednjeg snimanja, statusna datoteka će u centralnom panou imati oznaku `Modified` (izmenjeno). Ukoliko je datoteka definisana kao datoteka koja se može samo čitati, u centralnom panou će biti upisano `Read Only`.

Prozor editora na levoj margini ima traku koja se naziva *žljeb*. Žljeb se koristi da prikaže ikone u različitim fazama procesa razvoja. Na primer, kada želite da definišete mesto na kom će zastati debager (biće obrađeno u sutrašnjoj lekciji), u žljeb će biti postavljena crvena tačka. Ukoliko želite da definišete oznaku (uskoro će biti obrađena), ikona koja predstavlja oznaku će biti postavljena u žljeb.

NAPOMENA Ukoliko greškom kliknete na žljeb u pokušaju da odaberete tekst, ili postavite kursor, primetićete da se na toj liniji postavlja tačka prekida. Da biste je obrisali, kliknite ponovo mišem na žljeb.

Otvaranje i snimanje datoteka

Ništa nije nepoznato u vezi otvaranja i snimanja datoteka u okviru editora koda. Sigurno Vam je jasno da ne postoji razlika između otvaranja projekta i otvaranja datoteke izvornog koda. Kada odaberete opciju File➥Open Project u okviru glavnog menija, bićete upitani za naziv datoteke projekta koji želite da otvorite. Kada odaberete opciju File➥Open u okviru glavnog menija možete otvoriti zasebne Delphi-jeve datoteke izvornog koda, odnosno datoteke forme. U stvari, možete otvoriti bilo koji tip tekst datoteke (uključujući tu i tipove datoteka koji nisu prikazani u okviru za dijalog koji se koristi za otvaranje datoteka). Ovo uključuje datoteke `.pas`, `.rc`, `.txt`, pa čak i C++ datoteke izvornog koda kao i datoteke zaglavlja (`.cpp` i `.h`). Obe opcije menija Open i Open Project imaju svoje dugme na traci za alate.

Ukoliko otvorite datoteku junita (`.pas`) koja sadržu formu, Delphi će otvoriti datoteku sa izvornim kodom u okviru editora koda, a formu u okviru dizajnera forme.

Takođe možete otvoriti više datoteka istovremeno. Da biste otvorili više datoteka, treba da ih odaberete iz okvira za dijalog Open, a zatim da kliknete na dugme OK. Sve odabrane datoteke će biti učitane, i jezičak za svaku datoteku će se pojaviti u gornjem delu prozora editora.

Takođe možete koristiti i mogućnost prevlačenja i spuštanja kako biste otvorili datoteke. Na primer, možete odabrati datoteku (ili grupu datoteka) u programu Windows Explorer i prevući ih u editor koda, a zatim ih spustiti. U okviru editora koda će biti otvorena željena datoteka.

Da biste snimili datoteku, odaberite opciju File➥Save, odnosno File➥Save As u okviru glavnog menija, odnosno pritisnite kombinaciju tastera Ctrl+S. Ukoliko datoteka prethodno nije bila snimljena, pojaviće se okvir za dijalog Save As, pa ćete moći da upišete ime datoteke.

Označavanje teksta

Iako je označavanje teksta osnovna stvar koju sadrže svi editori teksta, mislim da ne bi bilo naodmet da se spomene nekoliko tehnika za označavanje teksta koje možete koristiti u Delphi-jevom editoru koda.

Da biste označili kratak blok teksta, možete da prevlačite miša preko teksta koji želite da označite. Nakon što ste odabrali tekst možete ga iseći, kopirati, odnosno zalepiti po potrebi. Da biste označili duži blok koda, možete koristiti metodu klik tasterom miša+Shift+klik tasterom miša. Prvo kliknite na početak bloka koji želite da označite. Zatim, držeći pritisnut taster Shift, kliknite mišem na kraj bloka. Tekst između početne i završne tačke je označen.

Još jedna korisna mogućnost je izbor određene reči. Da biste odabrali ključnu reč, naziv funkcije, ili promenljive, potrebno je da kliknete dva puta mišem na željenu reč. Sada možete izvršiti bilo koju operaciju editovanja nad rečju koju ste označili.

SAVET Dok programirate često ćete dodavati, brisati, ili pomerati blokove teksta. Ponekad će biti potrebno da uvučete kompletan blok koda. U drugim slučajevima ćete trebati da vratite nazad (ne uvučete?) kompletan blok koda. Da biste uvukli blok koda, označite linije koje želite da uvučete, a zatim pritisnite tastere Ctrl+Shift+I. Kompletan blok koda će biti uvučen. Da biste vratili nazad uvučeni deo koda, pritisnite testere Ctrl+Shift+U.

Editor koda takođe podržava i prevlačenje i spuštanje teksta. Da biste pomerili deo koda, prvo ga označite. Zatim postavite kursor miša na označeni tekst i prevucite ga. Tekst prevlačite sve dok ne dođete do mesta gde želite da premestite kod. Otpustite taster miša i označeni tekst će biti prebačen na novu lokaciju. Da biste kopirali označeni tekst, umesto da ga prevučete, ponovite prethodne korake držeći pritisnut taster Ctrl, pre nego što spustite označeni tekst.

Poništavanje prethodne operacije (Undo)

Editor koda teoretski ima neograničen broj nivoa za poništavanje prethodne operacije (32.767 generički). Praktično, možete poništiti prethodne operacije sve do trenutka kada ste poslednji put snimili datoteku. Promenom opcija editora, bićete u mogućnosti da poništite prethodnu operaciju čak iako ste snimili datoteku. Opcijama editora i karakteristikama će biti posvećeno poglavlje pod nazivom "Promena opcija editora".

U suštini, vredi da zapamtite ovo načelo: "Poništavanje prethodne operacije je Vaš prijatelj."

Pronalaženje i izmena (find, replace)

Pronalaženje i izmena se često koristi u toku programiranja. Opciju za pronalaženje možete koristiti da biste pronašli određeni deo koda, odnosno, određenu promenljivu u okviru koda. Opciju za izmenu možete koristiti da promenite naziv promenljive, odnosno da promenite naziv metode. Mogućnosti su neograničene.

Delphi-jevi okviri za dijalog za pronalaženje teksta (Find Text) i izmenu teksta (Replace Text) su implementirani slično kao i standardne operacije za pronalaženje i izmenu teksta. Da biste prikazali okvir za dijalog za pronalaženje teksta, odaberite opciju Search➥Find u okviru glavnog menija, odnosno pritisnite tastere Ctrl+F. U polje Text to find (traženi tekst) upišite tekst koji želite da pronađete, a zatim kliknite mišem na dugme OK, ili pritisnite taster Enter. Ukoliko traženi tekst bude pronađen, automatski će biti označen.

NAPOMENA Tekst koji je označen primenom okvira za dijalog za pronalaženje teksta, nije isti kao tekst koji ste označili mišem. Uočićete da tekst koji se traži biva označen crnim, dok se tekst označen mišem, označava plavom bojom (podrazumeva se da niste menjali opcije editora). Tekst koji ste označili nakon pretraživanja se ne koristi za editovanje; samo je označen da biste ga bolje videli.

Da biste pozvali okvir za dijalog za izmenu teksta (Replace Text) odaberite opciju Search➥Replace u okviru glavnog menija, odnosno pritisnite tastere Ctrl+R. Slika 9.12 prikazuje Delphi-jev okvir za dijalog za izmenu teksta. Uz nekoliko očiglednih izuzetaka, okvir za dijalog za traženje teksta sadrži potpuno iste opcije.

U većini slučajeva, opcije okvira za dijalog za traženje i izmenu teksta rade upravo ono na šta ukazuju. Ukoliko odaberete opciju Case Sensitive (razlikovanje velikih i malih slova), u polje za tekst koji se traži morate upisati upravo ono što se nalazi u datoteci izvornog koda.

Slika 9.12
Okvir za dijalog za izmenu teksta

Opciju Whole Words Only možete koristi samo kada želite da budete sigurni da tekst koji tražite nije deo duže reči, odnosno naziva promenljive. Na primer, recimo da želite da zamenite reč `Form` sa `MyForm`. U ovom slučaju, želećete da koristite opciju traženja samo celih reči pošto se reč `Form` može koristiti i u drugim nazivima promenljivih (kao što je `TForm`).

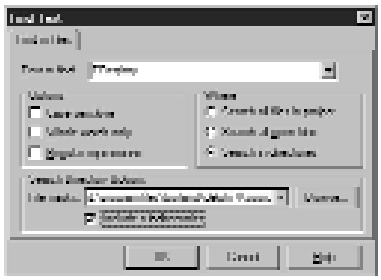
Opcija Regular Expressions (regularni izrazi) takođe zahteva objašnjenje. Ukoliko je ova opcija uključena, možete koristiti specijalne karaktere i džokere u toku pretraživanja. Specijalni karakteri Vam omogućavaju da pronađete elemente kao što je početak linije, odnosno kraj linije u okviru stringa koji tražite. Džokeri imaju sličnu namenu kao i kod operacija sa direktorijumima. Za kompletan opis regularnih izraza možete pogledati Delphi-jev sistem za pomoć u toku rada u odeljku "Regular Expressions" (Regularni izrazi).

Kada menjate tekst, najbezbednije je da ostavite uključenu opciju koja traži potvrdu za izmenu (Prompt on Replace). Prilikom korišćenja operacije Replace All i uključenom ovom opcijom editor će označiti svaku pronađenu reč i upitati Vas da li želite da je izmeni. Veoma je lako da se pogrešno pretpostavi rezultat operacije Replace All, zbog toga ovu opciju uvek koristite sa oprezom. Čak i tada vredi se setiti načela: "Poništavanje prethodne operacije je Vaš prijatelj". Ostale opcije za pretraživanje i izmenu su dovoljno jasne, pa ih nije potrebno dodatno objašnjavati.

Pretraživanje u okviru datoteka

Pretraživanje u okviru datoteka je veoma dobar alat za traženje teksta u više datoteka. Često koristim ovu opciju da bih pretražio VCL izvorni kod za određene metode, promenljive, ili klase. Ovaj alat je veoma koristan i zgodan i trebali bi da naučite kako da ga koristite.

Da bi okvir za dijalog za traženje teksta (Find Text) bio prikazan, odaberite opciju Search➥Find in Files u okviru glavnog menija. Možda je jednostavniji način korišćenje kombinacije tastera Ctrl+F, da bi se pojavio okvir za dijalog za pronalaženje teksta, a zatim izbor kartice Find in Files. Slika 9.13 prikazuje okvir za dijalog za traženje teksta sa karticom Find in Files.



Slika 9.13
Okvir za dijalog za traženje teksta

Opcija za traženje teksta u datotekama koristi neke od opcija za pretraživanje kao i standardna operacija za pretraživanje (razlikovanje velikih i malih slova, rad sa celim rečima i regularni izrazi). Dodatno, postoji opcija za pretraživanje svih datoteka u okviru projekta, svih otvorenih datoteka, odnosno datoteka u određenom dirtektorijumu, uključujući tu i poddirektorijume.

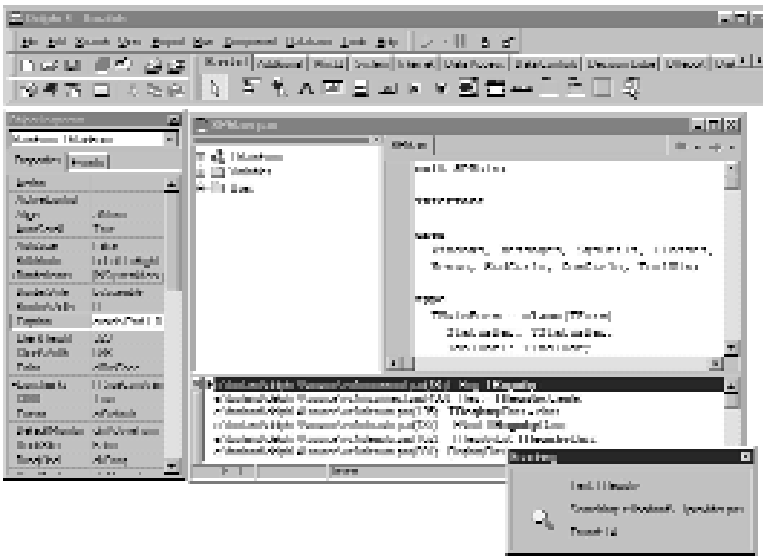
Kada pokrenete opciju za pretraživanje u datotekama, mali prozor sa naslovom Searching (pretraživanje) se pojavljuje u donjem desnom uglu Vašeg ekrana. Ovaj prozor prikazuje status operacije Find in Files. Pokazaće Vam koju datoteku trenutno pretražuje kao i broj do sada pronađenih reči. Da biste prekinuli pretraživanje, potrebno je da zatvorite prozor Searching.

Sve pronađene reči će se nalaziti u prozoru za poruke editora koda. Prozor za poruke prikazuje naziv datoteke u kojoj se nalazi traženi tekst, broj linije gde je tekst pronađen, kao i liniju koja sadrži traženi tekst, koji je istaknut podebljanim slovima. Da bi pregledali datoteku koja sadrži traženi tekst, dva puta kliknite mišem na prozor za poruke. Delphi će otvoriti odgovarajuću datoteku i prikazati liniju koja sadrži tekst koji ste tražili. Slika 9.14 prikazuje Delphi u trenutku pretraživanja grupe datoteka.

Kada definišete masku za datoteke, možete primeniti uobičajene džokere. Na primer, ako želite da pregledate sve tekst datoteke u tekućem direktorijumu, treba da u polje za masku datoteke unesete:

```
c: \MyStuff\*.txt
```

Alat za pretraživanje u datotekama je često neophodan. Veoma često ga koristim. Ukoliko naučite da koristite ovaj alat, uštedećete veoma mnogo vremena.



Slika 9.14
Delphi traži tekst

Dobijanje pomoći

Jedna od najkorisnijih opcija editora koda je integracija sa Delphi-jevim sistemom za pomoć. Jednostavno postavite kursor editora na ključnu reč jezika Object Pascal, VCL karakteristiku, ili metodu, odnosno na bilo koji drugi tekst vezan za Delphi i pritisnite taster F1. Ukoliko se u okviru sistema za pomoć nalazi element koji je označen kursorom, Windows sistem za pomoć će prikazati odgovarajuću stranu datoteke za pomoć. Ukoliko za odabrani tekst nije definisan element u okviru sistema za pomoć, biće prikazana poruka greške.

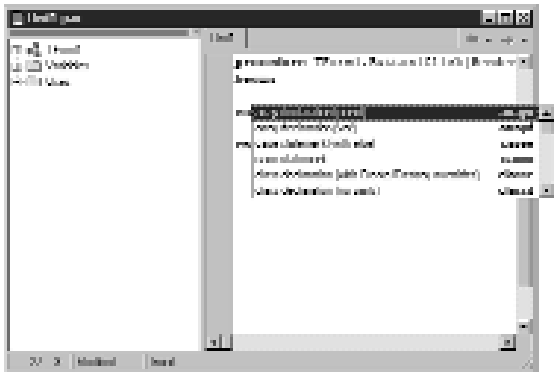
Ova mogućnost je veoma korisna kada ne možete da se setite određenog elementa Delphi-ja, jezika Object Pascal, ili VCL-a. Pomoć je, kako se to obično kaže, udaljena za samo jedan pritisak tastera.

Posebne mogućnosti editora

Delphi-jev editor koda ima nekoliko mogućnosti koje su od velike pomoći, ukoliko pišete veoma mnogo koda. Ove mogućnosti će biti objašnjene u nekoliko narednih poglavlja.

Obrasci koda (code templates)

Obrasci koda su još jedna sjajna opcija u okviru Delphi-jevog editora koda. Obrasci koda Vam omogućavaju da ubacite bilo koji unapred definisani kod (ili bilo koji tekst vezan za kod) u Vašu datoteku izvornog koda. Da biste koristili obrasce koda, pritisnite tastere Ctrl+J u toku rada sa editorom koda. Nakon toga će se pojaviti okvir za listu koji će Vam prikazati listu obrazaca. Odaberite obrazac sa liste, pritisnite taster Enter i tekst odgovarajućeg obrasca koda će biti ubačen u Vaš izvorni kod. Slika 9.15 prikazuje okvir za listu obrazaca koda koji se pojavljuje nakon pritiska na tastere Ctrl+J.



Slika 9.15
Delphi-jev okvir za listu obrasca koda

Nove obrasce možete dodavati, odnosno možete editovati postojeće obrasce korišćenjem kartice Code Insight okvira za dijalog opcija okruženja. Ukoliko više volite, možete otvoriti datoteku sa obrascima koda uz pomoć bilo kog tekst editora (kao što je Delphi-jev editor koda) i editovati obrasce koda. Datoteka obrazaca koda ima naziv `DELPHI32.DCI` i nalazi se u direktorijumu `Delphi 4\Bin`.

Slobodno možete menjati obrasce koda onako kako Vam odgovara. Na primer, obrazac koda koji predstavlja iskaz `for` sam izmenio, tako da ima sledeći izgled:

```
for I := 0 to Pred(¦) do begin
end;
```

Uočite znak uspravne crte (`¦`) koji se nalazi u isečku koda. Uspravna crta u okviru obrasca koda je mesto koje određuje gde će se kursor nalaziti prilikom unosa obrasca.

Ukoliko pravite mnogo modifikacija u okviru datoteke za obrasce koda, obezbedite rezervnu kopiju datoteke na bezbednom mestu. Rezervna kopija će Vam biti potrebna pošto će Delphi-jev instalacioni program presnimiti Vašu izmenjenu datoteku `DELPHI32.DCI` prilikom promene instalacije, odnosno ponovne instalacije Delphi-ja.

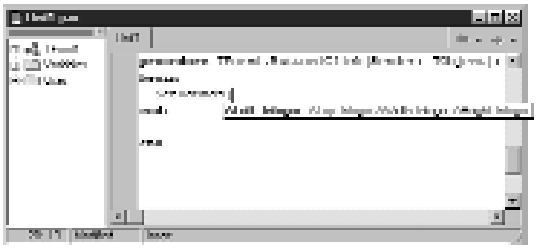
Obrasce koda možete koristiti i za stvari koje nisu vezane za kod. U firmi TurboPower Software datoteke izvornog koda uvek imaju zaglavlje na početku junita. Zaglavlje izgleda ovako:

```
{*******************************************************};
{*                 Filename and version                *};
{*        Copyright (c) TurboPower Software 1998       *};
{*                 All rights reserved.                *};
{*******************************************************};
```

Pošto većina ovog teksta ostaje neizmenjena, imam obrazac koji mogu brzo da ubacim u zaglavlje novih junita koje kreiram. Obrasce koda treba da koristite za bilo koji tekst koji često koristite u svakodnevnom programiranju.

Parametri koda (code parameters)

Parametri koda su mogućnost editora koda da prikaže savet koji Vas pita za potrebne paramerte VCL metoda, odnosno Windows API funkcija. Sa stotinama VCL metoda i Windows API funkcija realno ne postoji mogućnost da se zapamte svi parametri svih funkcija. Prikazivanjem parametara metode koje upravo kucate, ova mogućnost editora koda Vam štedi vreme. Na primer, recimo da pozivate metodu SetBounds. Kada otkucate otvorenu zagradu, oblačić za savet će se pojaviti, kao što je to prikazano na slici 9.16.



Slika 9.16
Parametri koda u toku rada

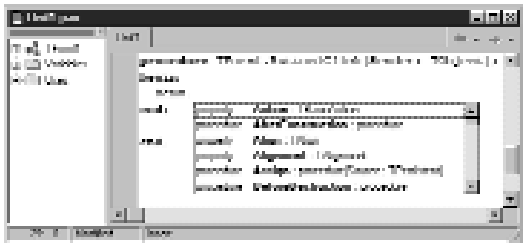
Kao što ste mogli da vidite na slici 9.16, svaki parametar je prikazan u oblačiću za savet. Parametar koji treba da sledeći kucate je prikazan podebljanim slovima. Nakon što upišete parametar, sledeći parametar sa liste će biti prikazan podebljenim slovima. Ovo se nastavlja sve dok ne upišete sve parametre metode. Nakon što upišete sve potrebne parametre, oblačić za savet sa parametrima koda nestaje. Opcije parametara koda se podešavaju u okviru za dijalog opcija okruženja na kartici Code Insight. Ova kartica će biti obrađena u poglavlju "Kartica izgleda koda".

Kompletiranje koda

Kompletiranje koda je još jedna mogućnost editora koda koja Vam može uštedeti vreme za razvoj programa. Upišite naziv promenljive klase zajedno sa operatorom tačka iza naziva klase, a editor koda će prikazati okvir za listu sa svim karakteristikama i metodama klase. Na primer, ukoliko imate memo komponentu sa nazivom Memo, upisaćete

```
Memo.
```

i zastati na trenutak. Pojaviće se okvir za listu, kao što je to prikazano na slici 9.17.



Slika 9.17
Kompletiranje koda prikazuje karakteristike i metode klase TMemo

Kada budete videli okvir za listu, možete odabrati opciju sa liste na jedan od dva načina. Jedan od načina je da pronađete karakterisktiku, ili metodu koristeći miš, ili tastaturu, a zatim je odaberete pritiskom na taster Enter. Karakteristika, ili metoda koju ste odabrali će biti ubačena u Vaš kod. Postoji mogućnost i da upišete prvih nekoliko slova karakteristike, ili metode koju želite da ubacite u kod. Dok kucate, Delphi će pretražiti listu karakteristika i metoda i označiti opciju na listi koja najviše odgovara tekstu koji kucate.

Kada ugledate karakteristiku, ili metodu koju želite, pritisnite taster Enter i karakteristika, odnosno metoda će biti uneta u Vaš kod. Ukoliko ne želite da koristite listu za kompletiranje koda, pritisnite taster Esc i okvir za listu za kompletiranje koda će nestati.

Kompletiranje koda Vam štedi vreme pružajući Vam listu karakteristika i metoda koje možete odabrati. Dodatna mogućnost je što omogućava proveru sintakse i proveru korišćenja velikih i malih slova kod naziva karakteristika i metoda. Potrebno je samo da odaberete karakteristiku, ili metodu koju želite, pritisnete taster Enter i Delphi će ubaciti identifikator u Vaš kod.

Oblačić za savet sa oznakom simbola

Oblačić za savet sa oznakom simbola se prikazuje kada postavite kursor miša na bilo koji identifikator u okviru Vašeg izvornog koda. Na primer, standardni projekt će sadržati sledeću liniju koda u odeljku interface:

```
TForm1 = class(TForm)
```

Kada postavite kursor miša na identifikator TForm, oblačić za savet će se pojaviti sa sledećim natpisom:

```
type Forms.TForm = class(TCustomForm) - Forms.pas (584)
```

Ova linija prikazuje dekleraciju klase `TForm` i saopštava Vam da je klasa `TForm` deklarisana u okviru linije 584 junita `Forms.pas`.

Oblačić za savet sa oznakom simbola Vam daje informacije o bilo kojoj promenljivoj u okviru Vašeg programa. Ovo je veoma zgodno ukoliko zaboravite tip promenljive (da li je X deklarisdano kao `Byte`, `Word`, ili `Integer`?).

Kompletiranje klasa

Kompletiranje klasa je novina u Delphi-ju 4. Upišite bilo koju dekleraciju karakteristike, ili metode u odeljak interface, a zatim pritisnite tastere Ctrl+Shift+C. Delphi će dodati odgovarajući kod u odeljak implementation, kako bi kompletirao klasu. Ovo ćemo najbolje ilustrovati izvođenjem vežbe. Izvršite sledeće korake:

1. Počnite sa praznim projektom.
2. Pređite na editor koda i pronađite dekleraciju klase forme u odeljku `interface`.
3. Upišite sledeći kod u odeljak `public` za deklarisanje klase forme:

```
procedure Test;
function GetSomething : Integer;
```

4. Pritisnite tastere Ctrl+Shift+C.

Nakon što ste izvršili korak četiri, Delphi je automatski dodao osnove metoda `Test` i `GetSomething` u odeljak `implementation`, a zatim postavio kursor za editovanje na prvu metodu. Dodati kod izgleda ovako:

```
function TForm1.GetSomething: Integer;
begin

end;

procedure TForm1.Test;
begin

end;
```

Sačuvajte ovaj junit pošto ćete ga koristiti u sledećem poglavlju, kada budemo obrađivali navigaciju između modula.

Kompletiranje koda radi sa dekleracijama karakteristika, isto kao i sa metodama. Upišite dekleraciju karakteristike, pritisnite tastere Ctrl+Shift+C i Delphi će završiti dekleraciju karakteristike. Čak će dodati i metodu `write` Vašoj karakteristici. Karaktertistike za pisanje će biti obrađene tek u lekciji dana 20, "Kreiranje komponenti", tako da Vam za sada ne izgledaju suviše poznato. Mnogo će Vam jasnije biti kada počnete pisati sopstvene komponente.

Za mene je kompletiranje klasa jedna od najboljih novina u Delphi-ju 4. Samo ova novina mi uštedi prilično mnogo vremena u toku pisanja komponenti. Već neko vreme je koristim i više ne bih mogao da radim bez nje.

Navigacija između modula

4 Navigacija između modula Vam omogućava da se brzo prebacite sa metoda u odeljku `implementation` na dekleraciju metode u odeljku `interface`, i obrnuto. Još jednom, vežba vredi koliko i hiljadu reči. Uradite sledeće:

1. Pređite na dekleraciju klasu forme koju ste izmenili u prethodnoj vežbi.
2. Kliknite na liniju koja sadrži dekleraciju procedure `Test`.
3. Pritisnite tastere Ctrl+Shift+strelica na dole. Editor koda preskače na sadržaj procedure u okviru odeljka `implementation`.
4. Pritisnite tastere Ctrl+Shift+strelica na gore. Editor koda se vraća na dekleraciju procedure `Test` u okviru odeljka `interface`.
5. Pomerite se na liniju sa dekleracijom funkcije `GetSomething`. Pritisnite ponovo tastere Ctrl+Shift+strelica na dole. Editor koda prelazi na metodu `GetSomething` u odeljku `implementation`.

Kao što ste mogli da vidite navigacija između modula omogućava jednostavni prelazak između odeljaka `implementation` i `interface`.

SAVET Nije važno da li koristite taster strelica na gore, ili strelica na dole na tastaturi. Pritisak na bilo koji od ovih tastera će preskakati između odeljaka interface i implementation.

Izbor modula

4 Izbor modula je još jedan od alata za navigaciju između modula. U okviru editora koda držite pritisnut taster Ctrl i postavite kursor miša na naziv identifikatora. Identifikator će biti označen plavom bojom i biće podvučen. Kliknite na identifikator i Delphi će Vas prebaciti na mesto u okviru izvornog koda, gde je identifikator deklarisan.

U ovom slučaju opcija za izbor modula liči na opciju za navigaciju između modula. Ipak, ova opcija ide i nešto dalje. Sa opcijom za izbor modula možete kliknuti na identifikator VCL komponente, kao i na Vaše sopstvene identifikatore. Ilustrovaću Vam ovo primerom. Ipak, pre nego što počnem, napominjem Vam da će ova vežba raditi samo ukoliko imate Delphi verziju Professional, ili Client/Server. Da bi mogao da radi, Delphi-jev izvorni kod će tražiti izvorni kod VCL komponenti. Izvršite sledeće korake:

1. Kreirajte novu aplikaciju. Postavite dugme memo polja na formu.
2. Odaberite opciju Project➥Options u okviru glavnog menija. Kliknite na jezičak kartice Directories/Conditionals okvira za dijalog opcije projekta. Upišite u polje za pretraživanje putanje:

```
$(DELPHI) \source\vcl;$(DELPHI) \source\rtl\win
```

Kliknite na dugme OK, kako biste zatvorili okvir za dijalog za opcije projekta.

3. Dva puta kliknite na dugme i dodajte sledeći tekst u upravljač događajem `OnClick` za dugme:

```
Memo1.Clear;
```

Metoda `Clear` klase `TMemo` briše sadržaj memo komponente. Da li ste radoznali kako će VCL izvorni kod izgledati za metodu `Clear`? Nastavite...

4. Držite pritisnut taster Ctrl na tastaturi i kliknite na metodu `Clear`.

5. Nakon par trenutaka, biće prikazan VCL junit `StdCtrls` u okviru editora koda sa kursorom na metodu `TCustomEdit.Clear` (metoda `Clear` je definisana u okviru klase `TCustomEdit` jedne od naslednice klase `TMemo`). Metoda će izgledati ovako:

```
procedure TCustomEdit.Clear;
begin
  SetWindowText(Handle, '');
end;
```

Interesantno. Samo jedna linija koda. Ali, odakle dolazi funkcija `SetWindowText`? Sledeći korak, molim.

6. Ponovo držite pritisnut taster Ctrl i ponovo kliknite na funkciju `SetWindowText`. Nakon par sekundi biće otvoren junit `Windows` i kursor će se naći na liniji:

```
function SetWindowText; external user32 name 'SetWindowTextA';
```

Ova linija Vam saopštava da je funkcija `SetWindowText` Windows funkcija koja se nalazi u DLL datoteci pod nazivom `USER32`. Slučaj zatvoren. Ipak, još nije sve završeno.

7. Sada pogledajte u jezičak kartice u gornjem desnom uglu prozora editora koda. Videćete dugmad za pomeranje *napred* i *nazad*. Kliknite na dugme za pomeranje unazad (ono koje pokazuje na levo). Editor koda Vas prebacuje na prethodnu tačku (metoda `Clear` u junitu `StdCtrls`).

8. Kliknite na dugme za pomeranje unapred. Editor koda prikazuje funkciju `SetWindowText` uvezenu u `junit Windows`.

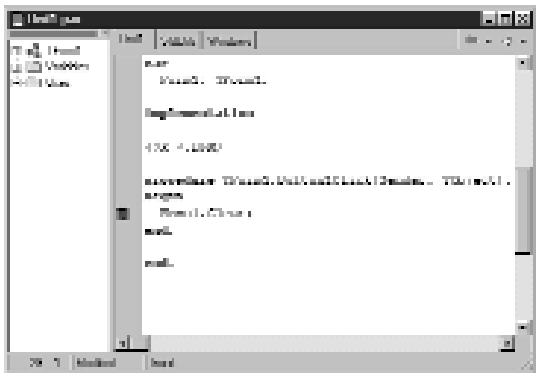
9. Kliknite na strelicu na dole pored dugmeta za pomeranje unazad. Videćete listu mesta sa izvornim kodom u okviru liste history. Kliknite na lokaciju da biste prebacili kursor editora koda na željenu poziciju.

Izbor modula je veoma dobar alat za navigaciju, ne samo kroz Vaš kod, nego i kroz VCL izvorni kod i izvorni kod bilo koje biblioteke komponenti nezavisnih proizvođača koju ste instalirali. Zapamtite da čitanjem VCL izvornog koda možete veoma mnogo naučiti, zato se nemojte bojati da prolazite kroz ovaj kod.

Korišćenje oznaka

Da biste privremeno označili poziciju u okviru izvornog koda, možete koristiti oznake. Na primer, često ćete ostavljati blok koda na kom trenutno radite da biste pregledali prethodno napisan kod, odnosno kopirali kod sa druge lokacije. Postavljanjem oznake na određeno mesto u okviru koda, pre nego što pređete na drugi posao, može Vas vratiti na određeni deo koda pritiskom na samo jedan taster. Možete postaviti do deset oznaka istovremeno.

Da biste postavili oznaku na određeno mesto, pritisnite tastere Ctrl+Shift i broj oznake koju želite da postavite. Na primer, da biste podesili oznaku 0 (prva oznaka), postavite kursor editora na mesto koje želite da označite, a zatim pritisnite tastere Ctrl+Shift+0 (Ctrl+K+0 će isto tako funkcionisati). Kada postavite oznaku u žljebu editora koda će se pojaviti ikona koja će označavati da oznaka postoji u toj liniji. Ikona pokazuje broj oznake. Slika 9.18 prikazuje editor koda sa oznakom postavljenom na liniju.



Slika 9.18
Editor koda sa postavljenom oznakom

Da biste se vratili na oznaku, pritisnite taster Ctrl i broj oznake na koju želite da se vratite. Koristeći isti primer, možete otkucati Ctrl+0, da biste se vratili na oznaku. (Takođe, možete da podesite oznake i prelazite na oznake koristeći meni sadržaja editora koda.) Da biste obrisali oznaku, postavite kursor editora bilo gde na liniju na kojoj se nalazi oznaka i ponovo pritisnite Ctrl+Shift+0.

NAPOMENA Oznake će biti postavljene za svaku datoteku koju ste otvorili u okviru editora koda. Na primer, možete imati oznaku broj #0 u okviru jedne izvorne datoteke i isto tako oznaku broj #0 u okviru druge izvorne datoteke. Ovo znači da se na oznake ne može prelaziti iz jedne u drugu datoteku izvornog koda. Ukoliko imate postavljenu oznaku broj #0 u datoteci `Unit1.pas`, ne možete nakon pritiska na tastere Ctrl+0 u okviru datoteke `Unit2.pas` preći na oznaku broj 0 u okviru datoteke `Unit1.pas`.

Da bi ilustrovali korišćenje oznaka uradite sledeće:

1. Otvorite bilo koju izvornu datoteku u okviru editora koda.
2. Spustite se do skoro poslednje linije datoteke i kliknite na jednu liniju koda.
3. Pritisnite tastere Ctrl+Shift+0 da biste postavili oznaku. Ikona oznake će biti prikazana u žljebu editora koda.
4. Pritisnite tastere Ctrl+Home da biste se prebacili na početak datoteke izvornog koda.
5. Sada pritisnite tastere Ctrl+0 da biste se vratili nazad na oznaku. Editor koda prikazuje liniju koda gde je postavljena oznaka, a kursor se nalazi na istom mestu gde je bio postavljen kada ste definisali oznaku.
6. Pritisnite ponovo tastere Ctrl+Shift+0 kako biste obrisali oznaku. Oznaka je obrisana i ikona je nestala sa žljeba editora koda.

Uočite da su oznake privremene. Kada zatvorite datoteku izvornog koda, oznake nisu sačuvane. Uočite takođe da za postavljanje i uklanjanje oznaka, morate koristiti tastere sa brojevima na glavnom delu tastature. Za postavljanje i uklanjanje oznaka se ne može koristiti numerički deo tastature.

Inkrementirano pretraživanje

Za brzo pronalaženje malih grupa karaktera možete koristiti inkrementirano pretraživanje. Da bi pokrenuli inkrementirano pretraživanje, odaberite opciju SearchÐIncremental Search u okviru glavnog menija, odnosno pritisnite tastere Ctrl+E na tastaturi. Da biste lakše shvatili kako radi opcija inkrementiranog pretraživanja, najjednostavnije će biti da uradimo vežbu. Izvršite sledeće korake:

1. Kreirajte novu tekst datoteku iz skladišta objekata. (Nije važno da li imate trenutno otvoren projekt.)
2. Upišite sledeći tekst:

   Učenje pisanja Windows

   programa deo po deo nije

   tako loše. Zar nije vreme

   da se vratite nazad na posao?
3. Vratite se kursorom na početak datoteke (Ctrl+Home).
4. Pritisnite tastere Ctrl+E da biste počeli sa inkremintiranim pretraživanjem. Tražićete reč posao. Uočite da statusna traka editora koda glasi `Searching for:`. (Traženje:.)

5. Otkucajte slovo p. Slovo p u reči pisanja će biti označeno. To nije ono što ste tražili.
6. Sada upišite slovo `o`. Sledeća pojava karaktera `po` će biti pronađena, ovog puta u reči `po`. Ovo još nije ono što ste tražili.
7. Upišite slovo `s`. Slova `pos` će biti osvetljena u reči `posao`. Sada upišite slova `a` i `o` i statusna traka editora koda će glasiti `Searching for: posao` i reč `posao` će biti istaknuta. Čestitam, našli ste ono što ste tražili!
8. Pritisnite taster Esc, ili Enter, kako biste prekinuli inkrementirano pretraživanje. Zatvorite tekst datoteku i nemojte je snimiti.

To bi bilo sve. Inkrementirano pretraživanje je zgodno kada tražite mali deo teksta.

SAVET Ukoliko napravite grešku prilikom kucanja karaktera u toku inkrementiranog pretraživanja, možete koristiti taster Backspace (brisanje unazad) da biste uklonili poslednji karakter koji ste upisali u string za pretraživanje.

Pronalaženje odgovarajućih običnih i uglastih zagrada

Editor koda ima mogućnost da Vam pomogne u pronalaženju običnih i uglastih zagrada koje odgovaraju običnim i uglastim zagradama na kojima se kursor trenutno nalazi. Da bi pronašli odgovarajuću zagradu, postavite kursor na nju (nema veze da li je otvorena, ili zatvorena zagrada). Zatim pritisnite `Alt+[` na tastaturi. Kursor će preći na zagradu koja odgovara zagradi na kojoj ste se prethodno nalazili.

Pritisnite tastere `Alt+[` i kursor će se vratiti na mesto odakle ste pošli. Isti tasteri rade i sa običnim i sa uglastim zagradama. Uvek postoji mogućnost da se izgubite u lavirintu običnih i uglastih zagrada, ali sada bar imate mogućnost da rešite problem.

Meni sadržaja editora koda

Kao i većina prozora na koje ste naišli u okviru Delphi-ja, editor koda sadrži sopstveni meni sadržaja. Meni sadržaja editora koda se u osnovi može podeliti na dva dela: opcije editora i opcije debagera. Opcije debagera će biti obrađene u lekciji sutrašnjeg dana, kada će biti obrađeno i debagiranje, a u ovom delu ću objasniti opcije editora menija sadržaja. Tabela 9.4 sadrži listu opcija menija sadržaja koje se odnose na editor, pored kojih je dat i opis.

Tabela 9.4: Meni sadržaja editora koda sa opcijama

Opcija	Opis
Close Page	Zatvara aktivnu stranicu u prozoru za editovanje. Ukoliko je datoteka na stranici menjana nakon poslednjeg snimanja, bićete upitani da li želite da snimite datoteku.
Open File At Cursor	Otvara datoteku koja se nalazi ispod kursora. Ova opcija ima efekta samo ukoliko tekst na kom se nalazi kursor predstavlja datoteku izvornog koda. Na primer, ukoliko imate datoteku pod nazivom `MyUnit2`, u okviru Vaše liste `uses`, možete kliknuti na naziv datoteke i odabrati ovu opciju menija kako bi otvorili datoteku. Datoteka će bitipostavljena u novi prozor editora i fokus će biti podešen na prozor.
New Edit Window	Otvara novu kopiju editora koda. Zgodno je ukoliko želite da uporedite dve izvorne datoteke jednu pored druge.
Browse Symbol At Cursor	Pokreće brouzer sa tekućim simbolom kao uslovom za listanje.
Topic Search	Prikazuje pomoć za opciju na kojoj se nalazi kursor (ukoliko pomoć može biti pronađena). Isto kao i kod pritiska na funkcijski taster F1.
Add to Interface	Dodaje karakteristiku, funkciju, ili proceduru ActiveX komponenti. Ova opcija je deaktivirana, ukoliko projekt nije ActiveX projekt.
Toggle Bookmarks	Uključuje i isključuje oznake (od 0 do 9). Oznake se postavljaju u liniju izvornog koda u okviru kog se nalazi kursor za editovanje.
Goto Bookmarks	Pomera editor izvornog koda na oznaku.
View As Form	Ukoliko je aktivna datoteka editora koda prikazana kao sadržaj forme u obliku teksta, izbor ove opcije će ponovo prikazati formu u okviru diza jnera forme.
Read Only	Prebacuje trenutno aktivnu datoteku između moda samo čitanje (read-only) i čitanje/pisanje (read/write). Kada je mod podešen samo na čitan je, datoteka ne može biti izmenjena, mada odabrani tekst može biti prekopiran na Clipboard. Na statusnoj traci će biti prikazan tekst `Read Only`, kako bi označio da je datoteka predviđena samo za čitanje. Kada je datoteka zatvorena i ponovo otvorena vraća se u mod čitanje/pisanje.
Massage View	Prikazuje, odnosno sakriva Delphi-jev prozor za poruke. Prozor za poruke se automatski prikazuje, kada se pojave greške prilikom prevođenja, ili povezivanja; na ovaj način, prozor za poruke se može izričito ili prikazati, ili ukloniti.
View Explorer	Postavlja fokus na prozor za ispitivanje koda. Ukoliko prozor za ispiti vanje koda nije usidren, ovom komandom se prebacuje ispred svih ostal ih prozora.
Properties	Prikazuje okvir za dijalog opcija okruženja, tako da opcije editora mogu biti podešene.

U zavisnosti od trenutnog stanja editora koda i trenutno otvorene datoteke, neke opcije u tabeli 9.4 mogu u određenom trenutku biti deaktivirane.

Promena opcija editora

Opcije editora zauzimaju tri kartice okvira za dijalog opcije okruženja. Da biste videli ovaj okvir za dijalog odaberite opciju Tools➥Environment Options u okviru glavnog menija.

Takođe možete odabrati opciju Properties u okviru menija sadržaja editora koda da biste videli opcije editora. Razlika između ova dva metoda je što se korišćenjem ove opcije pojavljuju samo kartice koje se odnose na opcije editora. Ukoliko se okvir za dijalog pozove na ovaj način, imaće naslov karakteristike editora (Editor Properties) umesto opcije okruženja (Environment Options). Slike u narednim poglavljima su uzete korišćenjem ovog metoda.

Četiri kartice opcija okruženja koje se odnose na editor koda su:

- Editor
- Display (ekran)
- Color (boja)
- Code Insight (izgled koda)

Istražimo ove kartice u narednim poglavljima.

Kartica editora (Editor)

Kartica editora okvira za dijalog karakteristike editora Vam omogućava da kontrolišete kako će editor raditi za Vas. Kao što možete videti sa slike 9.19, na ovoj kartici se nalazi mnogo opcija.

Slika 9.19
Kartica editora u okviru za dijalog karakteristike editora

Na početku kartice se nalazi kombo okvir sa natpisom Editor SpeedSetting (podešavanje prečica editora). Možete odabrati opcije Default Keymapping, IDE Classic, BRIEF Emulation, ili Epsilon Emulation (generičke prečice, prečice klasičnog okruženja, emulacija editora BRIEF, ili Y emulacija) u kombo okviru. Ukoliko promenite podešavanje u ovom kombo okviru, opcije editora će se promeniti i editor će imati nove prečice tastature.

Ukoliko ste početnik u programiranju, ili ukoliko ste ranije koristili neki drugi Borlandov prevodilac koristeći generičko mapiranje testera, ne morate da brinete o stvarima koje propuštate. Ukoliko ste se prilagodili nekom od posebnih tipova editora, biće Vam drago da saznate da još uvek možete da koristite prečice i opcije editora koje znate i volite, menjajući sadržaj polja na kartici Editor SpeedSetting i na kartici Display.

Do kraja kartice, videćete polja Block Indent (uvlačenje bloka) i Tab Stops (tabulator). Možete koristiti ova dva polja da biste promenili broj mesta za koja će kod biti uvučen kada uvlačite blok, odnosno tabulator za sledeći pritisak na taster Tab. Uvlačenje bloka je bilo obrađeno u prethodnom delu lekcije u poglavlju, "Isticanje teksta".

Pravi programeri koriste tabulatore na dva, ili tri karaktera. (Ja koristim tabulatotore koji uvlače tekst za dva karaktera.)

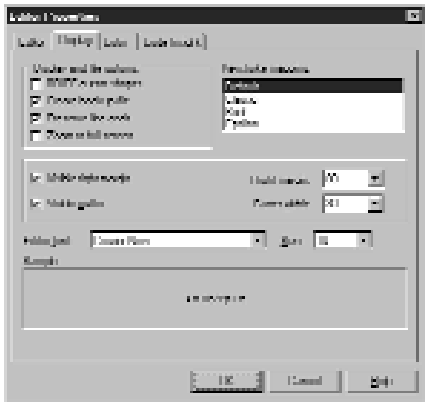
Undo Limit (ograničenje poništavanja prethodne operacije) ima vrednost 32.767, što je verovatno dovoljno za većinu potreba (nadam se!), pa sumnjam da ćete imati potrebu da izmenite ovu opciju. Polje Syntax Extensions Vam omogućava da odaberete tipove datoteka za koje će označavanje sintakse važiti. Na primer, verovatno nećete želeti da označavanje sintakse važi za regularne tekst datoteke (`.txt`), koje možete otvoriti u editoru koda, pa ovaj tip generički nije definisan.

U srednjem delu kartice editora ćete pronaći veliku grupu opcija editora koje možete birati. Pošto je dostupno mnogo opcija i pošto je određivanje koje su dostupne opcije najinteresantnije, veoma teško, pozvaću se na Delphi-jev sistem za pomoć u toku rada. Dok ste na ovoj kartici pritisnite taster F1, odnosno kliknite mišem na dugme Help i dobićete objašnjenja za sve opcije editora koje možete videti na ovoj kartici. Kao što je bio slučaj sa drugim opcijama koje ste danas mogli videti, verovatno ćete prihvatiti Delphi-jeve generičke opcije.

Jedna od stvari koje ću dodati je vraćanje na traženje teksta koji se nalazi na poziciji kursora (suprotno od definisane opcije na slici 9.19). Kada je ova opcija aktivirana, tekst na kom se nalazi kursor se automatski upisuje u polje Text to Find okvira za dijalog za traženje teksta u slučaju da se pozove ovaj okvir za dijalog. Za mene je pretraživanje teksta puno brže ukoliko ne moram da kucam tekst koji tražim. Ovo je posebno zgodno kada se koristi opcija traženja po datotekama.

Kartica Display (ekran)

Kartica Display okvira za dijalog opcije okruženja ima dodatne opcije koje možete da birate. Ove opcije se odnose na prikazivanje teksta u prozoru editora koda (videti sliku 9.20).



Slika 9.20
Kartica Display

U odeljku opcije ekrana i datoteka (Display and File Options), možete pronaći opciju BRIEF Cursor Shapes (oblici kursora slični editoru BRIEF). Aktivirajte ovu opciju ukoliko želite da editor ima horizontalni, umesto vertikalnog kursora. Aktivirajte opciju za kreiranje bekap datoteke (Create Backup File), ukoliko želite da Delphi kreira bekap datoteku svaki put kada snimite Vašu datoteku u okviru projekta. Produžetak bekap datoteke počinje znakom tilda (~). Na primer, bekap datoteka za izvornu datoteku pod nazivom `MyApp.pas` će biti `MyApp. ~pa`.

Obično sam pretrpan svim bekap datotekama koje se nalaze u mom direktorijumu projekta i stoga uvek isključujem opciju za bekap. Odaberite ono što Vam odgovara.

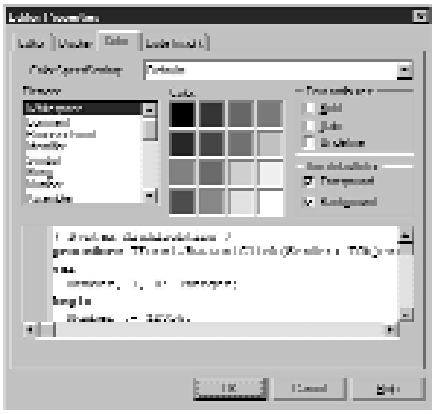
Opcija Zum to Full Screen (zumiranje na kompletan ekran) kontroliše kako će se editor koda ponašati kada bude maksimiziran. Ukoliko je ova opcija uključena, editor koda će u tom slučaju popuniti kompletan ekran. Ukoliko je ova opcija isključena (generički), gornji deo prozora editora koda će se nalaziti priljubljen za donji deo glavnog prozora Delphi-ja, prilikom maksimiziranja ovog prozora. Drugim rečima, Delphi-jev glavni prozor će uvek biti vidljiv kada je editor koda maksimiziran, a ova opcija isključena.

Možete takođe odabrati da li će u Vašim prozorima editora biti vidljiva desna margina. Desna margina ne predstavlja ograničenje - možete još uvek da kucate tekst iza nje - ali će Vam dati vizuelnu oznaku ukoliko Vaša programska linija bude suviše duga. Ovaj odeljak Vam takođe omogućava da odredite da li želite da se vidi žljeb i koje će veličine biti (u pikselima).

Možete takođe menjati veličinu i tip fonta editora koda. Za to postoji kombo okvir iz kog možete birati ove opcije. Prikazani su samo ekranski fontovi sa fiksnim razmakom; proporcionalni i fontovi štampača nisu prikazani. Odaberite tip i veličinu fonta koji Vam najbolje odgovara. Prozor za pregled možete koristiti da biste videli kako izgleda font koji ste trenutno odabrali.

Kartica Color (boja)

Kartica za boju okvira za dijalog opcija okruženja Vam omogućava da potpuno prilagodite prozor editora koda i opcije označavanja sintakse (videti sliku 9.21).



Slika 9.21
Kartica za boju okvira za dijalog karakteristike editora

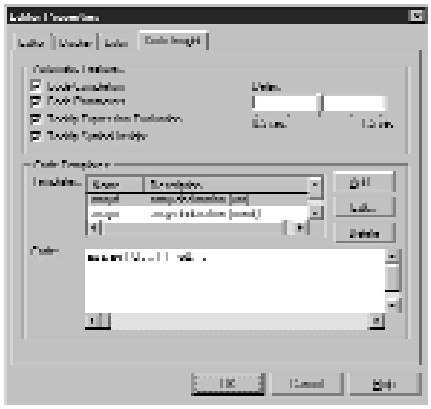
U gornjem delu kartice se nalazi kombo okvir Color SpeedSetting (brzo podešavanje boja). Ovaj kombo okvir Vam omogućava da definišete jednu od četiri unapred definisane šeme boja. Možete odabrati jednu od šema boja kao osnovu za kreiranje Vaše šeme boja.

Karticu za boju je veoma lako koristiti. U dnu stranice se nalazi tekst prozor koji sadrži primer koda. Ukoliko kliknete na ključne elemente koda, u okviru za listu u kom se nalaze elementi će biti odabran traženi element, a njegova trenutna boja će biti prikazana u delu kartice sa bojama.

Da biste promenili boju teksta, pozadine i atribute teksta za određeni element, odaberite boje koje Vam se sviđaju. Na primer, ključne reči su ispisane podebljanim tekstom sa crnom bojom slova i belom pozadinom (podrazumeva se generička šema boja). Da biste promenili ključne reči u zeleni podebljani tekst, kliknite na ključnu reč procedure u prozoru sa primerom koda, a zatim promenite boju slova u zelenu. Boje teksta u prozoru sa primerom će se promeniti kako bi odrazile novu boju koju ste odabrali. Nastavite sa promenama boja sve dok ne dobijete onakav prozor kakav ste želeli. Kada kliknete na dugme OK, editor koda će promeniti boje onako kako ste ih odabrali.

Kartica izgleda koda (code insight)

Kartica izgleda koda se koristi da omogući, ili isključi kompletiranje koda, parametre koda, oblačiće za savet za izračunavanje izraza (alat debagera), kao i izgled simbola oblačića za savet. Kazaljka sa oznakom Delay (odlaganje) se koristi za podešavanje vremena odlaganja, to jest, vremena koje će proteći pre nego što se pojavi aktivirana opcija u oblačiću za savet. Ova kartica Vam takođe omogućava da dodate, editujete, ili obrišete obrasce koda. Slika 9.22 prikazuje karticu izgleda koda okvira za dijalog opcije okruženja.



Slika 9.22
Kartica izgleda koda okvira za dijalog karakteristike editora

Dugme Add (dodavanje) Vam omogućava da dodate novi obrazac koda. Kada kliknete na dugme Add, biće prikazan okvir za dijalog dodavanje koda (Add Code). Upišite naziv obrasca, njegov opis, a zatim kliknite na dugme OK. Sada upišite kod za obrazac u memo polje Code. Kod, takođe, možete zalepiti sa Clipboard-a. Možete dodati onoliko obrazaca koliko želite. Kada kliknete na dugme OK u okviru za dijalog opcije okruženja, u datoteku obrazaca će biti dodat novi obrazac. Ukoliko želite da se kod ne snimi u datoteku obrazaca, kliknite na dugme Cancel.

SAVET Datoteku obrazaca koda možete editovati u Delphi-jevom editoru koda, ukoliko Vam je okvir za dijalog izgleda koda suviše mali. Datoteka obrasca koda ima naziv `DELPHI32.DCI` i nalazi se u direktorijumu `Delphi 4\Bin`.

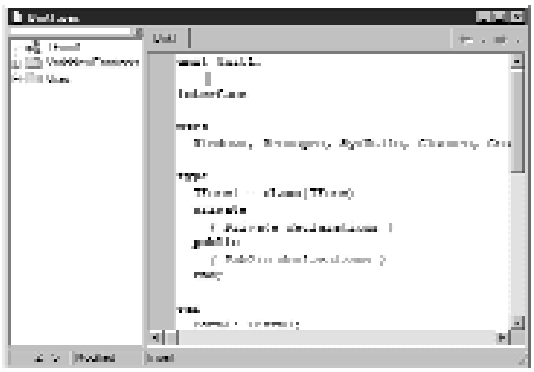
Da biste obrisali obrazac koda, prvo odaberite obrazac u okviru tabele obrazaca, a zatim kliknite na dugme Delete. Ukoliko greškom obrišete obrazac, kliknite na dugme Cancel okvira za dijalog opcije okruženja i Vaše izmene datoteke obrazaca koda neće biti snimljene.

Dugme Edit Vam omogućava da promenite naziv i opis obrasca koda. Kada kliknete na dugme Edit, biće prikazan okvir za dijalog Edit Code Template (editor obrasca koda). Ovaj okvir za dijalog je potpuno isti kao okvir dijaloga za dodavanje obrasca koda. Upišite novi naziv, ili opis obrasca, a zatim kliknite na dugme OK. Kao što je to slučaj sa dodavanjem i brisanjem obrazaca, izmene koje uradite kod obrazaca neće

biti snimljene sve dok ne kliknete na dugme OK u okviru za dijalog opcije okruženja. Da biste editovali kod za obrazac, izmenite ga u memo polju Code.

Prozor za ispitivanje koda (Code Explorer)

Prozor za ispitivanje koda je nova opcija koja je najbolje prihvaćena kao dodatak Delphi-jevom okruženju. Kako mu sam naziv govori, prozor za ispitivanje koda se koristi da brzo pretraži Vaše junite u okviru izvornog koda. Prozor za ispitivanje koda je normalno usidren na levoj strani editora koda. Kada prvi put pokrenete Delphi, editor koda i Prozor za ispitivanje koda će se pojaviti upravo onako kako je to prikazano na slici 9.23.



Slika 9.23
Prozor za ispitivanje koda za novi projekt

Prozor za ispitivanje koda prikazuje sve klase, funkcije, procedure i listu uses za određeni junit. Nod klase se širi kako bi prikazao sve karakteristike, promenljive, polja i metode određene klase.

Prozor za ispitivanje koda prikazuje strukturu junita koji se nalazi u aktivnom prozoru editora koda. Kada se prebacite na junite u editoru koda, Prozor za ispitivanje koda se menja, kako bi prikazao novi junit.

Meni sadržaja prozora za ispitivanje koda

Kao i mnogi Delphi prozori, prozor za ispitivanje koda ima meni sadržaja. Tabela 9.5 prikazuje opcije menija sadržaja prozora za ispitivanje koda.

Tabela 9.5: Opcije menija sadržaja prozora za ispitivanje koda

Opcija	Opis
New	Dodaje novu promenljivu, metodu, funkciju, ili proceduru junitu. Takođe se koristi za dodavanje junita na listu uses. Možete koristiti taster Insert na tastaturi za unos nove opcije.
Rename	Menja naziv identifikatora (promenljiva, metoda, funkcija, procedura, itd.). Takođe možete da koristite ovu opciju za promenu naziva identifikatora na mestu editovanja.
NewView Editor	Ukoliko je prozor za ispitivanje koda usidren na prozor editora koda, postavlja prozor editora koda ispred svih prozora.
Dockable	Određuje da li se mogu usidriti drugi prozori na prozor za ispitivanje koda.

Opcija menija Insert je veoma moćna. Ova opcija Vam omogućava da dodate proceduru, funkciju, metodu klase, odnosno promenljivu, ili junitu, ili klasi u okviru junita. Ovo će biti obrađeno nešto kasnije.

Navigacija junitom

Da biste došli do određene metode, funkcije, ili procedure, dva puta kliknite na naziv identifikatora u okviru prozora za ispitivanje koda. Editor koda se prebacuje na mesto u okviru izvornog koda gde se metoda nalazi. Da biste pronašli polje podataka u klasi, ili dekleraciji promenljive junita, pronađite identifikator promenljive u prozoru za ispitivanje koda i dva puta kliknite mišem na identifikator. Editor koda će prikazati dekleraciju promenljive.

Dodavanje koda korišćenjem prozora za ispitivanje koda

Prozor za ispitivanje koda se može koristiti za dodavanje metoda i dekleraciju promenljivih u Vaš izvorni kod. Na primer, da biste dodali promenljivu polja klasi odaberite opciju Insert u okviru menija sadržaja prozora za ispitivanje koda i upišite dekleraciju za promenljivu koju želite da dodate. Na primer, da biste dodali celobrojnu promenljivu pod nazivom X možete upisati:

```
X : Integer;
```

Kada pritisnete taster Enter na tastaturi, promenljiva će biti dodata klasi.

Metode možete dodati klasi i na jednostavniji način. Da biste videli kako ovo radi, pratite korake vežbe:

1. Pokrenite novu aplikaciju i pređite na prozor editora koda.
2. Kliknite desnim tasterom miša na nod `TForm1` u okviru prozora za ispitivanje koda i odaberite opciju New u okviru menija sadržaja (odnosno pritisnite taster Insert na tastaturi).

3. Upišite sledeći kod u okvir za editovanje prozora za ispitivanje koda, a zatim pritisnite taster Enter na tastaturi:

```
procedure Test;
```

Delphi dodaje nod pod nazivom Public u okviru noda klase `TForm1`. Raširite nod `Public`. Videćete na listi proceduru `Test`.

4. Desnim tasterom miša kliknite na klasu `TForm1` i ponovo odaberite opciju New. Upišite sledeći kod u okvir za editovanje, a zatim pritisnite taster Enter:

```
function GetSomething : Byte;
```

5. Unesite još jedan element. Ovaj put upišite:

```
AVariable : Integer;
```

Verovatno niste primetili, ali u toku dodavanja elemenata, Delphi je bio zauzet menjanjem junita. Listing 9.1 prikazuje junit nakon završetka ove vežbe.

Listing 9.1: Junit nakon dodavanja elemenata u prozor za ispitivanje koda

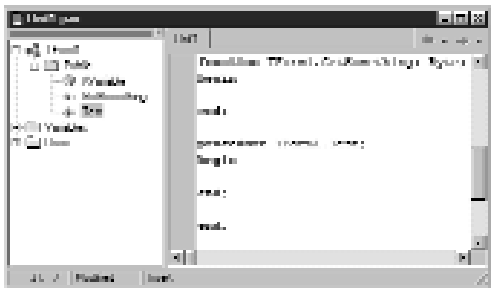
```
unit Unit1;

interface

uses
  Windows, Messages, SysUtils, Classes,
  Graphics, Controls, Forms, Dialogs;

type
  TForm1 = class(TForm)
  private
    { Private declarations }
  public
    AVariable: Integer;
    procedure Test;
    function GetSomething: Byte;
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

function TForm1.GetSomething: Byte;
begin

end;
```

nastavlja se

Listing 9.1: Junit nakon dodavanja elemenata u prozor za ispitivanje koda *nastavak*

```
procedure TForm1.Test;
begin

end;

end.
```

Uočite da nisu samo opcije koje ste dodali prisutne u dekleraciji klasa, nego i osnove metoda koje su kompletirane u odeljku `implementation`. Prozor za ispitivanje koda radi zajedno sa kompletiranjem klasa da bi Vam olakšao posao programiranja. Slika 9.24 prikazuje editor koda i Prozor za ispitivanje koda nakon završetka ove vežbe.



Slika 9.24
Prozor za ispitivanje koda prikazuje nove metode i polje podataka

Prozor za ispitivanje koda je i pretraživač junuita i alat za proizvodnju. Možete ga koristiti za administriranje junitima u toku pisanja koda. Po meni je veoma koristan za svrhu pisanja koda. Takođe možete koristiti Prozor za ispitivanje koda da biste dodavali kod u Vaše junite. Prozor za ispitivanje koda je moćan alat za ubrzavanje produktivnosti. Glavna prednost je što se ovaj alat veoma lako koristi.

Opcije prozora za ispitivanje koda

Opcije za prikazivanje prozora za ispitivanje koda su prikazane na kartici Explorer okvira za dijalog opcije okruženja (opcija Tools➥Environment Options) u okviru glavnog menija. Slika 9.25 prikazuje karticu Explorer.

Različite opcije na ovoj kartici kontrolišu ponašanje prozora za ispitivanje koda, odnosno koliko informacija će prikazivati. Pogledajte Delphi-jev sistem za pomoć vezan za karticu Explorer za deteljniji opis ovih opcija.

Slika 9.25
Kartica Explorer okvira za dijalog opcije okruženja

Zaključak

Danas je predstavljena lekcija u kojoj ste naučili dosta o mogućnostima koje se često previđaju. Nadam se da ste pronašli neke savete koji će Vam biti od koristi u radu sa Delphi-jevim projektima i Delphi-jevim editorom koda. Takođe imate objašnjenje čemu služe pojedine opcije projekta i editora. Čak i da Vam trenutno sve ovo ne izgleda potpuno jasno, na ovu lekciju se možete vratiti kasnije. Lekciju smo završili pregledom prozora za ispitivanje koda. Prozor za ispitivanje koda je fantastičan alat, stoga treba da provedete neko vreme kako biste naučili da ga koristite. Kada počnete da koristite Prozor za ispitivanje koda više nećete biti u mogućnosti da radite bez njega.

Radionica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstirte razumevanje obrađenog materijala, kao i vežbe koje Vam omogućavaju da steknete iskustvo u korišćenju gradiva koje ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A, "Odgovori na kviz pitanja".

Pitanja i odgovori

P **Da li moram da koristim projektne grupe, čak i ako imam samo jedan projekat?**

O Ne. Ne morate da koristite projektne grupe za jedan jedini projekat. Umesto toga možete da koristite generičku projektnu grupu.

P **ada pokrenem svoju aplikaciju, jedan od mojih okvira za dijalog je prikazan umesto glavne forme. Šta se dogodilo?**

O Verovatno ste greškom namestili da glavna forma aplikacije bude forma dijaloga. Pređite na okvir za dijalog opcije projekta, kliknite na jezičak kartice Forms i odaberite glavnu formu za projekt u kombo okviru Main Form u gornjem delu kartice. Ponovo pokrenite Vaš program i glavna forma će biti prikazana onako kako ste očekivali.

P **Sve ove opcije prevodioca i programa za povezivanje me zbunjuju. Da li je potrebno da znam svaku od ovih opcija da bih pisao programe u Delphi-ju?**

O Ne. Generičke opcije projekta rade dobro sa skoro svim Delphi aplikacijama. U jednom trenutku ćete morati da uđete dublje u misterije prevodioca i programa za povezivanje, a od tog trenutka ćete morati i da naučite više o opcijama projekta. Do tada nemojte da brinete.

P **Kada je moja aplikacija minimizirana, ikona i natpis ne odgovaraju onome što piše u glavnoj formi moje aplikacije. Zašto?**

O Podešavanje ikone i natpisa glavne forme ne utiče na prikaz Vaše aplikacije kada je minimizirana. Da biste postavili natpis i ikonu aplikacije, pređite u okvir za dijalog opcije projekta, odaberite karticu Application i dodelite aplikaciji natpis i ikonu.

P **Da li mogu da pronađem i izmenim naziv aplikacije u svim datotekama izvornog koda moje aplikacije?**

O Ne. Morate otvoriti svaku datoteku izvornog koda i pokrenuti okvir za dijalog Replace u okviru svake datoteke izvornog koda. Takođe možete koristiti taster F3 da ponovite poslednju komandu za pronalaženje i zamenu. Zapamtite da nazive promenljivih koje je Delphi generisao ne smete menjati.

P **Da li mogu otvoriti nekoliko datoteka izvornog koda istovremeno u editoru koda?**

O Da. Okvir za dijalog Open podržava izbor više datoteka. Takođe možete odabrati više datoteka u programu Windows Explorer i prebaciti ih u editor koda.

P **Otkrio sam da 32.767 nivoa za poništavanje poslednje komande nisu dovoljni za moje potrebe. Šta predlažete?**

O Obavezno, kada završite rad, snimite sve što ste u toku dana uradili.

Kviz

1. Kako možete brzo preći između forme junita i izvorne datoteke kada radite u Delphi-ju?
2. Ukoliko uklonite datoteku iz projekta koristeći menadžer projekta, da li uklanjate i datoteku sa Vašeg hard diska?

3. Kako možete da podesite glavnu formu aplikacije?
4. Šta znači ako nemate Delphi-jeve forme koje se automatski kreiraju?
5. Kako možete dodavati nove elemente u Vaše junite koristeći prozor za ispitivanje koda?
6. Koji je značaj generisanja informacija za debagiranje u slučaju Vaše aplikacije?
7. Zašto se koristi opcija Find in Files?
8. Koja je prečica tastature za snimanje datoteke u editor koda?
9. Kako možete podesiti oznaku u prozoru editora? Koliko oznaka možete najviše imati?
10. Kako možete podesiti da datoteka bude dostupna samo za čitanje (read-only) u okviru editora koda?

Vežbe

1. Kreirajte novu aplikaciju. Prikažite menadžer projekta. Kliknite na dugme Add Unit (dodavanje junita) da biste dodali novi junit u projekt. Pređite u direktorijum `\Demos\Coolstuf` i odaberite datoteku pod nazivom `main.pas`. Kliknite na dugme OK, kako biste dodali datoteku u projekat.
2. Uklonite junite `main.pas` iz projekta u vežbi jedan.
3. Otvorite projekast `ScratchPad`. Promenite glavnu formu tako da forma `AboutBox` postane glavna forma. Zatvorite okvir za dijalog opcije projekta i pokrenite program. Okvir za dijalog About će biti prikazan prilikom pokretanja programa. Zatvorite okvir za dijalog About, kako biste izašli iz programa i vratite formu `ScratchPad` kao glavnu formu projekta.
4. Kreirajte novu aplikaciju. Snimite projekt i projektnu grupu. Sada dodajte novi projekat projektnoj grupi.
5. Otvorite bilo koju datoteku izvornog koda u editoru koda. Postavite četiri oznake na slučajna mesta u okviru izvornog koda. Prelazite sa jedne oznake na drugu i posmatrajte efekte u editoru koda. Kada završite, obrišite sve oznake.
6. Otvorite projekt `ScratchPad` (ili bilo koji drugi projekt) i pređite u editor koda. Pogledajte datoteku izvornog koda glavnog projekta. Odaberite opciju Search➥Find u okviru glavnog menija. Upišite reč Click u okvir za pronalaženje teksta, a zatim kliknite na dugme OK, kako biste pronašli prvu pojavu reči Click.
7. Pritisnite taster F3 nekoliko puta, kako biste ponovili pretraživanje, sve dok ne bude bila pretražena cela datoteka.

8. Nastavljajući rad na istom projektu, pritisnite tastere Ctrl+Home da biste se vratili na početak datoteke. Pritisnite tastere Ctrl+R da biste prikazali okvir za dijalog Replace Text. Upišite u okvir za pronalaženje teksta reč `Click`, a u okvir za izmenu `Test`. Isključite opciju Prompt on replace (upit za zamenu), a zatim kliknite na dugme Replace All. Pregledajte kompletnu datoteku kako biste videli rezultate.

   VAŽNO: Odaberite opciju Edit➥Undo da biste poništili prethodnu operaciju. Zatvorite projekt i nemojte ga snimiti (da bi bili sigurni).

9. Otvorite datoteku u editoru koda. Odaberite opciju Properties iz menija sadržaja editora koda. Promenite oznaku sintakse za stringove, cele brojeve i brojeve u pokretnom zarezu na tamno sivu boju. Kliknite na dugme OK kako bi pogledali rezultate u editoru koda.

10. Promenite boje u generičku šemu boja.

Dan 10

Debagiranje Vaših aplikacija

Glavna odlika Delphi-jevog okruženja je integrisani debager. Debager Vam omogućava da jednostavno podesite tačke prekida (breakpoints) promenljive koje ćete pratiti, nadgledati objekte i još mnogo toga. Korišćenjem debagera, brzo možete otkriti šta se dešava (ili ne dešava) sa Vašim programom u toku rada. Dobar debager je vitalni element za efikasan razvoj programa.

Debagiranje je posao koji se može veoma lako prevideti. Nemojte reći nikome, ali kada sam počeo da pišem Windows programe, dugo sam ignorisao debager, pošto sam imao pune ruke posla oko učenja programiranja na Windows-ima. Kada sam otkrio kolika je vrednost dobrog debagera, osećao sam se pomalo glupo što sam se zavaravao i nisam dugo koristio ovaj alat. Čovek se uči dok je živ. Imate tu sreću da učite iz mojih grešaka. Danas ćete učiti o tome šta debager može da učini za Vas.

Debager u okviru okruženja pruža nekoliko različitih mogućnosti i alata koji Vam pomažu da obavite posao debagiranja. U današnjoj lekciji ćemo obraditi:

- Opcije menija debagera.
- Korišćenje tačaka prekida (breakpoints).
- Praćenje promenljivih korišćenjem liste za pregled (Wastch List).
- Praćenje objekata korišćenjem debager inspektora (Debug Inspector).
- Ostale alate za debagiranje.
- Prolazak kroz kod korak po korak.

4 Tehnike debagiranja.

Zašto koristiti debager?

Kratak odgovor bi glasio: Debager Vam pomaže u pronalaženju grešaka u programima. Proces debagiranja nije samo pronalaženje i popravljanje grešaka - to je takođe alat za razvoj. Iako je debagiranje važno, mnogi programeri nemaju vremena da nauče kako da koriste mogućnosti okruženja debagera. Kao rezultat sve ovo ih košta puno vremena i novca, da ne spominjemo frustraciju koju može prouzrokovati greška u programu koju je teško pronaći.

Proces debagiranja počinjete startovanjem programa pod debagerom. Kada kliknete mišem na dugme Run u okviru trake sa alatima, automatski koristite debager. Isto tako možete koristiti opciju Run➥Run u okviru glavnog menija, ili pritisak na funkcijski taster F9 tastature.

Opcije menija za debagiranje

Pre nego što detaljnije obradimo debager, pregledaćemo opcije menija koje se odnose na debager. Neke od ovih opcija se nalaze u okviru glavnog menija ispod opcije Run, dok se ostale nalaze u okviru menija sadržaja editora koda. Tabela 10.1 prikazuje opcije menija sadržaja editora koda koje se odnose na debager; uz svaku opciju je dato objašnjenje.

Tabela 10.1: Opcije za debagiranje menija sadržaja editora koda

Opcija	Prečica	Opis
Toggle Breakpoint	F5	Uključuje i isključuje tačku prekida za tekuću liniju u okviru editora koda.
Run to Cursor	F4	Pokreće (ukoliko je neophodno) i izvršava program dok ne stigne do linije programa na kojoj se nalazi kursor.
Inspect	Alt+F5	Otvara prozor debager inspektora vezanog za objekat na kom se nalazi kursor.
Goto Address	Ctrl+Alt+G	Omogućava Vam da definišete adresu u okviru programa odakle će se nastaviti izvršavanje programa.
Evaluate/Modify	Ctrl+F7	Omogućava Vam da pregledate i/ili izmenite vrednost promenljive u toku rada programa.
Add Watch at Cursor	Ctrl+F5	Dodaje u listu za pregled (Watch List) promenljivu na kojoj se nalazi kursor.

View CPU	Ctrl+Alt+C	Prikazuje prozor procesora (CPU window).

Opcija Run u okviru glavnog menija ima nekoliko izbora koji zavise od rada programa pod debagerom. Opcije menija Run, Vam omogućavaju da pokrenete program pod debagerom, da prekinete rad programa koji radi pod debagerom, da definišete parametre u okviru komandne linije za Vaš program; da spomenemo samo neke funkcije. Neke opcije koje ste ovde mogli pronaći su duplirane u meniju sadržaja editora koda. Tabela 10.2 prikazuje opcije menija Run koje kontrolišu operacije debagiranja.

Tabela 10.2: Opcije za debagiranje menija Run

Opcija	Prečica	Opis
Run	F9	Prevodi program (ukoliko je neophodno), a zatim pokreće program pod kontrolom okruženja debagera (isto kao dugme Run u okviru trake sa alatima).
Parameters	nema	Omogućava Vam da unesete parametre komandne linije za Vaš program i da dodelite aplikaciju domaćina (host) kada debagirate DLL datoteku.
Step Over	F8	Izvršava liniju izvornog koda od početka i zaustavlja se na sledećoj liniji izvornog koda. Trace IntoF7Prati metodu u trenutku izvršavanja.
Trace to Next Source Line	Shift+F7	Prebacuje tačku izvršavanja na sledeću liniju izvornog koda programa.
Run to Cursor	F4	Pokreće program i zastaje kada izvršavanje programa dostigne tekuću liniju izvornog koda.
Show Execution Point	nema	Prikazuje trenutnu tačku izvršavanja pro grama u editoru koda. Pomera prozor izvornog koda ukoliko je to neophodno. Radi samo kada je zaustavljeno izvršavanje programa.
Program Pause	nema	Zaustavlja izvršavanje programa nakon što se aktivira komanda u okviru izvornog koda programa.
Program Reset	Ctrl+F2	Bezuslovno prekida izvršavanje progra ma i vraća se u Delphi-jevo okruženje.
Inspect	nema	Prikazuje okvir za dijalog Inspect tako da možete uneti naziv objekta koji želite da ispitate.

nastavlja se

Evaluate/Modify	Ctrl+F7	Prikazuje okvir dijaloga Evaluate/Modify (prikazivanje/izmena).

nastavak

Tabela 10.2: Opcije za debagiranje menija Run

Opcija	Prečica	Opis
Add Watch	Ctrl+F5	Prikazuje okvir za dijalog Watch Properties (karakteristike pregleda).
Add Breakpoint	nema	Prikazuje podmeni koji sadrži opcije za dodavanje izvora, adrese, podataka odnosno modula za učitavanje tačke prekida.

Opcije ovog menija ćete veoma često koristiti u toku debagiranja Vaših programa. Bilo bi dobro da se priviknete na različite prečice tastature vezane za operacije debagiranja. Pređimo na tačke prekida i način na koji se tačke prekida koriste u Vašim programima.

Korišćenje tačaka prekida (breakpoints)

Kada pokrenete program iz Delphi-jevog okruženja, program će raditi punom brzinom i zastajati samo na mestima gde ste definisali tačke prekida.

NOVI TERMIN Tačka prekida (*breakpoint*) je marker koji saopštava debageru da zaustavi izvršavanje programa kada se u toku rada dostigne naznačeno mesto.

Postavljanje i uklanjanje tačaka prekida

Da biste postavili tačku prekida kliknite mišem na žljeb prozora editora koda na levoj strani linije koda na kojoj želite da se zaustavi izvršavanje programa (žljeb je siva margina duž leve ivice prozora editora koda). Ikona tačke prekida (crveni kružić) se pojavljuje na žljebu i kompletna linija je istaknuta crvenom bojom. Da biste obrisali tačku prekida, kliknite mišem na ikonu tačke prekida. Takođe, možete pritisnuti funkcijski taster F5, odnosno odabrati opciju Toggle Breakpoint u okviru menija sadržaja editora koda, kako bi uključili, odnosno isključili tačku prekida.

NAPOMENA Tačka prekida se može postaviti samo na liniju koja sadrži aktuelni kod programa. Tačke prekida se ne mogu postaviti na prazne linije, linije sa komentarom, odnosno linije sa dekleracijom. Ukoliko želite da postavite tačku prekida na ove tipove linija nećete biti sprečeni, ali će Vas debager na to upozoriti ukoliko pokušate. Pokušaj postavljanja tačke prekida na jednu od narednih linija će prouzrokovati upozorenje za pogrešno postavljenu tačku prekida:

```
{ This is a comment followed by a blank line. }

X : Integer;  { a declaration }
```

Tačke prekida mogu biti postavljene na iskaz end koji pripada funkciji, ili proceduri.

Ukoliko ste podesili tačku prekida na pogrešnu liniju, editor koda će prikazati tačku prekida zelenom bojom (podrazumeva se generička šema boja), a ikona tačke prekida u žljebu će biti siva.

Kada program bude pokrenut u okviru debagera, ponašaće se normalno - sve dok ne naiđe na tačku prekida. Kada dođe do tačke prekida, okruženje prelazi u prvi plan i linija u okviru izvornog koda koja sarži tačku prekida će biti označena. Ukoliko koristite generičke boje, linija na kojoj će program biti zaustavljen postaje crvena, pošto crvena boja označava liniju koja sadrži tačku prekida.

Tačka izvršavanja (execution point) označava liniju koja u okviru izvornog koda dolazi na red za izvršavanje.

Kako se krećete kroz program, tačka izvršavanja je osvetljena plavom bojom, a žljeb prozora editora prikazuje sličicu zelene strelice. Treba da shvatite da linija koja je označena plavom bojom još nije izvršena, a biće izvršena kada program nastavi sa radom.

Tekuća tačka izvršavanja programa je osvetljena plavom bojom, sve dok se ne dođe do tačke izvršavanja koja sadrži tačku prekida. U tom slučaju, linija je označena crvenom bojom. Zelena strelica na žljebu je najpouzdanija oznaka za izvršavanje programa, nezavisno od boje za osvetljavanje linije.

Kada zastanete na tački prekida, možete pregledati promenljive, pregledati stek poziva, pretražiti simbole, odnosno proći korak po korak kroz kod programa. Nakon što ispitate promenljive i objekte, možete se vratiti normalnom izvršavanju programa klikom miša na dugme Run. Vaša aplikacija će normalno raditi sve dok ne dođe do sledeće tačke prekida.

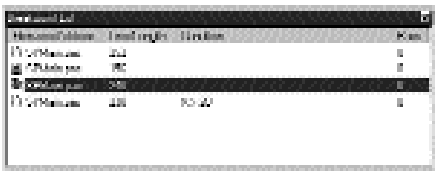
Uobičajeno je da otkrijete greške u pisanju programa nakon dolaska do tačke prekida. Ukoliko promenite izvorni kod programa u toku procesa debagiranja, a zatim odaberete opciju Run da biste nastavili izvršavanje programa, okruženje će Vas upitati da li želite da ponovo prevedete i povežete izvorni kod. Ukoliko odaberete opciju Yes, trenutni proces će biti prekinut, izvorni kod će biti ponovo preveden, a program će biti ponovo pokrenut.

Problem sa ovim pristupom je da Vaš program nema priliku da se završi normalno, a resursi koji se trenutno koriste možda neće regularno biti oslobođeni. Ovaj scenario u krajnjem slučaju vodi do rasipanja memorije. Iako operativni sistemi Windows 95 i Windows NT upravljaju rasipanjem resursa bolje od 16-bitnih Windows-a, još uvek se preporučuje normalno završavanje programa i njegovo prevođenje.

Prozor sa spiskom tačaka prekida

Delphi-jevo okruženje vodi računa o tačkama prekida koje ste postavili. Ove tačke prekida se mogu videti u prozoru sa spiskom tačaka prekida. Da biste videli spisak tačaka prekida, odaberite opciju View➥Debug Windows➥Breakpoints u okviru

glavnog menija. Prozor sa spiskom tačaka prekida će biti prikazan, što možete videti na slici 10.1.



Slika 10.1
Prozor sa spiskom tačaka prekida

Prozor sa spiskom tačaka prekida ima četiri kolone:

- Kolona Filename/Address prikazuje naziv datoteke i izvorni kod junita u okviru kog je postavljena tačka prekida.
- Kolona Line/Length prikazuje broj linije na kojoj se nalazi tačka prekida.
- Kolona Condition prikazuje uslove koji su postavljeni za tačku prekida.
- Kolona Pass prikazuje uslov brojanja prolaza koji je podešen za tačku prekida (uslovi tačke prekida i uslov brojanja prolaza će biti obrađeni u poglavlju "Uslovne tačke prekida".)

Kolonama se može promeniti širina povlačenjem linije za razdvajanje dve kolone u zaglavlju kolona.

NAPOMENA Kolona Pass ne prikazuje broj prolazaka kroz tačku prekida, nego samo uslov za prolazak koji ste postavili za tačku prekida.

Meni sadržaja prozora za spisak tačaka prekida

Prozor za spisak tačaka prekida ima dva menija sadržaja. Tabela 10.3 prikazuje meni sadržaja koji možete videti kada kliknete desnim tasterom miša na bilo koju tačku prekida. Ovaj meni ću nazvati primarni meni sadržaja prozora.

Tabela 10.3: Primarni meni sadržaja prozora spiska tačaka prekida

Opcija	Opis
Enable	Aktivira, odnosno deaktivira tačku prekida. Kada je tačka prekida deaktivirana, oznaka u okviru prozora za spisak tačaka prekida postaje siva. Takođe i u okviru prozora sa izvornim kodom tačka prekida je takođe siva, a linija tačke prekida je osvetljena zelenom bojom da bi označila deaktiviranu tačku prekida.
Delete	Uklanja tačku prekida.
View Source	Pomera izvorni kod programa u prozoru editora koda da bi prikazala liniju izvornog koda koja sadrži tačku prekida. (Prozor sa spiskom tačaka prekida zadržava fokus.)

Edit Source	Postavlja kursor za editovanje na liniju izvornog koda programa gde se nalazi tačka prekida i prebacuje fokus na editor koda.
Properties	Prikazuje okvir za dijalog karakteristike izvora tačke prekida.
Dockable	Određuje da li je prozor za spisak tačaka prekida aktiviran za usidravanje.

Da biste brzo editovali liniju izvornog koda na kojoj se nalazi tačka prekida, dva puta kliknite mišem na tačku prekida u koloni Filename u okviru prozora za spisak tačaka prekida. Potpuno isto kao izbor opcije Edit Source menija sadržaja prozora za spisak tačaka prekida.

Sekundarni meni sadržaja će biti prikazan ukoliko kliknete desnim tasterom miša, dok se kursor nalazi na bilo kom delu prozora za spisak tačaka prekida koji ne sadrži tačku prekida. Ovaj meni sadržaja ima opcije: Add, Delete All, Disable All, Enable All i Dockable (dodavanje, brisanje svih elemenata, deaktiviranje svih elemenata, aktiviranje svih elemenata i mogućnost usidrenja). Ove opcije su dovoljno jasne same po sebi, pa se neću truditi da Vam ih objasnim.

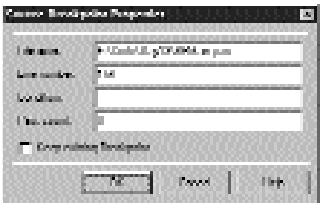
Po mom mišljenju, opcija Add menija sadržaja nije posebno korisna. Mnogo jednostavnije je postaviti tačku prekida u okviru editora koda, umesto dodati tačku prekida korišćenjem komande Add u okviru prozora sa listom tačaka prekida.

Aktiviranje i deaktiviranje tačaka prekida

Tačke prekida možete aktivirati, ili deaktivirati kad god to poželite. Tačke prekida možete deaktivirati ukoliko želite da Vam program na trenutak normalno radi; tačku prekida ponovo možete aktivirati kasnije, a da ne morate da je ponovo kreirate. Debager ignoriše tačke prekida koje su deaktivirane. Da bi aktivirali, odnosno deaktivirali tačku prekida, kliknite desnim tasterom miša na tačku prekida u okviru prozora sa spiskom tačaka prekida, a zatim aktivirajte, odnosno deaktivirajte tačku prekida korišćenjem opcije Enable menija sadržaja.

Izmena tačaka prekida

Ako želite da izmenite tačku prekida, odaberite opciju Properties koja se nalazi u primarnom meniju sadržaja prozora sa spiskom tačaka prekida. Kada odaberete ovu opciju, biće prikazan dijalog za okvir karakteristike izvora tačke prekida (pogledati sliku 10.2).



Slika 10.2
Okvir za dijalog karakteristike izvora tačke prekida (Source Breakpoint Properties)

Osnovni razlog za izmenu tačke prekida je izmena uslova u okviru tačke prekida. (Uslovne tačke prekida će biti obrađene u poglavlju "Uslovne tačke prekida".)

Da biste uklonili tačke prekida, odaberite tačku u okviru prozora sa spiskom tačaka prekida, a zatim pritisnite taster Delete na tastaturi. Da biste obrisali sve tačke prekida, kliknite desnim tasterom miša, a zatim odaberite opciju Delete All. Sada ćemo obraditi dva tipa tačaka prekida: jednostavne i uslovne.

Jednostavne tačke prekida (simple breakpoints)

Jednostavne tačke prekida (simple breakpoints) zaustavljaju izvršavanje programa svaki put kada izvršavanje programa naiđe na liniju koda koja je definisana kao tačka prekida. Kada inicijalno postavite tačku prekida, ona generički postaje jednostavna tačka prekida. Jednostavne tačke prekida ne treba posebno objašnjavati. Kada izvršavanje programa dođe do tačke prekida, program se zaustavlja, a debager čeka Vašu odluku. U većini slučajeva ćete koristiti jednostavne tačke prekida. Uslovne tačke prekida su rezervisane za posebne slučajeve u kojima Vam je potrebno više kontrole nad procesom debagiranja.

Uslovne tačke prekida (conditional breakpoints)

U slučaju uslovnih tačaka prekida (conditional breakpoints), izvršavanje programa se zaustavlja samo ukoliko je ispunjen unapred definisan uslov. Da biste kreirali uslovnu tačku prekida, prvo treba da podesite tačku prekida u editoru koda. Zatim odaberite opciju View➥Debug Windows➥Breakpoints u okviru glavnog menija, kako biste otvorili prozor sa spiskom tačaka prekida. Kliknite desnim tasterom miša na tačku prekida za koju želite da postavite uslov i odaberite opciju Properties. Kada se pojavi okvir za dijalog karakteristike izvora tačke prekida, podesite uslove za tačku prekida.

Uslovne tačke prekida se pojavljuju u dva oblika. Prvi oblik je tačka prekida uslovnog izraza (conditional expression breakpoint). Unesite uslovni izraz u polje uslov okvira za dijalog karakteristike izvora tačke prekida (pogledajte sliku 10.2). Kada se program pokrene uslovni izraz će biti izračunat svaki put kada izvršavanje programa naiđe na tačku prekida. Kada uslovni izraz postane tačan, izvršavanje programa se zaustavlja. Ukoliko uslovni izraz nije tačan, tačka prekida se ignoriše. Na primer,

pogledajte poslednju tačku prekida u okviru prozora sa spiskom tačaka prekida koja je prikazana na slici 10.1. Ova tačka prekida ima uslovni izraz X > 20. Ukoliko je u nekom trenutku izvršavanja programa promenljiva X veća od 20, program će se zaustaviti na tački prekida. Ukoliko promenljiva X nikada nije veća od 20, izvršavanje programa se neće zaustaviti na tački prekida.

Drugi tip uslovne tačke prekida je tačka prekida koja broji prolaske (*pass count breakpoint*). Sa tačkom prekida koja broji prolaske, izvršavanje programa će biti zaustavljeno samo kada se određeni broj puta dostigne tačka prekida. Da biste definisali tačku prekida koja broji prolaske, editujte tačku prekida i definišite vrednost za polje u koje se upisuje broj prolazaka, u okviru za dijalog sa karakteristikama izvora tačke prekida. Ukoliko podesite brojač prolazaka tačke prekida na 3, izvršavanje programa će se zaustaviti nakon trećeg prolaska kroz tačku prekida.

NAPOMENA Brojač prolazaka počinje sa brojanjem od 1, a ne od 0. Kao što je naznačeno u prethodnom primeru, broj prolazaka koji je definisan brojem 3, znači da će tačka prekida biti aktivirana kada izvršavanje programa treći put dostigne tačku prekida.

Brojač prolazaka tačke prekida možete koristiti kada želite da program prelazi preko tačke prekida određeni broj puta, pre nego što se zaustavi da biste pregledali promenljive, izvršavali program korak po korak, odnosno izvršili bilo koji drugi zadatak u okviru debagera.

NAPOMENA Uslovne tačke prekida usporavaju normalno izvršavanje programa pošto je potrebno izračunati uslove svaki put kada se dostigne uslovna tačka prekida. Ukoliko se Vaš program usporava u toku debagiranja, pogledajte listu tačaka prekida da biste videli da li ste zaboravili na neku uslovnu tačku prekida.

SAVET Činjenica da uslovne tačke prekida usporavaju izvršavanje programa u nekim slučajevima Vam može i koristiti. Ukoliko imate neki proces koji želite da pogledate usporeno, podesite jednu, ili više tačaka prekida u delu koda koji želite da pregledate. Postavite uslove koji nikada neće biti ispunjeni, a Vaš program će biti usporen, ali se neće zaustavljati.

Komanda za rad do kursora (Run to Cursor)

Postoji još jedna komanda za debagiranje koja zaslužuje da se spomene. Komanda za rad do kursora (Run to Cursor), koja se može pronaći u meniju Run glavnog menija, kao i u menju sadržaja editora koda, izvršava program sve dok ne naiđe na liniju izvornog koda na kojoj se nalazi kursor. Od te tačke, program se zaustavlja kao da je tačka prekida postavljena na liniju na kojoj se nalazi kursor.

Opcija Run to Cursor se ponaša kao privremena tačka prekida. Ukoliko želite da trenutno ispitate određenu liniju izvršnog koda, možete koristiti ovu komandu, a da ne postavljate tačku prekida. Treba samo da postavite kursor na liniju gde želite da

se zaustavi izvršavanje programa, a zatim da odaberete opciju Run to Cursor (odnosno pritisnete taster F4). Debager će se ponašati potpuno isto kao da ste postavili tačku prekida na liniju izvornog koda. Prednost je što ne morate da brišete tačke prekida nakon što ste završili debagiranje dela koda.

Praćenje promenljivih

Šta ćete raditi kada se zaustavite na tački prekida? Obično se na tački prekida zaustavljate da biste ispitali vrednosti jedne, odnosno više promenljivih. Možda ćete želeti da se uverite da određena promenljiva ima vrednost koju mislite da bi trebala da ima, odnosno možda ne znate koju bi vrednost promenljiva trebala da ima, a to želite da otkrijete.

Funkcije prozora spiska za praćenje (Watch List) su jednostavne: omogućavaju Vam da istražite vrednosti promenljivih. Programeri obično previđaju ovu jednostavnu, ali osnovnu mogućnost, pošto nemaju vremena da nauče kako da u potpunosti iskoriste debager. U listu za pregled možete dodati koliko god želite promenljivih. Slika 10.3 prikazuje listu za pregled u toku procesa debagiranja.

Slika 10.3
Lista za pregled u toku rada

Na listi za pregled je prikazan naziv promenljive iza kog sledi vrednost promenljive. Kako je vrednost promenljive prikazana, određeno je tipom promenljive i trenutnim postavljanjem prikaza opcije koja se posmatra. Prozor za spisak pregleda će biti nešto kasnije obrađen, a prvo ću Vam objasniti opcije koje mogu učiniti pregled promenljivih lakšim.

Oblačić za savet za izračunavanje izraza

Debager i editor koda imaju veoma dobru opciju koja omogućava jednostavnu proveru vrednosti. Ova opcija, oblačić za savet za izračunavanje izraza, je generički aktivirana, pa nije potrebno da uradite ništa posebno da bi mogli da je koristite. Ukoliko želite, možete isključiti oblačić za savet za izračunavanje izraza, koristeći karticu Code Insight okvira za dijalog opcije okruženja (ova kartica je bila obrađena u prethodnoj lekciji).

Šta je oblačić za savet za izračunavanje izraza (iako je to teško reći)? Oblačić za savet radi na sledeći način: kada se zaustavite na tački prekida, postavljate kursor za editovanje na promenljivu i pojavljuje se oblačić za savet koji prikazuje trenutnu vrednost promenljive. Ovo Vam omogućava da lako i jednostavno ispitate promenljive. Samo treba da postavite kursor na promenljivu i sačekate otprilike pola sekunde.

Oblačić za savet za izračunavanje izraza ima različite prikaze za različite tipove promenljivih. Za standardne tipove promenljivih (`Integer`, `Char`, `Byte`, `string`, itd.), biće prikazana aktuelna vrednost promenljive. Za dinamički kreirane objekte (slučajeve klasa, na primer), oblačić za savet za izračunavanje izraza prikazuje poziciju objekta u memoriji. Za slogove, oblačić za savet za izračunavanje izraza prikazuje sve elemente sloga. Slika 10.4 prikazuje oblačić za savet za izračunavanje izraza kojim se trenutno ispituje sadržaj sloga.

Slika 10.4
Oblačići za savet su velika prednost debagera

Ponekad se oblačić za savet za izračunavanje izraza ponaša kao da ne radi ispravno. Ukoliko, na primer, postavite kursor na promenljivu koja je van opsega, neće se pojaviti oblačić za savet. Oblačić za savet za izračunavanje izraza neće imati šta da prikaže za određenu promenljivu, pa stoga neće ništa ni prikazati.

Budite oprezni, pošto promenljive koje je prevodilac optimizovao mogu da ne prikažu ispravnu vrednost. Optimizacija je obrađena u jučerašnjoj lekciji, a biće obrađena i u narednom delu današnje lekcije.

Još jedan slučaj kada oblačić za savet za izračunavanje izraza neće raditi je petlja `with`. Na primer, pogledajte sledeći kod:

```
with Point do begin
  X := 20;
  Y := 50;
  Label1.Caption := IntToStr(X);
end;
```

Ukoliko postavite kursor miša na promenljivu X, oblačić za savet za izračunavanje izraza neće prikazati vrednost promenljive X, pošto promenljiva X pripada ciljnom objektu iskaza with (promenljivoj `Point`). Umesto toga, postavite kursor miša na promenljivu `Point`, pa će debager pokazati vrednost promenljive `Point` (uključujući i polje X).

Oblačić za savet za izračunavanje izraza je odlična opcija, pa ne bi trebali da zaboravite kako da je koristite.

Meni sadržaja liste za pregled

Kao i svaki drugi Delphi-jev prozor koji smo do sada obradili, prozor liste za pregled ima svoj meni sadržaja. (Bili biste razočarani ako ne bi imao, zar ne?) Tabela 10.4 prikazuje opcije menija sadržaja liste za pregled i njihove opise.

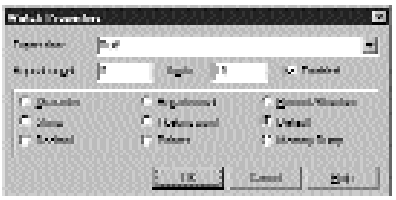
Tabela 10.4: Meni sadržaja liste za pregled

Opcija	Opis
Edit Watch	Omogućava Vam da editujete element za pregled koristeći okvir za dijalog Watch Properties.
Add Watch	Dodaje novi element u listu za pregled.
Enable Watch	Aktivira element za pregled.
Disable Watch	Deaktivira element za pregled.
Delete Watch	Uklanja element za pregled sa liste za pregled.
Enable All Watches	Aktivira sve elemente u prozoru sa spiskom za pregled.
Disable All Watches	Deaktivira sve elemente u prozoru sa spiskom za pregled.
Delete All Watches	Briše sve elemente u prozoru sa spiskom za pregled.
Stay on Top	Određuje da prozor sa spiskom za pregled bude na vrhu svih prozora okruženja.
Break When Changed	Kada se promenljiva u prozoru za pregled promeni, debager će prekinuti rad. Promenljiva koja se posmatra će biti prikazana crvenom bojom, kako bi označila da je ova opcija aktivirana.
Dockable	Određuje da li će prozor sa listom za pregled moći da se usidri.

Obe opcije menija Edit Watch i Add Watch pozivaju okvir za dijalog Watch Properties, pa ćemo ovaj okvir za dijalog obraditi u sledećem poglavlju.

Korišćenje okvira za dijalog Watch Properties (karakterisatike pregleda)

Okvir za dijalog Watch Properties ćete koristiti kada budete želeli da dodate, ili editujete element za pregled. Slika 10.5 pokazuje okvir za dijalog Watch Properties u trenutku kada se edituje promenljiva pod nazivom Buff.



Slika 10.5
Okvir za dijalog Watch Properties

Polje Expression (izraz) u gornjem delu okvira za dijalog Watch Properties je mesto na koje upisujete naziv promenljive za editovanje, ili dodavanje na spisak za pregled. Ovo polje je kombo okvir koji se može koristiti za izbor elemenata za pregled koji su prethodno bili odabrani.

Polje Repeat count (ponavljanje izračunavanja) možete koristiti kada istražujete nizove. Na primer, pretpostavimo da imate niz od 20 celih brojeva. Da biste ispitali prvih 10 celih brojeva u nizu, treba da upišete prvi element niza u polje Expression (na primer, `Array [0]`), a zatim upišete broj 10 u polje Repeat count. Prvih `10` elemenata niza će biti prikazano na spisku za pregled.

Ukoliko dodate samo naziv niza na spisak za pregled, svi elementi niza će biti prikazani. Polje Repeat count koristite samo onda kada želite da prikažete određeni broj elemenata niza.

Polje Digits (cifre) se koristi kada želite da ispitate brojeve u pokretnom zarezu. Upišite broj važnih cifara koje želite da vidite kada se Vaš broj u pokretnom zarezu prikaže na spisku za pregled. Prikazane cifre su zaokružene, nisu odsečene. Još jedno polje u ovom okviru za dijalog je polje Enabled (aktivirano), koje određuje da li će element za pregled trenutno biti aktiviran.

Ostatak okvira za dijalog Watch Properties je formiran tako da definiše različite opcije za prikazivanje. Svaki tip podataka ima generički tip prikaza, koji se koristi za izbor opcije za pregled Default. Generička (Default) opcija za pregled je generička. (Izvinite, ali ne postoji drugi način da se ovo objasni.) Da biste videli podatke na drugi način, odaberite drugu opciju za pregled. Slika 10.6 prikazuje prozor spiska za pregled sa dve promenljive i nekoliko primera opcija za pregled. Promenljiva Buff je karakter niz, a promenljiva I je ceo broj.

Slika 10.6
Prozor spiska za pregled koji prikazuje različite opcije za pregled promenljivih

Da biste izmenili element za pregled, kliknite mišem na element u prozoru spiska za pregled i odaberite opciju Edit Watch u okviru menija sadržaja spiska za pregled. Takođe možete dva puta kliknuti mišem na opciju za pregled ukoliko želite da ga editujete. Biće prikazan okvir za dijalog Watch Properties, pa ćete moći da editujete elemente za pregled ukoliko je to potrebno.

Najbrži način da editujete element za pregled je da dva puta kliknete mišem na njegov naziv u okviru prozora sa spiskom za pregled.

Aktiviranje i deaktiviranje elemenata za pregled

Kao što je to slučaj sa tačkama prekida, zasebni elementi spiska za pregled mogu biti aktivirani, odnosno deaktivirani. Kada je element za pregled deaktiviran, postaje siv i umesto vrednosti, pored elementa će pisati `<disabled>`.

Da biste deaktivirali element za pregled kliknite mišem na naziv elementa u okviru liste za pregled i odaberite opciju Disable Watch u okviru menija sadržaja prozora sa spiskom za pregled. Da biste ponovo aktivirali element za pregled, odaberite opciju Enable Watch u okviru menija sadržaja.

NAPOMENA Možda ćete poželeti da deaktivirate elemente za pregled, ukoliko trenutno ne želite da ih vidite, ali će Vam biti potrebni kasnije. Veliki broj aktiviranih elemenata na spisku za pregled usporavaju izvršavanje programa u toku debagiranja, pošto sve promenljive moraju biti izmenjene nakon izvršavanja svake zasebne linije koda.

Dodavanje promenljivih na listu za pregled

Promenljive na listu za pregled možete dodati na nekoliko načina. Najbrži način je da kliknete mišem na naziv promenljive u okviru prozora editora, a zatim odaberete opciju Add Watch at Cursor u okviru menija sadržaja editora koda, odnosno da pritisnete testere Ctrl+F5. Element za pregled će trenutno biti dodat na spisak za pregled. Ukoliko je potrebno, kasnije možete editovati element za pregled da biste promenili karakteristike prikaza elementa.

Da biste dodali promenljivu na spisak, a da je prethodno ne pronađete u okviru izvornog koda programa, odaberite opciju Run➥Add Watch u okviru glavnog menija. Kada se pojavi okvir za dijalog Watch Properties, upišite naziv promenljive koju želite da dodate u spisak za pregled, a zatim kliknite mišem na dugme OK.

NAPOMENA Iako možete dodati trenutnu promenljivu klase na spisak za pregled, prikazana vrednost najverovatnije neće biti od koristi. Za pregled svih elemenata klase, treba da koristite debager inspektor koji će biti uskoro obrađen.

Korišćenje spiska za pregled

Kada program naiđe na tačku prekida, spisak za pregled prikazuje trenutne vrednosti svih promenljivih koje su dodate na spisak za pregled. Ukoliko spisak za pregled trenutno nije otvoren, možete ga otvoriti korišćenjem opcije View➥Debug Windows➥Watches u okviru glavnog menija.

SAVET Usidrite prozor sa spiskom za pregled na donju ivicu editora koda kako bi uvek bio pristupačan u toku rada.

Pod izvesnim okolnostima može biti prikazana poruka pored promenljive umesto njene vrednosti. Ukoliko je, na primer, promenljiva van opsega, odnosno ako se ne može pronaći, spisak za pregled će pored naziva promenljive prikazati poruku

`Undeclared identifier: 'X'` (Nedeklarisan identifikator: `'X'`). Ukoliko program nije pokrenut, odnosno nije zaustavljen na tački prekida, prozor spiska za pregled će pored svih elemenata za pregled prikazati poruku: `[process not accessible]` (proces nije dostupan). Ukoliko je element za pregled deaktiviran, pored njega će biti prikazana poruka `<disabled>`. Ostale poruke mogu biti prikazane u zavisnosti od trenutnog stanja aplikacije, odnosno trenutnog stanja promenljive.

Slučajno vam se može dogoditi da vidite poruku: `Variable 'X' inaccessible here due to optimization` (Promenljiva `'X'` nije dostupna zbog optimizacije). Ovo je jedan od manjih nedostataka ukoliko koristite prevodilac za optimizaciju. Ukoliko želite da ispitate promenljive koje su optimizovane, morate isključiti optimizaciju. Isključite opciju Optimization na kartici Compiler okvira za dijalog opcije projekta. Pazite da promenljive koje još nisu inicijalizovane (nije im dodeljena vrednost) ne prijave slučajne vrednosti pre no što budu inicijalizovane.

Prozor spiska za pregled se može koristiti kao brzi konvertor vrednosti decimalnih i heksadecimalnih brojeva. Da biste konvertovali heksadecimalni broj u decimalni, odaberite opciju Run➥Add Watch u okviru glavnog menija. Upišite heksadecimalni broj u polje Expression, a zatim kliknite mišem na dugme OK. Na spisku za pregled će biti prikazani i heksadecimalni i decimalni ekvivalent broja. Da biste konvertovali decimalni broj u heksadecimalni, obavite istu proceduru, ali dodatno kliknite mišem na radio dugme Hexadecimal da biste prikazali broj u heksadecimalnom formatu. Pošto polje Expression može prihvatiti i matematički izraz, spisak za pregled možete koristiti i kao heksadecimalni kalkulator. Možete čak i mešati heksadecimalne i decimalne vrednosti u okviru istog izraza. Jedini nedostatak je što Vaša aplikacija mora biti zaustavljena na tački prekida ukoliko želite da ova opcija radi.

Prozor spiska za pregled je jednostavan, ali veoma važan alat za debagiranje aplikacija. Da bi ilustrovali korišćenje prozora spiska za pregled, izvršite sledeću vežbu:

1. Kreirajte novu aplikaciju i postavite dugme na formu. Promenite karakteristiku dugmeta `Name` u `WatchBtn`, a karakteristiku `Caption` u `Watch Test`. Promenite karakteristiku Name forme u `DebugMain`, a karakteristiku Caption u bilo koji naziv.

2. Dva puta kliknite mišem na dugme da bi bio prikazan upravljač događajem `OnClick` u okviru editora koda. Izmenite upravljač događajem `OnClick` tako da dobije sledeći izgled:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  S : string;
  X, Y : Integer;
begin
  X := Width;
  S := IntToStr(X);
  Y := Height;
  X := X * Y;
  S := IntToStr(X);
```

```
  X := X div Y;
  S := IntToStr(X);
  Width := X;
  Height := Y;
end;
```

3. Snimite projekt. Junit nazovite `DbgMain`, a projekt `DebugTst`.
4. Postavite tačku prekida na prvu liniju nakon iskaza begin u okviru upravljača događaja `OnClick`. Pokrenite program.
5. Kliknite mišem na dugme Watch Test. Debager će se zaustaviti na tački prekida. Kada se debager zaustavi na tački prekida, Delphi okruženje i editor koda dolaze u fokus (prelaze na vrh).
6. Dodajte elemente pregleda za promenljive `S`, `X` i `Y`. (Inicijalno, promenljive `X` i `Y` će biti nedostupne zbog optimizacije, ali ne brinite o tome.)
7. Uredite prozor spiska za pregled i prozor editora koda, tako da oba možete videti (usidrite prozor spiska za pregled na donju stranu prozora editora koda ukoliko želite).
8. Prebacite fokus na editor koda, a zatim pritisnite funkcijski taster F8, kako biste izvršili sledeću liniju koda. Ova linija je izvršena i tačka izvršavanja se pomera na sledeću liniju. Promenljiva `X` sada prikazuje vrednost.
9. Izvršite sledeći korak u okviru programa pritiskom na taster F8. Pogledajte rezultate promenljivih u prozoru za spisak pregleda.
10. Kada izvršavanje programa dođe do poslednje linije metode, kliknite mišem na dugme Run u okviru trake sa alatima da biste nastavili sa izvršavanjem programa.

Kliknite mišem na dugme Watch Test onoliko puta koliko želite, da biste osetili kako funkcioniše prozor spiska za pregled. Prilikom svakog prolaza eksperimentišite sa različitim opcijama za pregled.

Kod u ovom primeru pruža vrednosti širine i visine forme (karakteristike `Width` i `Height`), izvršava neke proračune, a zatim postavlja vrednosti karakteristika `Width` i `Height` na vrednosti koje su bile definisane u početku. Na kraju se ništa ne menja, ali postoji dobar razlog za dodeljivanje vrednosti karakteristikama `Width` i `Height` na kraju metoda.

Ukoliko ništa niste uradili sa promenljivama `X` i `Y`, ne morate ih proveravati pošto će ih prevodilac optimizovati, pa nećete biti u mogućnosti da ih pregledate. U principu, prevodilac može da gleda unapred, da pronađe promenljive koje se nikad ne koriste i da ih odstrani. Postavljanje promenljivih u upotrebu na kraju metode sprečava prevodilac da ih optimizuje. Ovo sam ponovio nekoliko puta, ali želim da budem siguran da ćete steći osnovno znanje o tome kako radi optimizovani prevodilac. Kada počnete sa debagiranjem Vaših aplikacija, ovo znanje će Vam pomoći da izbegnete frustracije kada počnete primati poruke: Variable `'X' inaccessible here due to optimization` (Promenljiva `'X'` nije dostupna zbog optimizacije).

Debager inspektor (Debug Inspector)

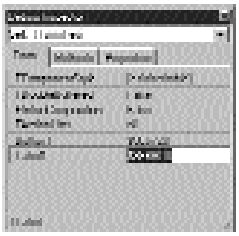
4 Debager inspektor je novina u Delphi-ju 4. Jednostavno rečeno, debager inspektor (Debug Inspector) Vam omogućava da pregledate objekte kao što su klase i slogovi. Takođe, možete istražiti jednostavne tipove podataka kao što su celi brojevi, nizovi karaktera, i slično, mada se ove vrste podataka mogu najbolje videti u okviru prozora sa spiskom za pregled. Debager inspektor je najkorisniji za ispitivanje klasa i slogova.

NAPOMENA Debager inspektor možete koristiti samo kada je zaustavljeno izvršavanje programa pod debagerom.

Da biste ispitali objekt, kliknite mišem na naziv objekta u datoteci izvornog koda, i odaberite opciju Inspect u okviru menija sadržaja editora koda (odnosno pritisnite tastere Alt+F5). Takođe, možete odabrati opciju Run➥Inspect u okviru glavnog menija.

Prozor debager inspektora sadrži detalje prikazanih objekata. Ukoliko je objekt jednostavan tip podataka, prozor debager inspektora će prikazati trenutnu vrednost (i u decimalnom i u heksadecimalnom obliku), a statusna linija u donjem delu prozora će prikazati tip podataka. Na primer, ako pregledate celobrojnu promenljivu, vrednost koja će biti prikazana u statusnoj traci će biti: `Integer`. U gornjem delu prozora debager inspektora je kombo okvir koji inicijalno sadrži opis objekta koji će biti ispitan.

Ukoliko ispitujete klase, debager inspektor će izgledati onako kako je to prikazano na slici 10.7.



Slika 10.7 Debager inspektor ispituje klasu forme

Da biste bolje razumeli debager inspektor, pratite sledeće korake:

1. Učitajte program `DebugTst` koji ste prethodno kreirali (ukoliko već nije učitan).
2. Postavite tačku prekida negde u okviru metode `WatchBtnClick`.
3. Pokrenite program i kliknite mišem na dugme WatchTest. Debager će se zaustaviti na tački prekida koju ste postavili.
4. U okviru glavnog menija odaberite opciju Run➥Inspect. Biće prikazan okvir za dijalog Inspect.
5. U polje Expression upišite tekst `Self`, a zatim kliknite na dugme OK.

6. Debager inspektor će biti prikazan i moćete da ispitate podatke vezane za glavnu formu.

NAPOMENA Promenljivu `Self` možete ispitati samo u okviru metode klase. Ukoliko postavite tačku prekida na regularnu funkciju, ili proceduru, a zatim pokušate da ispitate promenljivu `Self`, dobićete poruku koja opisuje da je `Self` pogrešan simbol. U prethodnom primeru promenljiva `Self` se odnosi na glavnu formu aplikacije.

Kartice debager inspektora

Kada istražujete klase, prozor debager inspektora sadrži tri kartice koje ste mogli da vidite na slici. Prvi elementi koji su prikazani, su elementi podataka koji pripadaju klasi koja je predak tekuće klase. Na kraju liste se nalaze elementi koji pripadaju tekućoj klasi. Možete odabrati da li ćete moći da vidite informacije o klasi koja je predak tekuće klase. Da biste isključili elemente klase pretka, kliknite desnim tasterom miša i odaberite opciju Show Inherited u okviru menija sadržaja debager inspektora.

Korišćenjem kursorskih tastera, pomerite se gore-dole po podacima liste elemenata, da biste mogli da vidite kom tipu podataka pripada svaki element (pogledajte u statusnu traku prozora debager inspektora). Da biste dalje ispitali tekući element, dva puta kliknite mišem na kolonu za vrednost u okviru linije koja prikazuje element podataka. Biće otvoren drugi prozor debager inspektora koji će prikazati odabrani element. Istovremeno možete imati nekoliko otvorenih prozora debager inspektora.

Kartica Methods debager inspektora prikazuje metode klasa. U nekim slučajevima, kartica metoda nije prikazana (na primer, kada ispitujete jednostavne tipove podataka). Statusna traka prikazuje dekleraciju odabranih metoda.

Kartica Properties debager inspektora prikazuje karakteristike klase koja se ispituje. Pregled karakteristika klase ima ograničen domet (informacije koje su predstavljene nisu baš korisne). U većini situacija možete shvatiti šta se dešava istraživanjem elemenata podataka koji su dodeljeni određenoj karakteristici na kartici Data.

NAPOMENA Kartica Methods i kartica Properties debager inspektora su dostupne samo kada ispitujete klase. Kada ispitujete jednostavne tipove podataka, prikazana je samo kartica Data.

SAVET Ukoliko želite da debager inspektor uvek bude na vrhu iznad editora koda, pređite na karticu Debugger okvira za dijalog opcija okruženja i odaberite polje za potvrdu Stay on Top debager inspektora.

Meni sadržaja debager inspektora

Meni sadržaja debager inspektora ima nekoliko opcija koje Vam omogućavaju da radite sa debager inspektorom i zasebnim promenljivama. Na primer, umesto da otvarate novi prozor debager inspektora za svaki objekat, možete kliknuti desnim tasterom miša i odabrati opciju Descend da biste zamenili tekući objekat u prozoru

debager inspektora, novim objektom. Na primer, ukoliko ispitujete formu sa dugmetom pod nazivom `Button1`, možete odabrati komponentu `Button1` u okviru debager inspektora i odabrati opciju Descend iz linije sadržaja. Debager inspektor će u tom slučaju ispitivati dugme `Button1`. Ovaj metod ima dodatnu prednost: okruženje čuva spisak objekata koje ste ranije ispitivali. Vratite se na objekat koji ste već ispitali, samo odaberite objekat iz kombo okvira u gornjem delu prozora debager inspektora. Izbor jednog od objekata sa spiska već korišćenih objekata će Vam ponovo pokazati željeni objekat u prozoru debager inspektora.

Da biste promenili vrednost promenljive, u meniju sadržaja debager inspektora odaberite opciju Change (promena).

Vodite računa kada menjate promenljive koristeći debager inspektor. Ukoliko upišete pogrešan tip podataka, odnosno definišete vrednost koja je pogrešna za određeni tip podataka, može se dogoditi da Vaš program krahira.

Opcija Inspect u okviru menija sadržaja debager inspektora Vam omogućava da otvorite drugi prozor debager inspektora sa elementom na kom se nalazi kursor. Opcija menija sadržaja New Expression Vam omogućava da upišete novi izraz koji ćete ispitati u debager inspektoru.

Opcija Show Inherited u okviru menija sadržaja debager inspektora je prekidač koji određuje koliko informacija može prikazati debager inspektor. Kada je opcija Show Inherited uključena, debager inspektor prikazuje sve tipove podataka, metode i karakteristike klase koja se ispituje, kao i tipove podataka, metode i karakteristike direktnog pretka klase. Kada je opcija Show Inherited isključena, samo tipovi podataka, metode i karakteristike same klase su prikazane. Isključivanje ove opcije može ubrzati debager inspektor, pošto tada nema puno podataka za prikazivanje.

Ukoliko imate podatke klase, a ne možete da se setite tipa, možete da kliknete na podatak u trenutku kada se program zaustavi na tački prekida i pritisnete testere Alt+F5 da biste prikazali debager inspektor. Statusna traka u donjem delu prozora debager inspektora će Vam saopštiti kog tipa je promenljiva.

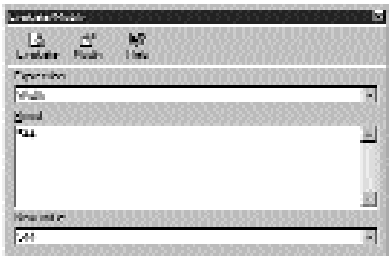
Ostali alati za debagiranje

Delphi ima nekoliko dodatnih alata za debagiranje koji Vam mogu pomoći u praćenju grešaka. Neki od ovih alata su, po svojoj prirodi napredni alati za debagiranje. Iako se napredni alati za debagiranje ne koriste toliko često kao ostali alati, oni mogu biti veoma moćni u rukama iskusnog programera.

Okvir za dijalog za prikazivanje/izmenu (Evaluate/Modify)

Okvir za dijalog za prikazivanje/izmenu (Evaluate/Modify) Vam omogućava da ispitate vrednost tekuće promenljive i promenite je ukoliko želite. Korišćenjem ovog okvira za dijalog, možete testirati različite izlazne vrednosti menjajući promenljivu

koju ispitujete. Ovo Vam omogućava da igrate igru šta-ako (what-if) u toku rada programa. Promena vrednosti promenljive u toku debagiranja Vam omogućava da testirate efekte različitih parametara Vaših programa, a da ga svaki put iznova na prevodite. Slika 10.8 prikazuje okvir za dijalog prikazivanje/izmena u trenutku ispitivanja promenljive.



Slika 10.8
Okvir za dijalog za prikazivanje /izmenu

NAPOMENA Traka sa alatima okvira za dijalog za prikazivanje/izmenu može prikazati veliku odnosno malu dugmad na traci za alate. Generički su prikazana mala dugmad. Mala dugmad nemaju natpise, pa treba da postavite kursor miša na dugme i pročitate oblačić za savet kako biste videli šta svako dugme može da radi. Da biste videli veliku dugmad na traci za alate, povucite na dole traku za promenu veličine (sizing bar) koja se nalazi odmah ispod trake sa alatima. Traka sa alatima će prikazati veću dugmad sa natpisima ispod svakog dugmeta. Slika 10.8. prikazuje okvir za dijalog za prikazivanje/izmenu sa velikim dugmadima na traci sa alatima.

Okvir za dijalog za prikazivanje/izmenu radi potpuno isto kao lista za pregled, odnosno debager inspektor. Ukoliko kliknete na promenljivu u okviru izvornog koda programa i odaberete opciju Evaluate/Modify menija sadržaja editora koda, vrednost promenljive će biti izračunata i prikazana. Ukoliko želite da upišete vrednost koja nije trenutno prikazana u izvornom kodu programa, možete odabrati opciju Run➥Evaluate/Modify u okviru glavnog menija, a zatim možete upisati naziv promenljive čiju vrednost želite da izračunate, odnosno prikažete.

Polje Expression se koristi za unos naziva promenljive, odnosno izraza čiju vrednost želite da vidite, odnosno izračunate. Kada kliknete na dugme Evaluate (ili pritisnete taster Enter) izraz će biti izračunat, a rezultat prikazan u polju Result.

SAVET Okvir za dijalog za prikazivanje/izmenu se može koristiti kao jednostavan kalkulator. Možete uneti heksadecimalne, odnosno decimalne brojeve (ili kombinaciju ovih brojeva) u okviru matematičke formule, pa ćete kao rezultat dobiti izračunatu vrednost. Na primer, ukoliko upišete

```
$400 - 256
```

u polje Evaluate, a zatim pritisnete taster Enter u polju Result će se pojaviti rezultat: `768`.

Takođe možete uneti logički izraz u polje Evaluate kako biste u polju Result videli da li je taj izraz tačan, ili netačan. Na primer, ukoliko upišete

```
20 * 20 = 400
```

u polju Result će biti prikazana vrednost `True`. Program mora biti zaustavljen na tački prekida, ukoliko želite da okvir za dijalog za izračunavanje/izmenu funkcioniše.

Ukoliko želite da promenite vrednost promenljive upišite novu vrednost u polje New Value, a zatim kliknite na dugme Modify. Vrednost promenljive će biti izmenjena. Kada kliknete na dugme Run, kako biste ponovo pokrenuli program (ili nastavili da se krećete korak po korak), nova vrednost promenljive će se koristiti u radu programa.

NAPOMENA Okvir za dijalog za prikazivanje/izmenu se ne osvežava automatski u toku rada programa, kao što je to slučaj sa listom za pregled i debager inspektorom. Ukoliko Vaš kod menja promenljivu u okviru za dijalog za prikazivanje/izmenu, morate kliknuti na dugme Evaluate kako biste videli nove rezultate. Ovaj aspekt okvira za dijalog za prikazivanje/izmenu ima jednu glavnu prednost; prolazak kroz izvorni kod je brži pošto debager ne mora da izračunava izraz prilikom svakog koraka (kao što je to slučaj sa listom za pregled i debager inspektorom). Ono što ćete uglavnom raditi sa ovim okvirom za dijalog je pregled promenljive, odnosno izračunavanje izraza, a zatim njegovo zatvaranje.

Prozor steka poziva

U toku rada Vašeg programa, možete videti stek poziva, kako biste ispitali bilo koju funkciju, ili proceduru koju Vaš program poziva. U okviru glavnog menija odaberite opciju View➥Debug Windows➥Call Stack da biste prikazali prozor steka poziva (Call Stack). Ovaj prozor prikazuje listu funkcija i procedura koje su pozvane iz Vašeg programa, kao i redosled kojim su pozvane. Poslednja pozvana funkcija, ili procedura se nalazi na vrhu prozora.

Dva puta kliknite mišem na naziv metode u okviru prozora steka poziva i preći ćete na liniju izvornog koda metode, ukoliko se metoda nalazi u Vašem programu. U slučaju da funkcije i procedure nemaju izvorni kod (na primer VCL metode), prozor za poziv steka sadrži samo adresu i naziv modula u kom se procedura nalazi. Dvostruki klik mišem na funkciju, odnosno proceduru koja nema izvorni kod će prikazati prozor procesora (prozor procesora će biti obrađen u sledećem poglavlju).

Pregledanje steka poziva je od velikе pomoći nakon greške u pristupu koju generiše Windows operativni sistem. Pregledom steka poziva, možete videti gde se nalazio Vaš program, pre nego što se pojavila greška. Prvi korak u otkrivanju greške, odnosno otkrivanju onoga što je krenulo naopako, predstavlja saznanje gde se Vaš program nalazio pre no što je krahirao.

SAVET Ukoliko lista steka sadrži podatke bez ikakvog smisla, mogući uzrok bi bio greška na steku. Poremećeni stek poziva je obično indikator prepunjenosti steka, odnosno zapisivanja podataka u memoriju preko već postojećih, potrebnih podataka. Prepunjenost steka se ne pojavljuje toliko često u 32-bitnim programima, kao što se to dešavalo u 16-bitnim programima, ali se još uvek dešava.

Prozor procesora (CPU window)

4 Zvanično, prozor procesora je novina Delphi-ja 4. U prethodnim verzijama Delphi-ja ste mogli da dođete do prozora procesora, ali samo ako ste znali magične Registry podatke. Prozor procesora je sada službeni deo Delphi-ja i može se naći u okviru opcije glavnog menija View➥Debug Windows➥CPU (Ctrl+Alt+C sa tastature).

Prozor procesora Vam omogućava da pratite Vaš program na nivou instrukcija asemblera. Korišćenjem ovog prozora možete korak po korak izvršavati asemblerske instrukcije. Takođe možete pokrenuti program počev od određene instrukcije asemblera, tako što pokrećete program od određene linije izvornog koda kod standardnog debagera. Prozor procesora ima pet panoa: pano za disasembliranje, pano registara, pano flegova, pano neobrađenog steka i pano za pregled mašinskog koda.

Svaki pano ima pridružen zaseban meni sadržaja. Meni sadržaja omogućava korišćenje svih funkcija koje su neophodne određenom panou. Da biste ga efikasno koristili, potrebno je da poznajete asembler. Očigledno je da prozor procesora spada u napredne alate za debagiranje.

Komanda za prelazak na adresu (Go to Address)

Još jedan napredan alat za debagiranje je komanda za prelazak na adresu (Go to Address). Kada Vaš program krahira, Windows prikazuje grešku sa adresom u okviru memorije na kojoj je greška nastala. Da biste pronašli gde se u okviru Vašeg programa pojavila greška, možete koristiti komandu Go to Address. Kada od Windows operativnog sistema dobijete poruku Access Violation eror (greška pogrešnog pristupa), možete videti okvir za dijalog koji je sličan okviru dijaloga prikazanom na slici 10.9.

Slika 10.9 Windows poruka izveštava o pogrešnom pristupu

Kada ugledate ovu poruku greške, zapišite adresu na kojoj se greška pojavila, a zatim odaberite opciju Debug➥Go to Address menija sadržaja editora koda, kako bi bio prikazan okvir za dijalog Goto Address. Upišite adresu koju ste prethodno zapisali u polje Address okvira za dijalog Goto Address.

Kada kliknete na dugme OK, debager će pokušati da pronađe liniju izvornog koda programa gde je nastala greška. Ukoliko je greška nastala u Vašem kodu, kursor će biti postavljen na liniju koja je generisala grešku. Ukoliko se greška dogodila van Vašeg koda, dobićete poruku da adresu koju tražite nije moguće pronaći. Kao što sam i objasnio, ovo je napredan alat za debagiranje, pa ga možda i nikada nećete koristiti.

Prolazak kroz kod korak po korak

Prolazak kroz kod korak po korak je jedna od osnovnih operacija debagiranja, pa bi trebali da je spomenemo. Ponekad ne možete da vidite šumu od drveća. (Pošto ponekad autori knjiga za programiranje ne uspevaju da uključe očigledne stvari!) Vraćanje na osnovne stvari s vremena na vreme može otkriti nešto čega ranije niste bili svesni.

Simboli žljeba za debagiranje

Na početku ovog poglavlja ću Vam ukratko objasniti simbole koji se pojavljuju u žljebu editora koda u toku procesa debagiranja. U poglavlju "Postavljanje i brisanje tačaka prekida", objasnio sam Vam da se pojavljuju crveni kružići u žljebu, kada postavite tačku prekida na liniju izvornog koda. Takođe sam Vam objasnio da zelena strelica označava tačku izvršavanja u toku prolaska kroz izvorni kod.

Nisam spomenuo da se pored određenih linija izvornog koda pojavljuju plave tačke u žljebu editira koda. Ove tačke označavaju linije u okviru izvornog koda programa koje generišu asemblerski kod. Slika 10.10 prikazuje editor koda u trenutku kada je debager zastao na tački prekida. Na slici su prikazane tačke koje označavaju generisani kod, strelicu koja označava tačku prekida, kao i oznaku za tačku prekida. Znak potvrde na simbolu tačke prekida označava da je tačka prekida određena kao važeća tačka prekida.

Slika 10.10
Editor koda prikazuje simbole u žljebu

Detaljnije pogledajte sliku 10.10. Uočite da se tačke pojavljuju samo na nekim linijama koda. Linije bez tačaka ne generišu prevedeni kod. Na primer, pogledajte ove linije:

```
var
  S : string;
  X : Integer;
```

Zašto ove linije ne generišu kod? Zato što su to dekleracije promenljivih. Šta je sa ovom linijom:

```
X := 20;
```

Zašto ova linija ne generiše kod? Ponoviću Vam ovu reč: optimizacija. Prevodilac gleda unapred i primećuje da se promenljiva X ne koristi, pa ignoriše sve što ukazuje na tu promenljivu. Na kraju, pogledajte sledeće linije koda:

```
{$IFNDEF WIN32}
S := 'Something's very wrong here...';
{$ENDIF}
```

Prevodilac ne generiše kod za ovu liniju izvornog koda, pošto se ona nalazi između direktiva prevodioca i pošto je simbol `WIN32` definisan u programu Delphi 4. Direktiva prevodioca `$IFNDEF WIN32` saopštava prevodiocu: "Prevedi ovu liniju koda samo ako ciljna platforma nije 32-bitni Windows operativni sistem." Pošto je Delphi 4 32-bitni prevodilac, ova linija koda neće biti prevedena. Ukoliko koristite Delphi 1 (16-bitno okruženje), ova linija koda će biti prevedena.

Preskoči (Step Over) i prati kroz (Trace Into)

Uredu, vraćamo se na prolazak kroz kod korak po korak. Kada se zaustavite na tački prekida možete da uradite mnogo stvari kako bi utvrdili šta se dešava sa Vašim kodom. Možete podesiti promenljive koje želite da posmatrate u listi za pregled, pratite objekte sa debager inspektorom, odnosno da pregledate stek poziva. Takođe možete da prolazite korak po korak kroz kod i posmatrate šta se dešava sa Vašim promenljivama i objektima nakon izvršavanja svake linije koda.

Kako prolazite kroz Vaš kod primetićete da linija izvornog koda, koja će se sledeća izvršiti, postaje osvetljena plavom bojom. Ukoliko ste otvorili listu za pregled i prozore debager inspektora, nakon svake linije koda koja će biti izvršena, podaci o ovim prozorima će biti obnovljeni. Bilo koja izmena promenljivih, ili objekata će se trenutno prikazati u prozoru liste za pregled, odnosno u prozoru debager inspektora. Okruženje debagera ima dve primarne komande za izvršavanje programa korak po korak koje će Vam pomoći u Vašim operacijama debagiranja: Step Over (preskoči) i Trace Into (prati kroz).

Step Over (preskoči)

Step Over predstavlja izvršavanje sledeće linije izvornog koda i zaustavljanje na liniji koja sledi iza tekuće linije. Ova opcija je neka vrsta greške u nazivu. Naziv ukazuje da možete da preskočite preko linije izvornog koda, a da je ne izvršite. Ipak, to nije slučaj. Step Over znači da će tekuća linija biti izvršena i da će funkcije, odnosno procedure koje su pozvane iz te linije biti pokrenute punom brzinom. Na primer, pretpostavimo da ste postavili tačku prekida na liniju koja poziva metodu u Vaš program. Kada saopštite debageru da preskoči metodu, debager će izvršiti metodu i zaustaviti se na sledećoj liniji. (Za razliku od načina na koji radi opcija Trace Into, o kojoj ćete ubrzo učiti, pa će sve imati više smisla.) Da biste koristili opciju Step Over da prolazite korak po korak kroz program, takođe možete pritisnuti funkcijski taster F8, odnosno odabrati opciju Run➥Step Over u okviru glavnog menija.

Kako prolazite kroz različite junite izvornog koda u Vašem programu, editor koda će automatski učitavati i prikazivati sve junite izvornog koda koji su potrebni, ukoliko već nisu učitani.

Trace Into (prati kroz)

Komanda Trace Into Vam omogućava da pratite rad programa kroz bilo koju funkciju, ili proceduru koja se pojavljuje u toku rada Vašeg programa. Bolji način od izvršavanja funkcije, ili procedure, ili vraćanja na sledeću liniju, kao što je to slučaj kod izvršavanja opcije Step Over, opcija Trace Into postavlja tačku izvršavanja na prvu liniju izvornog koda funkcije, ili procedure koja se poziva. Zatim možete prolaziti liniju po liniju kroz funkciju, odnosno proceduru koristeći komande Step Over, odnosno Trace Into, ukoliko je to potrebno. Prečica tastature za komandu Trace Into je F7.

Nakon što ispitate promenljive i uradite ono što je potrebno, možete ponovo pokrenuti program punom brzinom, ukoliko kliknete na dugme Run. Program će funkcionisati normalno sve dok ne naiđe na sledeću tačku prekida.

Ukoliko imate verziju Delphi-ja Professional, ili Client/Server, možete ući i u VCL izvorni kod. Kada uđete u VCL metodu, komanda Trace Into će Vas provesti kroz VCL izvorni kod tekuće metode. Možete ispitati bilo koju promenljivu koju želite da vidite. Morate dodati putanju do VCL izvornog koda u polju Search path okvira za dijalog opcije projekta (kartica Directories/Conditionals). Da biste aktivirali ovu opciju morate pokrenuti opciju Build nakon što dodate VCL direktorijum u polje Search path. Ulazak u VCL izvorni kod je bez sumnje prednost za mnoge programere. Iskusni programeri će verovatno ovo smatrati veoma korisnim.

Prelazak na sledeću liniju izvornog koda (Trace To Next Souce Line)

Još jedna ređe korišćena komanda za debagiranje je prelazak na sledeću liniju izvornog koda (Trace To Next Source Line), sa prečicom tastature Shift+F7. Najverovatnije nećete puno koristiti ovu komandu, sve dok se ne upoznate bolje sa debagiranjem i uopšte programiranjem pod Windows-ima. Neke Windows API funkcije koriste nešto što se naziva ponovno pozivanje funkcije (callback function). Ovo znači da će Windows funkcija pozvati jednu od Vaših funkcija za izvršavanje nekog zadatka.

Ukoliko se tačka izvršavanja nalazi na Windows API funkciji koja koristi ponovni poziv funkcije, korišćenje komande prelazak na sledeću liniju izvornog koda će prebaciti tačku izvršavanja na prvu liniju ponovo pozvane funkcije. Efekat je potpuno isti kao kod komande Trace Into, ali specifična situacija u kojoj se koristi komanda prelazak na sledeću liniju izvornog koda je potpuno različita. Ukoliko Vam sve to nema smisla, ne brinite. Sve ovo nije potrebno da danas naučite.

Kada prolazite kroz metodu, tačka izvršavanja će eventualno doći do iskaza end tekuće metode. Ukoliko metoda kroz koju prolazite vraća kontrolu Windows-ima nakon izvršavanja, pritisak na taster F8, ukoliko se nalazite na iskazu end, će kao rezultat dati izlazak iz metoda i vraćanje kontrole programu koji se debagira. Ne postoji nikakva očigledna naznaka da je program zaustavljen, pošto je okruženje još uvek u fokusu. Ovo ponašanje može biti zbunjujuće, ukoliko prvih nekoliko puta naiđete na njega, sve dok ne budete postali svesni šta se dešava. Da biste se vratili u Vaš program, ponovo ga aktivirajte, kao što bi to uradili sa bilo kojim drugim programom (kliknite na njegovo dugme u okviru trake sa zadacima Windows operativnog sistema, odnosno pritisnite tastere Alt+Tab).

Kao što sam i rekao, prolazak kroz kod korak po korak je osnovna tehnika debagiranja, ali i jedina koju ćete stalno koristiti u toku debagiranja. Od svih prečica tastature koje su Vam dostupne u okviru Delphi-ja, F7 (Trace Into), F8 (Step Over) i F9 (Run) će definitivno biti u Vašem arsenalu.

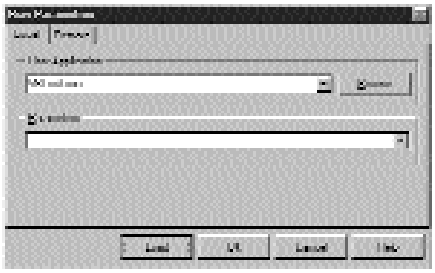
Debagiranje DLL datoteka

U većini slučajeva, debagiranje DLL datoteka (dynamic link library - dinamički povezana datoteka) je potpuno isto kao i debagiranje izvršnih datoteka. U DLL kod možete postaviti tačke prekida, pa kada izvršavanje programa dostigne date tačke prekida, debager će se zaustaviti kao što je to slučaj kod debagiranja EXE datoteke. Normalno, DLL datoteku ćete testirati kreiranjem test aplikacije i pokretanjem test aplikacije pod debagerom.

Ponekad je potrebno da testirate DLL datoteku koja će se koristiti sa drugim izvršnim datotekama, koje su napravljene u drugim razvojnim okruženjima. Na primer, recimo da ste napravili DLL datoteku koja će biti pozvana iz Visual Basic aplikacije. Sigurno nećete moći da pokrenete VB aplikaciju pod Delphi-jevim debagerom. Ono što možete da uradite je da saopštite Delphi-jevom debageru da

startuje VB aplikaciju kao host aplikaciju. (Prirodno, host aplikacija treba da sadrži kod koji će učitavati DLL datoteku.) Saopštićete Delphi-ju da startuje eksternu host aplikaciju, koristeći okvir za dijalog Run Parameters.

Da biste prikazali okvir za dijalog Run Parameters, odaberite opciju Run➥Parameters u okviru glavnog menija. Upišite naziv EXE datoteke u polje Host Application, kliknite na dugme Load, pa će host aplikacija biti pokrenuta. Slika 10.11 prikazuje okvir za dijalog Run Parameters koji se pojavljuje pre debagiranja DLL datoteke.



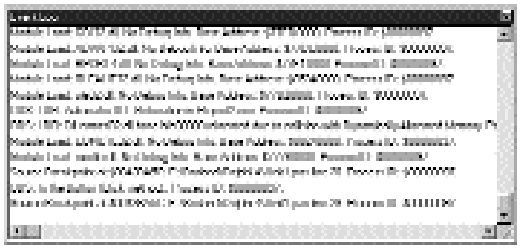
Slika 10.11
Definisanje host aplikacije korišćenjem okvira za dijalog Run Parameters

Nakon što pokrenete host aplikaciju, možete debagirati Vaš DLL onako kao što ste testirali bilo koju drugu Delphi aplikaciju: postavite tačke prekida u DLL datoteku i počnite sa debagiranjem.

Okvir za dijalog Run Parameters sadrži karticu pod nazivom Remote. Ova kartica Vam omogućava da podesite parametre za debagiranje aplikacija koje se nalaze na udaljenim mašinama. Debagiranje na udaljenim mašinama je napredna opcija i neće biti obrađena u ovoj knjizi.

Prozor za evidenciju događaja (Event Log window)

Prozor za evidenciju događaja (Event Log) je posebna Delphi datoteka koja prikazuje dijagnostičke poruke - poruke koje generiše Delphi, Vaša aplikacija, a ponekad i sam Windows. Na primer, evidencija događaja sadrži informacije o modulima koji su učitani (uglavnom DLL datoteke), da li uključuju informacije debagera, kada je Vaša aplikacija pokrenuta, kada je prestala sa radom, kada se došlo do tačke prekida, itd. Evidenciju događaja možete pregledati koristeći prozor za evidenciju događaja. Da biste videli prozor za evidenciju događaja, odaberite opciju View➥Debug Windows➥Event Log u okviru Delphi-jevog glavnog menija. Slika 10.12 prikazuje prozor evidencije događaja u toku debagiranja aplikacije.



Slika 10.12
Prozor evidencije događaja

Prozor evidencije događaja sadrži meni sadržaja koji Vam omogućava da obrišete evidenciju, snimite je u tekst datoteku, odnosno komentarišete evidenciju događaja. Snimanje evidencije događaja u tekst datoteku Vam omogućava da pregledate poruke podrobnije, odnosno da tražite određeni tekst koji želite da vidite. Prozor evidencije događaja u okviru menija sadržaja ima opciju Properties (karakteristike) koja Vam omogućava da dodatno prilagodite evidenciju događaja. Kada odaberete ovu opciju menija, okvir za dijalog će biti prikazan i omogućiće Vam da izmenite opcije evidencije događaja. Ovaj okvir za dijalog je potpuno isti kao kartica Event Log u okviru za dijalog opcije debagera (biće obrađena u poglavlju "Kartica za evidenciju događaja").

Svoje poruke možete poslati u evidenciju događaja korišćenjem Windows API funkcije `OutputDebugString`. Funkcija `OutputDebugString` će biti obrađena u poglavlju "Funkcija `OutputDebugString`".

Prozor modula

Prozor modula pokazuje module koji su trenutno učitani, izvorne datoteke koje su pridružene ovim modulima, kao i simbole (funkcije, procedure i promenljive) koje su izvezene iz tog modula. Prozor modula možete pozvati birajući opciju View➥Debug Windows➥Modules u okviru glavnog menija. Prozor modula je prvenstveno napredna alatka za debagiranje, pa je neću detaljnije objašnjavati. Možete neko vreme ekperimentisati sa prozorom modula da biste videli kako radi. Slika 10.13 prikazuje prozor modula u toku rada.

Slika 10.13
Prozor modula

Tehnike debagiranja

Već sam objasnio neke tehnike debagiranja, dok sam Vam izlagao razne aspekte debagera u ovoj lekciji. Želeo bih da napomenem još nekoliko dodatnih tehnika koje će Vaš posao debagiranja učiniti lakšim.

Funkcija `OutputDebugString`

Ponekad je veoma korisno pratiti izvršavanje Vaših programa u toku samog rada programa. Možda ćete poželeti da vidite vrednost promenljive, a da ne zaustavite program na tački prekida. Funcija `OutputDebugString` Vam omogućava da upravo to i uradite. Ova funkcija je veoma zgodan alat za debagiranje koji mnogi programeri previđaju, prvenstveno zbog opšteg nedostatka objašnjenja ovog alata. Pogledajte u poslednji element u okviru evidencije događaja koja je prikazana na slici 10.12 (u prethodnom delu lekcije). Ovaj element je generisan korišćenjem koda:

```
OutputDebugString('In the Button1Click method...');
```

To je sve što treba da uradite. Pošto je Delphi instaliran kao sistemski debager, bilo koji string poslat korišćenjem funkcije `OutputDebugString` će biti prikazan u evidenciji događaja. Pozive funkciji `OutputDrbugString` možete uputiti iz bilo kog dela Vašeg programa.

Da biste videli vrednost promenljive, morate formatirati string i poslati ga u funkciju `OutputDebugString`. Na primer:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  X : Integer;
  S : string;
begin
  { Some code here...}
  S := Format('X := %d', [X]);
  OutputDebugString(PChar(S));
end;
```

Korišćenjem funkcije `OutputDebugString` možete videti šta se dešava sa Vašim progamom čak i u delovima koda koji veoma kratko traju.

Praćenje grešaka pristupa

Kada program pokuša da upiše podatke u memoriju koja mu ne pripada, Windows prosleđuje poruku o pogrešnom pristupu. Svi Windows programeri nailaze na grešku o pogrešnom pristupu u toku razvoja aplikacija.

Pojam GPF (General Protection Fault) se koristio u 16-bitnim Windows-ima. Njegovo korišćenje preovladava i u 32-bitnom Windows programiranju, iako 32-bitni Windows programi generišu grešku pristupa (Access Violation), umesto General Protection Fault.

Greške pristupa se mogu teško pratiti, a to važi i za početnike i za iskusne Windows programere. Često, dok programeri stiču iskustvo u pisanju programa, razvijaju šesto čulo za pronalaženje uzroka pogrešnog pristupa. Evo nekih ključeva za rešenje problema kada pokušavate da pronađete lukavu grešku pristupa. Ovo nisu jedine situacije koje prouzrokuju pad programa, ali su najčešće.

Neinicijalizovani pointeri

Neinicijalizovani pointer (uninitialized pointer) je pointer koji je deklarisan, ali ne ukazuje ni na šta što ima smisla u okviru Vašeg programa. Neinicijalizovani pointer će sadržati slučajne podatke. U najboljem slučaju pokazuje na neku bezopasnu tačku u memoriji. U najgorem slučaju, neinicijalizovani pointer ukazuje na slučajnu adresu u memoriji u okviru Vašeg programa. Ovo može voditi do pogrešnog ponašanja programa, pošto pointer može pokazivati na različite memorijske adrese svaki put kada se program pokrene. Pre nego što počnete da koristite pointer podesite ga na `nil`, a isto to uradite i kada objekat na koji pointer ukazuje bude obrisan. Ukoliko pokušate da pristupite pointeru koji sadrži vrednost `nil`, Vaš program će se zaustaviti, prijavljujući grešku pristupa, a pogrešna linija izvornog koda će biti osvetljena u okviru debagera, pa ćete moći da trenutno identifikujete grešku sa pointerom.

Brisanje prethodno obrisanog pointera

Brisanje pointera koji je već bio obrisan, kao rezultat daje grešku pristupa. Savet koji ste dobili za rad sa neinicijalizovanim pointerima i ovde važi: dodelite obrisanim pointerima `nil`. Potpuno je bezopasno ukoliko želite da obrišete `nil` pointer. Podešavanjem Vašeg obrisanog pointera na `nil`, osiguravate se da neće biti loših efekata ukoliko obrišete pointer po drugi put.

Pisanje preko kraja niza

Pisanje preko kraja niza može da prouzrokuje grešku pristupa. U nekim slučajevima memorija preko koje je nešto napisano ne mora biti od vitalne važnosti, pa se problem neće manifestovati odmah, nego kasnije; tada će najverovatnije krahirati. Kada se to dogodi, najverovatnije ćete tražiti grešku u delu programa u trenutku pada, ali će se problem pojaviti u potpuno drugom delu programa. U drugim slučajevima, memorija preko koje je pisano može biti od vitalne važnosti, pa će program trenutno stati. U ekstremnim slučajevima možda će doći i do pada Windows-a.

Pisanje preko kraja niza može ponekad biti svedeno na minimum proverom opsega. Kada je uključena provera opsega (generički), prevodilac će ispitati reference niza da bi proverio da li pristupate elementima niza van dozvoljenog opsega. Na primer, ovaj kod će kao rezultat dati grešku prevodioca:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  A : array [0..20] of Char;
begin
  A[30] := 'a';
end;
```

U ovom slučaju pristupam tridesetom elementu niza koji sadrži samo 21 element. Prevodilac primećuje da se nizu pristupa van deklarisanog opsega i generiše grešku prevodioca. Provera opsega ipak ne radi sa promenljivama. Na primer, ovaj kod neće kao rezultat dati grešku prevodioca.

```
procedure TForm1.Button1Click(Sender: TObject);
var
  X : Integer;
  A : array [0..20] of Char;
begin
  X := 30;
  A[X] := 'a';
end;
```

Iako je preko kraja niza napisano 9 bajtova, kao rezultet nećete dobiti grešku prevodioca, pošto prevodilac ne zna u toku prevođenja vrednost promenljive `X`.

Pogrešan pristup prilikom završetka programa

Kada se program zaustavi sa greškom u pristupu prilikom zatvaranja programa, to obično ukazuje da je veličina steka suviše mala. Iako ovo nije slučaj kod 32-bitnih programa, ovo se često dešava u ekstremnim okolnostima. Greška u pristupu prilikom zatvaranja programa može nastati brisanjem već obrisanog pointera, što je obrađeno u jednom od prethodnih poglavlja.

Kratki saveti za debagiranje

Kao dodatak mnogobrojnim savetima koji su Vam ponuđeni na prethodnim stranama knjige, možda ćete poželeti da primenite još neke:

- Promenite karakteristiku `Caption` forme, kako biste prikazali promenljive, a da ne koristite tačke prekida. Pošto je postavljanje komponente `Label` na formu jednostavno, možete da koristite i ovu komponentu. Promenite tekst natpisa tako da prikazuje vrednost promenljive, odnosno bilo koju drugu informaciju koju želite da bude prikazana.
- Aktivirajte uslovne tačke prekida, odnosno tačke prekida za pregled podataka, da bi privremeno usporili Vaš program (verovatno da pogledate efekte usporeno). Ove tačke prekida će usporiti izvršavanje Vašeg programa dok budu proveravale uslove tačaka prekida.
- Koristite okvir za dijalog za prikaz/izmenu da biste privremeno izmenili vrednost promenljive u toku rada programa. Ovo Vam omogućava da vidite efekte koje različite vrednosti mogu imati na Vaš program, a da ponovo ne prevodite Vaš kod svaki put kada želite da izmenite vrednost promenljive.
- Odaberite opciju Run➥Inspect u okviru glavnog menija i upišite promenljivu `Self` u polje Expression kako biste proverili klasu na kojoj se debager trenutno zaustavio.
- Koristite funkciju `MessageBeep ($FFFF)` kao zvučni indikator da je dostignuta određena tačka u Vašem programu. Ova Windows API funkcija aktivira zvučnik Vašeg računara, ukoliko se pozove korišćenjem parametra -1.
- Odaberite opciju Run➥Program Reset u okviru glavnog menija, odnosno pritisnite tastere Ctrl+F2, kako bi zaustavili program prilikom pojave greške u toku debagiranja.
- Koristite privremene promenljive kako biste podelili duge jednačine, odnosno povezane pozive metoda i na taj način podelili rezultate na više delova kojima se lakše upravlja.
- Koristite funkcije `ShowMessage`, `MessageBox`, ili `MessageDlg` da biste prikazali informacije za praćenje programa. (Najbolje je koristiti funkciju `ShowMessage` pošto ova funkcija uzima samo string poruke kao jedini parametar.)

Ukoliko radite sa Delphi-jem na Windows-ima 95, opciju Program Reset koristite oprezno. U nekim slučajevima korišćenje opcije Program Reset za ukidanje aplikacije može dovesti do kraha operativnog sistema. Pošto se svi Windows 95 sistemi ne ponašaju na isti način, može se dogoditi da na ovaj problem nikad ne naiđete. Windows NT ne pati od ovog problema kao što je to slučaj sa Windows 95 operativnim sistemom, pa ćete moći da koristite opciju Program Reset puno slobodnije pod Windows NT operativnim sistemom. Lično koristim opciju Program Reset samo kada se aplikacija koju debagiram zakoči.

Jedan od najboljih saveta koji Vam mogu dati za debagiranje je korišćenje programa za proveru memorije kao što je Memory Sleuth firme TurboPower Software. Možda ćete primiti ovaj program kao deo Delphi-ja 4 (uključen je u Delphi 4 na ograničen vremenski period). Memory Sleuth proverava da li Vaš program rasipa memoriju. Ovaj tip programa može Vas poštedeti nerviranja kada servisirate Vašu aplikaciju. Ukoliko Vaša aplikacija rasipa memoriju, prouzrokovaće probleme Vašim korisnicima. Problemi Vaših korisnika predstavljaju probleme za Vas. Brinući o rasipanju memorije u toku rada, poštedećete sebe i Vaše korisnike od frustracije.

Ukoliko niste dobili Memory Sleuth sa Delphi-jem 4 možete ga pronaći na Web sajtu firme TurboPower (`www.turbopower.com`). Još jedan program za proveru rasipanja memorije je BoundsChecker firme NuMega Tehnologies.

Opcije debagera

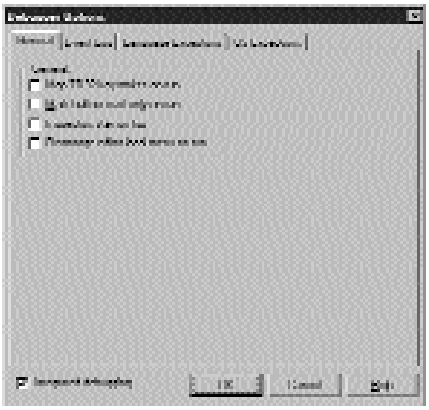
Opcije dabagera mogu biti postavljene na dva nivoa: nivo projekta i nivo okruženja. Opcije debagera projekta su bile obrađene u jučerašnjoj lekciji u poglavlju "Kartica Compiler" i "Kartica Linker". Opcije debagera koje ste postavili na globalnom nivou se mogu pronaći u okviru za dijalog opcije debagera. Da biste pozvali okvir za dijalog opcije debagera, odaberite opciju Tools➥Debugger Options u okviru glavnog menija.

U donjem delu okvira za dijalog je polje za potvrdu sa nazivom Integrated debugging. Ova opcija kontroliše da li se za debagiranje koristi debager iz okruženja. Ukoliko je polje za potvrdu za integrisani debager odabrano, koristiće se debager okruženja. Ukoliko ova opcija nije odabrana neće se koristiti debager iz okruženja. Ovo znači da kada kliknete mišem na dugme Run, program može biti izvršen, ali je debager deaktiviran, pa tačke prekida neće funkcionisati.

Okvir za dijalog za opcije debagera ima četiri kartice: General, Event Log, Language Exceptions i OS Exceptions. Ove kartrice će biti obrađene u narednim poglavljima.

Kartica General

Kartica General se koristi za podešavanje opštih opcija debagera. Ova kartica je prikazana na slici 10.14.



Slika 10.14
Okvir za dijalog opcije debagera; kartica General

Opcija Map TD32 keystokes on run (TD32 mapiranje tastature u toku rada) u ovom odeljku saopštava editoru koda da koristi mapiranje tastera koje se koristi u Borlandovom zasebnom debageru, Turbo Debugger-u. Ovo je veoma dobra opcija ukoliko ste proveli dosta vremena koristeći Turbo Debugger i upoznati ste sa mapiranjem tastature ovog programa.

Opcija Mark buffers read-only on run (markiranje bafera u toku rada tako da se samo mogu čitati) postavlja bafere editora koda tako da se mogu samo čitati u toku rada programa pod debagerom. Nakon što pokrenete program pod debagerom, ne možete editovati Vaš izvorni kod sve dok se program ne zatvori. Ovu opciju sam ostavio isključenu pošto u toku debagiranja često menjam svoj izvorni kod.

Polje za potvrdu Inspectors stay on top kontroliše da li prozori debager inspektora uvek ostaju na vrhu editora koda. Ovo je odlična opcija pošto uglavnom želite da prozori debager inspektora ostaju na vrhu u toku prolaska kroz Vaš kod.

Opcija Rearrange editor local menu on run menja pojavu menija sadržaja editora koda kada je program pokrenut pod debagerom. Kada je ova opcija uključena, meni sadržaja editora koda ima opcije koje se odnose na debagiranje pomerene u gornji deo menija sadržaja, tako da se lakše mogu pronaći.

Kartica Event Log

Kartica Event Log (praćenje događaja) Vam omogućava da podesite opcije za praćenje događaja. Možete odabrati maksimalan broj poruka koje se istovremeno mogu pojaviti u kartici događaja, odnosno možete ostaviti neograničen broj poruka. Takođe, možete odabrati tipove poruka koje želite da vidite u kartici događaja.

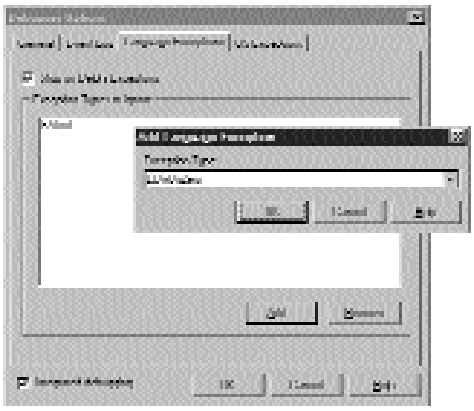
Kartica Lenguage Exceptions

Kartica Language Exceptions (izuzeci jezika) se koristi za kontrolu VCL izuzetaka koje će uhvatiti debager (izuzeci su obrađeni u lekciji dana 14, "Napredno programiranje"). Najvažnija opcija na ovoj kartici je Stop on Delphi Exceptions. Kada je ova opcija uključena, debager zaustavlja izvršavanje programa ukoliko je izbačen izuzetak. Ukoliko je opcija isključena, VCL izuzetkom se upravlja na uobičajen način koristeći okvir za poruke koji obaveštava korisnika, šta nije u redu sa programom.

Kada je opcija Stop on Delphi Exceptions uključena, debager se zaustavlja na izuzecima, čak iako upravljate njima u Vašem programu. Ukoliko ne želite da debager zastaje na svakom izuzetku, isključite ovu opciju. Ova opcija zamenjuje opciju Break on exception koja se mogla pronaći u prethodnim verzijama Delphi-ja.

Opcija Exception Types to Ignore se koristi da definiše tipove izuzetaka koje želite da debager ignoriše. Bilo koja klasa izuzetaka na ovoj listi će biti ignorisana od strane debagera i izuzetkom će se upravljati na način koji je generički definisan. Ovo je efektivno potpuno isto kao da ste isključili opciju Stop on Delphi Exceptions za odabrane tipove izuzetaka.

Da biste dodali tip izuzetka na listu, jednostavno kliknite na dugme Add a zatim upišite naziv klase izuzetaka. Da biste saopštili debageru da ignoriše izuzetke deljenja nulom (na primer), kliknite na dugme Add i upišite u polje Exception Type, `EDivByZero`. Slika 10.15 prikazuje ovaj proces.



Slika 10.15
Dodavanje tipa izuzetka u listu Exception Types to Ignore

Bilo koji tip izuzetka koji ste dodali na listu će biti prihvaćen u svim projektima (uključujući bilo koji novi projekt).

Kartica OS Exceptions

Kartica OS Exceptions se koristi za kontrolu izuzetaka operativnog sistema; da li će ovim izuzecima upravljati debager, ili program korisnika. Slika 10.16 prikazuje karticu OS Exceptions okvira za dijalog opcije debagera.

Slika 10.16
Okvir za dijalog opcije debagera sa karticom OS Exceptions

Kada je opcija Handled By podešena na User Program, debager zaustavlja izvršavanje programa kada se izbaci izuzetak. Kada je ova opcija postavljena na Debugger, VCL izuzetkom se upravlja na uobičajen način - sa okvirom za poruke koje obaveštavaju korisnika šta nije u redu sa programom.

NAPOMENA Kada je opcija Handled By podešena na Debugger, debager će zastati na izuzecima, čak iako se događajima upravlja u Vašem programu. Ukoliko ne želite da debager zastane na svakom izuzetku, podesite ovu opciju na User Program. Ova opcija zamenjuje opciju Break on exception u prethodnim verzijama Delphi-ja.

Opcija On Resume definiše kako će izuzetak biti tretiran kada izvršavanje programa nastavi rad nakon izuzetka.

Okvir za dijalog Exceptions sadrži listu mogućih izuzetaka operativnog sistema. Da biste podesili opcije za određeni tip, kliknite na izuzetak u okviru za listu Exceptions, a zatim postavite opcije Handled By, ili On Resume po želji. Sličice na desnoj margini okvira za listu Exceptions označavaju upravljanje i ponovno vraćanje podešenih opcija.

Zaključak

Debagiranje je zadatak koji se nikad na završava. Debagiranje predstavlja više od samog praćenja grešaka u Vašem programu. Pametni programeri uče da koriste debager koristeći novi projekt. Debager je alat za razvoj, a takođe i alat za traženje grešaka. Počev od današnjeg dana, imaćete u najmanju ruku, osnovno razumevanje

za korišćenje debagera. Još uvek ćete morati da provedete neko vreme koristeći debager, pre nego što se potpuno priviknete na njega, ali sada bar imate odakle da počnete.

Radionica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstite razumevanje obrađenog materijala, kao i vežbe koje Vam omogućavaju da steknete iskustvo u korišćenju gradiva koje ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A, "Odgovori na kviz pitanja".

Pitanja i odgovori

P **Moj program obično radi normalnom brzinom kada ga pokrenem iz okruženja. Sada je spor kao puž. Zašto se to dešava?**

O Više nego očigledno je da imate veliki broj tačaka prekida koje ste deaktivirali, a zaboravili jednu, ili više uslovnih tačaka prekida u okviru koda. Pređite na spisak tačaka prekida i obrišite sve tačke prekida koje trenutno ne koristite. Takođe se uverite da nemate puno promenljivih na listi za pregled.

P **Imam promenljive koje želim da vidim i u decimalnom i u heksadecimalnom formatu. Da li to mogu da uradim koristeći listu za pregled?**

O Da, prvo dodajte promenljivu na listu za pregled, zatim dva puta kliknite na promeljivu u okviru liste za pregled. Kada se pojavi okvir za dijalog Watch Properties, odaberite opciju za pregled Decimal. Sada ponovo dodajte promeljivu u listu za pregled, ali ovaj put odaberite opciju Hexadecimal. Oba elementa će biti prikazana u listi za pregled, jedan u decimalnom formatu, drugi u heksadecimalnom formatu.

P **Želeo bih da se zaustavim na tački prekida, kada promenljiva dostigne određenu vrednost i kada tačka prekida bude aktivirana određen broj puta. Da li to mogu da uradim?**

O Naravno. Upišite uslovni izraz u polje Conditional u okvir za dijalog Source Breakpoint Properties i vrednost u polje Pass Count. Kada uslov koji zadovoljava određeni broj prolaza bude jednak vrednosti polja Pass Count, program će se zaustaviti na tački prekida.

P **Prolazim kroz kod programa i naišao sam na funkciju u programu koju želim da debagiram. Kada pritisnem taster F8, tačka izvršavanja prelazi preko funkcije. Kako da uđem u funkciju?**

O Kada tačka izvršavanja programa bude na liniji koja poziva funkciju, pritisnite taster F7 (Trace Into), umesto tastera F8. Sada možete da prolazite kroz funkciju liniju po liniju.

P **Kada prolazim kroz kod, debager mi ne dozvoljava da vidim vrednosti određenih promenljivih. Zašto se to dešava?**

O Jednom rečju: optimizacija. Prevodilac optimizuje određeni deo koda i ne dozvoljava Vam da vidite vrednosti promenljivih koje su optimizovane. U suštini, po pitanju debagera, ove promenljive ne postoje. Da biste izbegli optimizaciju, isključite je na kartici Compiler okvira za dijalog opcije projekta. Ne zaboravite da ponovo uključite optimizaciju, pre nego što isporučite Vašu aplikaciju.

P **Prolazim kroz metodu liniju po liniju. Ponekad kada dođem do iskaza end u okviru metode, pritisnem taster F8 još jednom i ništa se ne dešava. Zašto?**

O Kada se završi određena metoda, Vaš program nema ništa više da radi, pa se vraća u besposleno stanje. Ustvari, od tog trenutka nema više koda kroz koji treba da prođe, pa debager vraća kontrolu programu koji se dabagira.

P **Kako da koristim prozor procesora prilikom debagiranja?**

O Odaberite opciju View➥Debug Windows➥CPU u okviru glavnog menija da biste prikazali prozor procesora. Potpuno druga stvar je ono što treba da znate da radite sa prozorom procesora.

Kviz

1. Kako možete da postavite tačku prekida na određenu liniju koda?
2. Šta je pogrešna tačka prekida?
3. Kako postavljate uslovne tačke prekida?
4. Kako možete promeniti karakteristike elementa liste za pregled?
5. Koji je najbrži način za dodavanje promenljive u listu za pregled?
6. Koji alat koristite da biste pregledali polja podataka i metode klase?
7. Kako možete ući u izvorni kod metoda kada prolazite kroz izvorni kod koristeći debager?
8. Kako možete promeniti vrednost promenljive u toku rada programa?
9. Kako možete poslati sopstvene poruke u listu događaja?
10. Šta radi opcija Integrated debugging u okviru za dijalog opcije debagera?

Vežbe

1. Učitajte program `ScratchPad` koji ste kreirali u lekciji dana 6, "Rad sa dizajnerom forme i dizajnerom menija". Postavite tačke prekida na metode `FileOpenClick` i `FileSaveClick`. Pokrenite program. Kada se zaustavi izvršavanje programa, ispitajte klase `OpenDialog` i `SaveDialog`.
2. Nastavite sa vežbom jedan. Kada prolazite kroz program i zastanete na tački prekida, ispitajte operacije programa dok prolazite kroz metode.
3. Učitajte program `DebugTst` koji ste kreirali u prethodnom delu ove lekcije. Postavite tačku prekida na metodu `WatchBtnClick`. Dodajte promenljive `S` i `X` na listu za pregled. Dodajte svaku promenljivu na listu za pregled četiri puta. Editujte svaki element liste za pregled i promenite opcije za prikaz. Pokrenite program i prođite kroz metodu da biste videli efekte na listi za pregled.
4. Dodajte uslovnu tačku prekida metodi u vežbi tri. Postavite je na liniju koja se nalazi odmah iza linije `X := Width`. Kao uslov postavite `X = 0` i pokrenite program. Šta se dešava? (Ništa se ne dešava, pošto vrednost `X` nikada ne dostigne `0`.)
5. Nastavite sa vežbom četiri, editujte tačku prekida i promenite uslov `X > 400`. Pokrenite program. Promenite veličinu prozora i kliknite mišem na dugme Watch Test. Ponovite ovaj proces nekoliko puta, menjajući pri tome veličinu prozora. Šta se dešava? (Ovaj put debager zastaje na tački prekida, kada je širina prozora veća od 400 piksela.)
6. Učitajte bilo koji program i pređite na editor koda. Postavite kursor na bilo koju liniju koda i odaberite opciju Run to Cursor u okviru menija sadržaja editora koda. Eksperimentišite sa programom, dok ne bude pronađena tačka prekida.
7. Ponovo učitajte program `DebugTst` koji ste ranije kreirali. Postavite tačku prekida na metodu `WatchBtnClick` i pokrenite program. Kada dostignete tačku prekida, koristite debager inspektor, kako biste ispitali .

Dan 11

Delphi-jevi alati i opcije

Delphi sadrži nekoliko alata koji će Vam pomoći u razvoju Vaših aplikacija. Danas ćete naučiti da koristite nekoliko Delphi-jevih alata. Tačnije, naučićete da koristite:

- Image Editor (editor slika)
- Poruke i Windows sistem za poruke
- WinSight
- Package Collection Editor (editor zbirke paketa)
- Configuring the Tools menu (meni za konfigurisanje alata)
- Delphi Environment Options (opcije Delphi-jevog okruženja)

Prvo ćete pogledati na kratko ove alate. Zatim ćete naučiti kako da dodate nove alate u meni Delphi-jevih alata (Delphi Tools). Lekciju ćemo završiti pregledom okvira za dijalog opcija Delphi-jevog okruženja (Delphi Environment Options).

Pored alata koji su navedeni, Delphi sadrži i dodatne alate za baze podataka, kao što su Database Desktop, BDE Administrator, SQL Builder i SQL Explorer. Današnja lekcija bi bila suviše duga, ukoliko bi obradili i navedene alate.

Korišćenje editora slika (Image Editor)

Delphi-jev editor slika je alat koji Vam omogućava da kreirate i editujete bitmape (`.bmp`), ikone (`.ico`) i kursore (`.cur`). Takođe možete kreirati resursni projekt koji sadrži više bitmapa, ikona i kursora u jednoj jedinoj resursnoj datoteci (`.res`).

Resursne datoteke zatim mogu biti dodate u Vaš Delphi projekt, tako da po potrebi možete koristiti resurse. Slika 11.1 prikazuje editor slika u trenutku editovanja bitmape.

Slika 11.1
Editor slika
(Image Editor)

NAPOMENA Sve Windows slike su bitmape, bilo da su prave Windows datoteke koje sadrže bitmapiranu sliku (.bmp), ikone, odnosno kursori. U ovoj lekciji ću sve slike nazvati bitmapama.

Editor slika radi samo sa Windows slikama koje su bitmape. Drugi formati kao što su: PCX, TIFF, JPEG i GIF nisu podržani.

Editor slika možete pokrenuti dva puta kliknuvši mišem na ikonu editora slika u Delphi-jevom folderu, odnosno birajući opciju Tools➥Image Editor u okviru glavnog menija Delphi-ja. Editor slika je zaseban program, pa ne morate da ga pokrećete iz Delphi-jevog okruženja.

Boje prednjeg plana (foreground) i pozadina (background)

Editor slika Vam omogućava da kreirate slike sa dve boje, 16 boja i kod bitmapiranih datoteka, slike sa 256 boja. Kada crtate bitmapu, možete odabrati bilo koju boju koja Vam je dostupna. U donjem levom uglu editora slika se nalaze dva okvira koja sadrže trenutne boje za prednji plan i pozadinu. (Boja prednjeg plana je predstavljena u krajnjem levom okviru.)

Kada koristite alat za crtanje, možete crtati, ili sa bojom prednjeg plana, ili sa bojom pozadine. Da biste crtali koristeći boju prednjeg plana, treba da koristite levi taster miša. Na primer, ukoliko odaberete alat za popunjeni četvorougao i nacrtate četvorougao na bitmapi, četvorougao će biti ispunjen bojom prednjeg plana. Da biste nacrtali ispunjen četvorougao koristeći boju pozadine, treba da koristite desni taster miša za razvlačenje četvorougla. Na isti način radi i većina drugih alata za crtanje.

NAPOMENA Alat Text koristi samo boju prednjeg plana. Ne možete postaviti tekst sa bojom pozadine. Ukoliko želite da koristite boju pozadine, treba da promenite boju prednjeg plana, a zatim postavite Vaš tekst.

Da biste promenuli boju prednjeg plana, kliknite levim tasterom miša na boju koju želite da odaberete u okviru palete boja. (Paleta boja se nalazi duž donje ivice prozora editora slika.) Kada odaberete novu boju prednjeg plana, kvadrat koji predstavlja boju prednjeg plana će prikazati novoizabranu boju. Da biste promenili boju pozadine, kliknite na željenu boju u okviru palete komponenti desnim tasterom miša.

NAPOMENA Ukoliko editujete sliku sa 256 boja, paleta boja će imati dugme za pomeranje na obe strane, kako biste mogli da pogledate sve dostupne boje.

Boje koje se pojavljuju u paleti su definisane na osnovu bitmape, ukoliko učitavate bitmapu koja već postoji. Ukoliko počinjete sa radom na novoj slici sa 256 boja, biće korišćena generička paleta sa 256 boja.

U alatu Eyedropper takođe možete podešavati boje prednjeg plana i pozadine. Alat Eyedropper Vam omogućava da podesite boje prednjeg plana i pozadine biranjem boje koja se već koristi na slici.

Da biste podesili boju prednjeg plana koristeći Eyedropper, odaberite ovaj alat sa palete alata, postavite vrh kursora alata Eyedropper na deo slike koji sadrži boju koju želite da koristite, a zatim kliknite levim tasterom miša. Boja prednjeg plana će biti promenjena u boju na kojoj se nalazi kursor miša. Da biste podesili boju pozadine, kliknite desnim tasterom miša, umesto levim. Alat Eyedropper je nezamenljiv kada želite da ponovo odaberete boje koje ste koristili u toku rada sa slikom.

NAPOMENA Boje prednjeg plana i pozadine, mogu raditi drugačije nego što je to slučaj kod drugih editora bitmapa koje ste koristili. Na primer, u nekim editorima bitmapa, okvir popunjenog četvorougla je nacrtan bojom prednjeg plana, a ispunjen bojom pozadine. Kod editora slika, popunjeni objekti nemaju okvir druge boje. Popunjeni četvorougao ima boju prednjeg plana, ili pozadine.

Transparentne i inverzne boje

U slučaju ikona i kursora, takođe možete odabrati i transprentne boje (reč boja možete ovde shvatiti relativno). Kada koristite transparentne boje, pozadina iza ikona se pokazuje bez obzira da li se transparentna boja koristi. Ovo može biti Winldows pozadina, odnosno može biti naslovna traka Vaše aplikacije.

Da li ćete koristiti transparentne boje zavisi od Vašeg ličnog ukusa, odnosno ikone koju ste već kreirali. U slučaju kursora, skoro uvek ćete koristiti transpasrentne boje kao pozadinu ikone. Retko se dešava da kursor bude popunjen.

Izbor inverzne boje omogućava da pozadina pod ikonom bude inverzna (kao inverzni video). Korišćenje inverzne boje nije uobičajeno, ali ona ipak postoji kada Vam zatreba.

I transparentna i inverzna boja su prikazane pored palete boja u slučaju kada editujete ikone i kursore. Ove boje su predstavljene kvadratićima u boji kroz koje prolazi kriva linija.

NAPOMENA Generički, nova ikona i novi kursor imaju pozadinu popunjenu transparentnom bojom.

Alati za crtanje editora slika

Alati za crtanje editora slika su slični kao i kod većine programa za crtanje. Pošto su ovo uobičajeni alati za crtanje, neću posebno obrađivati svaki od njih. Petnaest minuta vežbanja sa editorom slika će imati više efekta nego bilo šta što ću Vam objasniti. Pokrenite editor slika i eksperimentišite sa njim. Sačekaću Vas.

Na vrhu palete alata editora slika možete pronaći alate Marquee (marker) i Lasso (laso). Oba ova alata rade na potpuno isti način, pa ću ih zajedno objasniti. Oba alata Vam omogućavaju da odaberete oblast na slici. Alat Marquee se koristi da definiše pravougaonu oblast. Odaberite ovaj alat i razvucite pravouganik po slici. Kada prestanete sa razvlačenjem, pravougaonik se nalazi oko oblasti koju treba da označi. Alat Lasso radi na sličan način, ali Vam omogućava da definišete proizvoljnu oblast. Oblast koju obuhvata laso je popunjena kosim linijama kako bi mogli da je bolje uočite.

SAVET Kada koristite alat Lasso, ne morate da zatvorite oblast. Kada otpustite dugme miša, editor slika automatski zatvara region crtajući liniju za vezu između početne i krajnje pozicije.

Kada definišete oblast, možete da isečete, ili kopirate deo slike koji se nalazi u oblasti, a zatim je zalepite na drugi deo slike, odnosno na drugu sliku sa kojom radite (možete otvoriti više bitmapa istovremeno). Kada odaberete opciju Edit➥Paste u okviru glavnog menija, deo slike unutar naznačene oblasti će biti postavljen u gornjem levom uglu bitmape sa odgovarajućom oznakom oko isečka. Zatim možete taj deo slike prevući do željenog mesta na slici.

Kada postavite kursor miša unutar markera, oblik kursora se promeni u oblik ruke. Kada vidite kursor u obliku ruke, markirani deo bitmape možete prevući na dugu poziciju i spustiti je. Bitmapu možete pomerati sve dok ne pronađete odgovarajuće mesto gde želite da je postavite. Pošto ste pronašli odgovarajuće mesto, ponovo kliknite na deo slike izvan marikrane oblasti koja se nalazi u okviru tekuće slike.

SAVET Editor slika ima prečice za isecanje i lepljenje. Kreirajte oblast Maquee, ili Lasso, postavite kursor miša unutar oblasti i povlačite. Deo slike unutar oblasti će se pomerati zajedno sa kursorom miša.

Kada isečete oblast, ili je pomerite prevlačenjem, trenutna boja pozadine će popuniti oblast koju originalni deo slike trenutno zauzima. Pozadina se prikazuje kroz rupu koju ste kreirali operacijom isecanja.

SAVET Možete kopirati delove bitmape sa jedne bitmape na drugu koristeći opcije za isecanje i lepljenje. Prvo otvorite obe slike u okviru editora slika. Postavite marker oko dela slike koji želite da kopirate, a zatim odaberite opciju Edit➥Copy u okviru glavnog menija. Pređite na drugu sliku i odaberite opciju Edit➥Paste u okviru glavnog menija. Ukoliko je potrebno, pomerite zalepljenu sliku.

NAPOMENA Ukoliko prilikom lepljenja imate već odabranu oblast i označenu markerom, zalepljena slika će se prilagoditi veličini markera.

Alat Eraser radi obrnuto od ostalih alata, sa stanovišta upotrebe levog i desnog tastera miša. Kod alata Eraser levo dugme miša crta koristeći boju pozadine, a desno dugme miša crta koristeći boju prednjeg plana.

Alat Text Vam omogućava da postavite tekst na sliku. Tekst će biti nacrtan korišćenjem aktuelnih opcija za podešavanje teksta. Opcije za podešavanje teksta se mogu menjati klikom miša na opciju Text u okviru glavnog menija. Tada možete podesiti poravnanje teksta (levo, desno, ili centriranje teksta), odnosno font (izgled i tip fonta). Da biste promenili izgled i tip fonta, odaberite opciju Text➥Font u okviru glavnog menija. Sada možete odabrati novi tip fonta, odnosno definisati oblike kao što su podebljeno, italik, podvučeno i slično.

Ostali alati za crtanje su dovoljno jasni sami po sebi. Kao što sam napomenuo i ranije, provedite neko vreme koristeći editor slika i naučićete sve što je potrebno da znate o ovim alatima.

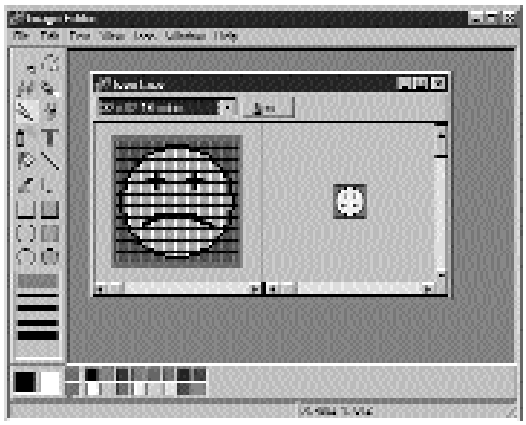
SAVET Kada crtate pravougaonike, možete pritisnuti i držati taster Shift, kako biste napravili pravougaonik. Isto tako možete nacrtati savršen krug birajući elipsu, ili popunjenu elipsu i u toku prevlačenja miša držati pritisnut taster Shift. Korišćenje Shift tastera kod alata Line Vam omogućava da crtate prave linije (vertikalne i horizontalne, odnosno uglove pod 45 stepeni).

Zumiranje

Editor slika Vam omogućava da zumirate sliku kako biste mogli bliže da pogledate Vašu bitmapu. Možete zumirati, ili korišćenjem alata Zoom, ili korišćenjem menija View. Da biste zumirali na određeni deo Vaše slike korišćenjem alata Zoom, prvo odaberite ovaj alat sa palete alata, a zatim prevlačite pravougaonik oko dela slike koji želite da uveličate. Uvećanje će se menjati u zavisnosti od veličine pravougaonika koji ste kreirali prevlačenjem miša. Nakon toga ćete moći da vidite dovoljno kako biste izmenili sitnije detalje Vaše bitmape.

Da biste zumirali sliku korišćenjem menija, odaberite opciju View➥Zoom In, odnosno pritisnite tastere Ctrl+I. Kada odaberete opciju Zoom In u okviru menija, slika će biti uvećana na unapred definisanu veličinu. Da biste vratili zumirani sliku korišćenjem menija, odaberite opciju View➥Zoom Out (Ctrl+U), odnosno View➥Actual Size (Ctrl+Q).

Kada kreirate kursor, ili ikonu, editor slika će prikazati podeljenu sliku. Slika 11.2 prikazuje prozor editora slika u toku kreiranja ikone.



Slika 11.2
Editovanje ikone

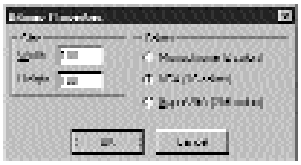
Iako možete da zumirate bilo koju stranu podeljenog prozora, uglavnom ćete raditi sa zumiranom kopijom na levoj strani i aktuelnom veličinom slike na desnoj strani, kao što je to prikazano na slici 11.2.

Paleta Line Width

Paleta za debljinu linije je prikazana direktno ispod palete sa alatima. U zavisnosti od trenutno odabranog alata, palete debljine linije će možda prikazati debljine linija, odnosno oblike četki koje možete odabrati. Da biste odabrali debljinu linije, kliknite na jednu od pet debljina linija koje su prikazane. Određene operacije za crtanje će koristiti nove debljine linija kada ih izmenite. Slično je i sa oblikom četke; potrebno je da kliknete na oblik četke koji želite da koristite. Ukoliko se vratite na sliku 11.1, moći ćete da vidite kako paleta za debljinu linija pokazuje oblike četki.

Rad sa bitmapiranim datotekama

Možete kreirati bitmapu od početka, odnosno učitati već postojeću i izmeniti je. Da biste otvorili već postojeću bitmapiranu datoteku, odaberite opciju File➥Open u okviru glavnog menija (bitmapa datoteke imaju nastavak BMP). Da biste kreirali novu bitmapiranu datoteku, odaberite opciju File➥New u okviru glavnog menija, a zatim odaberite opciju Bitmap File iz padajućeg menija. Okvir za dijalog Bitmap Properties će biti prikazan onako kao što možete videti na slici 11.3.



Slika 11.3
Okvir za dijalog Bitmap Properties

Ovde možete podesiti inicijalnu veličinu bitmape (u pikselima), kao i broj boja. Možete kreirati bitmapu sa 2, 16, odnosno 256 boja.

NAPOMENA Editor slika ne podržava bitmape sa više od 256 boja.

Odaberite veličinu i broj boja koji želite, a zatim kliknite na dugme OK. Prazna bitmapa će biti prikazana u prozoru editora. Nacrtajte šta želite na bitmapu. Kada završite odaberite opciju File➥Save, odnosno File➥Save As da biste snimili bitmapu na disk.

Svaki put kada radite sa bitmapiranim datotekama, editor slika u okviru glavnog menija, sadrži opciju pod nazivom Bitmap. Ovaj meni ima samo jednu opciju pod nazivom Image Properties. Izborom opcije Bitmap➥Image Properties će biti prikazan okvir za dijalog sa karakteristikama bitmape, kao kada ste kreirali novu datoteku bitmape. Okvir za dijalog sa karakteristikama bitmape Vam omogućava da promenite veličinu i broj boja bitmape. Odaberite novu širinu, visinu, odnosno novi broj boja, a zatim kliknite na dugme OK.

NAPOMENA Postoji jedna razlika između okvira za dijalog sa karakteristikama bitmape prilikom prikazivanja već postojeće bitmape i okvira za dijalog koji se pojavljuije prilikom kreiranja nove bitmape. Kada se prikazuje sa već postojećom bitmapom okvir za dijalog sa karakteristikama bitmape sadrži polje za potvrdu sa nazivom Stretch. Ovo polje za potvrdu se koristi kada treba promeniti veličinu bitmape. Ukoliko je opcija Stretch isključena, bitmapa neće biti razvučena (na manju, ili veću površinu), ukoliko se menja veličina bitmape. Ukoliko je opcija Stretch uključena, bitmapa će biti razvučena tako da popuni sav prostor izmenjene veličine bitmape. Razvlačenje bitmape nije egzaktna nauka, pa ponekad rezultati razvlačenja mogu biti nezadovoljavajući.

Sve u svemu, ne postoji mnogo toga što možete uraditi sa bitmapiranim datotekama. Iako je editor slika dobar za jednostavne bitmape, verovatno neće biti adekvatan za finiju grafiku. Ukoliko su Vam potrebne bitmape dobrog kvaliteta razmislite o porudžbini paketa za editovanje slika, odnosno angažovanju umetnika koji radi na računaru koji će kreirati Vaše bitmape.

SAVET Nemojte zaboraviti da proverite preporuke umetnika koji rade na računaru. Ovi ljudi znaju svoj posao bolje od najtelentovanijih programera i često su veoma realni po pitanju cena. Ponovite ovo deset puta: "Ja sam programer, a ne umetnik." (Uredu, možda su neki od Vas talentovani za oba posla.)

Rad sa ikonama

Kreiranje ikona je isto tako umetnost, ali ikone nisu ni približno zahtevne kao bitmape u punom koloru. Možete uglavnom kreirati sopstvene ikone, ali dobre ikone još uvek zahtevaju veštinu. Ukoliko se vratite na sliku 11.2 možete videti ikonu koja je prikazana u editoru slika u toku rada.

SAVET Učitajte datoteku ikona iz bilo koje datoteke koju možete pronaći i zumirajte je kako biste mogli da vidite način na koji su kreirane ikone koje dobro izgledaju. Kreiranje 3D ikona zahteva praksu (mada uvek nešto ispadne pogrešno).

Ikona u 32-bitnom Windows operativnom sistemu je ustvari predstavljena sa dve ikone. Velike ikone su dimenzija 32x32 piksela. Velike ikone mogu biti postavljene u okvir za dijalog, kao što je okvir About (opis programa). To su iste ikone koje Windows koristi za kreiranje prečica do Vaših aplikacija. Dodatno, velike ikone se koriste u programu Windows Explorer, kada je spisak datoteka podešen da pokazuje velike ikone.

Male ikone su dimenzija 16x16 piksela, a koriste se u Windows operativnom sistemu za traku sa naslovom aplikacije, na Windows traci za poslove (taskbar), u okviru za dijalog File Open i u programu Windows Explorer, kada je pogled podešen na male ikone. I velike i male ikone se nalaze u istoj datoteci ikona (`.ico`).

NAPOMENA Ne morate da kreirate i velike i male ikone. Ukoliko isporučujete samo velike ikone, Windows će umanjiti veliku ikonu, ukoliko je potrebno da je prikaže kao malu. Ponekad rezultat smanjivanja ikone neće biti onakav kakav ste očekivali i možda neće imati dovoljan kvalitet za Vaš ukus. U takvim slučajevima, možete kreirati i male ikone, tako da možete kontrolisati izgled Vaše aplikacije, umesto da se oslanjate na Windows koji bi trebao da uradi pravu stvar.

Kreiranje novog resursa za ikone

Da biste kreirali nov resurs za ikone, odaberite opciju File➥New u okviru glavnog menija, a zatim odaberite opciju Icon u okviru padajućeg menija. Kada kreirate novu ikonu u editoru slika, videćete okvir za dijalog sa karakteristikama ikona (Icon Properties), kao što je to prikazano na slici 11.4.

Slika 11.4
Okvir za dijalog Icon Properties

Ovaj okvir za dijalog Vam omogućava da odaberete ikonu koju kreirate (bilo veliku, ili malu) kao i broj boja koje ćete koristiti za ikonu. Generička vrednost je kreiranje standardne ikone (velike ikone) i korišćenje 16 boja. (U principu, ikone sa dve boje se retko koriste. Kada ste ih poslednji put videli?)

NAPOMENA Čak iako kreirate i velike i male ikone, morate odabrati jednu, odnosno drugu za početak. Na primer, ako kreirate novu ikonu, trebalo bi da počnete sa velikom ikonom. Nakon što kreirate veliku ikonu, možete kreirati malu.

Kada editujete ikonu, traka za meni sadrži opciju pod nazivom Icon. Meni Icon sadrži opcije pod nazivom New, Delete i Test. Opcije menija New, Vam omogućavaju da kreirate novu veliku, ili malu ikonu. Na primer, ukoliko ste već kreirali veliku ikonu, možete odabrati opciju Icon➥New u okviru glavnog menija da biste kreirali malu ikonu.

SAVET Prozor editor ikona sadrži dugme pod nazivom New kojim se isto tako može kreirati nova ikona, a i brže je od korišćenja glavnog menija (pogledajte sliku 11.2).

Kada odaberete opciju New da biste kreirali drugu ikonu, okvir za dijalog sa karakteristikama ikona će biti prikazan kao i u prethodnom slučaju. Ukoliko ste već kreirali veliku ikonu, generički će biti odabrana mala ikona i sve što treba da uradite je da kliknete na dugme OK. Prozor editora će se izmeniti da prikaže novu praznu ikonu. Editor slika Vam neće dozvoliti da kreirate ikonu koja već postoji u datoteci ikona.

SAVET Kada postoje obe ikone, možete prelaziti sa jedne na drugu korišćenjem kombo okvira u gornjem delu prozora editora ikona.

Opcije za editovanje ikona

Opcija Delete menija Icon Vam omogućava da obrišete ili veliku, ili malu ikonu iz resursa ikona. U resursu ikona ne možete obrisati poslednju ikonu.

Opcija Test u okviru menija Icon prikazuje okvir za dijalog Icon Tester, koji Vam pokazuje kako će ikona izgledati na ekranu. Slika 11.5 prikazuje okvir za dijalog Icon Tester u toku rada.

Slika 11.5
Okvir za dijalog Icon Tester

Okvir za dijalog Icon Tester Vam omogućava da promenite boju pozadine, tako da možete isprobati kako različite boje pozadine utiču na izgled Vaše ikone. Ukoliko trenutno editujete veliku ikonu, u okviru za dijalog će biti prikazana velika ikona. Ukoliko editujete malu ikonu, okvir za dijalog Icon Tester će prikazati malu ikonu.

Rad sa kursorima

Rad sa kursorima se ne razlikuje od rada sa ikonama. Kursor ima samo dve boje: belu i crnu. (Kursori sa više boja i animirani kursori nisu podržani u editoru slika.) Nacrtajte kursor onako kako želite da izgleda.

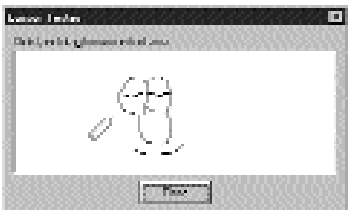
NAPOMENA Karakteristično za editor kursora je da boja koja je postavljena kao sistemska boja pozadine ujedno predstavlja i transparentnu boju, dok je kod editora ikona transparentna boja predstavljana tamno zelenom bojom. Ukoliko imate boju pozadine Windows-a postavljenu na veoma svetlu boju, možda će Vam biti teško da uočite šta je transparentno, a šta je belo. Ukoliko teško možete da razlikujete boju pozadine od bele boje, postavite boju Vaše pozadine na neku drugu vrednost (na primer, tamno zeleno).

Kao i kod editovanja bitmapiranih datoteka i ikona, meni editora slika prikazuje opciju menija pod nazivom Cursor onda kada editujete kursor. Ova opcija menija ima dve podopcije: Set Hot Spot i Test.

Opcija Set Hot Spot Vam omogućava da definišete tačku pokazivanja kursora (hot spot). Tačka pokazivanja (hot spot) predstavlja određeni piksel na kursoru koji Windows koristi da bi saopštio koordinate na kojima se kursor nalazi. Na primer, na kursoru koji izgleda kao krst, tačka pokazivanja se nalazi u centru preseka linija. Za kursor u obliku strelice, tačka pokazivanja je postavljena na sam vrh strelice. Da biste postavili tačku pokazivanja kursora, odaberite opciju Cursor➥Set Hot Spot u okviru glavnog menija. Pojaviće se okvir za dijalog Set Cursor Hot Spot, gde možete upisati x i y koordinate tačke pokazivanja kursora.

SAVET Morate upisati tačne x i y koordinarte tačke pokazivanja kursora. Da bi Vam bilo lakše pre nego što podesite tačku pokazivanja, podesite kursor za editovanje na tačku koju želite da definišete kao tačku pokazivanja. Statusna traka editora slika će prikazati x i y koordnate tačke na kojoj se nalazi kursor. Zapišite ove koordinate, a zatim odaberite opciju Cursor➥Set Hot Spot u okviru glavnog menija, a zatim unesite x i y koordinate.

Opcija Test u okviru menija Cursor Vam pruža priliku da isprobate Vaš novi kursor. Odaberite opciju Cursor➥Test u okviru glavnog menija, nakon čega će biti prikazan okvir za dijalog Cursor Tester, kao što to možete videti na slici 11.6.



Slik 11.6
Testiranje kursora

Pritisnite bilo koji taster miša kako biste crtali u prozoru Cursor Tester. Ukoliko još uvek niste definisali tačku pokazivanja kursora, verovatno ćete zapaziti da je tačka pokazivanja trenutno postavljana u gornji levi ugao kursora; što je generičko mesto

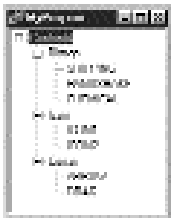
tačke pokazivanja. Tačku pokazivanja možete uvek postaviti na mesto gde bi to bilo najlogičnije korisnicima Vaše aplikacije.

Meniji sadržaja editora slika

Editor slika ima menije sadržaja za svaki mod editovanja (Bitmap, Cursor i Icon). Možda ćete se setiti da se desni taster miša koristi za crtanje, tako da meni sadržaja ne možete pozvati klikom na desni taster miša u trenutku kada se kursor nalazi na slici. Da biste prikazali meni sadržaja editora slika, kliknite desnim tasterom miša kada je kursor na prozoru editora, ali se nalazi van slike. Meni sadržaja sadrži iste opcije koje se mogu pronaći na posebnim menijima, kao što ste mogli da vidite u prethodnim poglavljima.

Kreiranje resursnih projekata

Editor slika Vam omogućava da kreirate i datoteke resursnih projekata u kojima će se nalaziti sve Vaše bitmape, ikone i kursori. Da biste kreirali resursni projekat, odaberite opciju File➥New u okviru glavnog menija, a zatim odaberite opciju Resource File u okviru padajućeg menija, nakon čega će biti prikazan prozor projekta. Prozor projekta je predstavljen u obliku stabla na kome su prikazane bitmape, ikone i kursori u okviru projekta. Slika 11.7 prikazuje prozor projekta koji je uzet kao primer.



Slika 11.7
Prozor projekta editora slika

Prozor projekta sadrži meni sadržaja koji se može koristiti za kreiranje novih resursa, editovanje resursa, promenu naziva resursa, odnosno brisanje resursa. Opcije menija sadržaja su isto tako duplirane u okviru glavnog menija pod opcijom Resource.

Kada snimate resursni projekat, editor slika će ga prevesti u binarnu resursnu datoteku (`.res`). Zatim možete dodati binarne resursne datoteke u Vaš Delphi projekt.

Kreiranje novih resursa

Da biste kreirali nove resurse za Vaš projekat, odaberite opciju New u okviru menija sadržaja prozora projekta, odnosno odaberite opciju Resource➥New u okviru glavnog menija. Zatim možete odabrati kreiranje nove bitmape, ikone, odnosno

kursora kao što ste to činili kreirajući posebne resursne datoteke. Prozor za editovanje resursa će biti prikazan u zavisnosti od tipa resursa koji ste odabrali.

Editovanje resursa

Kada kreirate resurs, možete poželeti da ga editujete kako bi vršili izmene. Da biste editovali resurs u okviru resursnog projekta, pronađite resurs na stablu projekta, a zatim dva puta kliknite na naziv resursa. Prozor resursnog editora će biti prikazan zajedno sa resursom koji želite da editujete.

Promena naziva resursa

Promena naziva resursa se može postići direktnim editovanjem naziva u okviru stabla. Da biste odabrali element kome želite da promenite naziv, kliknite mišem na njega, a zatim ponovo kliknite mišem, kako biste editovali naziv. Isto tako možete odabrati opciju Rename u okviru menija sadržaja kako biste započeli editovanje u okviru samog stabla. Nakon što upišete novi naziv resursa, pritisnite taster Enter, odnosno kliknite na drugi element na stablu i naziv resursa će biti izmenjen.

Brisanje resursa

Da biste obrisali resurs iz projekta resursa, kliknite na naziv resursa u okviru stabla projekta, kako biste odabrali resurs koji želite da obrišete, a zatim odaberite opciju Delete u okviru menija sadržaja. Nakon izbora opcije će se pojaviti upit za potvrdu brisanja elementa; ukoliko potvrdno odgovorite, element će biti obrisan. Kod brisanja resursa ne postoji opcija za vraćanje, pa se prethodno uverite da Vam resurs neće trebati pre nego što ga obrišete.

Dodavanje resursa iz drugih resursnih datoteka

Na nesreću, ne postoji jednostavan način za dodavanje resursa koji se nalaze u izdvojenim datotekama u trenutno aktivnom projektu resursa. Jedino što možete da uradite je otvaranje projektne datoteke, a zatim otvaranje zasebne bitmape, ikone, odnosno datoteke kursora, koja sadrži resurs koji želite da dodate u projekat. Pređite na zasebnu datoteku i odaberite opciju Edit➥Select All u okviru glavnog menija, kako biste odabrali resurs; zatim odaberite opciju Edit➥Copy da biste kopirali objekat u Clipboard. Kreirajte novi resurs u projektu resursa. Nakon što bude prikazan prozor za editovanje resursa, odaberite opciju Edit➥Paste u okviru glavnog menija, kako biste zalepili snimljeni resurs na željeno mesto.

Ukoliko je objekat, koji ste dodali u resursni projekat, bitmapa, uverite se da ste odabrali odgovarajuće atribute za visinu, širinu i broj boja bitmape. Ovi atributi će Vam biti potrebni prilikom kreiranja nove bitmape u projektu resursa.

Editor slika nije najsavremeniji editor, ali je dovoljno dobar za većinu zadataka kojima se kreiraju slike. Veoma je jednostavan za korišćenje i adekvatan je za kreiranje većine ikona i kursora.

WinSight: špijuniranje Windows-a

WinSight je pomoćni program koji Vam omogućava da špijunirate Windows-e. WinSight će Vam pokazati svaku aplikaciju koja trenutno radi, kao i prozor koji radi u okviru aplikacije. (Zapamtite: kontrole su takođe prozori.) WinSight Vam prikazuje svaku poruku koju generiše Windows operativni sistem (svaka poruka koja je poslata prozoru će biti prikazana).

Možete odabrati da budu prikazane sve poruke, odnosno samo poruke koje su poslate određenom prozoru. Da biste pokrenuli WinSight, koristeći Windows Explorer pronađite datoteku `WS32.EXE`, a zatim kliknite mišem na ikonu (nalazi se u direktorijumu `Delphi 4\Bin`). WinSight je poput editora slika zaseban program koji možete pokrenuti i van Delphi-jevog okruženja. Slika 11.8 prikazuje kako WinSight špijunira Windows Explorer.

Slika 11.8
WinSight u toku rada

Kao što ste mogli da vidite na slici 11.8, prozor programa WinSight je podeljen na dva dela. Gornji deo prikazuje spisak aktivnih prozora, a donji deo prikazuje poruke koje su poslate određenom prozoru, odnosno svim prozorima. Veličine panoa možete podešavati linijom za promenu veličine koja se nalazi između ova dva panoa. Generički, ovaj prozor je podeljen hotizontalno, ali prozor možete podeliti i vertikalno. Da biste promenili izgled prozora, odaberite opciju Split Horizontal, odnosno Split Vertical u okviru menija View.

Pre nego što ispitamo detaljnije ova dva panoa, ukratko ćemo obraditi Windows poruke.

Windows sistem za poruke

Špijuniranje Windows-a nije posebno korisno ukoliko ne znate šta znače poruke. (Uzbuđenje pravog špijuna će ubrzo splasnuti.) Ukoliko želite da razumete sve informacije koje Vam prenosi WinSight morate biti prilično dobar Windows programer.

Delphi je odličan alat za veoma brzo pisanje pravih nezavisnih Windows aplikacija. Ukoliko postoji mana kod ovakvog tipa programskog okruženja, to bi bila nemogućnost da naučite šta je sve potrebno Windows programu da bi mogao da radi.

Ono što omogućava Windows programu da radi, su poruke; mnoštvo poruka. Windows šalje poruke prozoru da bi ga obavestio da treba nešto da se uradi, odnosno da naznači prozoru da se neki događaj aktivirao. Na primer, kada prozor treba ponovo iscrtati, Windows šalje poruku `WM_PAINT`. Poruka `WM_PAINT` poručuje prozoru da se ponovo iscrta.

Kada treba promeniti veličinu prozora, Windows šalje poruku `WM_WINDOWPOSCHANGING`, kako bi naznačio prozoru da se njegova veličina i/ili pozicija menjaju. Iza ove poruke slede poruke `WM_WINDOWPOSCHANGED` i `WM_SIZE`, nakon čega se menja veličina prozora. U prosečnom Windows okruženju ova poruka se šalje nekoliko desetina, odnosno nekoliko stotina puta svake sekunde.

Windows može poslati aplikaciji više od sto različitih poruka. Ove poruke sam delimično obradio u lekciji dana 5, "Model vizuelnih komponenti", kada je bio obrađen VCL. Većina događaja na koji Delphi program odgovara su Windows poruke. Događaj `OnCreate` se generiše kao odgovor na poruku `WM_CREATE`, događaj `OnSize` se generiše kao odgovor na poruku `WM_SIZE`, a događaj `OnMouseDown` odgovara na poruke `WM_LBUTTONDOWN` i `WM_RBUTTONDOWN`. Lista se produžava unedogled. Delphi Vam omogućava da obrađujete ove poruke na višem nivou, oslobađa Vas detalja i omogućava Vam da se posvetite važnijim delovima aplikacije.

Možda ćete poželeti da naučite više o stvarima koje pokreću Windows. Velika korist od Delphi-ja je što možete da pišete Windows programe, a da postepeno učite stvari koje se dešavaju na niskom nivou.

Neki veoma dobri Windows programeri bi tvrdili kako prvo treba da naučite pisanje Windows programa u jeziku C, kao što se to nekad radilo. Ova teorija tvrdi da se na taj način uči osnova koja je potrebna Windows programeru da bude efikasniji u bilo kom programskom okruženju. Iako bih mogao da se složim da je ovakav scenario optimalan, takođe mogu da shvatim da većina programera danas nema dovoljno vremena za ovakav proces učenja.

Nakon što smo videli primere Windows poruka, možemo se vratiti na WinSight i pogledati kako ovaj pomoćni program radi.

Stablo prozora

Gornji pano programa WinSight se naziva stablo prozora (Window Tree). Stablo prozora prikazuje sve trenutno otvorene prozore. Ovaj pano takođe sadrži detalje o prozorskim klasama određenog prozora. Detalji izgledaju ovako (izuzev što se na ekranu pojavljuju u jednoj liniji):

```
Overlapped 00000D74 {ExploreWClass}
  EXPLORER.EXE (730,14)-(935,460) 'Exploring - aTemplate'
```

Prvi element ove linije prikazuje stil prozora. U ovom slučaju to je preklapajući (overlapped) prozor (`WS_OVERLAPPED` za stare Windows hakere). Ostale mogućnosti su prozor potomak i samostalni prozor. (Postoje i drugi tipovi prozora, ali preklapajući prozor potomak i samostalni prozor su najčešći.) Druga kolona pokazuje upravljač (`HWND` - window handle) tekućeg prozora, koji je u vezi sa karakteristikom `Handle` VCL prozorske komponente.

U vitičastim zagradama možete videti naziv klase prozora. Ovo je naziv klase koju aplikacija koristi da bi definisala tekući prozor u okviru Windows-a. Često prozori mogu deliti jedan naziv klase. Na primer, uobičajena kontrola tipa dugme, za naziv klase koristi reč `Button`. U istom trenutku u različitim aplikacijama može postojati desetine dugmadi, a da sva dugmad pripadaju istoj prozorskoj klasi. U slučaju Delphi aplikacija, forme i komponente koje pokazuju VCL klasu predstavljaju datu komponentu. Za komponentu tipa dugme, WinSight će kao naziv klase prikazati `TButton`.

U toku razvoja Delphi-ja 1, šifrovani naziv projekta je bio Delphi. Prodajni naziv proizvoda je trebao da bude AppBuilder. Zatim se naziv Delphi održao i postao službeni naziv proizvoda. Ukoliko istražujete Delphi-jevo okruženje koristeći program WinSight, otkrićete da glavni Delphi-jev prozor sadrži klasu pod nazivom `TAppBuilder`, na osnovu čega je i nastala ova priča.

Iza naziva klase se nalazi naziv modula procesa na osnovu kog je kreiran prozor. Često je modul izvršni program. U ovom primeru naziv modula je `EXPLORER.EXE`. Nakon naziva modula, možete videti veličinu i poziciju prozora. Na kraju, možete videti tekst koji se nalazi u okviru prozora. Za prekrivajuće prozore, ovo obično predstavlja tekst koji se pojavljuje u naslovnoj traci. Za druge tipove prozora, ovaj tekst predstavlja nešto drugo u zavisnosti od tipa prozora. Na primer, za dugme, tekst predstavlja natpis na dugmetu.

Uobičajena praksa kod komercijalnih aplikacija je kreiranje skrivenih prozora koji izvršavaju određeni zadatak u okviru aplikacije. Skriveni prozori će nakon naziva modula imati prikazan tekst `hidden`. Pošto je prozor sakriven, ne postoji podatak o veličini i poziciji prozora, a isto tako ne postoji ni tekst prozora.

Stablo prozora je složeno po hijerarhiji. Na vrhu prozora se nalazi Windows Desktop (Windows radna površina). Svi prozori kreirani ispod radne površine su njeni potomci. Na primer, izvršna datoteka se pojavljuje ispod noda radne površine. Dati pro-

zor može imati prozore potomke. Linije mogu povezivati prozore roditelje, njihove potomke i elemente.

Ukoliko pogledate sliku 11.8 možete uočiti da se na levoj strani svakog elementa u okviru stabla prozora pojavljuje znak u obliku dijamanta. Ukoliko prozor sadrži potomke, unutar dijamanta će se pojaviti, ili plus, ili minus znak. Ukoliko dijamant sadrži znak plus, znači da je nod skupljen i da se može proširiti kako bi se videli prozori potomci. Ukoliko nod ima znak minus, sam nod je već raširen. Ukoliko kliknete mišem bilo gde na levoj strani elementa, nod možete raširiti, odnosno skupiti. Prazni dijamanti označavaju prozore bez potomaka.

NAPOMENA Ukoliko je određeni prozor aktiviran u okviru stabla prozora, dijamant pored elementa će blinkati svaki put kada prozor prima poruku od Windows-a.

Prozor za praćenje poruka

Prozor za praćenje poruka prikazuje zasebne poruke koje generiše Windows. Karakteristična stavka koja se pojavljuje u ovom prozoru izgleda ovako:

```
000684:00000854 {TMemo} WM_KEYDOWN Dispatched 48h 72d VK_H Scan 23h
Down
```

Pošto se detalji poruke razlikuju, nema efekta da se poruka detaljnije analizira. U ovom slučaju, memo komponenta je primila poruku `WM_KEYDOWN` sa parametrom `VK_H`. Drugim rečima, korisnik je pritisnuo taster h u trenutku kada se kursor nalazio unutar memo komponente. Kao što ste mogli da vidite, neke informacije koje se nalaze u prozoru za praćenje poruka se pojavljuju i u stablu prozora. Na primer, upravljač prozorom i naziv klase se pojavljuju u oba slučaja.

Praćenje poruka počinjete izborom opcije Start! u okviru glavnog menija. Poruke počinju da se pojavljuju u prozoru za praćenje poruka onim redom kako ih prozor, odnosno prozori koje ste odabrali primaju. Da biste prekinuli praćenje poruka, odaberite opciju Stop! u okviru glavnog menija.

NAPOMENA Opcije Stop! i Start! u okviru glavnog menija zauzimaju isto mesto u okviru menija. Ukoliko je praćenje poruka isključeno, opcija menija dobija naziv Start!. Ukoliko je praćenje poruka u toku opcija menija dobija naziv Stop!.

Poruke se ispisuju u okviru prozora za praćenje poruka onim redosledom kako su primljene. Uvek možete zaustaviti praćenje poruka i vratiti se na deo liste koja se pomerila; za pomeranje po listi možete koristiti traku za pomeranje teksta. Druga opcija je slanje izlaznih poruka u datoteku (log file). Opcije programa WinSight su obrađene u poglavlju "Opcije za praćenje poruka".

Špijuniranje prozora

WinSight Vam omogućava da vidite sve poruke poslate svim prozorima, odnosno poruke poslate određenom prozoru. Da biste videli sve poruke, odaberite opciju Messages➥All Windows u okviru glavnog menija. Iako možete da odaberete pregled svih poruka koje su poslate svim prozorima, ovo često nije produktivno. Pošto postoji veoma mnogo poruka koje lete unaokolo, veoma je teško pronaći upravo one poruke koje tražite.

Bolji način za rad, je odabiranje određenog prozora, a zatim praćenje poruke samo za odabrani prozor. Na ovaj način će nered u okviru prozora za praćenje poruka biti sveden na nivo koji se može kontrolisati. Da biste videli poruke za određeni prozor, pronađite ga u stablu poruka, a zatim kliknite na prozor kako biste ga odabrali. Zatim odaberite opciju Messages➥Selected Windows u okviru glavnog menija. Odaberite opciju Start! i sve poruke poslate odabranom prozoru će se pojaviti u prozoru za praćenje poruka.

SAVET Kada koristite WinSight najbolje rezultete ćete postići ukoliko imate ideju šta tražite. Pošto se svakom prozoru šalje puno poruka, poruka koju tražite će brzo nestati sa dela prozora koji se može videti. Da biste uvećali efekat programa WinSight, odaberite prozor za koji želite da pratite poruke, počnite sa praćenjem poruka, manipulišite prozorom onako kako želite, a zatim isključite praćenje poruka.

Opcije za praćenje poruka

Poruke koje su prikazane u prozoru za praćenje poruka možete kontrolisati okvirom za dijalog opcije za praćenje poruka (videti sliku 11.9). Takođe možete izmeniti način na koji će poruka biti prikazana. Da biste aktivirali okvir za dijalog opcije za praćenje poruka, odaberite opciju Messages➥Options u okviru glavnog menija.

Slika 11.9
Okvir za dijalog opcije za praćenje poruka

Odeljak koji nosi naziv Messages to Trace (poruke za praćenje) predstavlja osnovu ovog okvira za dijalog. Ovaj odeljak prikazuje nekoliko grupa poruka na levoj strani i okvir za listu na desnoj strani koji sadrži sve Windows poruke. U zavisnosti od

izbora grupe poruka, odgovarajuće poruke će biti odabrane, odnosno neće biti odabrane u okviru liste poruka.

NAPOMENA Poruke koje se nalaze na listi, a prikazane su velikim slovima, predstavljaju standardne Windows poruke. Poruke ispisane malim slovima su nedokumentovane Windows poruke. Nedokumentovane poruke se koriste uglavnom interno u okviru Windows operativnog sistema i nisu dokumentovane u datotekama sistema za pomoć Windows API-ja.

Ukoliko je opcija All Messages odabrana, WinSight će pratiti sve Windows poruke. Da biste definisali poruke koje ćete ubuduće pratiti, odnosno da biste smanjili gužvu u prozoru za praćenje poruka, možete odabrati samo određene grupe poruka. Na primer, ukoliko želite da vidite samo poruke vezane za miša, možete deaktivirati polje za potvrdu All Messages i aktivirati samo polje za potvrdu Mouse. Ukoliko želite da definišete neku posebnu poruku koja se nalazi u okviru za listu, isključite sve opcije i odaberite samo poruku koju želite da pratite. Na primer, ukoliko želite da vidite samo poruku `WM_LBUTTONDOWN`, možete isključiti sve opcije i odabrati samo poruku `WM_LBUTTONDOWN` sa liste na desnoj strani odeljka za praćenje poruka. U prozoru za praćenje poruka će biti prikazane samo poruke `WM_LBUTTONDOWN`.

Još jedna grupa opcija u okviru za dijalog opcije za praćenje poruka, kontroliše kako će poruke biti prikazane u prozoru za praćenje poruka. Opcija za interpretiranje vrednosti (Interpret Values) saopštava programu WinSight da prikaže parametre svake poruke u čitljivijem formatu. Na primer, poruka `WM_KEYDOWN` može biti prikazana u obliku

```
0000309:00000474 {TMemo} WM_KEYDOWN Dispatched
```

odnosno

```
0000309:00000474 {TMemo} WM_KEYDOWN Dispatched 44h 68d VK_D Scan 20h
Repeat
```

Pošto uglavnom želim da vidim detalje poruke, u većini slučajeva opciju za interpretiranje vrednosti ostavljam uključenu.

Opcija Hex Values (heksadecimalne vrednosti) saopštava programu WinSight da prikaže vrednosti u prozoru za praćenje poruka u heksadecimalnom formatu. Ovo je u određenim situacijama korisno, ali ovu opciju verovatno nećete koristiti sve dok se ne uhodate sa programiranjem. Opcija za prikazivanje vremena (Show Times) je uglavnom beskorisna, pošto prikazuje sistemsko vreme koje se u principu ne koristi.

Opcija za zapisivanje u tekst datoteku (Log file) Vam omogućava da poruke zapišete u datoteku. Ukoliko je ova opcija uključena, poruke će biti prikazane u prozoru za praćenje poruka, a biće i zapisane u datoteku. Ova opcija je korisna ukoliko želite da imate evidenciju poruka koje su generisane.

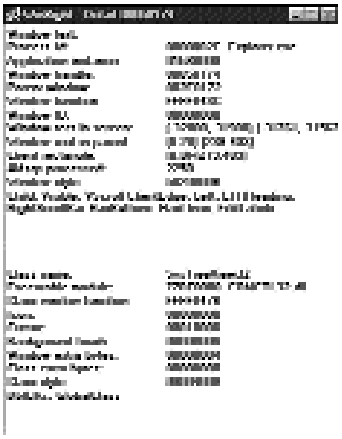
SAVET Kreiranje tekst datoteke za praćenje (log file) ima jednu prednost. Nakon što ste kreirali datoteku i učitali je u tekst editor, možete koristiti funkciju za pretraživanje teksta, kako biste pronašli određenu poruku, odnosno parametar poruke koju želite da pogledate.

Druge mogućnosti programa WinSight

Program WinSight ima druge mogućnosti koje olakšavaju traženje i pregledanje prozora.

Detaljniji pregled prozora

Mogućnost za detaljniji pregled prozora programa WinSight Vam prikazuje sve pripadajuće karakteristike traženog prozora. Da biste videli sve karakteristike prozora, odaberite prozor u okviru stabla prozora, a zatim odaberite opciju Spy➥Open Detail u okviru glavnog menija. Isti efekat postižete ako kliknete dva puta mišem na prozor u okviru stabla poruka, odnosno pritiskom na taster Enter Vaše tastature. Slika 11.10 prikazuje detaljne karakteristike programa Windows Explorer.



Slika 11.10
Prozor Detail programa WinSight

Kao što možete videti, prozor Detail Vam prikazuje veliki deo karakteristika prozora koji istražujete (verovatno više nego što biste želeli da znate!).

Praćenje fokusa

Ova opcija Vam omogućava da odaberete prozor u okviru aplikacije, tako da odabrani prozor postane aktivan u stablu prozora. Stablo prozora obično sadrži desetine prozora. Često je teško pronaći prozor koji tražite, pa opcija za praćenje fokusa može biti veoma korisna.

Da biste koristili ovu mogućnost, odaberite opciju glavnog menija Spy➥Follow Focus. Zatim se prebacite na aplikaciju koja sadrži prozor za koji želite da pogledate poruke, a zatim, ukoliko je to potrebno, kliknite na kontrolu u okviru prozora koju želite da ispitate. WinSight će automatski odabrati prozor u stablu prozora. Da biste počeli praćenje prozora, možete odabrati opciju Start! u okviru glavnog menija.

SAVET Opcija za praćenje fokusa ostaje uključena sve dok je ne isključite. Sve dok ne poželite da vidite druge prozore, odnosno kontrole u okviru Vaše aplikacije, treba da ostavite uključenu opciju za praćenje fokusa, kako bi pratili prozor koji ste odabrali.

Traženje prozora

Opcija za traženje prozora (Find Window) programa WinSight je suprotnost opcije za praćenje fokusa: u okviru stabla prozora. Pronađite prozor koji tražite, a zatim odaberite opciju glavnog menija Spy➥Find Window. Program WinSight će oko prozora postaviti tanak okvir i učiniće da okvir prozora blinka.

Okvir ostaje oko prozora sve dok ne kliknete na sam prozor, odnosno ne odaberete drugi prozor u okviru stabla. Tanak okvir će biti nacrtan na vrhu svih prozora, tako da možete da pronađete traženi prozor, iako se nalazi ispod drugih prozora. Prozor koji je pronađen opcijom za traženje prozora se ne pojavljuje na vrhu, odnosno ne prima ulazni fokus; samo će biti označen kako biste ga lakše uočili.

NAPOMENA Opcija za traženje prozora ne može pronaći skrivene prozore, odnosno prozore koji su minimizirani.

Mod za traženje prozora se isključuje u trenutku kada fokus napusti stablo prozora. Ukoliko kliknete na drugu aplikaciju, odnosno na neku drugu opciju programa WinSight, mod za pretraživanje prozora će se isključiti.

Prebacivanje

Opcija za prebacivanje (Switch To) Vam omogućava da se prebacite na određeni prozor u okviru aplikacije. Da biste se prebacili na prozor, potrebno je da odaberete željeni prozor u okviru stabla prozora, a zatim da odaberete opciju glavnog menija Spy➥Switch To. Odabrani prozor će se postaviti na vrh svih prozora i dobiti ulazni fokus. Ukoliko je prozor minimiziran, biće obnovljen na prethodnoj lokaciji. Opcija za prebacivanje nema efekta kod skrivenih prozora.

Trebaće Vam dosta iskustva u radu sa programom WinSight da biste shvatili značenje svake poruke. Ipak, vremenom će sve doći na svoje mesto.

TDUMP

`TDUMP` je program koji se pokreće iz komandne linije, kako bi prikazao strukturu datoteka tipa `.exe`, odnosno `.dll` (kao i drugih tipova datoteka). Generički, program `TDUMP` rezultat prikazuje na konzoli, ali se može preusmeriti u tekst datoteku koju kasnije možete ispitati, kako bi dobili informacije o programu. Program `TDUMP` će Vam saopštiti detalje o strukturi datoteke, koje DLL datoteke program koristi, koje funkcije u okviru datih DLL datoteka program poziva i slično. Na primer, ovo je deo analize datoteke koja pripada programu kreiranom u Delphi-ju:

```
program produced by Delphi:
Portable Executable (PE) File

Header base: 00000200

CPU type                80386
Flags                   818E [ executable 32bit ]
DLL flags               0000 [ ]
Linker Version          2.19
Time stamp              00000000
O/S Version             1.0
User Version            0.0
Subsystem Version       4.0
Subsystem               0002 [ Windows GUI ]
Object count            00000008
Symbols offset          00000000
Symbols count           00000000
Optional header size    00E0
Magic #                 10B
Code size               0003C000
Init Data size          00009000
Uninit Data size        00001000

...(other information)...

Imports from OLEAUT32.dll
    SysAllocStringLen
    SysStringLen
    VariantChangeTypeEx
    VariantClear
    VariantCopyInd

Imports from MPR.dll
    WNetGetConnectionA

Imports from USER32.dll
    AdjustWindowRectEx
    BeginPaint
    CallNextHookEx
    CallWindowProcA
    CharLowerA

... (more DLL imports) ...
```

Obično ćete program `TDUMP` pokretati iz komandne linije. Pošto je obično prikaz rezultata veoma dug, bolje je da ga preusmerite u tekst datoteku. Navedena komanda ovo ilustruje:

```
tdump MyApp.exe > dump.txt
```

Na ovaj način ćete dobiti ASCII tekst datoteku koju možete pregledati koristeći Delphi-jev editor koda, odnosno bilo koji drugi tekst editor. Iako možda nećete često koristiti program TDUMP, biće Vam od velike koristi kada Vam ustreba.

NAPOMENA TDUMP se uglavnom koristi za prikazivanje liste funkcija i procedura koje su izvezene u DLL.

Package Collection Editor (editor zbirke paketa)

Package Collection Editor Vam omogućava da uposlite nekoliko paketa koji treba da rade zajedno. Da biste pozvali Package Collection Editor odaberite opciju Tools➥Package Collection Editor u okviru glavnog menija. Ovaj program ima meni za pomoć koji dobro objašnjava način njegove upotrebe.

Ovaj alat je primarno bio namenjen isporučiocima komponenti koji isporučuju nekoliko paketa zajedno. Package Collection Editor je pretrpeo neke izmene u verziji Delphi 4 i korisniji je nego u verziji Delphi 3. Možete eksperimentisati sa ovim alatom, kako biste videli da li postoji nešto što ćete moći da koristite. Ipak, možete ga smatrati nedovoljno robusnim za intenzivno korišćenje u komercijalne svrhe.

Konfigurisanje Delphi-jevog menija alata

Delphi-jev meni alata se može konfigurisati. Generički, Delphi-jev meni alata ima elemente za programe Database Desktop i editor slika (Image Editor). U meni alata možete dodati sopstvene alate, druge Delphi alate, odnosno dodati bilo koji drugi program koji često koristite. Takođe možete menjati redosled opcija menija, odnosno, čak ih možete i brisati.

Korišćenje okvira za dijalog za konfigurisanje alata

Svaku aplikaciju koju često koristite možete dodati u meni Tools. Dodavanjem opcija u meni alata, možete brzo učitati bilo koji program koji često koristite u toku razvoja programa. Dodavanje alata u meni alata je veoma jednostavan posao i zahteva samo nekoliko minuta rada. Dodavanje, brisanje i rezmeštanje opcija u okviru menija alata je omogućeno korišćenjem okvira za dijalog za opcije alata. Da biste prikazali okvir za dijalog opcija alata, odaberite opciju glavnog menija Tools➥Configure Tools (videti sliku 11.11).

Slika 11.11
Okvir za dijalog opcija alata

Okvir za listu pod nazivom Tools (alati) prikazuje alate koji se trenutno prikazuju u meniju Tools. Alati su poređani onim redosledom kojim se pojavljuju u meniju za alate. Da biste promenili redosled alata, kliknite na alat kome želite da promenite mesto, a zatim kliknite na dugme sa strelicom gore, ili dole. Da biste uklonili alat iz menija alata, odaberite odgovarajući alat sa liste, a zatim klinite na dugme Delete. Alat će biti uklonjen iz menija.

Dodavanje alata u meni

Da biste dodali alate u meni alata kliknite na dugme Add. Okvir za dijalog karakteristike alata (Tool Properties) će biti prikazan, tako da ćete moći da unesete karakteristike za alat koji dodajete. Okvir za dijalog karakteristike alata je prikazan na slici 11.12 sa karakteristikama programa Windows Explorer.

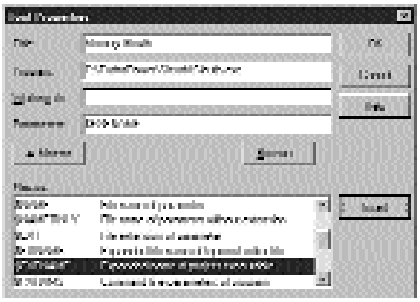
Slika 11.12
Okvir za dijalog karakteristike alata (Tool Properties)

Polje Title (naslov) je mesto gde možete upisati naslov alata koji će se pojaviti u meniju alata. Možete koristiti znak & (ampersand) kako bi označili karakter koji će se koristiti kao prečica tastature. U polje Program upisujete putanju i naziv datoteke. Da biste bili sigurni da će program biti pronađen, možete uneti kompletnu putanju programa. Ukoliko ne znate kompletnu putanju i naziv programa, možete kliknuti na dugme Browse i potražiti datoteku. Polje Working dir (radni direktorijum) se koristi za određivanje radnog direktorijuma alata. Ukoliko pretražujete disk da biste pronašli alat korišćenjem dugmeta Browse, radni direktrorijum će automatski biti postavljen na direktorijum u kome se nalazi sam alat.

Polje Parameters zahteva dodatno objašnjenje. U najjednostavnijoj formi, ovo polje se koristi za postavljanje parametara komandne linije alata. Možete upisati bilo koje

parametre koji su potrebni za pokretanje alata. Čak možete koristiti i Delphi-jeve makroe u polju Parameters. Ukoliko kliknete na dugme Macros, u okviru za dijalog karakteristika alata, dobićete listu makroa koje možete odabrati.

Da biste koristili makroe, možete ih upisati direktno, odnosno možete ih odabrati sa liste, a zatim kliknuti mišem na dugme Insert. Slika 11.13 prikazuje okvir za dijalog karakteristike alata zajedno sa listom makroa.



Slika 11.13
Okvir za dijalog karakteristike alata (Tool Properties) koji prikazuje makroe

Makro `$EXENAME` će vratiti kompletnu putanju izvršne datoteke projekta. Na primer, ukoliko imate projekt pod nazivom MyApp u direktorijumu pod nazilvom Projects, parametar `$EXENAME` će proslediti string c: `\Projects\MyApp.exe` odgovarajućem programu. Neki makroi zahtevaju korišćenje parametara. Da biste koristili samo putanju Vašeg ciljnog projekta, a da ne koristite naziv datoteke, treba da definišete string sa parametrom `$PATH($EXENAME)`. Na osnovu ovog primera, kombinacija parametara će kao rezultat dati string c: `\Projects\`. Da biste pregledali spisak svih dostupnih makroa pogledajte Delphi-jev sistem za pomoć u toku rada.

Alati za editovanje u okviru menija

Alati za editovanje u okviru menija su jednostavni kao i klik miša na dugme Edit u okviru za dijalog opcije alata. Biće prikazan okvir za dijalog karakteristike alata, tako da možete da promenite bilo koje polje, ukoliko je to neophodno.

Kada završite sa dodavanjem, editovanjem i razmeštanjem Vaših alata, kliknite na dugme Close. Vaše promene će se odraziti na Delphi-jev meni alata.

Podešavanje opcija okruženja

Delphi-jeve opcije okruženja Vam omogućavaju da napravite izmene Delphi-jevog okruženja na globalnom nivou. (Okvir za dijalog opcije projekta kontolišu promene na nivou projekta.) Da biste prikazali okvir za dijalog opcije okruženja, odaberite opciju Tools➥Environment Options u okviru glavnog menija. Pojaviće se okvir za dijalog sa karticama koji sadrži devet kartica. O karticama Editor, Display, Color,

Code Insight i Explorer ste se već upoznali u lekciji dana 9, koja je obrađivala editor koda i Code Explorer. Sada ćemo obraditi kartice Preferences, Library i Palette.

NAPOMENA Okvir za dijalog opcije okruženja takođe sadrži i karticu za podešavanje brouzera (Browser). Ova kartica zahteva detaljnije objašnjenje, pa ću pokušati da je obradim u poglavlju koje obrađuje opcije brouzera.

Kartica Preferences (prioriteti)

Kartica Preferences okvira za dijalog opcije okruženja je mesto gde možete definisati generalne prioritete Delphi-ja: kako Delphi-jevo okruženje kontroliše prevođenje, automatsko snimanje (autosaving) i pojavu dizajnera forme. Takođe, možete podesiti i prioritete dizajnera forme na ovoj kartici (videti sliku 11.14).

Slika 11.14 Kartica Preferences okvira za dijalog opcija okruženja

Odeljak Desktop contents određuje koji deo radne površine će biti snimljen kada se snima projekat.

Odeljak Autosave options Vam omogućava da definišete da li će datoteke editora, ili radna površina biti automatski snimljeni prilikom svakog pokretanja programa. Lično ne volim da podesim automatsko snimanje datoteka editora; više volim mogućnost da eliminišem stvari koje sam editovao, ukoliko je to neophodno. Ipak, znam programere kojima se ova opcija veoma sviđa i koji ne mogu da programiraju bez korišćenja ove opcije.

Na primer, ukoliko Vaša aplikacija počne da se čudno ponaša i ruši Delphi-jevo okruženje, odnosno Windows operativni sistem, bićete sigurni da su poslednje izmene snimljene. Opcije radne površine snimaju trenutnu veličinu i poziciju editora koda, svih formi, palete Alignment, prozora Object Inspector itd.

Većim delom, odeljak Form Designer je jasan sam po sebi. Opcija Display grid uključuje, odnosno isključuje mrežu dizajnera forme. Ovo ima efekta samo na prikazivanje mreže, a ne na povezivanje objekta sa mrežom. Opcija Snap to grid

definiše da li će komponenta koja je postavljena na dizajner forme, odnosno pomerena na drugu poziciju u okviru dizajnera forme biti povezana sa tačkama mreže. Polja X grid size i Y grid size Vam omogućavaju da podesite veličinu mreže. Generički, veličina je podešena na osam piksela. Opcija Show component captions se odnosi na nevizuelne komponente koje su postavljene na formu. Ukoliko je ova opcija uključena dizajner forme će prikazati karakteristiku `Name` nevizuelnih komponenti ispod ikone koja predstavlja komponentu.

Oblasti Compiling i Runing (prevođenje i pokretanje) imaju četiri opcije:

- Opcija Show compiler progress kontroliše pojavljivanje okvira za dijalog koji prikazuje tok prevođenja. Uglavnom ćete ovu opciju držati isključenom, pošto Delphi prevodi Vaš program toliko brzo da Vam neće biti potrebno da pratite tok prevođenja.
- Opcija Warn on package rebuild Vas upozorava ukoliko je potrebno da ponovo napravite paket koji je trenutno učitan u paletu komponenti.
- Opcija Hide designers on run sakriva dizajner forme, Object Inspector i prozore za podršku dizajneru forme u toku rada programa. Naravno, Delphi-jev glavni prozor i ediltor koda će još uvek biti vidljivi.
- Opcija Minimize on run je slična prethodnoj opciji za skrivanje prozora za dizajniranje u toku rada. Ukoliko je ova opcija uključena, kompletno Delphi okruženje će biti minimizirano, bez obzira da li se program izvršava iz okružanja.

NAPOMENA Kada je opcija Minimize on run uključena, svaka aktivnost koja zahteva rad debagera (izuzeci, tačke prekida, itd.) kao rezultat prikazuje ponovo Delphi-jevo okruženje.

Odeljak Shared Repository (skladište koje korisnici mogu da dele) Vam omogućava da definišete gde će informacije o skladištu objekata biti snimljene; u okvir za editovanje direktorijuma upisujete odgovarajuću putanju. Ovo može biti veoma korisno u slučaju da se elementi skladišta objekata koriste u grupi koja zajedno radi na istom projektu. Ukoliko definišete direktorijum koji se nalazi na mrežnom disku, na ovaj način podešavate direktorijum za deljenje skladišta objekata, tako da su svi objekti pristupačni drugim članovima grupe.

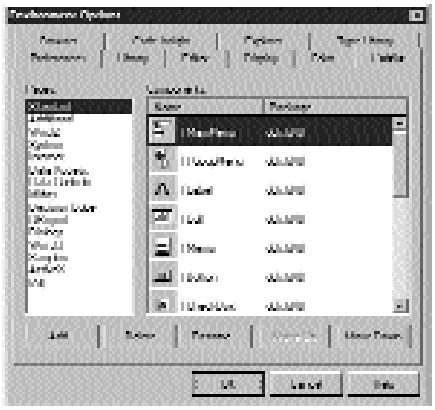
Kartica Library

Kartica Library (biblioteka) okvira za dijalog opcije okruženja ima samo tri polja. Polje Library Path se koristi za definisanje putanje do datoteka biblioteka koju Delphi koristi za paletu komponenti. U normalnim okolnostima nije potrebno da menjate ovu putanju. Polja za izlazni direktorijum BPL i izlazni direktorijum DCP se koristi za definisanje direktorijuma u kojima će se nalaziti izvršne datoteke paketa. Na primer, možete poželeti da prebacite Vaše pakete (.bpl datoteke) kreirane u

Vašem direktorijumu `Windows\System` ili `Winnt\System32` (datoteke paketa moraju biti u poddirektorijumu sistemskog direktorijuma).

Kartica Palette

Kartica Palette okvira za dijalog opcije okruženja Vam omogućava da prilagodite paletu komponenti (videti sliku 11.15).



Slika 11.15
Kartica Palette okvira za dijalog opcije okruženja

Ova kartica Vam omogućava da promenite redosled kojim se pojavljuju kartice u okviru palete komponenti. Okvir za listu Pages na levoj strani okvira za dijalog pokazuje sve kartice koje se trenutno nalaze u okviru palete komponenti. Uočite da u zavisnosti od verzije Delphi-ja koju posedujete, kartice Palette mogu biti različite od kartica koje su prikazane na slici 11.15. (Na primer, kartica Midas je dostupna samo u Client/Server verziji Delphi-ja.) Na kraju liste ćete videti opciju pod nazivom `[ All ]`. Ova opcija Vam omogućava da vidite sve komponente koje su instalirane na svim karticama palete komponenti. Okvir za listu Components Vam prikazuje komponente koje se pojavljuju na kartici odabranoj u okviru za listu Pages.

Da biste promenili redosled kartica, prevucite karticu koja se nalazi u okviru za listu Pages, na mesto koje želite da zauzme na paleti komponenti. Ukoliko želite da se kartica Samples pojavi prva na paleti komponenti, možete je prevući do vrha liste i pustiti. Takođe, možete kliknuti na karticu i koristiti dugmad Move Up, odnosno Move Down, da biste pomerili karticu na novu poziciju.

Možete dodavati, brisati, odnosno preimenovati kartice; ove operacije rade tačno ono što ste očekivali. Da biste dodali karticu, kliknite na dugme Add. Bićete upitani za naziv nove kartice. Nakon što upišete naziv kartice, biće kreirana nova kartica. Možete dodati novu karticu, ukoliko ste, na primer, kreirali sopstvene komponente i želite da ih dodate na zasebnu karticu u paleti komponenti. Takođe, možete da

uzmete bilo koju VCL komponentu koju često koristite i premestite je na novu karticu, tako da joj brzo možete pristupiti.

SAVET Možete pomerati komponente sa kartice na karticu, ali ne možete kopirati komponente sa jedne kartice na drugu. Takođe, možete da dodate bilo koju instaliranu komponentu bilo kojoj kartici u okviru palete komponenti. Prvo treba da odaberete opciju [All] u okviru za listu Pages. Zatim treba da prevučete komponentu iz okvira za listu Components na bilo koju stranu koja se nalazi u okviru za listu Pages.

Brisanje i promena naziva kartice je veoma jednostavno. Kao što je to slučaj sa skladištem objekata, karticu možete obrisati samo ukoliko uklonite sve njene komponente.

Komponente na kartici se mogu preurediti isto kao i kartice. Da biste pomerili komponentu, prevucite je na novu poziciju u okviru za listu Components, odnosno koristite dugmad Move Up, ili Move Down. Da biste prebacili komponentu na novu karticu prevucite je i pustite na naziv nove kartice u okviru za listu Pages. Da biste obrisali komponentu, prvo odaberite samu komponentu , a zatim kliknite na dugme Delete.

Zaključak

Sada znate mnogo više o alatima koji su deo Delphi-jevog paketa. Neki od ovih alata možete prevideti, pa je dobro da imate pregled svega što Vam je dostupno. Možda trenutno nećete koristiti baš sve alate, ali u toku rada možete ponovo pogledati šta Vam je dostupno. Lekcija je završena pregledom okvira za dijalog opcija okruženja. U većini slučajeva možete ostaviti generičke definicije ovih opcija. Kada počnete prilagođavati Delphi-jevo okruženje, morate se bolje upoznati sa mogućnostima pojedinih opcija.

Radionica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstirte razumevanje obrađenog materijala, kao i vežbe koje Vam omogućavaju da steknete iskustvo u korišćenju gradiva koje ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A, "Odgovori na kviz pitanja".

Pitanja i odgovori

P Kreirao sam ikonu, pokušavajući da dobijem elipsu sa crnom ivicom i žutim središtem. Podesio sam boju prednjeg plana u crno, a boju pozadine u žuto, ali sam dobio crni krug. Šta da uradim da bih dobio ono što želim?

O Editor slika radi nešto drugačije nego neki drugi editori bitmapa. Da biste ostvarili ono što želite, treba prvo da nacrtate crnu praznu elipsu, a zatim je popunite žutom bojom.

P **Kako mogu nacrtati idealne krugove, odnosno prave linije u editoru slika?**

O Držite pritisnut taster Shift, dok crtate krug, odnosno liniju. Na isti način možete nacrtati savršen kvadrat.

P **Kreirao sam sopstveni kursor sa strelicom, ali kada kliknem mišem, korišćenjem ovog kursora, dobija se utisak da nije kliknuto vrhom strelice. Šta nije u redu?**

O Potrebno je da postavite tačku pokazivanja kursora (hot spot). Generički, tačka pokazivanja kursora se nalazi u gornjem levom uglu. Editujte kursor u editoru slika i promenite tačku pokazivanja na piksel koji se nalazi na vrhu strelice.

P **Da li je potrebno, kada kreiram ikonu, da kreiram i veliku i malu verziju ikone?**

O Nije neophodno. Ukoliko kreirate samo veliku ikonu, kada je potrebno Windows će je umanjiti. Ukoliko Vam se ne svidi način na koji je ikona prikazana nakon umanjenja, možete kreirati i malu ikonu isto kao što ste to uradili sa velikom. Kada postoji mala ikona, Windows će je koristiti.

P **Kako mogu kopirati deo jedne od svojih bitmapa u novu bitmapu?**

O U editoru slika možete otvoriti prozor originalne bitmape, a zatim nov prozor nove bitmape kao zaseban prozor. Prevucite marker oko oblasti originalne bitmape, a zatim odaberite opciju Edit➥Copy u okviru menija editora slika. Pređite na novu bitmapu, a zatim odaberite opciju Edit➥Paste. Bitmapa će biti zalepljena na bitmapu koju ste kreirali kao novu.

P **Želim da se neke komponente u okviru palete komponenti pojavljuju ne samo na njihovoj početnoj poziciji, nego i na novoj kartici koju sam kreirao. Da li to mogu da uradim?**

O Da. Pređite na karticu Palette u okviru za dijalog opcije okruženja. Kliknite na opciju [All] koja se nalazi u okviru za listu Pages. Sada prevucite komponentu iz okvira za listu Components na bilo koju karticu u okviru za listu Pages.

Kviz

1. Kako se transparentna boja koristi za ikone i kursore?
2. Kako možete da odaberete boju u editoru slika?
3. Kako možete da odaberete oblast na bitmapi koju treba da isečete, ili kopirate?

4. Koji je maksimalan broj boja dozvoljen u radu sa bitmapama u editoru slika?
5. Šta je tačka pokazivanja kursora (hot spot)?
6. Da li program WinSight može istražiti skrivene prozore?
7. Koji je najbrži način da pronađete prozor u stablu prozora programa WinSight?
8. Kako možete da omogućite automatsko snimanje datoteka, svaki put kada program bude pokrenut preko debagera?
9. Gde možete reorganizovati sadržaj palete komponenti?

Vežbe

1. Korišćenjem editora slika kreirajte bitmapu koja će se koristiti kao uvodna slika Vaše firme, odnosno kao uvodna slika Vaše aplikacije.
2. Korišćenjem editora slika kreirajte ikonu sa transparentnom pozadinom koja sadrži i veliku i malu ikonu.
3. Kreirajte resurs kursora u editoru slika. Uverite se da ste podesili tačku pokazivanja kursora, a zatim testirajte kursor koristeći Cursor Tester.
4. Kreirajte novu resursnu datoteku. Kreirajte dve bitmape i kursor. Snimite datoteku.
5. Korišćenjem resursnih datoteka koje ste kreirali u vežbi četiri, kreirajte novi resurs ikone, a zatim kreirajte ikonu kreiranu u vežbi dva, u novi resurs ikone.
6. Otvorite okvir za dijalog opcija okruženja. Pređite na karticu Palette. Kreirajte novu karticu u paleti komponenti. Nazovite je `MyStuff`. Kopirajte komponente `Label`, `Image` i `OpenDialog` na novu karticu. Zatvorite okvir za dijalog opcija okruženja. Pogledajte novi izgled palete komponenti. Nakon toga obrišite karticu `MyStuff`.
7. Učitajte program u Delphi. Otvorite okvir za dijalog opcija okruženja. Na kartici Preferences uključite opciju Minimize on Run. Kliknite mišem na dugme Run kako biste pokrenuli program. Posmatrajte ponašanje Delphi-jevog okruženja.

Dan 12

Programiranje grafike i multimedije

Programiranje grafike i multimedije predstavlja zabavni deo programiranja. U ovoj lekciji će Vam biti predstavljeno programiranje grafike i multimedije uz pomoć Delphi-ja. U slučaju programiranja grafike većina saveta će biti vezana za istraživanje klasa `TCanvas` i `TBitmap`.

Počeću sa pregledom najjednostavnijih načina da se prikaže grafika na Delphi-ju. Zatim ćete naučiti o Windows grafičkom interfejsu za uređaje (GDI - Graphics Device Interface), kao i o komponentama koje čine ovaj interfejs. U toku rada naučićete o različitim rutinama za crtanje linija i oblika, kao i o različitim načinima za prikazivanje bitmapa. U nastavku lekcije ćete naučiti o vanekranskim bitmapama i o tome kakvu Vam korist vanekranske bitmape mogu doneti. Odeljci za programiranje multimredije Vam objašnjavaju kako da puštate muzičke datoteke koristeći Windows API. Takođe ćete naučiti kako da puštate audio wave, MIDI i AVI video datoteke, koristeći klasu `TMediaPlayer`. Stoga počnimo!

Grafika na jednostavniji način

Programiranje grafike ne bi trebalo da bude teško. Ponekad je dovoljno da prikažete sliku, ili jednostavan oblik na formi. VCL Vam pruža već spremne komponente za poduhvate ovog tipa. Obradićemo neke od ovih komponenti, pre nego što pređemo na pravo programiranje grafike.

Komponenta `Shape` (nalazi se na kartici Additional u okviru palete komponente) se može koristiti za dodavanje jednostavnih oblika na formu. Korišćenje komponente Shape je jednostavno. Potrebno je da spustite komponentu na formu i po želji

promenite karakteristike `Brush`, `Pen` i `Shape`. Možete crtati krugove, elipse, kvadrate, pravougaonike i pravougaonike sa oblim ivicama. Promenite karakteristiku `Brush` da biste izmenili boju pozadine oblika. Promenite karakteristiku `Pen` da biste promenili boju, odnosno debljinu okvira oko oblika.

Komponenta `Image` se može koristiti za prikazivanje bitmapa na formi. Ova komponenta je idealna za većinu grafičkih operacija, uključujući i pozadinu bitmape na formi. Karakteristika `Picture` klase `TImage` je slučaj klase `TPicture`. Možete odabrati sliku u toku dizajniranja, koristeći Object Inspector, mada isto tako možete učitati sliku i u toku rada programa. Na primer, ovo bi mogao biti način kako da učitate sliku u komponentu u toku rada programa:

```
Image1.Picture.Bitmap.LoadFromFile('bkgnd.bmp');
```

Karakteristika `Stretch` određuje da li će slika biti uvećana, ili umanjena da bi se uklopila u veličinu komponente. Karakteristika `Center` određuje da li će bitmapa biti centrirana u okviru komponente. Karakteristika `AutoSize` se koristi za prilagođavanje same komponente veličini slike.

Takođe ćemo spomenuti komponentu `PaintBox`. Ukoliko želite da Vaš crtež bude zatvoren unutar određene oblasti unutar forme, ova komponenta Vam obezbeđuje kanvas (platno) po kom možete crtatati. Jedina značajna karakteristika komponente `PaintBox` je karakteristika `Canvas`. Ova karakteristika je slučaj klase `TCanvas`. Sa ovom klasom možete mnogo toga nacrtati u Delphi-jevim aplikacijama. Sada ćemo obraditi klasu TCanvas.

Kontekst uređaji i klasa `TCanvas`

Windows koristi termin kontekst uređaja (*device context*) da bi opisao kanvas po kom možete crtati. Kontekst uređaja možete koristiti da bi crtali na različitim površinama:

- U klijent oblast prozora, ili u okviru samog prozora.
- Na radnu površinu (desktop).
- U memoriju.
- Na štampač, odnosno drugi izlazni uređaj.

Sledi nekoliko primera. Iako postoje drugi skriveni kontekst uređaji (na primer, meniji), kontekst uređaji koji će biti navedeni predstavljaju najinteresantnije primerke.

Rad sa kontekst uređajima na nivou Windows API-ja ponekad može biti složen. Prvo treba da obezbedite upravljač kontekst uređajima u okviru Windows-a. Zatim treba da odaberete različite objekte u okviru kontekst uređaja (pera, četke, odnosno fontove). Nakon toga, možete crtati na kontekst uređaju. Nakon što završite sa

crtanjem, treba da se uverite da su objekti koje ste odabrali u okviru kontekst uređaja uklonjeni, pre nego što obrišete kontekst uređaje. Ukoliko zaboravite da uklonite objekte u okviru kontekst uređaja, Vaša aplikacija će rasipati memoriju (koristiti dodatnu memoriju koju nikada neće osloboditi i prepustiti sistemu). Najblaže rečeno, to je veoma nezgodan proces.

Dobra vest je da VCL obezbeđuje klasu TCanvas koja Vam omogućava da jednostavnije upravljate kontekst uređajima. Daću Vam kratak primer. Naredni kod koristi Windows API da bi nacrtao krug na ekranu koji ima plavi okvir i crveno središte:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  DC : HDC;
  Brush, OldBrush : HBrush;
  Pen, OldPen : HPen;
begin
  DC := GetDC(Handle);
  Brush := CreateSolidBrush(RGB(255, 0, 0));
  Pen := CreatePen(PS_SOLID, 1, RGB(0, 0, 255));
  OldBrush := SelectObject(DC, Brush);
  OldPen := SelectObject(DC, Pen);
  Ellipse(DC, 20, 20, 120, 120);
  SelectObject(DC, OldBrush);
  SelectObject(DC, OldPen);
  ReleaseDC(Handle, DC);
end;
```

Ovaj kod nije suviše težak, ali lako možete zaboraviti da vratite objekte, nakon što ste ih upotrebili. Ukoliko se to dogodi, Vaša aplikacija će rasipati resurse. Pogledajmo sada ekvivalent istog programa pisan u VCL kodu:

```
Canvas.Brush.Color := clRed;
Canvas.Pen.Color := clBlue;
Canvas.Ellipse(20, 20, 120, 120);
```

Ovaj kod, ne samo da je kraći i čitljiviji, on je isto tako i robusniji. Klasa TCanvas vodi računa o oslobađanju resursa ukoliko je to potrebno, pa o tome ne treba da brinete. Klasa `TCanvas` je jednostavnija i ima efektniji pristup u odnosu na Windows API.

Klasa `TCanvas` ima mnogo karakteristika i metoda. Obradićemo neke od ovih karakteristika i metoda u toku rada sa primerima u današnjoj lekciji. Tabela 12.1 prikazuje primarne karakteristike klase `TCanvas`, a tabela 12.2 prikazuje primarne metode.

Tabela 12.1: Primarne karakteristike klase `TCanvas`

Karakteristika	Opis
`Brush`	Boja četke, odnosno dezen se koristi da bi se popunio oblik.
`ClipRect`	Tekući pravougaonik za isecanje kanvasa. Bilo kakvo crtanje ograničeno na tekući pravougaonik. Ova karakteristika se može samo čitati (read-only).
`CopyMode`	Određuje kako će crtež biti postavljen (normalno, inverzno, `xor`, itd.).
`Font`	Font koji kanvas koristi za pisanje teksta.
`Handle`	Upravljač (`HDC`) kanvasa. Koristi se prilikom direktnog poziva Windows API-ja.
`Pen`	Određuje stil i boju linija koje se crtaju na kanvasu.
`PenPos`	Trenutna pozicija za crtanje u x i y koorinatama.
`Pixels`	Niz piksela koji pripadaju kanvasu.

Tabela 12.2: Primarne metode klase `TCanvas`

Metoda	Opis
`Arc`	Crta luk na kanvasu koristeći trenutno odabrano pero.
`BrushCopy`	Prikazuje bitmapu sa transparentnom pozadinom.
`CopyRect`	Kopira deo slike na kanvasu.
`Draw`	Kopira sliku iz memorije na kanvas.
`Ellipse`	Koristi trenutno odabrano pero za crtanje elipse na kanvasu i popunja va je trenutno odabranom četkom.
`FloodFill`	Popunjava oblast kanvasa trenutno odabranom četkom.
`LineTo`	Crta liniju počev od trenutne pozicije do pozicije definisane parametrima x i y.
`MoveTo`	Postavlja trenutnu poziciju za crtanje.
`Pie`	Crta oblik pite na kanvasu.
`Polygon`	Crta poligon na kanvasu koristeći tačke iz niza, a zatim popunjava poligon koristeći trenutno odabranu četku.
`Polyline`	Crta liniju u okviru kanvasa preuzimajući tačke iz niza i koristeći tenut no odabrano pero. Tačke se ne zatvaraju automatski.
`Rectangle`	Crta pravougaonik u okviru kanvasa sa okvirom koji ima boju trenutno odabranog pera i popunjen je bojom trenutno odabrane četke.
`RoundRect`	Crta popunjeni pravougaonik sa zaobljenim ivicama.
`StretchDraw`	Kopira bitmapu iz memorije na kanvas. Bitmapa je prilagođena (uvećana ili umanjena) u zavisnosti od veličine pravougaonika.

`TextExtent`	Vraća širinu i visinu stringa prosleđenog u parametar `Text`; širina i visina su dati u pikselima. Širina se izračunava korišćenjem trenutno odabranog fonta kanvasa.
`TextHeight`	Vraća visinu u pikselima stringa prosleđenog u parametar `Text`. Širina jeizračunata korišćenjem trenutno odabranog fonta kanvasa.
`TextOut`	Koristeći trenutno odabrani font, ispisuje tekst na kanvas, na definisanoj poziciji.
`TextRect`	Ispisuje tekst unutar pravougaonika za odsecanje kanvasa.

Verovali ili ne, ove karakteristike i metode predstavljaju samo mali deo funkcionalnosti Winldows kontekst uređaja. Dobra vest je, da ove karakteristike i metode pokrivaju 80 procenata zadataka potrebnih da se obave poslovi vezani za grafiku. Ipak, pre nego što budemo detaljnije obradili klasu `TCanvas`, obradićemo grafičke objekte koji se koriste u Windows programiranju.

GDI objekti

Windows grafički interfejs za uređaje (GDI - Graphics Device Interface) sadrži mnogo tipova objekata koji definišu način funkcionisanja kontekst uređaja. Najčešće korišćeni GDI objekti su pera, četke i fontovi. Ostali GDI objekti su palete, bitmape i regioni. Prvo ćemo obraditi pera, četke i fontove, a zatim ćemo preći na komplikovanije objekte.

Pera, četke i fontovi

Pera, četke i fontovi su jasni sami po sebi. Ukratko ćemo obraditi svaki od ovih GDI objekata, i pojasniti način na koji ih klasa TCanvas koristi.

Pera (pens)

Pero definiše objekat koji se koristi za crtanje linija. Linija može biti zasebna linija, od jedne do druge tačke, odnosno može biti okvir koji se može nacrtati oko pravougaonika, elipsi i poligona. Peru možete pristupiti koristeći karakteristiku `Pen` klase `TCanvas`, odnosno koristeći slučaj klase `TPen`. Tabela 12.3 prikazuje karakteristike klase `TPen`. Za klasu TPen ne postoje metode i događaji koje vredi spomenuti.

Tabela 12.3: Karakteristike klase `TPen`

Karakteristika	Opis
`Color`	Postavlja boju linije.
Handle	Upravljač perom (`HPEN`). Koristi se prilikom direktnog poziva GDI-ja.
`Mode`	Određuje kako će linija biti nacrtana (normalno, inverzno, `xor`, itd.).
`Style`	Stil pera. Stilovi mogu biti: pun, tačkast, sa crticama, crta-tačka, prazan (solid, dotted, dashed, dash-dot, clear), itd.
`Width`	Širina pera u pikselima.

U većini slučajeva, ove karakteristike se koriste upravo onako kako ste očekivali. Sledeći primer crta crvenu tačkastu liniju:

```
Canvas.Pen.Color := clRed;
Canvas.Pen.Style := psDash;
Canvas.MoveTo(20, 20);
Canvas.LineTo(120, 20);
```

Da biste testirali kod, postavite dugme na formu i upišite u upravljač događajem `OnClick` navedeni kod. Kada kliknete mišem na dugme, linija će biti iscrtana na formi.

Jednostavni primeri u ovoj lekciji se mogu testirati na ovaj način. Ipak, ukoliko budete napravili aplikaciju i u toku rada, vratite formu na vrh nakon određenog vremena, crtež će nestati. Razlog za ovakvo ponašanje je činjenica da je crtež privremen. Ukoliko želite da crtež ostane na formi, postavite kod za crtanje u upravljač događajem `OnPaint`, koji je povezan sa formom. Svaki put kada se prozor bude ponovo iscrtavao, događaj `OnPaint` prozora će biti generisan i Vaš kod za crtanje će biti izvršen.

Stilovi pera sa crtama i tačkama se mogu koristiti samo sa debljinom (širinom) pera koja je jednaka 1. Stil pera `psClear` se može koristiti da bi se eliminisala linija koju Windows GDI crta oko objekata kao što su pravougaonici, elipse i popunjeni poligoni.

Možete eksperimentisati sa različitim karakteristikama klase `TPen`, postavljajući komponentu `Shape` na formu i menjajući karakteristiku `Pen` postavljene komponente. Ovo je veoma korisno za posmatranje efekata karakteristike `Mode` klase `TPen`.

Četke (brushes)

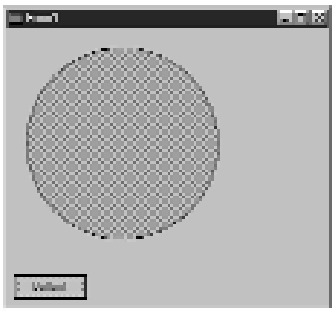
Četka predstavlja popunjenu oblast oblika. Kada nacrtate elipsu, pravouganik, odnosno poligon, oblik će biti popunjen trenutno odabranom bojom četke. Kada razmišljate o četki, verovatno mislite na određenu boju. U većini slučajeva ovo je tačno, ali četka je više od same definicije boje. Četka može biti dezen, odnosno bitmapa. Klasa `TCanvas` sadrži karakteristiku pod nazivom `Brush`, koju možete koristiti za kontrolu izgleda četke. Karakteristika `Brush` je slučaj klase `TBrush`. Kao i kod klase `TPen`, klasa `TBrush` nema metode, ili događaje koje treba napomenuti. Karakteristike klase `TBrush` su prikazane u tabeli 12.4.

Tabela 12.4: Karakteristike klase `TBrush`

Karakteristika	Opis
`Bitmap`	Bitmapa koja će se koristiti kao pozadina četke. Kod Windows 95 operativnog sistema, bitmapa ne sme biti veća od 8x8 piksela.
`Color`	Postavlja boju četke.
`Handle`	Upravljač četkom (HBRUSH). Koristi se prilikom direktnog poziva GDI-ja.
`Style`	Stil četke. Stil može biti pun (solid), prazan (hollow), odnosno bilo koji tip dezena.

Generički, karakteristika `Style` je postavljena na vrednost `bsSolid` (pun). Ukoliko želite da popunite oblik dezenom, postavite karakteristiku `Style` na bilo koji stil dezena (`bsHorizontal`, `bsVertical`, `bsFDiagonal`, `bsBDiagonal`, `bsCross`, `bsDiagCross`). Primer koji sledi crta krug na formi koristeći dezen sa linijama ukrštenim pod uglom od 45 stepeni. Slika 12.1 prikazuje formu nakon izvršavanja narednog koda:

```
Canvas.Brush.Color := clBlue;
Canvas.Brush.Style := bsDiagCross;
Canvas.Ellipse(20, 20, 220, 220);
```



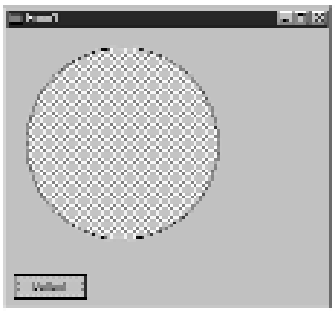
Slika 12.1. Slika popunjena dezenom sa linijama ukrštenim pod uglom od 45 stepeni

Kada koristite četku sa dezenom, karakteristika Color trenutno aktivne četke će odrediti boju linija na dezenu. Ukoliko koristite popunjavanje objekta dezenom, VCL automatski, iz nekog razloga, prebacuje da pozadina postane transparentna. Ovo znači da će boja pozadine četke biti ista kao boja pozadine prozora na kom je oblik nacrtan.

Pogledajte ponovo sliku 12.1 i videćete da je boja pozadine kruga ista kao i boja pozadine forme (znam da nije lako razlikovati nijanse sive). Ukoliko želite da definišete boju pozadine, morate zaobići VCL i koristiti API. Evo kako izgleda kod koji možete koristiti da biste nacrtali plavu mrežu na beloj pozadini:

```
Canvas.Brush.Color := clBlue;
Canvas.Brush.Style := bsDiagCross;
SetBkMode(Canvas.Handle, OPAQUE);
SetBkColor(Canvas.Handle, clWhite);
Canvas.Ellipse(20, 20, 220, 220);
```

Sada je boja pozadine četke bela. Slika 12.2 prikazuje krug nacrtan korišćenjem novog koda.

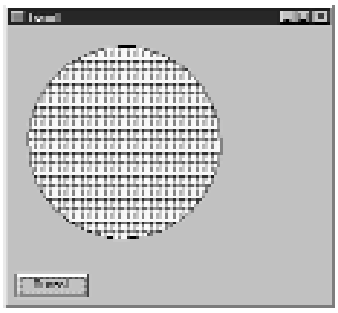


Slika 12.2.
Slika popunjena dezenom sa linijama ukrštenim pod uglom od 45 stepeni i belom pozadinom

Korišćenje bitmapa za definisanje pozadine predstavlja još jednu interesantnu osobinu četki. Prvo pogledajte kod, a zatim ću Vam detaljnije objasniti tip četke koja koristi bitmape. Sledi kod:

```
Canvas.Brush.Bitmap := TBitmap.Create;
Canvas.Brush.Bitmap.LoadFromFile('bkgnd.bmp');
Canvas.Ellipse(20, 20, 220, 220);
Canvas.Brush.Bitmap.Free;
```

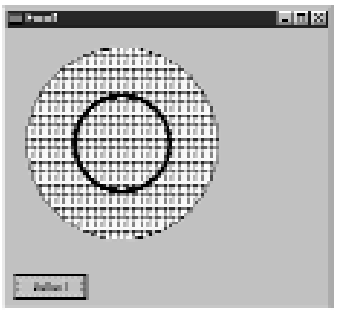
Prva linija u ovom isečku koda kreira objekat `TBitmap` i dodeljuje ga karakteristici Bitmap koja pripada četki. Karakteristika Bitmap nije generički dodeljena, pa morate izričito kreirati objekt `TBitmap` i dodeliti ga karakteristici `Bitmap`. Druga linija učitava bitmapu iz datoteke. Bitmapa ne može biti većih dimenzija od 8x8 piksela. Ukoliko koristite veće bitmape, iz ovih bitmapa će biti isečen deo dimenzija 8x8 piksela. Treća linija u ovom primeru crta elipsu. Nakon što elipsa bude nacrtana, karakteristika `Brush` će biti obrisana. Pošto u ovom slučaju VCL neće obrisati karakteristiku `Brush`, morate to sami učiniti. Ukoliko propustite da obrišete karakteristiku `Brush`, Vaš program će rasipati memoriju. Slika 12.3 prikazuje elipsu koja je nacrtana korišćenjem četke sa bitmapom.



Slika 12.3
Elipsa popunjena bitmapom koja je definisana u okviru četke

Ponekad Vam je potrebna prazna (hollow) četka. Prazna, odnosno čista (clear) četka Vam omogućava da definišete pozadinu kroz koju se može videti pozadina. Da biste kreirali praznu četku, postavite karakteristiku `Style` na vrednost `bsClear`. Vratimo se na prethodni primer i dodajmo drugi krug unutar prvog koristeći praznu četku. Na slici 12.4 možete videti rezultate. Sledi i kod:

```
Canvas.Pen.Width := 1;
Canvas.Brush.Bitmap := TBitmap.Create;
Canvas.Brush.Bitmap.LoadFromFile('bkgnd.bmp');
Canvas.Ellipse(20, 20, 220, 220);
Canvas.Brush.Style := bsClear;
Canvas.Pen.Width := 5;
Canvas.Ellipse(70, 70, 170, 170);
Canvas.Brush.Bitmap.Free;
```



Slika 12.4.
Krug sa praznom četkom

Sa četkama možete raditi i neke druge stvari, ukoliko direktno koristite API. U većini slučajeva, VCL klasa `TBrush` će moći da obavi posao.

Fontovi

Fontovi ne predstavljaju novinu za Vas; koristili ste ih u toku čitanja ove knjige. Fontovi koji se koriste sa klasom `TCanvas` se ne razlikuju od fontova koji se koriste u formama, odnosno drugim komponentama. Karakteristika `Font` klase `TCanvas` je ista kao karakteristika `Font` bilo koje druge komponente. Da biste promenili font kanvasa, uradite sledeće:

```
Canvas.Font.Name := 'Courier New';
Canvas.Font.Size := 14;
Canvas.Font.Style := Canvas.Font.Style + [fsBold];
Canvas.TextOut(20, 20, 'Testing');
```

To bi bilo sve. Nešto kasnije, u poglavlju "Crtanje teksta", biće detaljnije obrađene ostale mogućnosti korišćenja fontova.

Bitmape i palete

Bitmape i palete su u većini slučajeva povezane. Klasa `TBitmap` enkapsulira bitmapirane objekte u okviru Delphi-ja. Ukoliko koristite ovu klasu, učitavanje i prikazivanje bitmapa će Vam biti jednostavno. U lekciji dana 8, "Kreiranje aplikacija u Delphi-ju" ste mogli da vidite kako klasa `TBitmap` funkcioniše. Klasa `TBitmap` se može koristiti u raznim situacijama. Neke od situacija će biti obrađene u današnjoj lekciji, u delu koji se odnosi na crtanje bitmapa i memorijske bitmape. Klasa `TBitmap` je veoma kompleksna, pa neću objašnjavati svaku karakteristiku i metodu.

Palete su aspekti Windows programiranja koji Vas mogu najviše zbuniti. Najveći deo vremena palete održavaju objekti klase `TBitmap`, pa o tome ne treba da brinete. Pokazaću Vam primer, umesto da Vam objašnjavam važnost paleta.

Pokrenite novu aplikaciju, a zatim upišite sledeći kod u upravljač događajem `OnPaint`; takođe možete koristiti događaj klika mišem na dugme. Uverite se da ste uneli pravu putanju datoteke `HANDSHAK.BMP`. (Možete je pronaći u direktorijumu `Borland Shared Files\Images\Splash\256Color`.) Evo i koda:

```
var
  Bitmap : TBitmap;
begin
  Bitmap := TBitmap.Create;
  { Bitmap.IgnorePalette := True; }
  Bitmap.LoadFromFile('handshak.bmp');
  Canvas.Draw(0, 0, Bitmap);
  Bitmap.Free;
end;
```

Uočite da je jedna linija napisana kao komentar u vitičastim zagradama. Pokrenite program i videćete lepu bitmapu na formi. Sada uklonite vitičaste zagrade iz linije koja je korišćena kao komentar. Ova linija saopštava VCL-u da ignoriše informaciju o paletama u toku prikazivanja bitmape. Ponovo pokrenite program. Ovaj put ćete uočiti da su boje bitmape pogrešne (možda niste primetili ovaj efekat ukoliko je Vaša video kartica podešena da prikazuje više od 256 boja). Uzrok je nepodešena paleta. Paleta osigurava prikazivanje ispravnih boja bitmape koje su postavljene u sistemsku paletu.

Bitmapa i objekti palete igraju važnu ulogu u grafičkim operacijama. Trebaće Vam vremena da biste shvatili šta sve nasleđuju ovi objekti, pa ne bi trebalo da se loše osećate ukoliko sve to ne možete da shvatite u ovom trenutku.

Regioni za isecanje

Regioni (*regions*) su delovi ekrana koji se mogu koristiti za kontrolisanje delova kanvasa koji se mogu crtati. Klasa `TCanvas` sadrži karakteristiku `ClipRect`, koja se može samo čitati. Da biste promenili region za isecanje, treba da koristite

Windows API. Uzmimo prethodni primer i izmenimo ga, tako da ilustruje način rada regiona za isecanje. Sledi kod:

```
var
  Bitmap : TBitmap;
  Rgn : HRGN;
begin
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile('handshak.bmp');
  Rgn := CreateRectRgn(50, 50, 250, 250);
  SelectClipRgn(Canvas.Handle, Rgn);
  Canvas.Draw(0, 0, Bitmap);
  Bitmap.Free;
end;
```

Kada pokrenete program primetićete da je prikazan samo jedan deo bitmape. Funkcija `SelectClipRgn` podešava region isecanja kanvasa na pravougaonik sa koordinatama 50, 50, 250 i 250. Bitmapa će biti prikazana na istoj poziciji na kojoj je bila i ranije, ali sada samo kao deo bitmape (definisana regionom za isecanje). Sve što se nalazi van regiona za isecanje se ignoriše.

Regioni ne moraju biti pravougaoni. Pogledajmo prethodni primer i učinimo ga interesantnijim. Uklonite liniju koja kreira pravougaoni region i zamenite je sledećom linijom:

```
Rgn := CreateEllipticRgn(30, 30, 170, 170);
```

Sada ponovo pokrenite program. Ovaj put bitmapa je ograničena kružnim regionom. Slika 12.5 prikazuje program u kom se nalazi eliptični region.

Slika 12.5
Eliptični region za isecanje

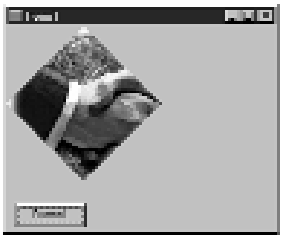
Isprobajmo još jedan tip regiona. Ponovo uklonite liniju koja definiše region i zamenite je sledećim linijama:

```
const
  Points : array[0..3] of TPoint =
    ((X:80;Y:0), (X:0;Y:80), (X:80;Y:160), (X:160;Y:80));
var
  Bitmap : TBitmap;
  Rgn : HRGN;
begin
  Bitmap := TBitmap.Create;
```

nastavlja se

```
  Bitmap.LoadFromFile('handshak.bmp');                    nastavak
  Rgn := CreatePolygonRgn(Points, 4, ALTERNATE);
  SelectClipRgn(Canvas.Handle, Rgn);
  Canvas.Draw(0, 0, Bitmap);
  Bitmap.Free;
end;
```

Ovaj put koristite poligon. Niz points definiše tačke koje kreiraju region. Funkcija `CreatePolygonRgn` kreira region od grupe tačaka. Možete koristiti onoliko tačaka koliko želite. Pošto se region automatski zatvara između prve i poslednje tačke, ne treba da definišete tačku zatvaranja regiona. Ponovo pokrenite program da biste videli efekte Vašeg rada. Slika 12.6. prikazuje rezultate ove vežbe.



Slika 12.6
Poligon regiona za isecanje

NAPOMENA Ovaj isečak koda takođe pokazuje kako da inicijalizujete niz slogova tipa const. Sledi kod koji to pokazuje:

```
const
  Points : array[0..3] of TPoint =
    ((X:80;Y:0), (X:0;Y:80), (X:80;Y:160), (X:160;Y:80));
```

Ovaj kod deklariše niz slogova `TPoint` i inicijalizuje ga upisujući podatke u niz. Niz `TPoint` sadrži dva polja: `x` i `y`. Uočite da je naziv polja ispisan tako da iza naziva sledi dvotačka, nakon koje se nalazi vrednost koja je dodeljena tom polju (`X:80`, na primer). Uočite da su i `x` i `y` polja vrednosti koje se dodeljuju u paru, i da su grupisane u zagradi. Pošto niz sadrži četiri elementa, ovo se ponavlja četiri puta. To je jedini način za deklarisanje i inicijalizaciju niza slogova tipa `const`.

Regioni Vam mogu biti od velike koristi kada obavljate određene operacije crtanja. Možda nećete često koristiti regione za isecanje, ali kada Vam budu zatrebali postaju nezamenljivi.

Osnovne operacije crtanja

Već ste se sreli sa nekim od osnovnih grafičkih rutina u toku rada na primerima u okviru knjige. Do sada ste saznali da se metoda `Rectangle` koristi za crtanje kvadrata i pravougaonika, metoda `Ellipse` koristi za crtanje krugova i ovala (elipsi), a metode `MoveTo` i `LineTo` koriste za crtanje linija.

Takođe postoji i metoda `Arc` koja se koristi za crtanje lukova, kao i metoda Pie koja se koristi za crtanje objekata oblika pite. Sve u svemu, ovo su osnovne funkcije. Nema puno smisla da detaljnije obrađujemo metode klase `TCanvas`. Umesto toga, pređimo na interesantnije (i komplikovanije) grafičke operacije koje ćete sresti u toku rada na Delphi-jevim aplikacijama.

Crtanje teksta

Crtanje teksta ne zvuči suviše komplikovano, zar ne? Istina je da postoji nekoliko zamki koje, ukoliko ih niste svesni, mogu da Vam zagorčaju crtanje teksta. Kao dodatak, postoji nekoliko lepih mogućnosti za crtanje teksta koje treba da upoznate.

Metode `TextOut` i `TextRect`

Metoda `TextOut` je najosnovniji način za crtanje teksta na kanvasu. Ne postoji puno toga što bi moglo da se kaže o metodi `TextOut`. Treba da prosledite X koordinatu, Y koordinatu i tekst koji će biti prikazan; na primer:

```
Canvas.TextOut(20, 20, 'Joshua Reisdorph');
```

Ovaj kod prikazuje dati string na poziciji 20,20 u okviru forme. X i Y koordinate definišu gornji levi ugao teksta koji će biti nacrtan, a ne osnovnu liniju. Da bih ilustrovao ono što želim da Vam objasnim pripremio sam sledeći kod:

```
Canvas.TextOut(20, 20, 'This is a test.');
Canvas.MoveTo(20, 20);
Canvas.LineTo(100, 20);
```

Ovaj kod prikazuje tekst na poziciji 20,20, a zatim crta liniju počev od početne pozicije teksta do pozicije 100,20. Slika 12.7 prikazuje rezultat koda. Uočite da je linija nacrtana na gornjoj ivici teksta.

Slika 12.7. Tekst nacrtan metodom `TextOut`

Metodu `TextOut` možete koristiti svaki put kada Vaš tekst nema zahteva za posebnih pozicioniranjem linije.

Metoda `TextRect` Vam omogućava da definišete pravougaonik za isecanje kao dodatak tekstu koji će biti prikazan. Ovu metodu možete koristiti kada tekst mora biti postavljen unutar određenih granica. Bilo koji tekst koji se nalazi van definisane granice će biti odsečen. Sledeći isečak koda osigurava da dužina teksta koja će biti prikazana na prelazi 100 piksela:

```
Canvas.TextRect(Rect(20, 50, 120, 70), 20, 50,
  'This is a very long line that might get clipped.');
```

Obe metode `TextOut` i `TextRect` mogu crtati samo jednu liniju teksta. Nije dozvoljeno prebacivanje teksta u sledeću liniju.

SAVET Da biste nacrtali tekst koristeći tabulatore, pogledajte Windows API funkciju `TabbedTextOut`.

Boja pozadine teksta

Pogledajte sliku 12.7. Uočite da tekst ima belu boju pozadine, što i ne predstavlja dobro rešenje, pošto je forma sive boje. Pozadina teksta je preuzeta iz tekuće boje četke (generički, bela boja). Da biste popravili ovo stanje na slici 12.7, treba da preduzmete jednu od sledeće dve aktivnosti: ili promenite boju četke kanvasa, ili učinite boju pozadine teksta transparentnom. Promena boje pozadine teksta je veoma jednostavna. Postavlja se samo pitanje da li znate u koju boju treba da promenite boju pozadine? U ovom slučaju, boja pozadine teksta može biti ista kao i boja forme, pa možete uraditi sledeće:

```
Canvas.Brush.Color := Color;
```

U većini situacija ovaj kod će imati efekta, ali u nekim slučajevima Vam je potrebna veća kontrola. Jednostavnije će biti ukoliko učinite pozadinu teksta transparentnom. Dobra vest je da to možete da učinite. Evo kako treba da izgleda kod:

```
var
  OldStyle : TBrushStyle;
begin
  OldStyle := Canvas.Brush.Style;
  Canvas.Brush.Style := bsClear;
  Canvas.TextOut(20, 20, 'This is a test.');
  Canvas.Brush.Style := OldStyle;
end;
```

Prvo, snimite tekući stil četke. Zatim podesite stil četke da bude transparentan (`bsClear`). Nakon prikazivanja teksta, podesite stil četke, na stil koji je bio definisan pre prethodne operacije. Treba da steknete naviku, da nakon što ste obavili crtanje teksta, vratite prethodno snimljeni stil. Vraćanje stila na prethodni, uvek predstavlja dobru ideju, pošto najverovatnije nećete želeti da ostavite stil četke definisan kao transparentan.

Korišćenje transparentne pozadine ima nekih drugih prednosti. Recimo da želite da prikažete tekst preko pozadine koja je predstavljena bitmapom. U tom slučaju ne možete koristiti pozadinu koja nije transparentna. Naredni isečak koda ilustruje ovaj problem (datoteka `FACTORY.BMP` se nalazi u direktorijumu `Borland Shared Files\Images\Splash\256Color`):

```
var
  OldStyle : TBrushStyle;
  Bitmap   : TBitmap;
begin
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile('factory.bmp');
  Canvas.Draw(0, 0, Bitmap);
  Canvas.Font.Name := 'Arial Bold';
  Canvas.Font.Size := 13;
  OldStyle := Canvas.Brush.Style;
  Canvas.Brush.Style := bsClear;
  Canvas.TextOut(20, 5, 'Transparent Background');
  Canvas.Brush.Style := OldStyle;
  Canvas.TextOut(20, 30, 'Solid Background');
  Bitmap.Free;
end;
```

Ovaj kod crta bitmapu na formi. Zatim se iscrtava tekst u okviru forme (preko bitmape) korišćenjem transparentne pozadine. Nakon toga, naredni tekst je nacrtan korišćenjem standardne pozadine. Slika 12.8 prikazuje rezultat rada ovog isečka koda. Kao što možete videti, ukoliko je pozadina transparentna, slika će puno bolje izgledati.

Slika 12.8. Tekst nacrtan preko bitmape sa transparentnom i netransparentnom pozadinom

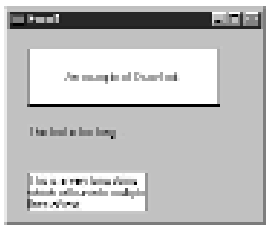
Još jedan razlog za korišćenje transparentne pozadine teksta je prikazan u lekciji dana 13 u poglavlju "Statusni panoi koje definiše korisnik". U ovom primeru ćete tekstu statusne linije dati 3D izgled crtajući beli tekst malo pomeren u odnosu na crni tekst. Jedini način da ovaj kod postigne željeni efekat, je korišćenje transparentne pozadine. Kao što možete videti, ponekad je transparentna pozadina jedini način za postizanje određene vrste efekta.

Funkcija `DrawText`

Windows API funkcija `DrawText` Vam pruža puno veću kontrolu nad tekstom koji se crta na kanvasu u odnosu na funkciju `TextOut`. Iz nekog razloga klasa `TCanvas` nema metodu `DrawText`. Da biste koristili metodu `DrawText`, potrebno je da direktno koristite Windows API. Prvo ćete pogledati osnovni `DrawText` primer, a zatim ću Vam objasniti detaljnije snagu ove funkcije:

```
var
  R : TRect;
begin
  R := Rect(20, 20, 220, 80);
  Canvas.Rectangle(20,20, 220, 80);
  DrawText(Canvas.Handle, 'An example of DrawText.',
    -1, R, DT_SINGLELINE or DT_VCENTER or DT_CENTER);
end;
```

Slika 12.9 prikazuje rezultat ovog koda, kao i rezultat primera koji će Vam biti predstavljeni kasnije



Slika 12.9
Primeri funkcije `DrawText`

Prvo se inicijalizuje slog `TRect` korišćenjem Windows API funkcije pod nazivom `Rect`. Nakon toga, na kanvasu će biti nacrtan običan pravougaonik. (Pravougaonik je na kanvas nacrtan tako da veličinu pravougaonika, u koji ćete crtati, možete lako da vidite.) Na kraju, funkcija `DrawText` se poziva kako bi se iscrtao tekst.

Posvetićemo nekoliko trenutaka pažnje različitim parametrima ove funkcije:

- Prvi parametar se koristi da definiše kontekst uređaj na kom će se crtati. Parametar `Handle` objekta `TCanvas` je `HDC` (upravljač kontekst uređajem) kanvasa, pa ćete ga proslediti kao prvi parametar.
- Drugi parametar je string koji će biti prikazan.
- Treći parametar se koristi za definisanje broja karaktera koji će biti iscrtani. Ukoliko je ovaj parametar podešen na vrednost `-1`, svi karakteri u okviru stringa će biti iscrtani.
- Četvrti parametar funkcije `DrawText` je promenljiva `TRect`. Ovaj parametar je tipa `var`, pošto neke operacije funkcije `DrawText` menjaju prosleđeni pravougaonik.
- Način na koji će se funkcija `DrawText` ponašati je predstavljen poslednjim parametrom. Ovaj parametar se koristi da definiše zastavice (flags) koje se koriste prilikom crtanja teksta. U ovom primeru koristite zastavice `DT_SINGLELINE`, `DT_VCENTER` i DT_CENTER. Ove zastavice saopštavaju Windows operativnom sistemu da je tekst jedna linija i da treba da bude centriran i po vertikali i po horizontali. U osnovi postoji skoro dvadeset zastavica koje možete

definisati u funkciji `DrawText`. Neću posebno obrađivati svaku zastavicu, pošto kompletan spisak možete videti u delu sistema za pomoć u toku rada, u koji se odnosi na Win32 API.

Prethodni primer ilustruje jednu od najčešćih upotreba funkcije `DrawText`: centriranje teksta po horizontali, po vertikali, odnosno centriranje teksta i po horizontali i po vertikali. Ovo je veoma dobra mogućnost kada kreirate komponente po kojima korisnik može da crta. U suštini, okviri za listu, kombo okviri i meniji koje kreira korisnik obično zahtevaju da tekst bude centriran. Možda nećete shvatiti pogodnosti ove funkcije u ovom trenutku, ali kada počnete da pravite komponente koje će korisnik moći sam da upotrebljava, odnosno kada budete počeli da pišete sopstvene grafičke komponente, prednosti ove funkcije će Vam biti mnogo jasnije.

Još jedna interesantna zastavica funkcije `DrawText` je `DT_END_ELLIPSIS`. Ukoliko je tekst suviše dug da bi ispunio predviđeni pravougaonik, Windows će odseći string i na kraj stringa dodati tri tačke (ellipsis), označavajući da je string odsečen. Kao primer možete uzeti sledeći kod:

```
var
  R : TRect;
begin
  R := Rect(20, 100, 120, 150);
  DrawText(Canvas.Handle, 'This text is too long to fit.',
    -1, R, DT_END_ELLIPSIS);
end;
```

Nakon izvršavanja ovog koda kao rezultat biće ispisan tekst:

```
This text is too long...
```

Ovu zastavicu možete koristiti svaki put kada pretpostavite da bi tekst mogao biti suviše dug u odnosu na pravougaonik u kom će biti iscrtan.

Od neprocenjive vrednosti je još jedna zastavica: `DT_CALCRECT`. Ova zastavica izračunava visinu pravougaonika koja je potrebna da bi se definisani tekst prikazao. Kada koristite ovu zastavicu, Windows izračunava potrebnu visinu i vraća rezultat, ali ne ispisuje tekst. Potrebno je da saopštite Windows operativnom sistemu koje će širine biti pravougaonik, a Windows će Vam saopštiti koja je visina pravougaonika potrebna da bi se prikazao kompletan tekst. U suštini, Windows istovremeno modifikuje elemente `bottom` i `left` definisanog pravougaonika, tako da može da prikaže željene vrednosti. Ovo je veoma važno ukoliko iscrtavate više linija teksta.

Sledeći primer postavlja pitanje Windows operativnom sistemu kako bi otkrio koja je visina potrebna da bi pravougaonik prikazao kompletan tekst. Nakon toga će pravougaonik biti nacrtan na ekranu. Na kraju će u pravougaonik biti ucrtan tekst. Sledi kod:

```
var
  R : TRect;
  S : string;
begin
  R := Rect(20, 150, 150, 200);
  S := 'This is a very long string which will ' +
    'run into multiple lines of text.';
  DrawText(Canvas.Handle, PChar(S),
    -1, R, DT_CALCRECT or DT_WORDBREAK);
  Canvas.Brush.Style := bsSolid;
  Canvas.Rectangle(R.left, R.top, R.right, R.bottom);
  Canvas.Brush.Style := bsClear;
  DrawText(Canvas.Handle, PChar(S), -1, R, DT_WORDBREAK);
end;
```

Uočite kako je prosleđena promenljiva `S` funkciji `PChar` koja predstavlja drugi parametar funkcije `DrawText`. Ovo je neophodno pošto funkcija `DrawText` zahteva pointer kao parametar teksta, a ne promenljivu tipa `string`.

Ukoliko pišete komponente koje će moći da se koriste u svim verzijama Delphi-ja, ne možete proslediti vrednost stringa funkciji PChar. U Delphi-ju 1, ovo prosleđivanje neće moći da bude prevedeno. Umesto toga treba da koristite funkciju StrPCopy, kao što je prikazano na primeru:

```
var
  Temp : array [0..50] of Char;
  R : TRect;
  S : string;
begin
  DrawText(Canvas.Handle, StrPCopy(Temp, S),
    -1, R, DT_CALCRECT or DT_WORDBREAK);
{ etc. }
```

Ukoliko ne morate da brinete o podršci Delphi-ju 1, funkciji `PChar` možete proslediti `string` kao što je to navedeno u prethodnom primeru.

Ovaj kod treba da postavite u upravljač događajem `OnPaint` tekuće forme. Pokrenite program nekoliko puta, menjajući dužinu tekst stringa koji treba da bude prikazan. Uočite da će, bez obzira koliko teksta dodavali Vašem stringu, pravougaonik uvek biti tačno iscrtan oko teksta. Pogledajte sliku 12.9 koja prikazuje rezultate tekuće i ranijih vežbi koje koriste funkciju `DrawText`.

Ukoliko Vam je potrebno još više opcija za crtanje teksta, možete koristiti funkciju `DrawTextEx`. Za detaljnije informacije možete pogledati Win32 API sistem za pomoć u toku rada.

Crtanje teksta korišćenjem funkcije `DrawText` je nešto sporije od korišćenja metode `TextOut`. Ukoliko su Vaše operacije za iscrtavanje teksta osetljive na brzinu, treba da koristite metodu `TextOut`, umesto funkcije `DrawText`. Na ovaj način ćete morati samostalno da odradite više stvari, ali će i brzina izvršavanja biti veća. Ipak, kod većine primera nećete uočiti nikakvu razliku između metode `TextOut` i funkcije `DrawText`.

Funkcija `DrawText` je veoma korisna i moćna. Kada budete pisali sopstvene komponente, nema sumnje da ćete veoma često koristiti ovu funkciju.

Crtanje bitmapa

Crtanje bitmapa Vam može zvučati kao veoma teško, ali ste nekoliko puta do sada mogli videti da je crtanje bitmapa veoma jednostavno. Klasa `TCanvas` sadrži nekoliko metoda za crtanje bitmapa. Najčešće korišćena metoda je `Draw`. Ova metoda jednostavno crta bitmapu (predstavljenu kao naslednicu klase `TGraphic`) na kanvas; bitmapa se nalazi na definisanoj poziciji. Do sada ste već mogli da vidite nekoliko primera, pa nije na odmet da pogledate još jedan:

```
var
  Bitmap : TBitmap;
begin
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile('c:\winnt\winnt256.bmp');
  Canvas.Draw(0, 0, Bitmap);
  Bitmap.Free;
end
```

Ovaj kod kreira objekt `TBitmap`, učitava datoteku pod nazivom `WINNT256.BMP`, a zatim prikazuje datoteku u gornjem levom uglu forme. Ukoliko želite da prikažete bitmape bez izmena, treba da koristite metodu `Draw`.

Metoda `StretchDraw` se koristi kada je potrebno menjati veličinu bitmape. Možete definisati pravougaonik na poziciji bitmape i sliku koju treba iscrtati. Ukoliko je predviđeni pravougaonik veći od originalne veličine bitmape, bitmapa će biti raširena. Ukoliko je manji od originalnih dimenzija bitmape, slika će biti umanjena kako bi mogla da stane u pravougaonik. Sledi primer:

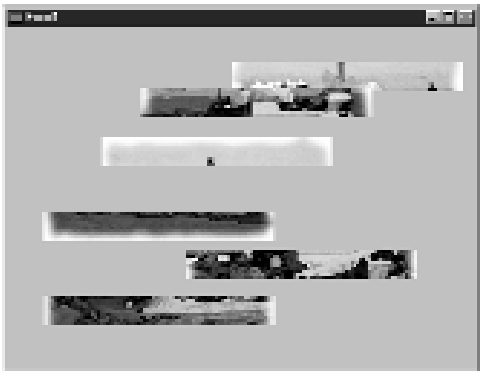
```
var
  Bitmap : TBitmap;
  R : TRect;
begin
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile('c:.bmp');
  R := Rect(0, 0, 100, 100);
  Canvas.StretchDraw(R, Bitmap);
  Bitmap.Free;
end;
```

Metoda `StretchDraw` ne pokušava da održi originalni odnos širine i visine. Na Vama je da obezbedite originalni odnos širine i visine bitmape.

Još jedna metoda za crtanje bitmapa je `CopyRect`. Ova metoda Vam omogućava da definišete i izvorni i odredišni pravougaonik. Ovo Vam omogućava da podelite bitmapu u delove kada želite da je prikažete. Kao primer možete pogledati sledeći kod:

```
var
  Bitmap : TBitmap;
  Src : TRect;
  Dst : TRect;
  I, X, Y : Integer;
  Strips : Integer;
  Stripsize : Integer;
  OldPal : HPalette;
begin
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile('factory.bmp');
  Strips := 6;
  Stripsize := (Bitmap.Height div Strips);
  OldPal := SelectPalette(Canvas.Handle, Bitmap.Palette, True);
  for I := 0 to Pred(Strips) do begin
    Src := Rect(0, i * Stripsize,
      Bitmap.Width, (i * Stripsize) + Stripsize);
    X := Random(Width - Bitmap.Width);
    Y := Random(Height - Stripsize);
    Dst := Rect(X, Y, X + Bitmap.Width, Y + Stripsize);
    Canvas.CopyRect(Dst, Bitmap.Canvas, Src);
  end;
  SelectPalette(Canvas.Handle, oldPal, True);
  Bitmap.Free;
end;
```

Ovaj kod učitava bitmapu, seče je na trake, a zatim prikazuje isečene trake na slučajnim delovima forme. Slika 12.10 prikazuje primer rada koda. Upišite dati kod u upravljač događajem `OnPaint` Vaše glavne forme, a zatim pokrenite program. Pokrijte glavnu formu, a zatim je ponovo prebacite na vrh. Slike će biti ponovo iscrtane nakon iscrtavanja same forme.



Slika 12.10
Delovi bitmape iscrtani nasumice na ekranu, korišćenjem metode `CopyRect`

Kopiranje delova bitmape kao što je bitmapa `FACTORY.BMP` Vam u početku možda neće izgledati kao operacija koja ima smisla. Uobičajena tehnika programiranja

grafike zahteva kreiranje velike bitmape koja se sastoji od nekoliko manjih slika i njihovog kopiranja u sliku koju želite da prikažete na ekranu. U ovakvim slučajevima, metoda `CopyRect` predstavlja način na koji treba da radite.

U prethodnom primeru koda sam koristio funkciju `SelectPalette`, kako bih uskladio paletu forme sa karakteristikom `Palette` bitmape. Iz nekog čudnog razloga, klasa `TCanvas` nema karakteristiku `Palette`, pa ćete morati da koristite API, kako biste podesili paletu formi. Da nisam podesio paletu forme, boje bitmape bi bile različite od originalnih prilikom prikazivanja bitmapi na formi. Metoda `CopyRect` koristi različite mehanizme za prikazivanje bitmapa u okviru kanvasa, pa je korišćenje dodatnog koraka neophodno onda kada želite da koristite ovu metodu.

Postoji još jedna metoda za crtanje bitmapa koju želim da spomenem. Metoda `BrushCopy` Vam omogućava da definišete izvorni pravougaonik, pravougaonik odredište, sliku i transparentnu boju. Sistem za pomoć u toku rada Vas upućuje da umesto metode `BrushCopy` koristite komponentu `ImageList`. Ovo je po mom mišljenju pomalo ekstremno. Postoje trenutci kada metoda `BrushCopy` radi veoma dobro i puno je jednostavnije koristiti ovu metodu umesto komponente `ImageList`. Nemojte previdetи metodu `BrushCopy`, ukoliko koristite bitmape sa transparentnom pozadinom.

Vanekranske bitmape

Vanekranske bitmape se takođe nazivaju i memorijske bitmape i često se koriste u Windows programiranju. Vanekranske bitmape Vam omogućavaju da crtate slike u memoriji, a zatim ih prikazujete na ekranu korišćenjem metode `Draw`. Vanekranske bitmape Vam omogućavaju da izbegnete treperenje koje možete videti u pokušajima da suviše direktno crtate po ekranu u kratkom vremenskom periodu. Vanekranske bitmape su takođe dobre za programe koje koriste kompleksne programe za crtanje. Sliku možete pripremiti u memoriji, a zatim je prikazati na ekranu, kada je sve spremno. Vanekranske bitmape se koriste u animaciji, a najpopularnija nova tehnologija vezana za animaciju je Microsoft DirectX SDK.

Princip rada vanekranskih bitmapa je proces od tri koraka:

1. Kreirajte memorijsku bitmapu.
2. Upišite bitmapu u memoriju.
3. Kopirajte memorijsku bitmapu na ekran.

Kreiranje memorijske bitmape

Kreiranje memorijske bitmape je veoma jednostavno. U suštini, ovo ste već radili nekoliko puta u današnjoj lekciji. Iznenađeni ste? Svaki put kada kreirate objekt `TBitmap`, kreirali ste memorijsku bitmapu. U ovakvim slučajevima, učitali ste

datoteku u memorijsku bitmapu. U suprotnom, kreirali ste memorijsku bitmapu, podesili veličinu , a zatim je iscrtali. Na primer:

```
var
  Bitmap : TBitmap;
  I, X, Y, W, H : Integer;
  Red, Green, Blue : Integer;
begin
  Bitmap := TBitmap.Create;
  Bitmap.Width := 500;
  Bitmap.Height := 500;
  for I := 0 to 19 do begin
    X := Random(400);
    Y := Random(400);
    W := Random(100) + 50;
    H := Random(100) + 50;
    Red := Random(255);
    Green := Random(255);
    Blue := Random(255);
    Bitmap.Canvas.Brush.Color := RGB(Red, Green, Blue);
    Bitmap.Canvas.Rectangle(X, Y, W, H);
  end;
  Canvas.Draw(0, 0, Bitmap);
  Bitmap.Free;
end;
```

Da biste isprobali ovaj kod postavite dugme na formu, a zatim upišite kod u upravljač događajem `OnClick` dugmeta koje se nalazi na formi. Svaki put kada kliknete na dugme, novi skup slučajno razmeštenih pravougaonika će biti iscrtan na ekranu. Ovaj kod jednostavno prebacuje bitmapu u memoriju, a zatim kopira bitmapu iz memorije na kanvas forme. Ukoliko koristite video karticu sa 256 boja, boje će biti izmenjene, pošto za ovu vežbu ne koristite paletu.

Kada kreirate memorijsku bitmapu, ova bitmapa mora imati isti broj boja koji je definisan Windows operativnim sistemom u odeljku za podešavanje video kartice. Drugim rečima, ukoliko je Vaša video kartica podešena da prikazuje 256 boja, memorijska bitmapa će biti definisana kao bitmapa sa 256 boja. Ukoliko je video kartica Vašeg računara podešena da prikazuje grafiku sa 24, ili 32 bita, Vaša memorijska bitmapa će sadržati 32 hiljade, 64 hiljade, odnosno 16 miliona boja.

Snimanje memorijske bitmape

Snimanje memorijske bitmape u datoteku je veoma jednostavno. Evo kako se to radi:

```
Bitmap.SaveToFile('test.bmp')
```

Da, to je to. U suštini, na ovaj način možete kreirati sopstveni program za hvatanje ekrana. Sve što je potrebno je kreiranje određenog dela radne površine u memorijsku bitmapu, a zatim snimanje bitmape u datoteku.

Ovo možete videti na listingu 12.1.

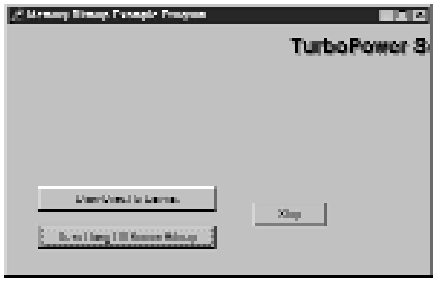
Listing 12.1: Rutina za hvatanje ekrana (screen capture)

```
procedure MainForm.CaptureButtonClick(Sender: TObject);
var
  DtCanvas  : TCanvas;
  Bitmap    : TBitmap;
  NumColors : Integer;
  LogPal    : PLogPalette;
  Src, Dst  : TRect;
begin
  { Create a TCanvas object for the desktop DC. }
  DtCanvas := TCanvas.Create;
  DtCanvas.Handle := GetDC(0);

  { Create a new TBitmap object and set its }
  { size to the size of the form.           }
  Bitmap := TBitmap.Create;
  Bitmap.Width := Width;
  Bitmap.Height := Height;

  { Create a palette from the form's Canvas  }
  { and assign that palette to the Bitmap's  }
  { Palette property.                        }
  NumColors := GetDeviceCaps(Canvas.Handle, SizePalette);
  GetMem(LogPal, SizeOf(TLogPalette) +
    (NumColors - 1) * SizeOf(TPaletteEntry));
  LogPal.palVersion  := $300;
  LogPal.palNumEntries := NumColors;
  GetSystemPaletteEntries(
    Canvas.Handle, 0, NumColors, LogPal.palPalEntry);
  Bitmap.Palette := CreatePalette(LogPal^);
  FreeMem(LogPal);

  { Copy a section of the screen from the   }
  { desktop canvas to the Bitmap.           }
  Src := BoundsRect;
  Dst := Rect(0, 0, Width, Height);
  Bitmap.Canvas.CopyRect(Dst, DtCanvas, Src);

  { Save it to disk. }
  Bitmap.SaveToFile('form.bmp');
  { Clean up and go home. }
  Bitmap.Free;
  DtCanvas.Free;
end;
```

NAPOMENA Ovaj kod proširuje izloženo gradivo i implementira paletu za formu u slučaju da forma prikazuje grafiku. Ovaj kod je preveden korišćenjem koda koji je originalno definisan u članku koji sam napisao za Cobb Group's C++Builder Developer's Journal. Iako možda niste zainteresovani za novine vezane za C++Builder, možda ćete biti zainteresovani za Delphi Developer's Journal. Možete se pretplatiti na besplatan primerak časopisa Delphi Developer's Journal na Web sajtu firme Cobb Group na adresi: `http://www.cobb,com/ddj`.

Primer programa za memorijsku bitmapu

Listing 12.2 sadrži program pod nazivom `MemBmp` koji prikazuje korišćenje memorijskih bitmapa. Ovaj program pomera marker preko ekrana, ukoliko kliknete na jedno od dva dugmeta. Prvo dugme pomera tekst preko ekrana, a da ne koristi memorijsku bitmapu (piše direktno na kanvas forme). Drugo dugme koristi memorijsku bitmapu, kako bi se bitmapa ravnomernije pomerala. Treće dugme se koristi za zaustavljanje markera. Slika 12.11 prikazuje program `MemBmp` u toku rada.



Slika 12.11 Program `MemBmp` u toku rada

Listing 12.2: `MemBmpU.pas`.

```
unit MemBmpU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

const
  DisplayText = 'TurboPower Software Co.';
type
  TForm1 = class(TForm)
    Button1: TButton;
    Button2: TButton;
    Button3: TButton;
    procedure Button1Click(Sender: TObject);
    procedure Button2Click(Sender: TObject);
    procedure Button3Click(Sender: TObject);
```

```
  private
    { Private declarations }
    Done : Boolean;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.Button1Click(Sender: TObject);
var
  I : Integer;
begin
  Canvas.Font.Name := 'Arial Bold';
  Canvas.Font.Size := 16;
  Canvas.Brush.Color := clSilver;
  Done := false;
  while not Done do begin
    for I := -Canvas.TextWidth(DisplayText)
      to Pred(Width) do begin
      Sleep(1);
      Application.ProcessMessages;
      if Done then
        Break;
      Canvas.Font.Color := clGray;
      Canvas.Brush.Style := bsClear;
      Canvas.TextOut(i + 2, 12, DisplayText);
      Canvas.Font.Color := clBlack;
      Canvas.Brush.Style := bsClear;
      Canvas.TextOut(i, 10, DisplayText);
      Canvas.Font.Color := clSilver;
      Canvas.TextOut(i + 2, 12, DisplayText);
      Canvas.TextOut(i, 10, DisplayText);
    end;
  end;
end;

procedure TForm1.Button2Click(Sender: TObject);
var
  Bitmap : TBitmap;
  I : Integer;
begin
  Bitmap := TBitmap.Create;
  Bitmap.Width := Width;
  Bitmap.Height := 40;
```

nastavlja se

Listing 12.2: `MemBmpU.pas`. *nastavak*

```
    Bitmap.Canvas.Font.Name := 'Arial Bold';
    Bitmap.Canvas.Font.Size := 16;
    Bitmap.Canvas.Brush.Color := clSilver;
    Bitmap.Canvas.FillRect(Rect(0, 0, Width, 40));
    Done := False;
    while not Done do begin
      for I := -Bitmap.Canvas.TextWidth(DisplayText)
        to Pred(Width) do begin
        Application.ProcessMessages;
        if (Done) then
          Break;
        Sleep(1);
        Bitmap.Canvas.Font.Color := clGray;
        Bitmap.Canvas.Brush.Style := bsClear;
        Bitmap.Canvas.TextOut(2, 12, DisplayText);
        Bitmap.Canvas.Font.Color := clBlack;
        Bitmap.Canvas.Brush.Style := bsClear;
        Bitmap.Canvas.TextOut(0, 10, DisplayText);
        Canvas.Draw(I, 0, Bitmap);
      end;
    end;
    Bitmap.Free;
  end;

  procedure TForm1.Button3Click(Sender: TObject);
  begin
    Done := True;
  end;

  end.
```

Program `MemBmp` je uključen u kod knjige zajedno sa ostalim programima primerima (pod nazivom `BrushTst`, `Capture`, `Gradient` i `Rotate`) koji ilustruju koncepte koji su danas obrađeni.

Programiranje multimedije

Programiranje multimedije je fraza koja Vas može obmanuti svojom nevinošću. Razlog leži u činjenici što prograniranje multimedije poseduje širok spektar mogućnosti. Multimedija tipično uključuje wave audio, MIDI audio, AVI video inserte i animaciju. Ne bih želeo da mešam programiranje multimedije sa programiranjem igara.

Sigurno je da programiranje igara uključuje multimediju, ali je multimedija više uključena nego što je to slučaj sa dodavanjem zvuka i video inserata u klasičnu aplikaciju. U ovom poglavlju ću delimično pokriti multimediju i pokazaću Vam šta

možete uraditi sa alatima dostupnim u Delphi-ju koji se mogu pronaći u okviru Delphi-jevog paketa. Neću objašnjavati grafiku, ili multimedija API, kao što je OpenGL, odnosno Direct Draw. Detaljnije informacije o programiranju grafike koja koristi date biblioteke će Vam pružiti dobra knjiga specijalizovana za programiranje grafike. *Delphi 4 do kraja* (ISBN: 0-672-312-859) bi bio dobar izbor.

Wave audio u saradnji sa Windows API-jem

Očigledno je da neću puno objašnjavati Windows API funkcije, pošto u većini slučajeva VCL pruža mnogo bolji način za izvršavanje zadataka. U slučaju pokretanja zvučne datoteke nema ništa jednostavnije od Win32 API funkcije `PlaySound`. Izvršavanje zvučne datoteke korišćenjem ove funkcije je veoma jednostavno. Prva stvar koju treba da uradite je dodavanje datoteke `MmSystem` u Vašu uses listu. Zatim možete pozvati funkciju `PlaySound` koristeći odgovarajuće parametre:

```
PlaySound('test.wav', 0, SND_FILENAME);
```

Ovo je prilično jednostavno, zar ne? Kao što ste mogli da vidite prvi parametar funkcije `PlaySound` se koristi za definisanje naziva zvučne datoteke koja se izvršava. Poslednji parametar se koristi da definiše zastavicu koja određuje kako će zvuk biti izveden. Kada izvršavate wave audio datoteku sa diska, treba da parametar `SND_FILENAME` definišete kao poslednji (isto tako možete definisati i druge zastavice, koje će biti obrađene u nastavku).

Funkcija `PlaySound` takođe može izvoditi i sistemske zvuke, kao što izvodi datoteke koje se nalaze na disku. Da biste pokrenuli sistemski zvuk, treba da definišete alias sistemskog zvuka kao prvi parametar koji se prosleđuje funkciji `PlaySound`, a kao parametar zastavicu treba da definišete `SND_ALIAS`. Na primer :

```
PlaySound('SystemAsterisk', 0, SND_ALIAS);
```

Ovaj kod izvršava sistemski zvuk dodeljen događaju Asterisk (pretpostavljajući da je takav zvuk definisan na Vašem sistemu). Da biste videli spisak sistemskih događaja za zvuk, pogledajte aplet Sound u prozoru Control Panel. Da biste odredili alias zvuka za događaj, možete pogledati Windows Registration Database (Regristry datoteku). Spisak aliasa je prikazan pod ključem `HKEY_CURRENT_USER`.

Ukoliko ne možete da pronađete traženi zvuk, Windows će izvršiti generički zvuk (ukoliko koristite generičku šemu zvuka, to će biti zvuk *ding*). Možete sprečiti da Windows izvrši generički zvuk, ukoliko definišete zastavicu `SND_NODEFAULT`. Na primer, ukoliko želite da izvršite sistemski zvuk, a ne želite da se izvrši generički zvuk, ukoliko sistem nije u mogućnosti da pronađe sistemski zvuk, možete koristiti sledeći kod :

```
PlaySound('MailBeep', 0, SND_ALIAS or SND_NODEFAULT);
```

Uočite da su zastavice `SND_ALIAS` i `SND_NODEFAULT` ispisane tako da se operator or nalazi između ove dve zastavice.

NAPOMENA Funkcija Win32 API-ja, `MessageBeep` se isto tako može koristiti za izvršavanje sistemskih zvukova, ukoliko se koriste indeksni brojevi. Za dodatne informacije možete pogledati sistem za pomoć u toku rada, koji je vezan za funkciju `MessageBeep` u okviru Win32 API-ja.

Postoje još dve zastavice koja su veoma bitne u radu sa funkcijom `PlaySound`:

- `SND_ASYNC` zastavica obezbeđuje asinhrono izvršavanje zvuka. Ukoliko koristite ovaku zastavicu, zvuk će početi da se izvršava, a zatim će trenutno vratiti kontrolu aplikaciji koja ga je pozvala. Ovo znači da će zvuk moći da se izvršava u toku rada aplikacije.
- `SND-SYNC` zastavica omogućava da se kontrola izvršavanja vraća aplikaciji koja je zvuk pozvala tek pošto prestane izvršavanje zvuka. Zastavica `SND_SYNC` se koristi kao generička zastavica funkcije `PlaySound`, stoga ovu zastavicu ne treba definisati posebno.

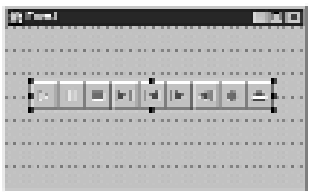
Postoje mnoge druge zastavice koje se mogu koristiti za kontrolu izvršavanja zvuka korišćenjem funkcije `PlaySound`. Za kompletne informacije možete pogledati Win32 sistem za pomoć u toku rada definisan za funkciju `PlaySound`.

Komponenta `TMediaPlayer`

VCL sadrži komponentu `MediaPlayer` koja se koristi za jednostavne multimedija operacije. Ova komponenta se nalazi u kartici System u okviru palete komponenti i može izvršavati zvučne datoteke, MIDI datoteke, AVI video datoteke, itd. Korišćenje klase `TMediaPlayer` je, takođe, jednostavno. Za samostalno izvršavanje zvučnih datoteka obično koristim funkciju `PlaySound` koja je opisana u prethodnom poglavlju. Za kompleksnije stvari možete koristiti komponentu `MediaPlayer`.

Najočigledniji primer korišćenja klase `TMediaPlayer` je jednostavno postavljanje ove komponente na formu. Nakon što postavite komponentu na formu, biće prikazana kontrola za izvođenje multimedijalnih datoteka. Ova kontrola sadrži dugmad za izvođenje, pauzu, zaustavljanje, brz prelazak unapred i unazad, preskakanje unapred i unazad, snimanje i izbacivanje (play, pause, stop, next, previous, step, back, record, eject). Slika 12.12. prikazuje formu sa komponentom `MediaPlayer`.

Slika 12.12 forma sa komponentom MediaPlayer.



Slika 12.12 Forma sa komponentom `MediaPlayer`

Korišćenje komponente MediaPlayer u osnovnoj formi je veoma jednostavno. Sve što treba da uradite je da postavite karakteristiku `FileName` na naziv odgovarajuće multimedijalne datoteke, a zatim kliknete na dugme Play komponente `MediaPlayer`. Možete odabrati bilo koju .WAV, .MID, odnosno .AVI datoteku. Komponenta `MediaPlayer` ima ugrađen mehanizam za automatsko prepoznavanje tipa datoteke, pa dodatna intervencija nije potrebna. U većini slučajeva, biste želeli da uradite nešto interesantnije sa komponentom `MediaPlayer`, mada je za takve aktivnosti potreban ozbiljniji rad.

Iako je u nekim situacijama vizuelna kontrolna traka komponente `MediaPlayer` veoma lepa, verovatno ćete ovu komponentu koristiti povremeno bez kontrolne trake. Komponentom `MediaPlayer` možete manipulisati koristeći kod kako biste izvršili, startovali, zaustavili, odnosno premotali medija datoteku. Ukoliko ne želite da vidite kontrolnu traku komponente MediaPlayer u toku izvršavanja, potrebno je da karakteristiku `Visible` postavite na `False`.

Karakteristike, metode i događaji komponente `MediaPlayer`

Klasa `TMediaPlayer` sadrži mnoštvo karakteristika. Mnoge od njih možete veoma lako razumeti, dok je manji broj nešto komplikovaniji. Tabela 12.5. prikazuje primarne karakteristike klase `TMediaPlayer`.

Tabela 12.5: Primarne karakteristike klase `TMediaPlayer`

Kakteristika	Opis
`AutoOpen`	Određuje da li će uređaj biti otvoren nakon kreiranja komponente MediaPlayer. Generički: `False`.
`AutoRewind`	Kada je vrednost `True`, pozicija pokazivača će biti vraćena na početak datoteke, nakon što se završi izvođenje datoteke. Generički: `True`.
`DeviceType`	Tip multimedija uređaja. Ukoliko želite da se tip uređaja automatski prilagođava nastavku datoteke, odaberite `dtAutoSelect`. Generički: `dtAutoSelect`.
`Display`	Koristi se da prikaže prozor na odgovarajuću komponentu (za video uređaje).
`DisplayRect`	Koristi se da podesi veličinu i poziciju prozora za izvođenje kod video uređaja. Veličina video platna će biti prilagiđena veličini pravougaonika.
`EnabledButtons`	Dugmad komponente `MediaPlayer` će biti aktivirana. Generički: All Buttons (sva dugmad).
`EndPos`	Krajnja pozicija medija datoteke. Medija datoteka se izvodi od `StartPos` (početna pozicija) do `EndPos` (krajnja pozicija). Ukoliko nije definisana krajnja pozicija (`EndPos`) medija datoteka se

nastavlja se

Tabela 12.5: Primarne karakteristike klase `TMediaPlayer` *nastavak*

Kakteristika	Opis
	izvodi do kraja. Vrednost koju dodelite oznaci `EndPos` zavisi od tipa medija datoteke.
`Error`	Kod greške za poslednju operaciju.
`ErrorMessage`	Tekst opis poslednje greške.
`Frames`	Broj delova medija datoteke za koje će se pokazivač pomeritii prilikom poziva metoda `Back` ili `Next`, odnosno nakon pritiska na dugmad Back, ili Next na kontrolnoj traci komponente `MediaPlayer`.
`Length`	Dužina medija datoteke. Vrednost karakteristike `Length` zavisi od tipa medija datoteke koja se izvodi i trenutne vrednosti karakteristike `TimeFormat`.
`Mode`	Stanje uređaja. Može biti: mpNotReady, `mpStopped`, mpPlaying, `mpRecording`, `mpSeeking`, `mpPaused` i `mpOpen`.
`Notify`	Kada je vrednost `True`, generiše se događaj `OnNotify`, nakon što komponenta `MediaPlayer` završi operaciju.
`NotifuValue`	Rezultat poslednje operacije za obaveštavanje. Vrednosti mogu biti: `nvSuccessful`, `nvSuperseded`, `nvAborted` i `nvFailure`.
`Position`	Trenutna pozicija u okviru medija datoteke.
`StartPos`	Početna pozicija medija datoteke. Medija datoteka se izvodi od oznake `StartPos` do oznake `EndPos`. Ukoliko karakteristika `StartPos` nije definisana, medija datoteka se izvodi od početka. Vrednost koju dodeljujete karakteristici `StartPos` zavisi od tipa medija datoteke koja se izvodi.
`TimeFormat`	Format za vreme koji će se koristiti za tekući uređaj. Format za vreme se može definisati u milisekundama, okvirima (frames), bajtovima, uzorci ma, trake/minute/sekunde, sati/minute/sekunde, itd.
`Tracks`	Broj traka koje se nalaze na mediju (za CD audio uređaje).
`VisibleButtons`	Određuje koja će dugmad biti prikazana na kontrolnoj traci komponente `MediaPlayer`. Generički:All Buttons (sva dugmad).
`Wait`	Definiše da li će kontrola biti vraćena aplikaciji trenutno, ili nakon što se završi sa izvršavanjem medija datoteke.

Komponenta `MediaPlayer` sadrži i mnoštvo metoda. Većina ovih metoda izvršavaju istu funkciju kao i dugmad kontrolne trake. Tabela 12.6. prikazuje spisak primarnih metoda klase `TMediaPlayer`.

Tabela 12.6: Primarne metode klase `TMediaPlayer`

Metod	Opis
`Back`	Vraća mediju za broj koraka koji je definisan karakteristikom `Frames`.
`Close`	Zatvara uređaj.
`Eject`	Izbacuje mediju, ukoliko je to moguće (na primer, izbacuje audio CD).
`Next`	Prelazi na početak sledeće trake za uređaje koji podržavaju trake.
`Open`	Otvara uređaj. (Koristi se kada je karakteristika `AutoOpen` podešena na `False`.)
`Pause`	Pauza za izvršavanje, ili snimanje medije.
`Play`	Pokreće izvršavanje medije na uređaju.
`Previous`	Pomera se na početak prethodne trake.
`Resume`	Nastavlja aktivnost (snimanje, ili izvršavanje) prekinutu metodom `Pause`.
`Rewind`	Vraća poziciju na mediji na početak medije.
`Save`	Snima mediju u datoteku čiji je naziv definisan karakteristikom `FileName`.
`StartRecord`	Počinje snimanje podataka.
`Step`	Pomera izvršavanje medije unapred za broj koraka koji je definisan karakteristikom `Frames`.
`Stop`	Zaustavlja trenutnu aktivnost (izvršavanje, ili snimanje).

Kompnenta `MediaPlayer` poseduje jedan veoma važan događaj. Događaj `OnNotify` se poziva svaki put kada se kompletira komanda, ali samo u slučaju kada je karakteristika Notify podešena na `True`. Možete istražiti karakteristike `Error` i `NotifyValue`, kako biste videli da li je operacija uspešno izvedena.

Wave audio

Izvršavanje wave audio datoteke je jedna od najosnovnijih multimedija operacija. Ona je verovatno i najčešća. Sinhrono izvršavanje zvučne datoteke bi trebalo da izgleda ovako:

```
Player.Wait := True;
Player.FileName := 'test.wav';
Player.Open;
Player.Play;
```

Uočite da je karakteristika `Wave` podešena na `True`, kako bi se obezbedilo da se wave audio datoteka izvršava sinhrono. Ovo je neophodno ukoliko želite da izvršavate zvučne datoteke jednu iza druge. Na primer:

```
Player.FileName := 'Sound1.wav';
Player.Open;
Player.Wait := True;
Player.Play;
Player.FileName := 'Sound2.wav';
Player.Wait := True;
Player.Play;
```

Uočite da sam pre izvršavanja svake datoteke postavio karakteristiku `Wait` na vrednost `True`. Karakteristika `Wait` se resetuje nakon operacije, pa je morate podesiti svaki put, ukoliko želite da se izvršavanje programa zaustavi sve dok se tekuća operacija ne završi.

Ukoliko ne podesite karakteristiku `Wait` na vrednost `True`, prvi zvuk će početi da se izvršava, a nekoliko milisekundi kasnije će početi da se izvršava i drugi zvuk. Ukoliko želite da se zvuk izvršava u pozadini, postavite vrednost karakteristike `Wait` na `False`.

Da biste izveli deo zvučne datoteke, možete podesiti karakteristike `StartPos` (početna pozicija) i `EndPos` (krajnja pozicija) pre izvršavanja datoteke. Naredni primer otvara zvučnu datoteku i izvršava dve sekunde audio zapisa počev od prve, završano sa trećom sekundom zapisa:

```
Player.FileName := 'test.wav';
Player.Open;
Player.StartPos := 1000;
Player.EndPos   := 3000;
Player.Play;
```

Vrednosti karakteristika `StartPos` i `EndPos` su definisane u milisekundama, što je generička vrednost za wave audio uređaje.

Ukoliko podesite karakteristike `StartPos` i `EndPos` tako da bilo koja od njih ima pogrešnu vrednost, wave datoteka se neće izvršiti. Pogrešne vrednosti uključuju slučajeve kada je vrednost `StartPos` veća od vrednosti karakteristike `EndPos`, odnosno kada je vrednost karakteristike `EndPos` veća od dužine medija datoteke.

Podešavanje izlazne jačine

Podešavnje izlazne jačine wave izlaznog uređaja je relativno jednostavno, ali Vam je za to potrebna pomoć Windows API-ja. Funkcije `waveOutGetVolume` i `waveOutSetVolume` se mogu koristiti za preuzimanje vrednosti jačine, odnosno podešavanje vrednosti jačine.

Vrednost jačine se definiše kao celobrojna vrednost. Viša reč definiše vrednost jačine desnog kanala, dok niža reč definiše vrednost jačine levog kanala. Ukoliko uređaj nema mogućnost nezavisnog podešavanja levog i desnog kanala, niža reč će se koristiti za podešavanja jačine, dok će viša reč biti ignorisana.

Vrednost `0` znači da nema zvuka, dok heksadecimalna vrednost `$FFFF` označava punu jačinu. Na osnovu toga naredni kod podešava jačinu oba kanala na 50%:

```
waveOutSetVolume(0, $80008000);
```

Naredni primer podešava jačinu na maksimalnu vrednost:

```
waveOutSetVolume(0, $FFFFFFFF);
```

Uočite da sam za vrednost prvog parametra funkcije `waveOutSetVolume` koristio vrednost `0`. Ovo je delom mala prevara, pošto podrazumevam da će wave uređaj biti uređaj broj 0.U većini slučajeva ovo je tačno, pa ćete uglavnom moći da se provučete, ukoliko koristite ovaj trik.

Podešavanje vrednosti jačine je jednostavno, baš kao što ste mogli da vidite u primerima. Uočite da funkcija `waveOutSetVolume` podešava samo vrednost jačine izlaznog wave uređaja, a ne jačinu glavnog izlaza. Jačina glavnog izlaza se može podesiti samo preko kontrola multimedija miksera, što ne spada u domen ove knjige. Shvatam da je diskusija o podešavanju pomalo u domenu naprednog programiranja. Ukoliko je niste u potpunosti shvatili, nemojte se nervirati. Uvek se možete vratiti na ovo poglavlje ukoliko vam zatreba.

Snimanje wave audio datoteke

Snimanje wave audio datoteke nije tako jednostavno, iako bi to trebalo da bude. Možda ćete pomisliti da je potrebno samo pozvati metodu `StartRecording`. Ovo nije baš tako jednostavno, zato što postoje izvesni problemi vezani za klasu `TMediaPlayer`.

Da biste snimili wave datoteku, potrebno je da otvorite već postojeću wave datoteku koja ima iste parametre za snimanje koje želite da poseduje i vaša nova datoteka. Zatim možete snimiti nove podatke u datoteku, promeniti karakteristiku `FileName` u novi naziv datoteke, a zatim snimiti datoteku. Ovo je malo komplikovano, ali radi.

Na primer, pretpostavimo da imate datoteku pod nazivom `DUMMY.WAV` koja je snimljena u wave formatu sa 22050 kHz, sa 8 bita po uzorku i jednim kanalom (lako možete kreirati ovaj tip datoteke korišćenjem programa Windows Sound Recorder). U ovom slučaju možete početi sa snimanjem wave audio datoteke klikom miša na dugme:

```
procedure TForm1.StartBtnClick(Sender: TObject);
begin
  with MediaPlayer do begin
    { Set FileName to the dummy.wav file to }
    { get the recording parameters. }
    FileName := 'dummy.wav';
    { Open the device. }
    Open;
    { Start recording. }
```

```
    Wait := False;                                          nastavak
    StartRecording;
  end;
end;
```

Od ovog trenutka snimanje je počelo i kontrola je vraćena Vašoj aplikaciji. Sada je potrebno da zaustavite snimanje, što možete uraditi klikom miša na drugo dugme. Na primer:

```
procedure TForm1.StopBtnClick(Sender: TObject);
begin
  with MediaPlayer do begin
    { Stop recording. }
    Stop;
    { Change the filename to the new file we want to write. }
    FileName := 'new.wav';
    { Save and close the file. }
    Save;
    Close;
  end;
end;
```

NAPOMENA U vreme pisanja ove knjige, da bi se snimila wave audio datoteka, bilo je potrebno izvršiti ove korake kako bi se izbegla greška u klasi `TMediaPlayer`. Verovatno će nakon izlaska Delphi-ja 4 ova greška biti otklonjena. Da biste ovo otkrili, postavite komponentu `MediaPlayer` na formu, a zatim pozovire metodu `Record`. Ukoliko Delphi ne izbaci izuzetak, greška je popravljena.

Faktori koji kontrolišu način na koji je wave audio datoteka snimljena, uključuju frekvenciju uzorka (sample rate), broj kanala (mono, ili stereo), i broj bitova po uzorku (obično 8, ili 16). Uobičajena frekvencija uzorka (sampling rate) je 8000 kHz, 11025 kHz, 22050 kHz i 44100 kHz. Što je veća frekvencija uzorka, veći je i kvalitet snimljene audio datoteke. U većini slučajeva, verovatno nećete koristiti stereo snimanje, ukoliko ne budete pisali igre. Čak i u tom slučaju, stereo zvuk treba da koristite samo kada je to potrebno. Broj bitova po uzorku (bits per sample) takođe utiče na kvalitet snimljenog zvuka. Broj bitova po uzorku može biti postavljen na 8, ili 16 bita.

NAPOMENA Što je veći kvalitet zvuka, snimljena datoteka će zauzeti više prostora na disku. Wave datoteka snimljena u stereo tehnici će, naravno, biti dva puta duža od iste datoteke snimljene korišćenjem samo jednog kanala. Slično je i sa korišćenjem 16 bita po uzorku, čime ćete duplirati veličinu wave datoteke u odnosu na datoteku koja je snimljena sa 8 bita po uzorku. Datoteka snimljena na 22050 kHz u mono tehnici i sa 8 bita po uzorku može imati dužinu 200 K, dok ekvivalent ove datoteke snimljen na 22050 kHz u stereo tehnici i 16 bita po uzorku može imati dužinu 800 K. U većini slučajeva, stereo tehnika i 16 bita po uzorku ne pružaju dovoljan kvalitet da bi se opravdao veći zahtev za prostorom na disku. Wave format od 22050 kHz, mono i 8 bita po uzorku pruža dobar kompromis između kvaliteta i veličine datoteke.

MIDI audio datoteke

Nije potrebno mnogo objašnjavati MIDI audio datoteke. Potrebno je samo da podesite karakteristiku `FileName` komponente `MediaPlayer` na MIDI datoteku i pozovete metodu `Play`. `MIDI` datoteke imaju nastavak, ili `.mid`, ili `.rmi`.

Ukoliko nameravate da obavljate poslove koji se ne odnose samo na izvršavanje MIDI datoteka, potrebno je da proučite `midiInXXX` i `midiOutXXX` funkcije niskog nivoa. Ove funkcije su dokumentovane u datoteci `MEDIA.HLP`. Ova datoteka se najverovatnije nalazi u Vašem Delphi Help direktorijumu (najverovatnije ima naziv `MM.HLP`). Ukoliko navedena datoteka ne postoji, verovatno ćete je moći nabaviti od firme Microsoft.

Većina zvučnih kartica nije u mogućnosti da izvršava dve zvučne datoteke istovremeno. Slično je i sa MIDI datotekama; nije moguće istovremeno izvršiti više od jedne MIDI datoteke. Ipak, većina zvučnih kartica omogućava istovremeno izvršavanje zvučne datoteke i MIDI datoteke. Da biste istovremeno izvršili i zvučnu i MIDI datoteku, treba da koristite dve `MediaPlayer` komponente. Alternativno, možete koristiti komponentu `MediaPlayer` za MIDI datoteku, a funkciju `PlaySound` za izvršavanje bilo koje zvučne datoteke.

MIDI datoteka se obično koristi kao muzika u pozadini kod igara. Ukoliko koristite MIDI datoteke na ovaj način, potrebno je ponovo pokrenuti muziku kada se stigne do kraja datoteke. Klasa `TMediaPlayer` nema ugrađenu mogućnost kontinualnog izvođenja zvuka. Ipak, možete imati koristi od događaja `OnNotify` komponente `MediaPlayer`, kako biste postigli efekat petlje. Prvo je potrebno da saopštite komponenti `MediaPlayer` da Vas obavesti ukoliko se nešto dogodi. Ovaj deo je jednostavan:

```
MediaPlayer.Notify := True;
```

Nakon toga, potrebno je da obezbedite upravljač događajem `OnNotify`. U okviru upravljača događajem potrebno je da obezbedite deo koda koji će ponovo pokrenuti MIDI datoteku nakon uspešnog završetka. Događaj `OnNotify` bi mogao izgledati otprilike ovako:

```
procedure TForm1.MediaPlayerNotify(Sender: TObject);
begin
  with MediaPlayer do
    if NotifyValue = nvSuccessful then begin
      Position := 0;
      Play;
    end;
end;
```

Prvo sam proverio da li karakteristika `NotifyValue` sadrži vrednost `nvSuccessful`. Ukoliko je to tačno, resetovaću poziciju datoteke na `0`, a zatim pozvati metodu Play, kako bih ponovo pokrenuo izvršavanje datoteke. Ovo je veoma jednostavno, ali postoji nekoliko stvari na koje treba da obratite pažnju.

Prvo, uočite da sam podesio karakteristiku `Position` na `0`. Na ovaj način se postiže efekat "premotavanja" datoteke na početak. Ukoliko je karakteristika `AutoRewind` podešena na `True`, ovaj korak nije potreban. Drugi element na koji treba da obratite pažnju su neke aktivnosti komponente `MediaPlayer` koje mogu da pozovu događaj `OnNotify` preko karakteristike `NotifyValue` koja ima vrednost `nvSuccessful`.

Na primer, jednostavna komanda `Stop` će kao rezultat dati vrednost `nvSuccessful`, kompletirajući izvršenje komande bez problema. Možda ćete podesiti stanje mašine tako da budete obavešteni samo kad događaj `OnNotify` biva pozvan kao odgovor na dostizanje kraja datoteke, a da ne reaguje na rezultat neke druge operacije komponente MediaPlayer.

CD audio datoteke

Izvođenje CD audio datoteka (muzičkih kompakt diskova) je relativno jednostavno kada koristite klasu `TMediaPlayer`. Da biste pustili muzički CD, potrebno je samo da promenite karakteristiku `DeviceType` na `CDAudio`, a zatim kliknete na dugme `Play` (odnosno pozovete metodu `Play`).

Najteži aspekt za shvatanje programiranja CD audio uređaja su različiti formati koji se koriste kod muzičkih kompakt diskova. Možete koristiti format za vreme TMSF (vreme, minute, sekunde, okviri - time, minutes, seconds, frames), kako bi prikupili informacije o željenoj traci, koja se nalazi na tekućoj poziciji određene trake. Bilo koja vrednost minute, sekunde, ili okvira će biti relativno postavljena u odnosu na broj trake. Na primer, naredni kod formatira string da bi izvestio o poziciji na kojoj se trenutno nalazi pokazivač u okviru tekuće trake:

```
var
  Time    : Integer;
  Track   : Integer;
  Minutes : Integer;
  Seconds : Integer;
  TimeStr : string;
begin
  MediaPlayer.TimeFormat := tfTMSF;
  Time    := MediaPlayer.Position;
  Track   := mci_TMSF_Track(Time);
  Minutes := mci_TMSF_Minute(Time);
  Seconds := mci_TMSF_Second(Time);
  TimeStr := Format('Track Time: %2.2d:%2.2d', [Minutes, Seconds]);
  Label1.Caption := 'Track: ' + IntToStr(Track);
  Label2.Caption := TimeStr;
end;
```

Prvo, karakteristika `TimeFormat` je podešena na `tfTMSF`, kako bi se obezbedio odgovarajući format vremena. Sledeće, trenutna pozicija je upisana u promenljivu pod nazivom `Time`. Na osnovu toga, različite vrednosti vremena (trake, minuti i

sekunde) se dobijaju korišćenjem Windows makroa za konverziju vremena, `mci_TMSF_Track`, `mci_TMSF_Minute` i `mci_TMSF_Second`. Ovi makroi se nalaze u junitu `MMSystem`. Ukoliko želite da koristite ove makroe, treba da dodate ovaj junit u Vašu `uses` listu. Nakon što izvučete zasebne vrednosti vremena, kreirajte string koji sadrži vreme, kako biste prikazali dužinu trajanja trake. Tada će naziv i broj trake biti prikazan u komponenti `Label`.

Da biste prikupili kompletnu informaciju o kompakt disku, možete koristiti vremenski format MSF (minute, sekunde, okviri - minutes, seconds, frames). Na primer, vremenski format MSF ćete moći da koristite kako bi očitali trenutnu poziciju pokazivača u tekućem kompakt disku u odnosu na početak kompakt diska. Slično, možete koristiti format za vreme MSF, kako biste podesili trenutnu poziciju na 30 minuta od početka kompakt diska, bez obzira na traku. Naredni kod pokazuje kako da očitate trenutnu poziciju u minutama i sekundama na kompakt disku:`var`

```
  Time    : Integer;
  Minutes : Integer;
  Seconds : Integer;
  TimeStr : string;
begin
  MediaPlayer.TimeFormat := tfMSF;
  Time    := MediaPlayer.Position;
  Minutes := mci_MSF_Minute(Time);
  Seconds := mci_MSF_Second(Time);
  TimeStr := Format('Total Time: %2.2d:%2.2d', [Minutes, Seconds]);
  Label3.Caption := TimeStr;
end;
```

Izvorni kod knjige sadrži program pod nazivom `CDPlayer` koji ilustruje kako možete koristiti klasu `TMediaPlayer` za kreiranje plejera muzičkih kompakt diskova.

AVI video datoteke

Da biste izvodili AVI video datoteke koristeći klasu `TMediaPlayer`, odaberite AVI datoteku i pozovite metodu `Play` (odnosno kliknite mišem na dugme `Play`). Ukoliko koristite generičke komande komponente `MediaPlayer`, pojaviće se zaseban prozor u kom će se izvršavati AVI datoteka. Kao alternativu možete podesiti karakteristiku `Display` na bilo koju prozorsku komponentu, kako bi se video datoteka izvršavala u okviru klijent oblasti komponente.

Na primer, pretpostavimo da imate pano u okviru forme pod nazivom `AVIPanel` i da želite da izvodite AVI video datoteku na tom panou. U tom slučaju ćete podesiti karakteristiku `Display` na sledeći način:

```
MediaPlayer.Display := AVIPanel;
```

Kada bude pokrenuta AVI datoteka, njeno izvršavanje će se obavljati na panou. Ukoliko je video slika veća od pravougaonika panoa, slika će biti odsečena na veličinu panoa.

Podešavanjem karakteristike `DisplayRect` možete raširiti, ili skupiti video sliku. Naredni kod će raširiti, odnosno skupiti video sliku na veličinu panoa za prikazivanje:

```
MediaPlayer.DisplayRect := AVIPanel.ClientRect;
```

Postoje mnogi tipovi AVI video formata. Takođe, svi video formati neće raditi na svakom sistemu. Da bi izvršili određenu video datoteku, potrebno je da se uverite da je korisnik instalirao drajvere (veznike) za određeni tip video datoteke. Ukoliko želite da igrate na sigurno, držite se Microsoft AVI video tipova datoteka. Vaši korisnici će gotovo sigurno imati instalirane drajvere za Microsoft AVI format, zato što se ovaj tip drajvera nalazi u osnovnoj Windows instalaciji.

Komponenta TAnimate (nalazi se u kartici Win32 u okviru palete komponenti) se koristi za izvođenje malih video datoteka, sličnih datotekama koje koristi Windows operativni sistem. Primere ovih animacija možete videti kada program Explorer kopira datoteke, odnosno kada Explorer prikazuje dijalog u toku traženja datoteka. AVI datoteke koje mogu mogu biti izvođene klasom `TAnimate` ne smeju biti kompresovane, odnosno dozvoljeno je samo da budu kodirane sistemom RLE (Run Length Encoded - kodiranje na dužinu izvršavanja). Nikakva druga forma kompresije nije dozvoljena. Takođe, AVI datoteke ne mogu sadržati audio zapis.

Komponenta `MediaPlayer` može izvršavati više tipova video datoteka i animacija, ukoliko su instalirani odgovarajući drajveri. Na primer, Autodesk Animator (AA) animacije mogu imati nastavke `.fli` ili `.flc`. Da biste izvršavali AA animacije, potrebno je da podesite karakteristiku `DeviceType` na `dtAutoSelect`, a zatim da odaberete datoteku programa Animator. Sve dok budu instalirani AA drajveri, komponenta `MediaPlayer` će izvršavati i ovaj tip animacije.

Zaključak

Programiranje grafike može biti veoma interesantno i veoma isplativo, mada može doneti frustracije. VCL će otkloniti uzroke mnogih frustracija u toku programiranja grafike, pošto obezbeđuje klase `TCanvas` i `TBitmap` zajedno sa klasama za podršku `TPen`, `TBrush` i `TFont`. Ove klase Vam omogućavaju da obavite posao vizuelnog dela programiranja grafike, umesto da se brinete kako ćete da učitate, odnosno snimite bitmapiranu datoteku. Programiranje multimedije može biti veoma zabavno. Isplativo je što nakon pisanja par linija koda možete videti i čuti rezultate rada. Multimedija sigurno dodaje uzbudljivost Vašim programima, ali pazite da ne preterate.

Iako ne mogu da tvrdima da je današnja lekcija detaljan prikaz programiranja grafike i multimedije u Delphi-ju, ona predstavlja dobar početak i predstavlja Vam koncepte koje ćete moći da koristite veoma dugo.

Radionica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstite razumevanje obrađenog materijala, kao i vežbe koje Vam omogućavaju da steknete iskustvo u korišćenju gradiva koje ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A, "Odgovori na kviz pitanja".

Pitanja i odgovori

P **Da li koncepti vezani za grafiku koji su obrađeni u ovom poglavlju mogu biti korišćeni za štampanje, kao što se koriste za crtanje po ekranu?**

O Da. Što se tiče Windows operativnog sitema, kontekst uređaj je kontekst uređaj. Nije bitno da li se kontekst uređaj koristi za prikazivanje na ekranu, upisivanje u memorijsku bitmapu, ili štampanje na štampaču.

P **Upoznao sam metodu `Ellipse`, ali nisam mogao da pronađem metodu za crtanje krugova. Kako mogu da nacrtam krugove?**

O Koristite metodu `Ellipse`. Uverite se da je pravougaonik koji se koristi za crtanje elipse savršeni kvadrat, pa ćete dobiti savršen krug (ukoliko Vam ovo nešto znači).

P **Kako da promenim boju teksta koja se dobija funkcijom `DrawText`?**

O Promenite karakteristiku `Color` fonta na kanvasu.

P **Zašto bih trebao da se mučim sa memorijskim bitmapama?**

O Možda i ne morate. Ipak, ukoliko ste zapazili treperenje u vašim rutinama za crtanje, trebali bi da razmislite o korišćenju memorijskih bitmapa.

P **Koji je generički format za wave audio uređaje?**

O Generički, wave audio uređaji koriste milisekunde (jedna sekunda je 1000 milisekundi).

P **Da li mogu da pokrenem više wave datoteka istovremeno?**

O Ne, ukoliko koristite komponentu `MediaPlayer`. Microsoft-ov DirectX API poseduje mogućnosti za miksovanje, ukoliko se istovremeno izvršavate dva, ili više zvukova. Ukoliko Vam je potrebno miksovanje zvuka, preuzmite DirectX od firme Microsoft (program je besplatan).

P **Kako mogu da izvršavam AVI video datoteku direktno na svojoj formi?**

O Podesite karakteristiku `Display` klase `TMediaPlayer` na naziv forme.

Kviz

1. Koje komponente možete koristiti da biste crtali grafiku na formi?
2. Koja karakteristika `klaseTCanvas` kontroliše boju koja popunjava kanvas?
3. Za koje operacije služi region za isecanje?
4. Koje funkcije treba da koristite da biste nacrtali više linija teksta na kanvasu?
5. Koje metode klase `TCanvas` se mogu koristiti za crtanje bitmape sa transparentnom pozadinom?
6. Koja metoda klase `TCanvas` se koristi za kopiranje kompletne bitmape na kanvas?
7. Kako možete snimiti memorijsku bitmapu u datoteku?
8. Koju komponentu koristite da biste izvodili wave datoteke?
9. Za šta se koristi karakteristika `TimeFormat` klase `TMediaPlayer`?
10. Da li možete da snimite wave audio datoteku koristeći komponentu `MediaPlayer`?

Vežbe

1. Napišite program koji prikazuje krug na ekranu nakon klika mišem na dugme.
2. Napišite program koji crta slučajne linije na ekranu, svaki put kada se prikaže forma (čak i kada se forma pojavi nakon što je bila sakrivena).
3. Napišite program koji kreira memorijsku bitmapu, crta tekst i oblike na bitmapi, a zatim prikazuje bitmapu na kanvasu forme.
4. Izmenite program koji ste napisali u vežbi 3, kako bi snimili memorijsku bitmapu na disk.
5. Napišite program koji prikazuje AVI video datoteke na glavnoj formi aplikacije.
6. Napišite program koji izvodi wave datoteke prilikom startovanja programa.

Dan 13

Iza osnova Delphi-ja

U današnjoj lekciji ćete naučiti kako da od dobre, napravite odličnu Windows aplikaciju. Tačnije obradićemo sledeće:

- Ukrašavanje prozora: trake sa alatima i statusne trake.
- Aktiviranje komandi.
- Štampanje u okviru Delphi-jevih aplikacija.
- Korišćenje kursora.

Diskusija će biti nastavljena i u sledećoj lekciji kada ćete videti kako se mogu implementirati mogućnosti naprednog Windows programiranja u okviru Delphi aplikacija.

Kreiranje ukrasa za prozore

Ne, neću pisati o lepim sijalicama na prednjem prozoru Vaše kuće. Biće obrađene opcije kao što su trake sa alatima i statusne trake. Ove opcije se obično nazivaju ukrasi za prozore. U ovom poglavlju će biti obrađeni navedeni tipovi dekoracija, kao i način za njihovu implementaciju u Vašim aplikacijama.

Trake sa alatima (toolbars)

Trake sa alatima (takođe se zovu i kontrolne trake i trake sa prečicama) su skoro standardna oprema današnjih Windows programa. Korisnici očekuju određene

elemente, a traka sa alatima je jedan od njih. Traka sa alatima vrhunskog kvaliteta sadrži određene mogućnosti i osobine:

- Dugmad koja predstavljaju zadatke su isto tako dostupna i u meniju aplikacije.
- Aktiviranje i deaktiviranje dugmadi po želji.
- Oblačići za savete opisuju funkciju dugmadi.
- Dodatni tekst za savet koji je prikazan u statusnoj traci aplikacije.
- Mogućnost usidravanja prozora.
- Ostale kontrole, kao što su kombo okviri i dugmad za padajuće menije.

Neke od mogućnosti na ovoj listi trake sa alatima ne moraju da ih poseduju, pa se mogu smatrati neobaveznim. Koriteći Delphi, lako možete implementirati navedene mogućnosti. Kasnije u ovoj lekciji, u poglavlju "Dodavanje funkcionalnosti aktiviranjem komandi", ćemo obraditi aktiviranje komandi. Do tada neće biti obrađeno aktiviranje komandi za dugmad trake sa alatima.

Postavljanje na traku za alate samo onih dugmadi koja imaju odgovarajuće opcije menija se pokazalo kao dobra praksa. Traka sa alatima je zamena za menije i ne bi trebala da sadrži opcije koje se ne mogu pronaći u menijima programa.

U lekciji dana 8, "Kreiranje aplikacija u Delphi-ju", objasnio sam da je najjednostavniji način za kreiranje trake sa alatima korišćenje čarobnjaka za aplikacije (Application Wizard). Čak, iako imate već delimično napisanu aplikaciju, još uvek možete koristiti čarobnjaka za aplikaciju da biste kreirali traku sa alatima. Kreirajte aplikaciju koristeći čarobnjaka za aplikacije, a zatim kopirajte pano koji sadrži traku za alate na Clipboard., ponovo otvorite Vašu aplikaciju (nemojte se truditi da snimite aplikaciju koju je kreirao čarobnjak), a zatim zalepite traku sa alatima sa Clipboard-a na Vašu aplikaciju. Kratko i efikasno.

Ipak, čarobnjak za aplikacije Vam neće pružiti sve što Vam je potrebno za traku sa alatima. Čarobnjak za aplikacije će, najverovatnije, koristiti stari metod za kreiranje trake sa alatima - sa panoom i dugmadima prečicama. Bolji način za kreiranje trake sa alatima je korišćenje komponenti `ToolBar` i `CoolBar` (mogu se pronaći u kartici Win32 palete komponenti). Ove komponente ćemo obraditi u narednim poglavljima.

Komponente `CoolBar` i `ToolBar` zahtevaju da broj verzije datoteke `COMCLT32.DLL` bude `4.722106.4`, odnosno noviji. Ova datoteka treba da bude instalirana kao deo instalacije Delphi-ja. Ukoliko nemate najnoviju verziju ove dll datoteke, možete je pronaći na Web sajtu firme Microsoft (`http://www.microsoft.com`). Kada isporučujete vašu aplikaciju, treba da instalirate `COMCTL32.DLL` verziju koja nije starija od 4.70. Budite sigurni da koristite dobar instalacioni program, kako ne bi obrisali noviju verziju datoteke u toku instaliranja Vaše aplikacije.

Komponenta CoolBar

Komponenta CoolBar predstavlja enkapsulaciju Win32 elementa pod nazivom *cool bar* - traka kontejner (ponekad se naziva i rebar). Ova komponenta je specijalizovana kontrola kontejner. U većini slučajeva komponenta CoolBar se koristi kao kontejner za trake sa alatima, ali nije ograničena izričito na trake sa alatima.

Trake u okviru trake kontejnera (cool bar bands)

Traka kontejner (cool bar) sadrži trake koje se mogu pomerati, a isto tako im se može menjati veličina u toku rada programa. Traka na levoj strani sadrži hvataljku za promenu veličine, koja korisniku obezbeđuje vizuelnu oznaku da se traka može premestiti, odnosno da joj se može promeniti veličina. Trake u okviru trake kontejnera, su predstavljene klasom TCoolBand. Traka u okviru trake kontejnera može sadržati samo jednu komponentu. Obično je ta komponenta traka sa alatima, ali može biti i kombo okvir, odnosno bilo koja druga komponenta. Uradimo vežbu kako bi bolje shvatili način rada komponente CoolBar:

1. Pokrenite novu aplikaciju i postavite komponentu CoolBar na formu.
2. Postavite komponentu ComboBox na traku kontejner. Delphi kreira novu traku koja će sadržati kombo okvir. Uočite da kombo okvir ispunjava celu širinu trake kontejnera.
3. Postavite još jednu komponentu ComboBox na traku kontejner. Biće kreirana još jedna traka koja će se nalaziti ispod prve trake.
4. Postavite kursor miša između hvataljke i kombo okvira na drugoj traci. Izgled kursora će se promeniti na ruku koja pokazuje, označavajući da možete da pomerite traku. (Isto tako možete koristiti hvataljku za promenu veličine, kako bi prevukli traku.) Prevucite donju traku na traku koja se nalazi iznad nje. Nakon ovog koraka prva traka će se skupiti kako bi napravila mesta za traku koju ste prevukli. Postavite traku blizu sredine trake kontejnera. Sada možete koristiti hvataljke za promenu veličine, kako bi ste promenili veličinu bilo koje trake.
5. Postavite komponentu Panel na traku kontejner. Biće kreirana nova traka koja će sadržati pano.
6. Odaberite traku kontejner i promenite karakteristiku AutoSize ove komponente u True.
7. Postavite Memo komponentu na formu ispod komponente CoolBar. Podesite karakteristiku Align memo komponente na vrednost alClient.

Sada Vaša forma izgleda slično kao i forma na slici 13.1.

Slika 13.1
Forma sa trakom kontejnerom u okviru koje se nalaze tri trake

Sada pokrenite program. Eksperimentišite sa trakama u okviru trake kontejnera. Prevlačite ih gore, ili dole, odnosno menjajte im veličinu. Uočite da se, kako prevlačite trake gore, ili dole, traka kontejner, povećeva, onosno smanjuje prema potrebi, a da memo polje uvek popunjava ostatak klijent oblasti.

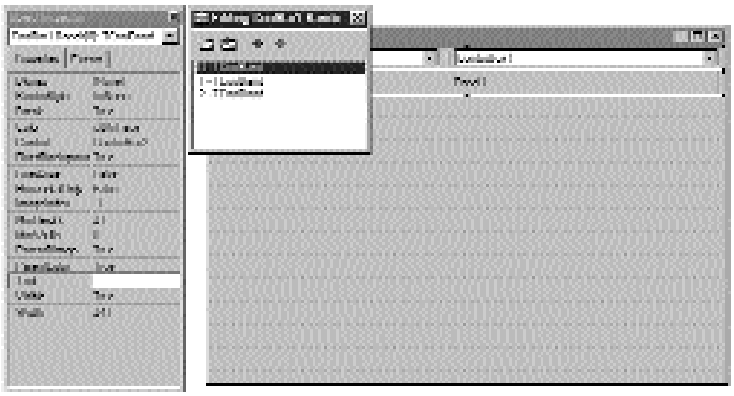
Trakama u okviru trake kontejnera možete pristupiti koristeći karakteristiku `Bands`. Ova karakteristika je predstavljena klasom `TCoolBands`, koja sadrži više komponenti tipa `TCoolBand`. Ukoliko želite da sakrijete drugu traku, možete uraditi sledeće :

```
CoolBar.Bands[1].Visible :=False;
```

Trake možete dodavati na dva načina. Kao što ste već mogli da vidite, traku možete kreirati postavljanjem komponente na traku kontejner, mada isto možete postići i korišćenjem prozora Bands Editor (editor traka). Da biste pozvali prozor Bands Editor, potrebno je da dva puta kliknete mišem na traku kontejner, odnosno na dugme sa tri tačke koje se nalazi pored karakteristike Bands u okviru prozora Object Inspector. Trake dodajete klikom miša na dugme Add. Trake brišete klikom miša na dugme Delete. Dugmad Move Up i Move Down Vam omogućavaju da promenite redosled traka.

Ukoliko je karakteristika `AutoSize` podešena na `True`, traku kontejner možete privremeno isključiti, ukoliko želite da dodate nove trake postavljanjem komponenti na traku kontejner. Da bi traka kontejner postala veća, podesite karakteristiku `AutoSize` na `False`, postavite komponentu na traku kontejner, a zatim ponovo podesite karakteristiku `AutoSize` na `True`.

Kada odabirate trake u prozoru Bands Editor, prozor Object Inspector prikazuje karakteristike trake. Slika 13.2. prikazuje prozor Bands Editor i prozor Object Inspector u trenutku kada je odabrana traka.



Slika 13.2
Prozor Bands Editor trake kontejnera

Karakteristika `Bitmap` Vam omogućava da postavite bitmapu kao pozadinu trake. Da bi ste odabrali sliku koja će se pojaviti na levoj strani trake možete koristiti karakteristiku `ImageIndex`. Karakterisrtika `ImageIndex` zahteva da karakteristika `ImageList` trake kontejnera bude podešena na odgovarajuću klasu `TImageList`. Koristeći karakteristike `MinHeight` i `MinWidth` možete podesiti minimalnu visinu i širinu trake. Da bi ste fiksirali traku (učinili da traka ne može da se pomera), podesite karakteristiku `FixedSize` na `True`.

Ostale karakteristike komponente `CoolBar`

Traka kontejner može biti vertikalna, ili horizontalna. Generički, karakteristika `Align` je podešena na `alTop`. Da bi ste napravili vertikalnu traku kontejner, promenite karakteristiku `Align` u alRight, odnosno `alLeft`. Neke komponente se orjentišu vertikalno, ili horizontalno, kada se postave na traku kontejner, što zavisi od položaja trake kontejnera. Postavljanje karakteristike `Vertical` predstavlja još jedan način za promenu orjentacije trake kontejnera.

Karakteristika `Bitmap` Vam omogućava da podesite bitmapu kao pozadinu trake kontejnera. Bitmapa koju ste odabrali će biti raspoređena tako da popuni pozadinu trake kontejnera. Uočite da se na ovaj način podešava bitmapa kao pozadina trake kontejnera, a ne kao pozadina pojedinih traka koje se nalaze u okviru trake kontejnera (što je bilo objašnjeno u prethodnom poglavlju). Karakteristiku `ImageList` koristite da bi podesili listu slika koje će trake koristiti da bi prikazale sliku na levoj strani trake koja je definisana u karakteristici `ImageIndex`.

Karakteristika `AutoSize` određuje da li će traka kontejner automatski promeniti veličinu nakon pomeranja traka koje se nalaze u okviru trake kontejnera. U prethodnom primeru ste mogli da vidite efekte za različita podešavanja karakteristike `AutoSize`.

4 Proverite komponentu `TControlBar` koja se nalazi u kartici Additional u okviru palete komponenti. Klasa `TControlBar` (odnosno komponenta ControlBar) je originalna VCL komponenta koja radi slično kao traka kontejner. Ova komponenta ne zavisi od datoteke `COMCTL32.DLL`, kao što je to slučaj sa komponentom `CoolBar`, pa je manje podložna hirovima Microsoft-a.

Komponenta `ToolBar`

Komponenta `ToolBar` enkapsulira Win32 kontrolu za traku sa alatima. Traka sa alatima će automatski urediti elemente i definisati veličinu elemenata koji se nalaze na traci sa alatima, tako da svi elementi trake imaju jednaku visinu. Komponentu `ToolBar` možete koristiti, a da ne koristite traku kontejner. Ukoliko imate samo jednu traku sa alatima, možete je koristiti bez trake kontejnera. Ukoliko imate više traka sa alatima i želite da omogućite korisniku da pomera, postavlja dve, ili više traka sa alatima, treba da koristite traku kontejner.

Kreiranje trake sa alatima i dodavanje dugmadi je veoma jednostavno. Ukoliko Vaša dugmad na traci sa alatima treba da imaju slike (a većina ih ima), morate koristiti komponentu `ImageList`, kako biste odabrali sliku sa liste. Da bi ilustrovali način postavljanja trake sa alatima korišćenjem komponente `ToolBar`, vratimo se na program `ScratchPad`. Rastavićete traku sa alatima na delove i sklopiti ponovo.

Uklanjanje stare trake sa alatima

Ukoliko možete da se setite trake sa alatima koju ste u početku kreirali u okviru programa `ScratchPad`, postojala je samo jedna komponenta koja je čuvala mesto traci sa alatima. Prvo što treba da uradite je da se otarasite stare trake sa alatima izvršavajući sledeće korake:

1. Kliknite na `Memo` komponentu i promenite karakteristiku `Align` ove komponente u `alNone`. Povucite na dole vrh `Memo` komponente, kako bi ostavili mesta za novu traku sa alatima.
2. Kliknite na komponentu trake sa alatima, a zatim je obrišite.

Dodavanje nove trake sa alatima

Sada ponovo možete dodavati komponente. Prvo treba da dodate sledeće koponente: traku kontejner (CoolBar) i traku sa alatima. U ovom trenutku Vam traka kontejner ne treba, pošto imate samo jednu traku sa alatima, ali pošto postoji mogućnost da kasnije dodate još traka sa alatima, najbolje je da planirate unapred. Izvršite sledeće korake:

1. Postavite komponentu `CoolBar` na formu. Traka kontejner se automatski poravnava sa vrhom forme. Promenite karakteristiku `Name` u `CoolBar`.

2. Postavite komponentu `ToolBar` na traku kontejner. Promenite karakteristiku Name trake sa alatima u `MainToolBar`.
3. Dva puta kliknite na karakteristiku `EdgeBorders` u okviru prozora Object Inspector, kako bi bili prikazani svi elementi ivica. Promenite stil `ebTop` u `False` (svi stilovi karakteristike `EdgeBorders` bi trebali da postanu `False`).
4. Promenite karakteristiku `Flat` u `True`. Na ovaj način dugmad trake sa alatima dobijaju ravan izgled, sve dok se kursor ne postavi na komponentu.
5. Kliknite na traku kontejner i promenite karakteristiku `AutoSize` u `True`. Traka kontejner će promeniti veličinu, kako bi obuhvatila traku sa alatima.
6. Kliknite na komponentu `Memo`, a zatim promenite karakteristiku `Align` ove komponente u `alClient`.

Dodavanje dugmadi na traku sa alatima

Sada možete da počnete sa dodavanjem dugmadi na traku sa alatima. Dodaćete nekoliko dugmadi i par oznaka za razmak. U početku dugmad neće imati sliku, ali ćete se o tome pobrinuti kasnije. Za sada, pratite sledeće korake:

1. Kliknite desnim tasterom miša na traku sa alatima i odaberite opciju New Button. Na traku sa alatima će biti postavljeno novo dugme. Promenite karakteristiku Name novog dugmeta u `FileNewBtn`. Postavite karakteristiku `Hint` na `New¦Create A New File`, a karakteristiku `ShowHint` u `True`. (Setite se pisanja koda za savet u lekciji dana 8. Ovaj kod je još uvek u programu, pa će novi saveti trenutno raditi.)
2. Kliknite desnim tasterom miša ponovo na traku sa alatima i ponovo odaberite opciju New Button. Na traku sa alatima će biti postavljeno drugo dugme sa desne strane prvog dugmeta. Promenite karakteristiku `Name` u `FileOpenBtn`. Postavite karakteristiku `Hint` na `Open¦Open An Existing File`, a karakteristiku `ShowHint` u `True`.
3. Dodajte još jedno dugme. Promenite karakteristiku Name ovog dugmeta u FileSaveBtn. Postavite karakteristiku `Hint` na `Save¦Save A File`, a karakteristiku `ShowHint` promenite u `True`.

Dugmad i prazna mesta se uvek dodaju na desnu stranu pored poslednje kontrole koja se nalazi na traci sa alatima. Dugme nije moguće postaviti na određenu lokaciju u okviru trake sa alatima, ali nakon što ste dodali dugme, odnosno prazno mesto, možete ga prevući na drugu poziciju u okviru trake sa alatima. Postojeća dugmad će napraviti mesta za novo dugme.

Na ovaj način smo završili sa ubacivanjem prve grupe dugmadi (ne računajući slike). Sada treba da dodate drugu grupu dugmadi, ali pre nego što to uradite potrebno je da dodate znak za razdvajanje prve i druge grupe. Nazad na posao:

1. Kliknite desnim tasterom miša ponovo na traku sa alatima, ali sada odaberite opciju New Separator. Znak za razdvajanje je dodat na traku sa alatima.
2. Dodajte još jedno dugme. Ovaj put promenite karakteristiku `Name` u `EditCutBtn`, a karakteristiku `Hint` u `Cut¦Cut To Clipboard`.
3. Dodajte dugmad za opcije kopiranja (Copy) i lepljenja (Paste). Promenite karakteristike `Name` i `Hint` za svako dugme.
4. Dodajte još jedan znak za razdvajanje.
5. Dodajte dugme pod nazivom `HelpAboutBtn`. Promenite karakteristiku `Hint` u `About¦About ScratchPad`.
6. Odaberite dugmad Cut, Copy, Paste i Help (koristite kombinaciju Shift+klik na taster miša kako biste odabrali svako dugme). Promenite karakteristiku `ShowHint` u `True`. Za svu odabranu dugmad ova karakteristika će biti promenjena.

Vaša forma sada liči na formu koja je prikazana na slici 13.3.

Slika 13.3
Glavna forma programa ScratchPad nakon dodavanja trake sa alatima

Dovođenje trake sa alatima u funkciju

Sada imate dobru osnovu za traku sa alatima, ali dugmad trake sa alatima nemaju funkciju, pošto im još uvek nisu dodeljeni upravljači događajem na odgovarajuće OnClick događaje. Ove operacije ćemo uraditi u sledećih nekoliko koraka.

1. Kliknite na `FileNewBtn` (prvo dugme), a zatim odaberite karticu Events u okviru prozora Object Inspector. Kliknite na strelicu koja se nalazi u polju za događaj `OnClick` i odaberite događaj `FileNewClick`. Dugme je prikačeno na upravljač događajem `FileNewClick`.
2. Ponovite prvi korak za preostalu dugmad; budite pažljivi prilikom odabiranja odgovarajućeg upravljača `OnClick` za svako dugme (`FileOpenClick`, `FileSaveClick`, `EditCutClick`, itd.).

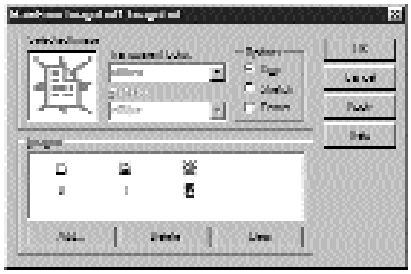
3. Ukoliko još uvek niste kreirali okvir za opis programa (About Box) za program `ScratchPad`, učinite to sada. Kada završite sa kreiranjem okvira za opis programa, kreirajte upravljač događajem za opciju menija Help➥About. Prikačite događaj `OnClick` na dugme `HelpAboutBtn` na upravljač događajem za opciju menija Help➥About.

Dodavanje bitmapa na dugmad trake sa alatima

Očigledno da ovoj traci sa alatima nedostaje nešto. Potrebno je dodati slike na dugmad trake sa alatima. Da biste to uradili, morate dodati komponentu `ImageList` na formu prateći korake:

1. Postavite komponentu `ImageList` na formu. (Pronaći ćete je na kartici Win32 u okviru palete komponenti.) Promenite karakteristiku `Name` u `ImageList`.
2. Kliknite desnim tasterom miša na ikonu komponente `ImageList` u okviru Vaše forme i odaberite prozor ImageList Editor. Nakon toga će biti prikazan prozor ImageList Editor. (Takođe možete da dva puta kliknete mišem na ikonu ImageList u okviru Vaše forme, kako bi bio prikazan prozor ImageList Editor.)
3. Kliknite mišem na dugme Add. Pronađite direktorijum `Common Files\Borland Shared\Images\Buttons`. Odaberite datoteku `FILENEW.BMP`, a zatim kliknite na dugme Open.

   Pojaviće se okvir sa porukom i upitati Vas da li želite da razdvojite bitmapu na dve slike. Ovo se dešava pošto su karakteristike liste slika `Width` i `Height` postavljene na vrednost 16. Bitmapa koju ste odabrali je veća od 16 piksela, pa se mora, ili podeliti na dve slike, ili umanjiti kako bi mogla da stane na dugme. Ukoliko se sećate, bitmape dugmadi koje se isporučuju uz Delphi su zasebne bitmape sa dve slike. Prva slika je bitmapa za normalno dugme, a druga za neaktivno dugme. Treba da dozvolite prozoru ImageList Editor da razdvoji bitmapu na dve slike, a zatim da obrišete drugi deo slike.
4. Kliknite mišem na dugme Yes, kako bi ImageList Editor podelio bitmapu na dve slike. Sada će prozor ImageList Editor prikazati dve slike. Vama treba samo prva, pa ćete kliknuti mišem na drugu sliku (slika deaktiviranog dugmeta), a zatim kliknuti mišem na dugme Delete.
5. Kliknite ponovo mišem na dugme Add. Ovaj put odaberite datoteku `FILEOPEN.BMP`. Kliknite ponovo mišem na dugme Yes nakon upita da li želite da podelite bitmapu na dve slike. Zatim kliknite mišem na sliku za deaktivirano dugme aktivne bitmape i obrišite je. Slika 13.4 prikazuje izgled editora slika, pre brisanja druge slike.



Slika 13.4
ImageList Editor nakon dodavanja tri slike

6. Ponovite korak pet za preostalu dugmad (File Save, Cut, Copy, Paste i About). Koristite bitmape koje želite, ali budite sigurni da ste obrisali dodatne bitmape, svaki put kada dodate sliku na listu. Takođe se uverite da slike u editoru liste slika prate redosled dugmadi koja se nalaze na traci sa alatima. Kada završite rad na listi, imaćete sedam slika sa brojevima od 0 do 6.
7. Kliknite mišem na dugme OK, kako bi zatvorili prozor ImageList Editor.

SAVET Možete odabrati više slika u okviru za dijalog Add Images prozora ImageList Editor, a zatim istovremeno dodati sve slike na spisak.

Sada ste spremni da povežete spisak slika i traku sa alatima. Kliknite mišem na traku sa alatima. Pronađite karakteristiku `Images` u okviru prozora Object Inspector, a zatim odaberite komponentu `ImageList` u okviru spiska komponenti. Ukoliko ste sve ispravno uradili, Vaša dugmad će dobiti slike. Možda niste zapazili da svaki put kada dodate dugme na traku sa alatima, Delphi automatski uvećava karakteristiku `ImageIndex` za novo dugme. Pošto ste kreirali dugmad i slike istim redosledom, slike na dugmadima bi trebale da budu ispravno postavljene. Ukoliko je pogrešna slika na dugmetu, možete izmeniti karakteristiku `ImageIndex` odgovarajućeg dugmeta, ili se vratiti u ImageList Editor i promeniti redosled slika na spisku slika.

SAVET Da biste promenili redosled slika na spisku, prevucite odgovarajuću sliku na novu poziciju u okviru prozora ImageList Editor.

Slike na neaktivnim dugmadima

Za sada imate slike samo za dugmad u aktivnom stanju. Takođe su Vam potrebne slike koje će moći da se prikažu u slučaju da su dugmad neaktivna. Za sada niste deaktivirali dugmad trake sa alatima, mada ćete to uraditi pre kraja današnje lekcije. Postoje dva načina za implementiranje slika na neaktivnu dugmad:

- Da dozvolite traci sa alatima da automatski kreira slike za neaktivno stanje.
- Da kreirate drugu listu slika koja sadrži bitmape za neaktivno stanje dugmadi.

Sigurno je jednostavniji način da dopustite traci sa alatima da automatski kreira neaktivno stanje dugmadi. U većini slučajeva ovo će biti dovoljno. Ipak, ponekad

algoritam za kreiranje slika neaktivnog stanja dugmadi ne radi dobro. (Slike gube svoju definiciju i ne izgledaju dobro.) Ovo se dešava ukoliko dugmad nemaju dovoljan kontrast. U tom slučaju možete kreirati drugu listu slika koja sadrži slike neaktivnih dugmadi. Postavite karakteristiku `DisabledImages` trake sa alatima na vrednost koja odgovara nazivu liste koja sadrži slike sa neaktivnim dugmadima, a ostatak će biti urađen automatski. Za program `ScratchPad` možete ostaviti automatsko deaktiviranje dugmadi u okviru trake sa alatima, tako da dodatne aktivnosti neće biti potrebne.

Jadni stari `ScratchPad` je ponovo sastavljen. Ovo je dobar trenutak da snimite projekat. Nakon što ste snimili projekat, kliknite mišem na dugme Run, kako biste pokrenuli program. Kliknite na svako dugme posebno, kako biste proverili da li dugmad rade ono za šta su predviđena. Ukoliko je sve u redu, imaćete ponovo program koji radi. Ukoliko Vaš program ne može da se prevede, pogledajte ponovo prethodne korake i pokušajte da popravite grešku. Ukoliko sve propadne, možete pronaći projekt `ScratchPad` u okviru koda knjige koji je dostupan na adresi `http://www.mcp.com/info` (lekcija dana 13 - Day 13).

Oblačići sa savetima i saveti trake sa alatima

U lekcijama dana 7, "VCL komponente" i dana 8, "Kreiranje aplikacija u Delphi-ju", obrađeno je skoro sve što ima veze sa oblačićima za savet i savete. U prethodnim lekcijama ste dodavali podršku za tekst saveta u program ScratchPad, dok ste u današnjoj lekciji ponovo kreirali traku sa alatima.

Postoji još jedan element koji nije obrađen, a to je promena karakteristika oblačića za savet. Klasa `TApplication` sadrži četiri karakteristike koje kontrolišu način ponašanja oblačića. Tabela 13.1 prikazuje ove karakteristike i opise karakteristika.

Tabele 13.1: Karakteristike klase `TApplication` koje se odnose na oblačiće za savete

Karakteristika	Opis
`HintColor`	Postavlja boju pozadine oblačića za savet. Generički: clInfoBk.
`HintHidePause`	Kontroliše koliko će vremena (u milisekundama) proteći pre nego što se sakrije oblačić za savet ukoliko miš ostane u nepokretnom stanju nad komponentom. Generički: 2500 milisekundi.
`HintPause`	Kontroliše interval nakon kog će se pojaviti oblačić za savet (u milisekundama); miš se nalazi na komponenti, a vreme se odnosi na vreme nakon kog će se oblačić za savet pojaviti. Generički: 500 milisekundi.
`HintShortPause`	Kontroliše koliko će vremena proteći između prikazivanja dva oblačića za savet nakon što je prethodni oblačić za savet već prikazan; na primer, korisnik prelazi kursorom preko grupe dugmadi na traci sa alatima. Generički: 50 milisekundi.

Generičke vrednosti navedenih karakteristika su dovoljne kod većine aplikacija. Ukoliko još uvek želite da promenite karakteristike saveta, opcije Vam stoje na raspolaganju.

Pravila kuće: Trake sa alatima i saveti

- Nemojte koristiti oblačiće za savet na kontrolama na kojima će biti pokriven tekst koji korisnik treba da pročita. U suštini, nemojte koristiti oblačiće za savet za edit kontrole i kombo okvire. U krajnjem slučaju, dajte korisniku mogućnost da isključi oblačiće za savet kod ovih tipova kontrola.
- Tekst oblačića za savet treba da bude (kod kratkih saveta) kratak i jasan.
- Tekst saveta statusne trake (kod dugih saveta) treba da bude jasan i da opisuje funkciju komponente.
- Razmotrite mogućnost da korisnik isključi sve savete.

Dodavanje ostalih kontrola trakama sa alatima

Pošto je komponenta `ToolBar` svestrana, ništa posebno nije potrebno uraditi kako bi se dodali drugi tipovi kontrola Vašoj traci sa alatima. Najčešći tip kontrole koji se dodaje traci za alate je kombo okvir. Kombo okvir na traci sa alatima možete koristiti da biste odabrali font, opciju za konfiguraciju, podešavanje uvećanja... Mogućnosti su beskonačne.

Da biste dodali komponente traci sa alatima, odaberite komponentu sa palete komponenti i postavite je na traku sa alatima. Traka sa alatima će sama poravnati komponentu. Dodajte prazna mesta ukoliko je to potrebno, kako biste vizuelno odvojili komponente. Kada se komponenta nalazi na traci sa alatima, njome možete upravljati na isti način kao što to radite na formi. Mogao bih da Vam ovo zako mplikujem, ali za to nema potrebe, sve je veoma jednostavno. Ukoliko nikad niste pokušali da implementirate kombo okvir na traku sa alatima koristeći Windows API, nećete moći da shvatite koliko Vam je truda Delphi uštedeo. Verujte mi, ušteda je značajna.

Trake sa alatima postoje u različitim oblicima i veličinama, a Delphi čini njihovo kreiranje i implementiranje veoma jednostavnim. Sa Delphi-jem više nećete imati izgovor: "Ovo je suviše teško!" U principu, čak ćete i uživati u kreiranju traka sa alatima radeći sa Delphi-jem.

Usidrene trake sa alatima

Usidrene trake sa alatima su uobičajene kod većine Windows programa. Usidrene trake sa alatima su paradoks same po sebi. Sa jedne strane, ova mogućnost je veoma dobra i većina iskusnih korisnika očekuje da dobra aplikacija ima trake sa alatima koje imaju mogućnost usidravanja. Sa druge strane, sumnjam da bilo ko stvarno koristi mogućnost usidrenja većine traka sa alatima, koje ovu mogućnost imaju. Ipak, usidrene trake sa alatima i njihovo implementiranje korišćenjem Delphi-ja je prilično jednostavno, pa će možda biti korisno da ih obezbedite.

Mogućnosti usidrenja obrađene u ovom poglavlju se odnose na bilo koju prozorsku kontrolu, ne samo na traku sa alatima.

Kreiranje usidrenih traka sa alatima

Da bi trake sa alatima mogle da se usidre, potrebno je ispuniti dva koraka:

- Podesite karakteristiku `DragKind` na `dkDock`.
- Podesite karakteristiku `DragMode` na `dmAutomatic`.

Nakon što ste podesili ove dve karakteristike, možete prevlačiti Vašu traku sa alatima po ekranu. Prevlačenje trake sa alatima po ekranu Vas neće puno zanimati. Da bi usidrene trake sa alatima imale smisla, treba da definišete odredište za deo jednačine koja se zove prevuci-i-spusti.

Mesta za usidrenje

Usidrene trake sa alatima zahtevaju mesto na koje mogu da se usidre. Kao što je Roger Waters rekao: "Svaka luda zna da je psu potrebna kuća." Kuća za usidrene trake sa alatima je mesto za usidrenje (dock site). Mesto za usidrenje može biti bilo koja prozorska komponenta čija je karakteristika `DockSite` postavljena na `True`. Komponente koje obično koriste mesta za usidrenje su: `TCoolBar`, `TControlBar`, `TPageScroller`, `TPanel` i `TPageControl`. I druge kontrole imaju karakteristiku `DockSite`, ali je manja verovatnoća da će se koristiti kao mesta za usidrenje.

Naredna vežba će nam pomoći za ilustrovanje načina korišćenja mesta za usidrenje. Sledite naredne korake:

1. Postavite komponentu `CoolBar` na praznu formu. Podesite karakteristiku `DockSite` na `True`.
2. Postavite komponentu `ToolBar` na komponentu `CoolBar`. Postavite karakteristiku `DragKind` na `dkDock`, a karakteristiku `DragMode` na `dmAutomatic`. Kreirajte nekoliko dugmadi u okviru trake sa alatima, kako bi lakše mogli da je vidite.
3. Postavite drugu komponentu `CoolBar` na formu. Promenite karakteristiku `Align` ove komponente na `alBottom`, a karakteristiku `DockSite` na `True`.

4. Postavite treću komponentu `CoolBar` na formu. Promenite karakteristiku Align na `alLeft`, a karakteristiku `DockSite` na `True`. Promenite veličinu komponente `CoolBar`, tako da širina bude približno 40 piksela.

Sada pokrenite program. Prevlačite traku sa alatima od jednog do drugog sidrišta. Uočite kako traka sa alatima menja orijentaciju kada je prevučete na traku kontejner koja se nalazi na levoj strani forme.

Eksperimentišite još malo. Podesite karakteristiku `AutoSize` svih komponenti `CoolBar` na `True`. Na ovaj način će komponenta `CoolBar` promeniti veličinu na osnovu kontrola koje se na njoj nalaze. Ponovo pokrenite program i pomerajte traku sa alatima sa jednog na drugo sidrište. Uočite da je svaka traka kontejner skoro nevidljiva, sve dok se traka sa alatima ne usidri na sidrište. Nakon usidrenja trake sa alatima, traka kontejner se proširuje kako bi obuhvatila traku sa alatima.

Plutajuće trake sa alatima (floating toolbars)

Traka sa alatima može postati plutajuća traka sa alatima (okvir za alate), ukoliko odvojite traku od sidrišta i postavite je bilo gde (bilo gde samo ne na drugo sidrište). Traka sa alatima postaje plutajući prozor. Možete čak i definisati tip prozora koji će služiti kao plutajuće sidrište postavljanjem karakteristike `FloatingDockSiteClass` na naziv klase koja će služiti kao sidrište plutajućoj traci za alate. Na primer, pretpostavimo da ste dizajnirali formu koja sadrži sve karakteristike koje su potrebne plutajućem okviru za alate koji kreira korisnik i da ste ovoj formi dodelili naziv `MyToolBox`. U tom slučaju možete ovu formu definisati kao plutajuću traku za alate, ukoliko koristite sledeći kod:

```
ToolBar.FloatingDockSiteClass := TMyToolBox;
```

Kada je traka sa alatima slobodna (nije usidrena), a korisnik otpusti taster miša, Delphi će automatski kreirati slučaj klase `TMyToolBox` i postaviti traku sa alatima na formu koja se koristi za takvu vrstu klase. Da biste ponovo usidrili plutajući okvir za alate, spustite ovu komponentu na bilo koje sidrište. Da bi sidrište moglo da prihvati plutajući okvir za alate, morate odgovoriti na događaje `OnDockOver` i `OnDockDrop` za odgovarajuće sidrište. Kod upravljača događajem `OnDockDrop` pozovite metodu `ManualDock` trake sa alatima, kako bi se traka sa alatima usidrila.

Statusne trake (status bars)

Statusna traka (status bar) je još jedna opcija koja Vašu aplikaciju čini marketinški prihvatljivom. Iako nemaju sve aplikacije koristi od statusne trake, u većini slučajeva statusna traka predstavlja poboljšanje. VCL komponenta `StatusBar`, koja enkapsulira Win32 kontrolu statusne trake čini kreiranje statusnih traka jednostavnim. Za početak pogledajte na trenutak glavne karakteristike komponente `StatusBar` koje su prikazane u tabeli 13.2.

Tabela 13.2: Karakteristike komponente `StatusBar`

Karakteristika	Opis
`AutoHint`	Automatski prikazuje savete na statusnoj traci kada kursor miša prelazi preko komponente čija je karakteristika `Hint` definisana.
`Panels`	Za statusne trake koje sadrže više panoa ova karakteristika definiše zasebne panoe.
`SimplePanel`	Određuje da li će statusna traka prikazati jednostavni pano, odnosno višestruke panoe.
`SimpleText`	Tekst za jednostavni pano statusne trake.
`SizeGrip`	Određuje da li će statusna traka prikazivati hvatače za promenu veličine u donjem desnom uglu. Hvatač za promenu veličine obezbeđuje oblast koju korisnik može da uhvati, kako bi promenio veličinu prozora. Odsustvo hvatača za promenu veličine neće sprečiti promenu veličine prozora, ali će prisustvo hvatača za promenu veličine učiniti promenu veličine prozora lakšim.
`UseSystemFont`	Uvek koristi tekući sistemski font, a preskače trenutnu postavku karak teristike `Font`. Ovo je posebno korisno za korisnike koji koriste teme programa Plus pack.

Kao što ste iz ove tabele mogli da primetite, statusna traka može biti jednostavna statusna traka, odnosno statusna traka sa više panoa. Ovaj izbor će biti obrađen u sledećem poglavlju.

Jednostavna, ili kompleksna?

Statusna traka može biti jednostavna statusna traka, ili kompleksna statusna traka. Jednostavna statusna traka (simple status bar) sadrži jedan pano koji obuhvata kompletnu statusnu traku. Ukoliko želite jednostavnu statusnu traku, podesite karakteristiku `SimplePanel` na `True`. Karakteristika `SimplePanel` funkcioniše kao prekidač. Između jednostavne i kompleksne statusne trake možete prelaziti u toku rada programa postavljajući vrednost karakteristike `SimplePanel` na `True`, odnosno `False` u zavisnosti od potrebe.

Kompleksna statusna traka (complex status bar) sadrži više panoa. Ukoliko odaberete kompleksnu statusnu traku, možete koristiti StatusBar Panels Editor, kako bi podesili panoe koje želite da budu prikazani u okviru Vaše statusne trake. Da biste pozvali prozor StatusBar Panels Editor, kliknite dva puta mišem na kolonu Value koja pripada karakteristici `Panels`. Da biste dodali pano, kliknite na dugme Add New u okviru prozora StatusBar Panels Editor. Za brisanje panoa kliknite mišem na dugme Delete Selected. Da biste editovali pano, odaberite željeni pano, a zatim promenite karakteristike panoa u prozoru Object Inspector. Slika 13.5 prikazuje prozor StatusBar Panels Editor i prozor Object Inspector u toku editovanja panoa.

Slika 13.5
Prozor StatusBar Panels Editor

NAPOMENA Zasebni panoi koji se nalaze u okviru kompleksne statusne trake su slučajevi klase `TStatusPanel`.

Većina karakteristika su jasne same po sebi, ali neke od njih zahtevaju detaljnije objašnjenje. Karakteristika `Text` sadrži tekst koji će biti prikazan na panou. Da bi u toku rada programa izmenili tekst koji će biti prikazan na panou, možete koristiti karakteristiku `Text`. Postavljanje teksta statusne trake će biti objašnjeno nešto kasnije; u toku dizajniranja, nije neophodno da pano ima obezbeđen tekst, ukoliko ćete menjati tekst panoa u toku rada.

Karakteristiku `Style` možete podesiti na `psText`, ili na `psOwnerDraw`. Ukoliko je karakteristika `Style` postavljena na `psText` (generička vrednost), pano će se ponašati onako kako ste očekivali. Tekst je poravnat u okviru panoa u zavisnosti od vrednosti karakteristike `Alignment`. Ukoliko je karakteristika `Style` postavljena na vrednost `psOwnerDraw`, moraćete sami da upišete tekst, odnosno prikažete sliku koja će se nalaziti na panou. Crtanje korisnika po panou će biti objašnjeno u poglavlju "Panoi statusne trake koje crta korisnik".

Karakteristike panoa `Width`, `Bevel` i `Alignment` su same po sebi jasne. Eksperimentišite sa ovim karakteristikama, kako biste videli način na koji ove karakteristike utiču na izgled statusne trake.

SAVET U okviru dizajnera forme možete trenutno videti rezultate izmena koje ste načinili u okviru statusne trake koristeći StatusBar Panels Editor. Postavite StatusBar Panels Editor tako da možete da vidite statusnu traku u toku rada sa ovim prozorom. Prilikom svake izmene rezultat će biti prikazan na dizajneru forme.

Nakon što ste završili sa dodavanjem panoa statusnoj traci, možete zatvoriti prozor StatusBar Panels Editor i vratiti se u dizajner forme.

NAPOMENA Kada menjate karakteristiku `Panels` komponente `StatusBar`, dizajner forme će automatski podesiti karakteristiku `SimplePanel` na `False`. Podrazumeva se da ćete koristiti višestruke panoe i da Vam nije potrebna jednostavna statusna traka.

Promena teksta u okviru statusne trake

Postoje dva načina za promenu teksta u okviru statusne trake:

- Manuelno izmenite karakteristiku `SimpleText` u okviru statusne trake (za jednostavne statusne trake), odnosno karakteristiku `Text` za zaseban pano (za kompleksne statusne trake).
- Dopustite da VCL automatski obezbedi tekst za statusnu traku postavljanjem karakteristike `AutoHint` na `True`.

Manuelna izmena teksta u okviru statusne trake je jednostavna, naročito ukoliko imate jednostavnu statusnu traku. Kada je karakteristika `SimplePanel` postavljena na `True`, karakteristiku `SimpleText` možete podesiti tako što ćete upisati tekst koji želite da bude prikazan u okviru statusne trake:

```
StatusBar.SimpleText := 'This shows up in the status bar.';
```

U slučaju kompleksnih statusnih traka, promena teksta je nešto komplikovanija. Ukoliko želite da promenite tekst prvog panoa kompleksne statusne trake, možete koristiti kod koji izgleda ovako:

```
StatusBar.Panels[0].Text := 'Status Bar Text';
```

Karakteristika Panels komponente `StatusBar` sadrži karakteristiku pod nazivom `Items` koja predstavlja niz panoa u okviru statusne trake. Postavljanje karakteristike `Text` za element u okviru niza `Items` menja tekst za dati pano (pošto karakteristika `Items` predstavlja generičku karakteristiku u obliku niza za objekt `Panels`, ne morate posebno da ukazujete na niz `Items`). Kao što možete videti, niz ima osnovu 0. Prvi pano u okviru statusne trake je element niza 0.

Automatski tekst saveta za statusnu traku ne zahteva posebno objašnjenje. Sve što treba da uradite je da karakteristiku `AutoHint` postavite na vrednost `True`. Ostatak je, kao što i sama karakteristika nagoveštava, automatizovan.

Čak, iako koristite opciju AutoHint, još uvek možete manuelno izmeniti tekst statusne trake. Ne postoji ništa što Vas može sprečiti da manuelno promenite tekst, ali zapamtite da će ovaj tekst biti zamenjen nakon što kursor miša pređe preko komponente koja sadrži tekst saveta.

Panoi statusne trake koje iscrtava korisnik

U prethodnom delu sam objasnio da karakteristika `Style` panoa može sadržati vrednosti `psText`, odnosno `psOwnerDraw`. Kada postavite stil panoa na `psOwnerDraw`, morate preuzeti odgovornost za crtanje svega što je potrebno da bude prikazano na panou. Najverovatnije nećete imati problema ukoliko panoe koje iscrtava korisnik koristite samo za prikazivanje teksta. To obično znači da ćete na statusnoj traci prikazivati neku vrstu ikone, ili bitmape. Bez obzira na način crtanja po panou, koraci su isti:

1. Postavite karakteristiku `Style` panoa na `psOwnerDraw` (obično koristeći StatusBar Panels Editor).
2. Odgovorite na događaj `OnDrawPanel`.

Očigledno da pravi posao u ovom slučaju ima veze sa upravljačem događajem za događaj `OnDrawPanel`. Dekleracija upravljača događajem `OnDrawPanel` izgleda ovako:

```
procedure TForm1.StatusBar1DrawPanel(StatusBar: TStatusBar;
  Panel: TStatusPanel; const Rect: TRect);
```

Parametar `StatusBar` je pointer na statusnu traku. U svakom slučaju imate pointer na statusnu traku (karakteristika Name komponente `StatusBar`), pa ovaj parametar nije u potpunosti koristan izuzev ako ne koristite više statusnih traka koje isctava korisnik. Karakteristika `Panel` je pointer na određeni pano koji trenutno treba iscrtati. Ovaj parametar možete koristiti da odredite koji pano ima potrebu da bude iscrtan, ukoliko u Vašoj statusnoj traci postoji više od jednog panoa koje korisnik iscrtava. Parametar `Rect` sadrži podatke o veličini i poziciji panoa. Parametar `Rect` je važan pošto Vam saopštava tačne dimenzije oblasti za crtanje.

Upravljač događajem `OnDrawPanel` se poziva jednom za svaki pano čija je karakteristika `Style` postavljena na `psOwnerDraw`. Ukoliko imate samo jedan pano koji treba iscrtati, ne treba da brinete; jedino treba da obratite pažnju na parametar `Rect`. Ukoliko je potrebno da iscrtate višestruke panoe, prvo je potrebno da odredite koji ćete pano iscrtati, a zatim je potrebno da ga iscrtate. Ilustracija će možda ovo objasniti. Kod u okviru knjige uključuje program pod nazivom `StatBar` koji ilustruje neke mogućnosti korišćenja statusnih traka. Pokrenite program i ispitajte njegov izvorni kod, kako biste pronašli savete za implementaciju statusnih traka u okviru Vaših aplikacija. Slika 13.6 prikazuje rad programa `StatBar`.

Slika 13.6
StatBar program sa panoima statusne trake koje iscrtava korisnik

Kao što ste mogli da primetite, statusna traka u ovom programu ima više panoa. Srednji od tri panoa je pano koji iscrtava korisnik. Panoi označeni sa `OVR` i `EXT` simuliraju statusnu traku na tekst procesoru, ili editoru koda. U programima ovog tipa modovi za izbor Overtype, odnosno Extended Selection mogu biti uključeni, ili isključeni. Ukoliko je mod uključen, tekst u panou statusne trake će biti prikazan crnom bojom. Ukoliko je mod isključen, tekst izgleda kao deaktivirani 3D tekst. Treći pano koji iscrtava korisnik prikazuje stek Windows ikona, kako bi ilustrovao

način korišćenja grafike na statusnoj traci. Pokrenite program i eksperimentišite, kako bi shvatili način rada programa.

Listing 13.1 prikazuje upravljač događajem `OnDrawPanel` iz programa `StatBar`. Pogledajte ga i pročitajte komentare, kako biste shvatili šta se događa u kodu.

Listing 13.1: Metoda `StatusBarDrawPanel` u okviru programa `StatBar`

```
procedure TMainForm.StatusBarDrawPanel(StatusBar: TStatusBar;
  Panel: TStatusPanel; const Rect: TRect);
var
  R : TRect;
  Icon : HIcon;
begin
  with StatusBar.Canvas do begin
    { Create a temporary TRect object. The Rect parameter
    { is const so we can't change it. }
    R := Rect;
    { Check to see if panel 3 is the panel which needs
    { to be drawn. If so, draw an icon in the panel. }
    if Panel.Index = 3 then begin
        { Load one of the stock Windows icons. This time
        { using the API is easier than using VCL. }
        Icon := LoadIcon(0, IDI_HAND);
        { Draw the icon and shrink it down to 15 x 15 pixels. }
        { Center it in the panel, too. }
        DrawIconEx(Handle, Rect.Left + 6, 3,
          Icon, 15, 15, 0, 0, DI_NORMAL);
        { Nothing more to do. }
        Exit;
    end;
    { This rather lengthy if statement checks to see if
    { either the Overtype Mode or Extended Selection
    { check boxes are checked. If so, then what we need
    { to do is to draw the text twice. First, we draw it
    { in white. Then we draw it again, offset by 1 pixel,
    { in gray. The effect is a 3D disabled-text look. }
    if ((Panel.Index = 1) and (OvrMode.Checked = False)) or
      ((Panel.Index = 2) and (ExtendedSel.Checked = False))
        then begin
      { Move over and down one pixel for the offset. }
      Inc(R.Left);
      Inc(R.Top, 2);
      { Change the text color to white. }
      Font.Color := clWhite;
      { Set the backround mode to transparent so the
      { text appears hollow and so that the white
      { text can be seen under the gray. }
      Brush.Style := bsClear;
      { Draw the text using the API function DrawText. }
```

nastavlja se

Listing 13.1: Metoda `StatusBarDrawPanel` u okviru programa `StatBar` *nastavak*

```
        { I use DrawText because it allows me to center
        { the text both horizontally and vertically within
        { the given rectangle. }
        DrawText(Handle, PChar(Panel.Text), -1,
          R, DT_CENTER or DT_VCENTER or DT_SINGLELINE);
        { Set the color to gray because we're going to
        { draw the text in gray in a moment. }
        Font.Color := clGray;
        { Set the rect back to the original size. }
        Dec(R.Left);
        Dec(R.Top, 2);
      end;
      { Display the text. If the item is not disabled then
      { the default color (black) is used to draw the text. }
      { If the item is disabled, then the text color has
      { been set to gray by the code above. }
      DrawText(Handle, PChar(Panel.Text), -1,
        R, DT_CENTER or DT_VCENTER or DT_SINGLELINE);
    end;
end;
```

Ovaj kod Vam možda izgleda zastrašujuće, ali njegov veći deo čine linije sa komentarom. Sam po sebi kod je relativno jednostavan. Linije komentara objašnjavaju šta se događa u svakom koraku: 3D prikaz za neaktivan tekst je ostvaren crtanjem teksta belom, a zatim crtanjem teksta sivom bojom gde je tekst nacrtan sivom bojom malo pomeren u odnosu na prethodni tekst. Kao rezultat tekst izgleda udubljen. Ikona koja je prikazana koristi Windows API funkcije `LoadIcon` i `DrawIconEx`.

Iscrtavanje panoa statusne trake kojim upravlja korisnik je u početku zbunjujuće, ali uskoro ste mogli i da se uverite da to i nije tako loše. Može se dogoditi da pišete Windows aplikacije veoma dugo i da Vam nikad u Vašoj statusnoj traci ne budu potrebni panoi koje iscrtava korisnik. Ukoliko Vam ikada budu zatrebali znaćete da ih nije nemoguće ostvariti.

Dodavanje funkcionalnosti sa aktiviranjem komandi

Aktiviranje komandi je proces aktiviranja, odnosno deaktiviranja dugmadi u zavisnosti od trenutnih uslova. Na primer, nema svrhe da dugmad Cut i Copy, odnosno opcije menija budu aktivirane u okviru tekst editora, ukoliko trenutno nije odabran tekst. Slično je i sa dugmetom Paste; ukoliko ne postoji tekst na Clipboard-u, u tom slučaju bi dugme Paste trebalo da bude deaktivirano.

Aktiviranje komandi nije teško naročito ukoliko koristite novu Delphi-jevu komponentu `TActionList`. Ipak, još uvek treba vremena da bi stvari ispravno radile. Potrebno je vremena pošto treba da posvetite pažnju detaljima. (Ponekad je pažnja koja je posvećena detaljima linija koja razdvaja odličnu aplikaciju od osrednje aplikacije.)

Aktiviranje komandi korišćenjem komponenti TActionList i TAction

Klasa TAction pruža zgodniji način za aktiviranje komandi. Klasa TActionList je nevizuelna komponenta koja upravlja akcijama i može se pronaći u kartici Additional u okviru palete komponenti. Kao što joj samo ime sugeriše, klasa TActionList sadrži spisak objekata TAction. Potrebno je da kreirate aktivnost, a zatim da dodelite tu aktivnost bilo kojoj kontroli koja treba da bude aktivirana, odnosno deaktivirana, u zavisnosti od željene aktivnosti. Pod kontrolama podrazumevam opcije menija, dugmad trake sa alatima, opcije menija sadržaja itd.

Uzmimo, na primer, opciju menija Edit➥Cut. Potrebno je da dodelite najmanje tri objekta ovom zadatku:

- Opcija glavnog menija.
- Dugme u okviru trake sa alatima.
- Opcija padajućeg menija.

Aktivnost kreirate koristeći ActionList Editor. Za prethodni primer opcije menija Edit➥Cut, treba da kreirate aktivnost za opciju Cut pod nazivom, pretpostavimo CutAction. Zatim, koristeći prozor Object Inspector, možete dodeliti CutAction karakteristici Action za svaki od objekata koji učestvuje u operaciji isecanja (na primer, dugmad trake sa alatima i meniji). U toku rada programa, kada je potrebno da aktivirate opciju Cut, to možete učiniti korišćenjem samo jedne linije koda:

```
CutAction.Enabled := True;
```

Ovo će aktivirati sve komponente sa odgovarajućim karakteristikama Action, koje su postavljene na CutAction. Deaktiviranje opcija Cut je jednostavno koliko i dodeljivanje vrednosti False karakteristici Enabled odgovarajuće aktivnosti. Događaj OnUpdate klasa TActon i TActionList obezbeđuje zgodno mesto za postavljanje Vašeg koda koji aktivira komande.

Aktiviranje komande korišćenjem klase TAction je nešto što treba da isprobate, kako biste je kasnije cenili u potpunosti. Aktiviranje komandi ćete dodati u program ScratchPad u narednom poglavlju.

Implementacija aktiviranja komandi

U ovom odeljku ćete implementirati aktiviranje komandi za program ScratchPad. Prvo što treba da uradite je da podesite komponentu ActionList, a zatim da prikačite različite komponente na listu aktivnosti.

Kreiranje komponente ActionList

Sve počinje sa komponentom ActionList koja je središte VCL sistema za aktiviranje komandi. Prvo treba da dodate aktivnosti za opcije menija Edit. Nakon toga treba da dodate aktivnosti za opcije Save i Save As menija File. Naredni koraci će Vas voditi kroz proces podešavanja komponente ActionList.

Kreiranje aktivnosti menija Edit

Naredni koraci Vam pokazuju kako da kreirate aktivnosti za opcije Cut, Copy i Paste u okviru menija Edit. Izvršite sledeće korake:

1. Postavite komponentu ActionList na formu i promenite karakteristiku Name u ActionList.
2. Dva puta kliknite mišem na ikonu komponente ActionList, kako biste pozvali ActionList Editor.
3. Kliknite desnim tasterom miša na prozor ActonList Editor i odaberite opciju New Standard Action u okviru menija sadržaja. Odaberite aktivnost TEditCopy a zatim kliknite na dugme OK. Uočite da se prozor Object Inspector menja, kako bi prikazao karakteristike klase aktivnosti TEditCopy.

   Želeo bih da zastanem na trenutak i objasnim detaljnije kako rade aktivnosti. Istražite Object Inspector u ovom trenutku. Uočite da aktivnost TEditCopy ima nekoliko sličnih karakteristika i da te karakteristike imaju svoje vrednosti. U suštini, uočite da karakteristike Caption, Hint, ImageIndex i ShortCut već imaju vrednosti koje odgovaraju na opciju Copy menija Edit. Ove karakteristike će biti prebačene na bilo koju kontrolu kojoj dodelite ovu aktivnost, što znači da morate da podesite bilo koju karakteristiku aktivnosti na vrednosti koje želite da prihvati odgovarajuća komponenta. Ovo nema mnogo smisla, sve dok ne prikačite aktivnost za komponentu u sledećem poglavlju, ali malo kasnije sve će Vam biti jasno. Nastavimo sa aktivnostima kreiranja procesa.
4. Promenite karakteristiku Name nove aktivnosti u CopyAction, karakteristiku Hint u Copy¦Copy to Clipboard, a karakteristiku ImageIndex u 3.
5. Kreirajte još jednu standardnu aktivnost; ovaj put aktivnost će biti TEditCut. Promenite karakteristiku Name u CutAction, karakteristiku Hint u Cut¦Cut to Clipboard, a karakterisitku ImageIndex u 4.
6. Kreirajte treću aktivnost; aktivnost TEditPaste. Promenite karakteristiku Name u PasteAction, karakteristiku Hint u Paste¦Paste from Clipboard, a karakteristiku ImageIndex u 5.

Sada ste kreirali aktivnosti za primarne opcije menija Edit.

Kreiranje aktivnosti menija File

Sledeći korak je kreiranje aktivnosti za opcije Save i Save As menija File. Izvršite sledeće korake:

1. Kliknite desnim tasterom miša na ActionList Editor i ovaj put odaberite opciju New Action. Biće kreirana nova klasa `TAction`, kao što je to prikazano u prozoru Object Inspector.
2. Promenite karakteristiku `Name` u `SaveAction`, karakteristiku `Caption` u `&Save...`, karakteristiku `Category` u `File`, karakteristiku `Hint` u `Save¦Save a File`, karakteristiku `ImageIndex` u 2, a karakteristiku `ShortCut` u Ctrl+S.
3. Kreirajte još jednu novu aktivnost. Promenite karakteristiku `Name` u `SaveAsAction`, karakteristiku `Caption` u `Save &As...`, a karakteristiku `Category` u `File`. Karakteristike `Hint` i `ImageIndex` nije potrebno da podešavate, pošto ove aktivnosti nemaju dodeljenu dugmad za traku sa alatima. Slika 13.7 prikazuje prozor ActionList Editor u ovom trenutku.

Slika 13.7 ActionList Editor nakon kreiranja elemenata aktivnosti

4. Zatvorite prozor ActionList Editor.

NAPOMENA Kategorije aktivnosti možete kreirati ukoliko imate nekoliko desetina aktivnosti koje treba da pratite. Da biste kreirali kategoriju aktivnosti, možete jednostavno podesiti karakteristiku Category jedne, ili više aktivnosti na naziv koji želite. Na primer, da biste kreirali kategoriju za prethodno kreirane aktivnosti grupe Edit, podesite karakteristiku Category aktivnosti, koje pripadaju grupi, na naziv Edit. Kategorije aktivnosti su uvedene samo zbog organizacije i nemaju nikakvog uticaja na način rada aktivnosti.

Priključivanje aktivnosti komponentama

Sledeći korak je priključivanje aktivnosti koje ste upravo kreirali odgovarajućim opcijama menija i dugmadima trake za alate na koje će ove aktivnosti odgovoriti. Izvršite sledeće korake:

1. Dva puta kliknite mišem na komponentu MainMenu, kako bi pokrenuli editor menija (Menu Editor).
2. Odaberite opciju menija File➥Save i promenite karakteristiku `Action` u `SaveAction`.

3. Odaberite opciju menija File➥Save As i promenite karakteristiku `Action` ove opcije u `SaveAsAction`.
4. Pređite na meni Edit i odaberite opciju menija Cut. Promenite karakteristiku `Action` u `CutAction`.
5. Ponovite korak četiri za opcije menija Copy i Paste koristeći nazive `CopyAction` i `PasteAction` za karakteristiku `Action`. Zatvorite editor menija.
6. U dizajneru forme kliknite na dugme File Save u okviru trake sa alatima, promenite karakteristiku `Action` dugmeta u `FileSaveAction`.
7. Ponovite šesti korak za dugmad Cut, Copy i Paste u okviru trake sa alatima, postavljajući karakteristiku `Action` na `CutAction`, `CopyAction` i `PasteAction`, respektivno.
8. Promenite karakteristiku `Action` opcije menija `MemoPopup` po želji.

Verovatno niste primetili da, kada ste dodelili naziv `SaveAction` karakteristici `Action` odgovarajuće komponente, karakteristike `Caption`, `Checked`, `Enabled`, `HelpContext`, `Hint`, `ImageIndex`, `ShortCut` i `Visible` su bile automatski promenjene na vrednosti odgovarajućih karakteristika, definisanih u objektu `SaveAction`. Važno je da shvatite da će karakteristike aktivnosti obrisati karakteristike bilo koje komponente kojoj je aktivnost i dodeljena. Morate podesiti aktivnost imajući to na umu. Zbog toga sam morao da promenim karakteristike `Hint` i `ItemIndex`, kada ste kreirali aktivnost. Ukoliko ovo ne uradite, tekst saveta i slike na dugmadima trake za alate neće odgovarati aktivnostima.

Sada je svaka komponenta koja se nalazi na spisku prikačena na aktivnost. Kada se određena aktivnost izmeni, bilo koja komponenta koja je prikačena na tu aktivnost, takođe će biti promenjena. Kao primer možete uzeti sledeći kod:

```
SaveAction.Enabled := False;
```

Nakon izvršavanja ovog koda, bilo koja komponenta čija karakteristika `Action` je podešena na vrednost `SaveAction` će biti neaktivna (opcija Save glavnog menija i dugme Save u okviru trake sa alatima). Da biste aktivirali svu dugmad dodeljenu ovoj aktivnosti, koristite sledeći kod:

```
SaveAction.Enabled := True;
```

Ovo je veoma jednostavno. Pošto glavni meni i komponenta Save u okviru trake sa alatima imaju karakteristike `Action` podešene na vrednost `SaveAction`, sledeća dva isečka koda su potpuno jednaka:

```
{ One-shot using the Action. }
SaveAction.Enabled := False;

{ The hard way. }
FileSave.Enabled := False;
FileSaveBtn.Enabled := False;
```

Na ovaj način štedite samo jednu liniju koda, kao što ste mogli da vidite u okviru prethodnog primera, ali ako imate nekoliko komponenti koje bi trebalo aktivirati, odnosno deaktivirati, ove aktivnosti će Vam uštedeti mnogo vremena. Nakon što kreirate aktivnost i dodelite ovu aktivnost jednoj komponenti, odnosno kompletnoj grupi komponenata, aktiviranje komandi je jednostavno i obavlja se jednom linijom koda. Lepota ovog načina rada je što, bez obzira koliko komponenti treba da aktivirate, odnosno deaktivirate, za čitavu operaciju Vam je potrebna samo jedna linija koda.

Pokrenite program `ScratchPad`. Uočite da su dugmad Cut i Copy deaktivirana. Upišite bilo kakav tekst u memo polje i označite ga. Dugmad Cut i Copy u okviru trake sa alatima se nekom čarolijom aktiviraju. Kliknite bilo gde u okviru memo polja, kako bi ukinuli izbor teksta. Dugmad Cut i Copy su ponovo neaktivna. Da li je dugme Paste aktivirano? Ukoliko je dugme aktivirano, pritisnite tastere Alt+Print Screen na tastaturi. (Na ovaj način kopirate tekući prozor u Clipboard kao bitmapu.) Kada pritisnete tastere Alt+Print Screen, dugme Paste će postati neaktivno, pošto bitmapu ne možete zalepiti u memo polje. Odaberite bilo koji tekst, a zatim kliknite, ili na dugme Cut, ili na dugme Copy. Dugme Paste je sada aktivirano, pošto Clipboard sadrži tekst koji možete zalepiti u memo polje.

Kako ovo radi? Standardne aktivnosti `TEditCopy`, `TEditCut` i `TEditPaste` automatski određuju kada treba da budu aktivirane, a kada deaktivirane njima odgovarajuće komponente, u trenutku kada bilo koja edit kontrola dobije ulazni fokus. Ovo nije magija, ali je nešto što liči na magiju! Treba samo da kreirate standardne aktivnosti, dok će ostatak biti automatski obavljen. Nije potrebno da pišete bilo kakav kod da bi komande menija Edit mogle da budu aktivne, odnosno neaktivne. Ovo ne možete izbeći!

Aktiviranje komandi za opcije Save i Save As

Opcije menija Edit su bile jednostavne zato što niste morali da pišete ni jednu liniju koda. Opcije menija File, Save i Save As zahtevaju nešto više rada, pošto ne postoje standardne aktivnosti za ove opcije menija. Ipak, nemojte brinuti, pošto aktiviranje komandi za ove opcije ne zahteva puno rada. Da biste implementirali aktiviranje komandi za ove opcije menija, možete koristiti događaj `OnUpdate`. Prvo je potrebno nešto dodatnih informacija da biste postavili događaj `OnUpdate` u perspektivu.

Događaj `OnUpdate` pruža dobro mesto za postavljanje Vašeg koda za aktiviranje komande. Kada Vaša aplikacija ostane bez poruka koje treba da obradi, Windows joj šalje poruku `WM_ENTERIDLE`. Ustvari, Windows saopštava Vašem programu: "Za sada nemam ništa što bi mogao da radiš, stoga se opusti i odmori na trenutak." Kada Delphi aplikacija prihvati poruku `WM_ENTERIDLE`, aktivira događaj `OnUpdate` klase `TAction`. Sve što treba da uradite je kreiranje upravljača događajem za događaj OnUpdate i aktiviranje Vaše komande. Upravljač događajem možete koristiti za proveru stanja memo komponente i u skladu s tim možete aktivirati opcije Save i Save As.

Poslednji korak koji nam je ostao je kreiranje upravljača događajem za događaj `OnUpdate` definisanih aktivnosti. Izvršite sledeće korake:

1. Dva puta kliknite na komponentu `ActionList`, kako biste pokrenuli ActionList Editor.
2. Odaberite aktivnost `SaveAction` sa spiska dostupnih aktivnosti. Kliknite na kategoriju aktivnosti File, odnosno (`All Actions`), ukoliko ne vidite aktivnost na listi aktivnosti.
3. U okviru prozora Object Inspector dva puta kliknite na kolonu Value koja se nalazi pored događaja `OnUpdate`. Editor koda će prikazati upravljač događajem `OnUpdate`. Nešto kasnije ćete u upravljač događaja upisati kod.
4. Pronađite ActionList Editor (koristite opciju View➥Window List, ukoliko ne možete da pronađete prozor ActionList Editor). Odaberite opciju `SaveAsAction` sa spiska aktivnosti.
5. U okviru prozora Object Inspector kliknite na strelicu na dole, koja se nalazi pored događaja `OnUpdate`. Odaberite sa liste `SaveActionUpdate`. Ovo omogućava da opcije menija Save i Save As koriste isti upravljač događajem za događaj `OnUpdate`.
6. Zatvorite prozor ActionList Editor.

Kreiranje upravljača događajem je, naravno, jednostavniji deo. Teži aspekat je pisanje koda koji se nalazi između iskaza `begin` i end. Listing 13.2 prikazuje kompletirani upravljač događajem `OnUpdate`. Prebacite se na editor koda, a zatim unesite kod koji je prikazan u okviru listinga 13.2 u Vaš upravljač događajem `OnUpdate`.

Listing 13.2: Upravljač događajem OnUpdate programa `ScratchPad`

```
procedure TMainForm.SaveActionUpdate(Sender: TObject);
begin
  { Command enabler for Save and Save As. }
  SaveAction.Enabled :=
    Memo.Modified and (Length(Memo.Lines.Text) > 0);
  SaveAsAction.Enabled := SaveAction.Enabled;

  { The following two command enablers don't use actions. }
  { Instead the Enabled property of the two menu items    }
  { is accessed directly. }
  { Command enabler for Select All. }
  EditSelectAll.Enabled := Memo.Lines.Count > 0;
  { Command enabler for Undo. }
  EditUndo.Enabled := Memo.Modified;
end;
```

Karakteristika Enabled aktivnosti `SaveAction` je postavljena na osnovu izmene, odnosno neizmenjenog stanja memo polja, i na osnovu toga da li memo polje sadrži

tekst. U školjki, aktivnost Save je aktivirana, ukoliko je memo polje bilo menjano nakon učitavanja i ukoliko sadrži tekst. Ista vrednost je dodeljena i karakteristici Enabled koja pripada aktivnosti `SaveAsAction`. Ovo omogućava da se istovremeno podrže opcije Save i Save As koristeći iste kriterijume.

Uočite da sam preskočio par dodatnih komandi u okviru upravljača događajem `OnUpdate`. Pokretači komande za opcije Select All i Undo menija Edit ne koriste aktivnosti. Umesto toga, karakteristika `Enabled` opcije menija se postavlja direktno. Korišćenje aktivnosti za ove dve opcije predstavlja ubijanje dve muve jednim udarcem, pošto je potrebna samo jedna linija koda za oba slučaja. Sve dok imate upravljač događajem `OnUpdate`, možete ga koristiti za bilo koji tip komande koji želite da aktivirate. Postavimo to na drugi način; ovaj upravljač događajem `OnUpdate` se ne koristi ekskluzivno za opcije menija Save i Save As koje pripadaju meniju File. Bilo koje aktiviranje komandi može biti urađeno u okviru događaja `OnUpdate`.

Upravljač događajem `OnUpdate` može biti pozvan hiljadu puta u sekundi. Iz tog razloga, kod u okviru ovog metoda mora biti što kraći.

Debagiranje koda upravljača događajem OnUpdate je neugodno. Problem predstavljaju tačke prekida koje u upravljaču događajem OnUpdate mogu biti aktivirane ubrzo nakon pokretanja programa. Ukoliko želite da debagirate kod upravljača događajem OnUpdate, trebali bi da koristite metode za debagiranje koje su drugačije od direktnog definisanja tačaka prekida. Dva ovakva metoda su: korišćenje uslovne tačke prekida i korišćenje funkcije OutputDebugString za slanje poruka u datoteku za evidentiranje događaja (Event Log).

Dodatak klasi `TCommandList`

Postoji dodatna korist od spiska komandi koju još nisam napomenuo. Da biste videli ovu (do sada) skrivenu korist, potrebno je da izvršite sledeće korake:

1. Odaberite ikonu MainMenu u okviru glavne forme programa `ScratchPad`.
2. Promenite karakteristiku `Images` u `ImageList`.
3. Uradite iste korake za komponentu `PopupMenu`.

Sada pokrenite program i pogledajte menije File i Edit. Hej! Instant bitmape menija! Opcije menija nasleđuju karakteristiku `ImageIndex` njihovih dodeljenih aktivnosti. Ono što je potrebno da uradite je da aktivirate bitmape i omogućite korišćenje iste liste slika za aktivnosti karakteristike `Images` koja pripada meniju. Ostatak se obavlja automatski.

Štampanje u Delphi aplikacijama

Štampanje je svakodnevna neminovnost za većinu Windows korisnika. Iako većina programa nema mogućnost štampanja, većina Windows aplikacija ima neku vrstu podrške za štampu. U ovom poglavlju ću obraditi osnove štampanja.

Obezbeđivanje mogućnosti štampanja u DOS aplikacijama je bilo prilično nezgodno. DOS programi su trebali da obezbede i instaliraju drajvere za štampače, za svaki tip štampača koji program podržava. Ovo je zadavalo mnogo muka proizvođačima softvera, naročito u malim firmama, odnosno kod programera shareware aplikacija. Windows operativni sistem je sve ovo promenio. U većini slučajeva, Windows se brine o radu sa različitim štampačima, drajverima za štampače itd. Sve što treba da uradite je da pošaljete zadatak za štampu na štampač, kao što ste poslali zadatak za prikazivanje slike prozoru. Do toga ćemo ubrzo doći.

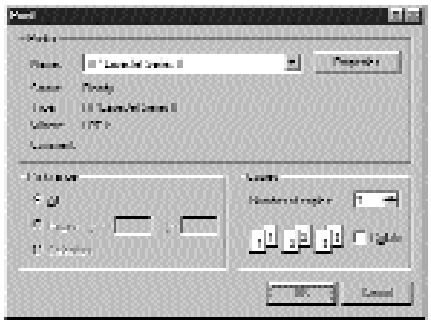
Štampanje u okviru Delphi-jevih aplikacija se može obaviti na nekoliko načina. Verovatno ćete odahnuti kada ovo budete naučili, pošto je u većini slučajeva štampanje već ugrađeno u VCL komponente i prilično je automatizovano. Ipak, u drugim slučajevima treba da obavite neku vrstu posebnog štampanja. Pre nego što naučite kako da i to obavite, obradićemo uobičajene okvire za dijalog koji se odnose na štampanje. Nakon toga ću obraditi različite načine na koje možete da štampate iz Delphi aplikacija.

Uobičajeni okviri za dijalog Print (štampanje)

Windows pruža uobičajene okvire za dijalog za štampanje (Print) i podešavanje štampača (Print Setup) koje možete koristiti u Vašim aplikacijama. Okvir za dijalog Print koristite pre nego što počnete sa štampanjem, dok okvir za dijalog Print Setup koristite za konfigurisanje štampača. Ipak, prvo što treba da uradite je dodavanje komponenti na Vašu formu.

Okvir za dijalog za štampanje (Print)

Već sam napomenuo da se okvir za dijalog Print prikazuje pre početka štampanja, obično kada korisnik odabere opciju File➥Print u okviru glavnog menija. Ukoliko korisnik klikne na dugme OK, računar počinje sa štampanjem; ukoliko korisnik klikne mišem na dugme Cancel, štampanje je otkazano. Slika 13.8 prikazuje okvir za dijalog Print u okviru Windows operativnog sistema u svojoj osnovnoj formi.



Slika 13.8
Okvir za dijalog Print u okviru Windows operativnog sistema

Nema sumnje da ovo nije prvi put da vidite okvir za dijalog koji se koristi za štampanje. Kombo okvir u gornjem delu okvira za dijalog Vam omogućava da odaberete određeni štampač na kom želite da štampate. Dugme Properties prikazuje okvir za dijalog na osnovu trenutno odabranog štampača i omogućava Vam da podesite položaj strane, rezoluciju i druge karakteristike koje su specifične za trenutno odabrani štampač. Odeljak Print Range (opseg štampanja) Vam omogućava da štampate sve strane, određeni broj strana, odnosno bilo koji objekat, ili tekst koji je trenutno odabran u okviru aplikacije. Odeljak Copies (kopije) Vam omogućava da definišete broj kopija koje želite da budu odštampane kao i opciju za izbor grupisanja kopija.

Okvir za dijalog Print je enkapsuliran u VCL komponenti `PrintDialog`. Kao što je to slučaj sa drugim uobičajenim okvirima za dijalog, okvir za dijalog Print prikazujete pozivom pripadajuće metode `Execute`. Ne bi trebalo da Vas razočara kada naučite šta sve operativni sistem Windows nosi u okviru za dijalog Print. Izbor štampača, broj kopija i grupisanje kopija su opcije kojima upravlja Windows operativni sistem, pa o tome ne treba da vodite brigu. U zavisnosti od Vaše aplikacije, možda ćete omogućiti korisniku da definiše određeni opseg stranica koje se mogu odštampati, odnosno da odštampa odabrani deo podataka u okviru Vaše aplikacije. Ukoliko obezbedite ovaj tip podrške, treba da istražite neke karakteristike komponente `PrintDialog`, pre nego što počnete sa štampanjem.

Komponenta PrintDialog ima samo jednu metodu: `Execute` i nema događaje. Kompletna funkcionalnost komponente `PrintDialog` je vezana za karakteristike koje su prikazane u tabeli 13.3.

Tabela 13.3: Karakteristike komponente `PrintDialog`

Karakteristika	Opis
`Collate`	Definiše grupisanje kopija. Ukoliko je vrednost postavljena na `True`, Windows će štampati kopije grupisano.
`Copies`	Određuje broj kopija koje će biti odštampane. Ovu karakteristiku možete podesiti pre poziva okvira za dijalog Print, ukoliko u Vašoj aplikaciji pos toji opcija za definisanje broja kopija. Windows vodi računa o tome da se štampa korektan broj kopija.
`FromPage`	Definiše početnu stranu, ukoliko je aktivirana opcija za štampanje određenog opsega strana. Aplikacije koje podržavaju štampanje određenog opsega strana treba da očitaju ovu karakteristiku, kako bi mogle da odrede koje će se strane štampati.
`MaxPage`	Definiše broj poslednje strane koji se upisuje u polje `To`, ukoliko je aktivirana opcija za štampanje određenog opsega strana. Okvir za dija log Print vodi računa o ispravnosti unosa podataka u polja `From` i `To` (od i do).
`MinPage`	Definiše broj prve strane koji se upisuje u polje `From`, kada se štampa određeni opseg strana.

nastavlja se

Tabela 13.3: Karakteristike komponente `PrintDialog` *nastavak*

Karakteristika	Opis
`Options`	Sadrži skup opcija koje definišu koje opcije će biti aktivirane u okviru za dijalog Print. Možete da odaberete mogućnost da okvir za dijalog ima dugme Help, da bude prikazana opcija Print to File (štampanje u datoteku), odnosno da aktivirate opcije za štampanje opsega strana.
`PrintRange`	Kontroliše koje je radio dugme za opciju Print Range odabrano, kada je okvir za dijalog Print inicijalno prikazan.
`PrintToFile`	Označava da li je korisnik odabrao opciju Print to File. Aplikacija određuje način zapisivanja u datoteku.
`ToPage`	Definiše poslednji broj strane kada se štampa određeni opseg strana. Aplikacije koje podržavaju štampanje određenog opsega strana, treba da očitaju ovu karakteristiku kako bi odredile koje strane treba štampati.

Aplikacija ne mora puno toga da uradi kako bi mogla da odgovori na zatvaranje okvira za dijalog Print, sve dok su opcije Print Range i Print to File neaktivirane. Na primer, ukoliko Vaša aplikacija omogućava štampanje određenog opsega strana, potrebno je pročitati karakteristike `FromPage` i `ToPage`, kako bi se odredilo koje strane treba štampati. U suprotnom, štampanje počinje kada korisnik pritisne dugme OK.

Okvir za dijalog za podešavanje štampača (Print Setup)

Okvir za dijalog Print Setup, prikazan na slici 13.9 se koristi kada korisnik želi da promeni štampač, veličinu papira, izvor papira, odnosno orijentaciju strane.

Slika 13.9
Okvir za dijalog Print Setup

Okvir za dijalog Print Setup nije neophodna opcija u većini aplikacija, pošto korisnik uvek može da pritisne dugme Properties u okviru za dijalog Print, kako bi promenio opcije za podešavanje štampača (pogledajte sliku 13.8). U drugu ruku, implementacija okvira za dijalog Print Setup je toliko jednostavna da ćete najverovatnije poželeti da je uključite u Vaše aplikacije. Koliko je to jednostavno? Pa, komponenta

PrinterSetup nema karakteristika, metoda i događaja koji se posebno odnose na ovu komponentu. Kao što je to slučaj sa komponentom PrintDialog, metoda Execute je jedina metoda koja će Vas interesovati. Da još pojednostavimo stvari, Windows upravlja svim stvarima umesto Vas. Ukoliko korisnik klikne mišem na dugme Cancel, Windows neće ništa uraditi. Ukoliko korisnik klikne na dugme OK, Windows će izvršiti određene promene, kako bi pripremio štampanje. Sve što treba da uradite je da prikažete okvir za dijalog Print Setup i zaboravite na njega. Karakteristični upravljač događajem za opciju menija File➥Printer Setup bi trebao da izgleda ovako:

```
procedure TMainForm.FilePrintSetupClick(Sender: TObject);
begin
  PrinterSetupDialog.Execute;
end;
```

To bi bilo sve što je potrebno da se uradi. Kao što sam napomenuo, implementacija okvira za dijalog Print Setup je toliko jednostavna da ćete poželeti da ga dodate u Vašu aplikaciju.

Štampanje na jednostavan način

Štampanje je zadatak koji zavisi od aplikacije. Možda ovo ne zvuči toliko ozbiljno, ali je tačno. U zavisnosti od tipa aplikacije koju razvijate, štampanje može biti toliko jednostavno da se koristi samo jedna linija koda, odnosno može uključiti na stotine linija koda. Prvo ću obraditi najjednostavnije forme štampanja, a zatim ću preći na teže operacije štampanja.

Metoda Print za forme

Klasa TForm poseduje metodu pod nazivom Print koja se može koristiti za štampanje sadržaja forme. Biće odštampana samo klijent oblast forme, a okvir forme i traka menija će biti izostavljeni. Iako ovaj metod radi slično kao jednostavno kopiranje ekrana, ograničeno je svojom implementacijom. Koristeći karakteristiku PrintScale, možete odabrati jednu od tri opcije štampanja. Tabela 13.4 prikazuje opcije za štampanje u razmeri i njihove opise.

Tabela 13.4: Opcije karakteristike PrintScale

Opcija	Opis
poNone	Nije primenjeno štampanje u razmeri. Izgled odštampane strane zavisi od tipa štampača.
poProportional	Ova opcija pokušava da odštampa formu u skoro istoj veličini kao što je forma prikazana na ekranu.
poPrintToFit	Ova opcija povećava, ili umanjuje veličinu slike, kako bi se uklopila u trenutno definisane opcije štampača.

Možete podesiti karakteristiku `PrintScale` u toku rada programa, odnosno u toku dizajniranja programa. Korišćenje `Print` metode je ograničeno na jednostavno kopiranje ekrana i nije dobro koristiti ovaj način za ozbiljnije štampanje.

Metoda `Print` za komponentu `RichEdit`

Komponenta `RichEdit` je veoma moćna, prvenstveno zbog posla koji obavlja odgovarajuća Windows memo kontrola. Štampanje korišćenjem komponente `RichEdit` se obavlja pozivanjem metode `Print`. Ova metoda preuzima samo jedan parametar pod nazivom `Caption` koji se koristi u okviru prozora Print Manager u toku prikazivanja posla za štampu (print job). Štampanje sadržaja komponente `RichEdit` je veoma jednostavno:

```
RichEdit.Print('MyApp.exe - readme.txt');
```

Na ovaj način će sama aplikacija voditi računa o štampanju. Prebacivanje reči i definisanje strana se automatski implementiraju. Ukoliko koristite edit kontrole sa više linija koje je potrebno štampati, komponenta `RichEdit` je način na koji možete obaviti štampanje.

Da biste odštampali tekst datoteku, možete koristiti Windows API funkciju ShellExecute. Između ostalog, funkcija ShellExecute se koristi za pokretanje programa na osnovu nastavka naziva datoteke. Na primer, generički Windows registruje da .txt nastavak pripada programu Windows Notepad. Ukoliko dva puta kliknete mišem na datoteku koja ima nastavak .txt u okviru Windows Explorer-a, operativni sistem će potražiti nastavak .txt u Registry bazi, videti da je program Notepad.exe registrovan da upravlja .txt datotekama i pokrenuće program Notepad. Datoteka na koju ste dva puta kliknuli mišem će automatski biti učitana.

Ovo ponašanje možete koristiti kao prednost. Pogledajte narednu liniju koda:

```
ShellExecute(Handle, 'print', 'readme.txt', nil, nil,
SW_HIDE);
```

Ovaj kod učitava Notepad, štampa datoteku pod nazivom `Readme.txt`, a zatim izlazi iz programa Notepad. U suštini, niste ni videli glavni prozor programa Notepad, pošto je aktiviran stil `SW_HIDE` koji je definisan u okviru parametra `Show`. Korišćenje ove tehnike podrazumeva da korisnik nije izmenio generičku vrednost u bazi Registry, koja se odnosi na nastavak naziva datoteke `.txt`, i da nije obrisao Notepad sa svog sistema. Ukoliko koristite `ShellExecute` funkciju, potrebno je da dodate junit `Shell Api` u Vašu listu uses.

Štampanje korišćenjem komponente `QuickReport`

Programi koji sadrži baze podataka mogu koristiti komponentu `QuickReport` za štampanje izveštaja. Ovu komponentu sam spomenuo zato što je očigledno da komponenta `QuickReport` pripada delu koji je vezan za štampanje, ali isto tako komponenta `QuickReport` je jedna od komponenti koja je povezana sa bazama

podataka. Diskusija o detaljima vezanim za implementaciju ove komponente će biti odložena do lekcije dana 18, "Kreiranje aplikacija sa bazama podataka".

Štampanje na teži način

Ne dozvolite da Vas naslov ovog poglavlja oneraspoloži. Štampanje nije toliko teško; potrebno je samo vreme i organizacija. Prvo ćemo pogledati korake koji su potrebni da se nauče kako bi korisnici mogli da štampaju u okviru Vaših aplikacija. Nakon toga ćemo preći na konkretan kod.

Šta je to kontekst uređaj?

Kontekst uređaji i klasa `TCanvas` su detaljnije bili obrađeni u jučerašnjoj lekciji, ali mala rekapitulacija neće škoditi. Kontekst uređaj (DC - device context) liči na ploču po kojoj Windows programi mogu crtati. Bolji naziv bi bio kanvas (canvas - platno). Na ovom kanvasu možete crtati tekst, linije, bitmape, pravougaonike, elipse itd. Tip linije koji se koristi prilikom crtanja na kontekst uređaju zavisi od pera koje je odabrano u okviru kontekst uređaja. Tekuća boja i dezen za popunjavanje su preuzeti od trenutno odabrane četke u okviru kontekst uređaja. Kontekt uređajem se mora pažljivo upravljati. Windows operativni sistem sadrži ograničen broj kontekst uređaja, stoga treba da budete pažljivi prilikom otpuštanja pojedinih kontekst uređaja; treba da ih oslobodite odmah nakon završetka rada. Takođe, ukoliko adekvatno ne obrišete objekte koje ste odabrali u okviru kontekst uređaja, Vaš program će rasipati memoriju i najverovatnije može dovesti operativni sistem u nestabilno stanje. Kao što ste pomislili, rad sa kontekst uređajima može biti komplikovan.

Dobra vest je da Vas VCL štiti od detaljnijeg upoznavanja kontekst uređaja. VCL enkapsulira Windows kontekst uređaje u klasi `TCanvas`. Karakteristika Canvas Vas oslobađa brige o sitnim detaljima koji mogu da Vas izlude, ukoliko radite sa Windows kontekst uređajima. VCL se brine o obezbeđivanju kontekst uređaja, izboru odgovarajućih objekata u okviru kontekst uređaja i oslobađanju kontekst uređaja, ukoliko više ne postoji potreba za njim. Sve što treba da uradite je da crtate po kanvasu i dopustite VCL-u da brine o ostalim stvarima.

Kakve to ima veze sa štampanjem? (Radoznali umovi bi želeli da znaju.) Pa, to izgleda ovako: Windows Vam omogućava da obezbedite kontekst uređaj štampača (printer device context) na koji možete da crtate tekst, grafiku, linije itd. Drugim rečima, na kanvas štampača možete crtati onako kako crtate i po kanvasu ekrana. Ovaj koncept predstavlja dobar prelaz na način štampanja koji se primenjivao u dobra stara DOS vremena. U ovom slučaju, na Windows operativnom sistemu je da Vas spase, omogućavajući Vam korišćenje kontekst uređaja štampača. VCL Vam dodatno pomaže enkapsulirajući kontekst uređaj u karakteristiku Canvas. Posledica ovakvog pristupa je štampanje koje je lako kao nikad do sada.

Klasa `TPrinter` i funkcija `Printer`

VCL Vam pomaže u operacijama štampanja obezbeđujući Vam klasu `TPrinter`. Ova klasa enkapsulira kompletno štampanje u okviru Windows opratívnog sistema. Klasa `TPrinter` sadrži karakteristiku `Canvas` koju možete koristiti za slanje linija, teksta, grafike i ostalih objekata koji se crtaju na štampač. Ne bih želeo da Vam ovo zvuči veoma jednostavno, ali sve što treba da uradite da biste štampali u okviru Vaših Delphi programa je dodavanje junita `Printer` u Vašu listu uses, a zatim pisanje koda koji bi otprilike izgledao ovako:

```
Printer.BeginDoc;
Printer.Canvas.TextOut(20, 20, 'Hello There!');
Printer.EndDoc;
```

U ovom kodu funkcija `Printer` je VCL funkcija. Funkcija `Printer` Vam omogućava da pristupite objektu `TPrinter` koji je podešen i spreman za rad. Sve što treba da uradite je da pustite štampač u rad.

Pogledajmo sada na kratko karakteristike i metode klase `TPrinter`. Tabela 13.5 prikazuje primarne karakteristike klase `TPrinter`, a tabela 13.6 prikazuje primarne metode klase `TPrinter`.

Tabela 13.5: Karakteristike klase `TPrinter`

Karakteristika	Opis
`Aborted`	Ova karakteristika je postavljena na True, ukoliko je započeto štampan je, a zatim pre završetka prekinuto.
`Canvas`	Mehanizam na osnovu kog možete crtati na štampač (kontekst uređaj štampača).
`Capabilities`	Trenutna postavka drajvera za štampač.
`Copies`	Broj kopija koje će biti odštampane.
`Fonts`	Lista fontova koju podržava trenutno odabrani štampač.
`Handle`	Upravljač kontekst uređajem za štampač (`HDC`). Ovu karakteristiku možete koristiti kada treba da pozovete Windows API funkciju, kada se zahteva upravljanje kontekst uređajem.
`Orientation`	Orijentacija papira u štampaču (`poPortrait`, ili `poLandscape`). Automatski se postavlja kada korisnik odabere štampač, ili izmeni postavku štampača, mada isto tako možete ori jentaciju papira promeniti i ručno.
`PageHeight`	Visina tekuće strane štampača u pikselima. Ova vrednost se menja u zav isnosti od štampača. Dodatno, ova karakteristika može da sadrži drugačiju vrednost vezanu za orijentaciju papira u štampaču. Neki štampači mogu štampati u nekoliko rezolucija, što dodatno menja vred nost ove karakteristike.

`PageNumber`	Broj stranice koja se trenutno štampa. Ova karakteristika se povećava svaki put kada pozovete metodu `NewPage`, kako biste počeli štampanje nove strane.
`PageWidth`	Širina strane u pikselima. Kao i kod karakteristike `PageHeight`, ova vrednost se menja u zavisnosti og rezolucije štampača, orijentacije papira i dimenzija papira.
`PrinterIndex`	Indeksna vrednost trenutno odabranog štampača na listi dostupnih štampača. Ukoliko želite da odaberete generički štampač, upišite vrednost `-1`.
`Printers`	Lista dostupnih štampača u okviru sistema.
`Printing`	Ova karakteristika ima vrednost `True`, ukoliko štampač trenutno radi.
`Title`	Tekst koji identifikuje posao štampanja (print job) u prozoru Print Manager.

Tabela 13.6: Metode klase `TPrinter`

Metoda	Opis
`Abort`	Koristi se da prekine štampanje pre normalnog završetka rada.
`BeginDoc`	Počinje sa procesom štampanja. Postavlja parametre štampača, koristeći Windows operativni sistem za pripremu štampe.
`EndDoc`	Završava proces štampanja. Primorava štampač da odštampa tekuću stranu, a zatim izvršava spremanje nakon štampanja koristeći Windows operativni sistem.
`GetPrinter`	Prihvata trenutno aktivni štampač. Umesto ove metode možete koristiti karakteristiku `Printers` (bolje je koristiti karakteristiku `Printers` za pristup štampačima, pošto je možete iskoristiti i za pristup i za postavljanje parametara trenutno aktivnog štampača).
`NewPage`	Koristi se da primora štampač da odštampa tekuću stranu, a zatim počne sa štampanjem nove. Inkrementira karakteristiku `PageNumber`.
`SetPrinter`	Postavlja parametre trenutno aktivnog štampača. Umesto ove metode možete koristiti karakteristiku `Printers`.

Klasa `TPrinter` nema interfejs u toku dizajniranja programa. Sve se postiže u toku rada programa.

Puštanje u rad

Vreme je da uposlite Vaše novostečeno znanje. Vreme je da još jednom skinete prašinu sa programa `ScratchPad` i na trenutak ga upregnete. Napokon, kakva je korist od editora teksta koji ne može da štampa?

Prvo treba da delimično izmenite glavnu formu. Već imate podešene opcije menija za štampanje i podešavanje štampača, ali je potrebno da ih aktivirate i dodate okvire za dijalog Print i Printer Setup na formu. Krenimo:

1. Dva puta kliknite mišem na komponentu `MainMenu`, kako bi se pojavio dizajner menija.
2. Odaberite opciju File➥Print u okviru menija programa `ScratchPad` koji se nalazi u prozoru dizajnera menija. Promenite karakteristiku `Enabled` u `True`.
3. Učinite isto sa opcijom menija File➥Print Setup. Zatvorite dizajner menija.
4. Postavite komponentu `PrintDialog` na formu i promenite karakteristiku Name komponente u `PrintDialog`.
5. Postavite komponentu `PrinterSetupDialog` na formu i promenite karakteristiku `Name` komponente u `PrinterSetupDialog`.

Uredu; nakon kompletiranja forme, vreme je da pređemo na izmenu koda. Za početak treba da dodate nekoliko opcija u dekleraciju glavne forme. Izvršite sledeće korake:

1. Pređite na editor koda i dodajte junit `Printers` na listu `uses` u okviru glavne forme.
2. Pronađite dekleraciju klase `TMainForm` u odeljku interface. Dodajte sledeću liniju u odeljak private u okviru dekleracije klasa:

   ```
   procedure PrintFooter(var R : TRect; LineHeight : Integer);
   ```

   Ovo je dekleracija za metodu koja štampa podnožje (footer) na dnu svake strane.
3. Pritisnite tastere Ctrl+Shift+C, kako bi opcija Delphi-ja Class Completion (kompletiranje klasa) kreirala proceduru PrintFooter u odeljku `implementation`. Kod ćete upisati kasnije.
4. Pređite na dizajner forme i odaberite opciju File➥Print u okviru glavnog menija forme. Biće prikazana metoda `FilePrintClick`. Za sada će ova metoda ostati prazna.
5. Odaberite opciju File➥Print Setup u okviru glavnog menija. U okviru linije na kojoj se nalazi kursor upišite kod, tako da kompletna metoda `FilePrintSetupClick` izgleda ovako:

   ```
   procedure TMainForm.FilePrintSetupClick(Sender: TObject);
   begin
     PrinterSetupDialog.Execute;
   end;
   ```

Uredu, sada ste spremni da popunite metode `FilePrintClick` i `PrintFooter`. Listing 13.3 prikazuje metodu `FilePrintClick`. Kod u ovom metodu možete uneti ručno, odnosno možete učitati projekat `ScratchPad` koji se nalazi u okviru koda

knjige i istražiti ga u Delphi-jevom okruženju. Listing 13.4 prikazuje metodu `PrinterFooter`. Unesite kod u ove metode koje se nalaze u okviru Vaše datoteke `SPMain.pas`. Ne morate da upisujete linije sa komentarima, što se podrazumeva.

Listing 13.3: Metoda `FilePrintClick`

```
procedure TMainForm.FilePrintClick(Sender: TObject);
var
  I            : Integer;
  LineHeight   : Integer;
  LinesPerPage : Integer;
  LineCount    : Integer;
  R            : TRect;
  S            : string;
begin
  { Display the Print dialog. }
  if PrintDialog.Execute then begin

    { Set the title for the printer object. }
    Printer.Title := 'ScratchPad - ' + OpenDialog.FileName;

    { Set the printer font to the same font as the memo. }
    Printer.Canvas.Font := Memo.Font;

    { Determine the line height. Take the Size of the
    { font and and use the MulDiv function to calculate
    { the line height taking into account the current
    { printer resolution. Use the Abs function to get
    { the absolute value because the result could be a
    { negative number. After that add 40% for leading. }
    LineHeight := Abs(
      MulDiv(Printer.Canvas.Font.Size,
      GetDeviceCaps(Printer.Handle, LOGPIXELSY), 72));
    Inc(LineHeight, (LineHeight * 4) div 10);

    { Determine how many lines will fit on a page. Trim
    { it back by three lines to leave some bottom margin. }
    LinesPerPage := (Printer.PageHeight div lineHeight) - 4;

    { Start printing on line 4 rather than line 0 to leave
    { room for the header and to allow for some top margin. }
    LineCount := 4;

    { Tell Windows we're starting and print the header. }
    Printer.BeginDoc;
    R.Top    := LineHeight;
    R.Left   := 20;
    R.Right  := Printer.PageWidth;
    R.Bottom := LineHeight * 2;
    DrawText(Printer.Handle,
      PChar(OpenDialog.FileName), -1, R, DT_CENTER);
```

nastavlja se

Listing 13.3: Metoda `FilePrintClick` *nastavak*

```
    { Loop through all of the lines and print each one. }
    for I := 0 to Pred(Memo.Lines.Count) do begin

      { When we get to the bottom of the page reset the
      { line counter, eject the page, and start a new page. }
      Inc(LineCount);
      if LineCount = LinesPerPage then begin
        PrintFooter(R, LineHeight);
        LineCount := 4;
        Printer.NewPage;
      end;

      { Get the next string and print it using TextOut }
      S := Memo.Lines.Strings[I];
      Printer.Canvas.TextOut(0, LineCount * LineHeight, S);
    end;

    { All done. }
    PrintFooter(R, LineHeight);
    Printer.EndDoc;
  end;
end;
```

Listing 13.4: Metoda `PrintFooter`

```
procedure TMainForm.PrintFooter(var R: TRect; LineHeight: Integer);
var
  S : String;
begin
  with Printer do begin

    { Build a string to display the page number. }
    S := Format('Page %d', [PageNumber]);

    { Set up the rectangle where the footer will be drawn. }
    { Find the bottom of the page and come up a couple of
    { lines. }
    R.Top := PageHeight - (lineHeight * 2);
    R.Bottom := R.Top + lineHeight;

    { Display the text using DrawText so we can center the
    { text with no fuss. }
    DrawText(Handle, PChar(S), -1, R, DT_CENTER);

    { Draw a line across the page just above the 'Page X' text. }
    Canvas.MoveTo(0, R.Top - 2);
    Canvas.LineTo(R.Right, R.Top - 2);
  end;
end;
```

Ovaj kod ilustruje kako možete štampati direktno kroz Windows operativni sistem, umesto da se oslanjate na ugrađene mogućnosti štampanja koje pruža VCL. Iako uvek želim da uradim bilo šta na jednostavniji način, postoje trenuci kada jenostavni načini nisu dovoljno fleksibilni. U takvim slučajevima je dobro da posedujete znanje koje će posao obaviti bez problema.

Štampanje bitmape

Štampanje bitmape je jednostavno. Jedino je potrebno da kreirate slučaj klase `TBitmap`, učitate bitmapu u objekat bitmape i pošaljete je na štampač, korišćenjem metode Draw klase `TCanvas`. Sledi kompletan kod:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Bitmap : TBitmap;
begin
  Bitmap := TBitmap.Create;
  Bitmap.LoadFromFile('test.bmp');
  Printer.BeginDoc;
  Printer.Canvas.Draw(20, 20, Bitmap);
  Printer.EndDoc;
  Bitmap.Free;
end;
```

Kada štampate bitmapu, budite oprezni, pošto bitmapa može da ispadne veoma mala, što zavisi od rezolucije štampača. Da bi lepo izgledala, bitmapa mora biti raširena. Ukoliko imate potrebu za širenjem bitmape, treba da koristite metodu `StretchDraw`, umesto metode `Draw`.

Korišćenje kursora

Korišćenje kursora nije teško, ali ću ovde objasniti određene elemente, kako bi Vam dao osnove za shvatanje načina rada kursora. Ovo poglavlje radi sa kursorima koje možete menjati u toku rada programa. (Da biste promenili kursor u toku dizajniranja, odaberite novu vrednost za karakteristiku Cursor željene komponente.) Nakon pregleda osnovnih elemenata vezanih za kursor, objasniću kako da učitate stek kursore i korisnički definisane kursore.

Osnovni pojmovi o kursorima

Prvo, možete promeniti kursor za određenu komponentu, odnosno formu koji se odnosi na kompletnu klijent oblast Vaše aplikacije. Ukoliko želite da promenite kursor za kompletnu aplikaciju, potrebno je da promenite karakteristiku `Cursor` objekta `Screen`. Objekat `Screen` predstavlja ekran Vaše aplikacije. Promenom karakteristike `Cursor` za objekat `Screen` osiguravate da kursor ostane isti, bez obzira na komponentu iznad koje se kursor nalazi. Na primer, pretpostavimo da

želite da promenite kursor u oblik peščanog sata. Ukoliko promenite karakteristiku `Cursor` samo za formu, kursor će dobiti oblik peščanog sata, kada se bude nalazio nad formom, ali se nikad neće vratiti na generički kursor, ukoliko se nađe iznad bilo koje komponente u okviru forme. Promena karakteristike `Cursor` za objekat `Screen` Vam daje isti pristup promeni kursora.

Administriranje kursorom spada u nadležnost objekta `Screen`. Svi kursori koji su dostupni za korišćenje se nalaze u karakteristici `Cursors` objekta `Screen`. Uočite da naziv ove karaktertistike: `Cursors` nije isti kao naziv karakteristike `Cursor` koja je objašnjenja u prethodnom delu ovog poglavlja. Karakteristika `Cursors` je niz koji sadrži listu kursora dostupnih aplikaciji, a karakteristika `Cursor` je karakteristika koja se koristi za prikazivanje određenog kursora. Iako Vas ovo u početku može zbuniti, shvatićete veoma brzo. Sve što će Vam biti potrebno je malo iskustva u radu sa kursorima.

Windows obezbeđuje nekoliko ugrađenih kursora koje možete koristiti u Vašim aplikacijama. Dodatno, VCL obezbeđuje sam za sebe nekoliko kursora koje takođe možete koristiti u Vašim aplikacijama. Zajedno, ovi kursori se nazivaju stek kursori (*stock cursors*). Svaki stek kursor ima pridodatu konstantu sa nazivom. Na primer, kursor sa strelicom ima naziv `crArrow`, kursor u obliku peščanog sata ima naziv crHourGlass, a kursor za prevlačenje ima naziv crDrag. Svi ovi kursori se nalaze u nizu pod nazivom Cursors na pozicijama od -17 do 0. Generički kursor se nalazi u nizu na poziciji 0 (`crDefault`), nekursor je -1 (`crNone`), kursor strelica je -2 (crArrow), itd. Sistem za pomoć u toku rada za karakteristiku `Cursors` će prikazati sve kursore zajedno sa nazivima konstanti, pa možete pokrenuti Delphi-jev sistem za pomoć kako biste videli kompletan spisak dostupnih kursora.

Da biste koristili jedan od kursora u okviru niza `Cursors`, dodelite naziv kursora koji želite na koristite karakteristici `Cursor` objekta `Screen`:

```
Screen.Cursor := crHourGlass;
```

Od ovog trenutka VCL magija stupa na scenu i ispravan kursor će biti učitan i prikazan. Korišćenje karakteristike `Cursors` je za Vas transparentno, pošto ne pristupate direktno karakteristici `Cursors` u trenutku kada koristite kursor. Umesto toga, dodeljujete kursor karakteristici `Cursor`, a VCL brine o traženju odgovarajućeg kursora u okviru niza `Cursors` i prikazuje ga. Karakteristika Cursor komponente i karakteristika `Cursors` objekta `Screen` rade zajedno na prikazivanju različitih tipova kursora u okviru Vaše aplikacije.

U Vašim aplikacijama možete menjati kursore iz bilo kog razloga (kursor u obliku peščanog sata), ukoliko imate program za crtanje koji koristi posebne kursore, odnosno za implementiranje kursora za pomoć u okviru Vaše aplikacije.

Učitavanje i korišćenje stek kursora

Windows obezbeđuje nekoliko već pripremljenih kursora koje možete koristiti u okviru Vaših aplikacija. Kao dodatak, VCL poseduje još nekoliko korsora koje možete odabrati. Ove stek kursore možete koristiti kad god to poželite.

Jedna od najočiglednijih situacija za promenu kursora je proces koji veoma dugo traje, a koji Vaša aplikacija mora da izvrši. Smatra se kao loša praksa, ukoliko program uposlite, a da korisniku ne date znak da je program zauzet. Za ovakve slučajeve Windows obezbeđuje kursor u obliku peščanog sata (u okviru Windows operativnog sistema naziva se kursor za čekanje). Na primer, pretpostavimo da imate petlju koja radi u Vašoj aplikaciji i koja će najverovatnije dugo obavljati svoj posao. U tom slučaju biste mogli da uradite nešto što liči na naredni kod:

```
procedure TMainForm.DoItClick(Sender: TObject);
var
  I, Iterations : Integer;
  OldCursor     : TCursor;
begin
  Iterations := StrToInt(Time.Text);
  OldCursor := Screen.Cursor;
  Screen.Cursor := crHourGlass;
  for I := 0 to Iterations do begin
    Application.ProcessMessages;
    Time.Text := IntToStr(I);
  end;
  Screen.Cursor := OldCursor;
end;
```

Pošto ne znate koji kursor u datom trenutku koristi Vaša aplikacija, kao dobra ideja se nameće snimanje trenutne vrednosti kursora, pre nego što izmenite karakteristiku `Cursor`. Nakon što završite rad sa posebno definisanim kursorom, možete vratiti stari kursor.

Ponekad ćete promeniti karakteristiku Cursor pre procesa koji bi trebao dugo da traje, a da se ništa ne dogodi. Razlog za ovo leži u činjenici da Windows nema šansu da promeni Vaš kursor, pre nego što program uđe u petlju. Kada se nađe u petlji, Vaša aplikacija ne može da obrađuje nikakve poruke, uključujući tu i poruke za promenu kursora. Ovo možete ispraviti slanjem Windows poruke za čekanje u toku prolaska kroz petlju programa:

```
Application.ProcessMessages ;
```

Sada Vaša aplikacija omogućava protok poruka i kursor se menja prilikom ulaska u petlju.

Naravno, možete promeniti izgled kursora za pojedinu komponentu. Na primer, program za crtanje može promeniti klijent oblast kursora u zavisnosti od trenutnog alata za crtanje. U tom slučaju ne morate da menjate kursor za objekat `Screen`, pošto želite da se kursor strelica pojavljuje kada se kursor pomera iznad trake sa alatima, statusne trake, menija, odnosno bilo koje druge komponente koja se nalazi na

formi. U ovom slučaju, treba da podesite kursor samo za komponentu koja predstavlja klijent oblast Vaše aplikacije:

```
PaintBox.Cursor := crCross;
```

Sada je kursor promenjen samo za jednu komponentu, dok ostale komponente koriste njihove prethodno definisane kursore.

Učitavanje i korišćenje korisnički definisanih kursora

Učitavanje korisnički definisanih kursora zahteva nešto više rada. Već sam napomenuo ranije, da karakteristika `Cursors` klase `TScreen` sadrži listu kursora koja je dostupna u okviru Vaše aplikacije. Korišćenje korisnički definisanih kursora zahteva obavljanje sledećih koraka:

1. Kreirajte kursor u editoru slika, ili bilo kom drugom resursnom editoru i snimite ga kao `.res` datoteku.
2. Povežite resursnu datoteku sa Vašim programom, koristeći direktivu prevodioca `$R`.
3. Učitajte kursor u niz `Cursors`, koristeći funkciju `LoadCursor`.
4. Implementirajte kursor u Vaš program, dodeljujući indeksni broj karakteristici `Cursor` u okviru forme, karakteristici `Cursor` za objekat `Screen`, ili karakteristici `Cursor` u okviru neke druge komponente.

Prva dva koraka su obrađena u lekciji dana 11, kada sam obradio editor slika i u lekciji dana 8, kada sam obradio resurse. Nakon što ste prikačili kursor na izvršnu datoteku (`.exe`), možete ga učitati korišćenjem funkcije `LoadCursor`. Učitavanje kursora u niz `Cursors` je jednostavno:

```
Screen.Cursors[1] := LoadCursor(HInstance, 'MYCURSOR');
```

Ovaj kod podrazumeva da treba da obezbedite kursor pod nazivom `MYCURSOR` i da dodelite kursor poziciji broj 1 u okviru liste kursora (zapamtite, pozicije od -17 do 0 se koriste za stek kursore). Učitavanje kursora se vrši samo jednom, pa ćete to verovatno raditi u Vašoj glavnoj formi, koristeći upravljač događajem `OnCreate`.

Svi kursori će biti učitani u karakteristiku `Cursors` objekta `Screen`, bez obzira da li koristite kursor sa objektom `Screen`, formom, ili sa komponentom. Postoji samo jedan niz `Cursors` i on pripada objektu Screen.

Da biste nakon ove operacije koristili kursor, treba da dodelite vrednost indeksa kursora karakteristici `Cursor` u okviru objekta `Screen`, odnosno da to uradite u okviru bilo koje kompomnente:

```
PaintBox.Cursor := 1;
```

Ukoliko imate nekoliko kursora možete poželeti da kreirate konstante za svaki kursor, tako da mu dodelite naziv koji ima smisla i koji ćete koristiti umesto celobrojih vrednosti koje je lako pomešati. Koristeći ovaj metod prethodni kod bi izgledao ovako:

```
const
  MyCursor = 1;

{ later... }
Screen.Cursors[MyCursor] := LoadCursor(HInstance, 'MYCURSOR');

{ later still... }
PaintBox.Cursor := MyCursor;
```

Kao što ste mogli da vidite učitavanje i inplementiranje korisnički definisanih kursora nije tako teško, kada znate kako ovu operaciju možete obaviti. Kod u okviru knjige za današnju lekciju uključuje i program primer pod nazivom `CursTest` koji demonstrira teorijski deo koji je obrađen u ovom poglavlju.

Zaključak

U današnjoj lekciji ste naučili nešto o mogućnostima koje daju dodatni kvalitet Windows aplikacijama i način na koji možete da implementirate ove mogućnosti u Vaše programe. Moram da Vas upozorim da je veliko iskušenje ne preterati sa dodacima kao što su kontrolne trake, statusne trake i kursori. Ubacite bilo kakvu dekoraciju koja je potrebna Vašoj aplikaciji, ali nemojte preterivati. U današnjoj lekciji ste mogli da naučite i o štampanju. U nekim slučajevima, podrška štampanju je ugrađena u pojedine komponente, pa je u takvim slučajevima štampanje neverovatno jednostavno. U drugim slučajevima potrebno je da zasučete rukave i prionete na posao, koristeći klasu `TPrinter`. Čak i u ovom slučaju, štampanje je posao koga ne treba da se bojite.

Radionica

Radionica sadrži kviz pitanja koja Vam pomažu da učvrstite razumevanje obrađenog materijala, kao i vežbe koje Vam omogućavaju da steknete iskustvo u korišćenju gradiva koje ste naučili. Odgovore na kviz pitanja možete pronaći u Dodatku A, "Odgovori na kviz pitanja".

Pitanja i odgovori

P **Da li mogu istovremeno da deaktiviram sve komponente u okviru svoje trake sa alatima?**

O Da. Postavite karakteristiku `Enabled` trake sa alatima na `False`.

P **Primetio sam da klase `TCoolBar` i `TControlBar` rade potpuno iste stvari. Koju od njih bih trebao da koristim?**

O Moj predlog bi bio da koristite klasu `TControlBar`. Postoji veliki broj verzija Microsoft klasične kontrolne biblioteke `COMCTL32.DLL`. Klasa TCoolBar je vezana za Microsoft-ovu klasičnu kontrolu trake kontejner, i kao takva je zavisna od verzije datoteke `COMCTL32.DLL` koju je korisnik instalirao na svom računaru. Klasa `TControlBar` ne zavisi od datoteke `COMCTL32.DLL`, pa se ponašanje komponente predstavljene ovom klasom neće menjati od sistema do sistema.

P **Kako da postavim bitmapu na svoju statusnu traku?**

O Kreirajte statusnu traku sa više panoa. Promenite stil panoa, tako da sadrži bitmapu, koristeći vrednost `psOwnerDraw`. Zatim, u upravljaču događajem `OnDrawPanel` iskoristite metodu `Draw` klase `TCanvas`, kako bi nacrtali željenu bitmapu na odabranom panou.

P **Zašto bih trebao da se mučim sa aktiviranjem komandi za opcije menija i dugmad trake sa alatima?**

O Zato što korisnik očekuje konzistentan interfejs. Kada određene opcije nisu dostupne (bile one meniji, ili trake sa alatima), treba da budu označene sivom bojom. Na ovaj način dajete korisniku vizuelni nagoveštaj da su ove komande ponekad dostupe, ali da trenutno nisu aktivne.

P **Dugo koristim Delphi. Imam sistem za aktiviranje komandi koji već radi. Zašto bih trebao da se prebacim na aktivnosti i listu aktivnosti?**

O Aktivnosti Vam daju centralno mesto sa kog možete da obavljate aktiviranje svih Vaših komandi. Umesto da Vaš kod za aktiviranje komandi bude razbacan po celom programu, sada ga možete imati na jednom mestu.

P **Želeo bih da, u okviru svoje aplikacije, napravim osnovni sistem za prosleđivanje edit kontrole koja sadrži više linija. Koji je najjednostavniji način?**

O Najjednostavniji način je korišćenje komponente `RichEdit` i korišćenje metode `Print` da bi štampali sadržaj komponente.

P **Primetio sam da postoji karakteristika `Handle` za objekt `Printer`, a isto tako i karakteristika `Handle` za karakteristiku `Canvas` objekta `Printer`. Koja je razlika između ove dve karakteristike?**

O U ovom slučaju nema razlike. Ukoliko pozovete Windows API funkciju koja zahteva upravljač za kontekst uređaj štampača, možete koristiti, ili komandu `Printer.Handle`, ili komandu `Printer.Canvas.Handle`.

P **Kada promenim kursor za glavnu formu, izgled kursora je dobar kada se kursor nalazi na formi, ali se menja u kursor strelicu kada se nalazi na nekom dugmetu. Zašto?**

O Potrebno je da promenite kursor za objekt `Screen`, a ne za formu. Promena kursora za objekt `Screen` obezbeđuje da će novi kursor biti korišćen, bez obzira na kom delu Vaše aplikacije se kursor nalazi.

Kviz

1. Kako možete prikačiti upravljač događajem na događaj `OnClick` dugmeta trake sa alatima?
2. Da li možete postaviti kontrole na traku sa alatima, a da to ne budu dugmad?
3. Koji je naziv događaja u okviru klase `TActionList` na koji odgovarate prilikom aktiviranja komande?
4. Šta karakteristika `SimplePanel` komponente `StatusBar` radi?
5. Kako možete da manuelno izmenite tekst statusne trake?
6. Kako možete da aktivirate i deaktivirate opcije menija i dugmad?
7. Kako možete pristupiti štampaču u okviru Delphi aplikacije?
8. Koju metodu pozivate kada počinjete štampanje koristeći klasu `TPrinter`?
9. Koju metodu klase `TPrinter` pozivate kada želite da počnete štampanje nove strane?
10. Kako možete da, u toku rada programa, promenite izgled kursora za određenu komponentu?

Vežbe

1. Napišite program iz početka. Dodajte traku sa alatima i postavite pet dugmadi na traku. Sada dodajte statusnu traku. Aktivirajte savete tako da dugmad trake sa alatima imaju i oblačiće za savete i tekst saveta u okviru statusne trake.
2. Promenite statusnu traku programa `ScratchPad` i dodajte drugi pano. Na ovom panou prikažite da li je datoteka snimljena (Saved), ili menjana (Modified) u zavisnosti od vrednosti karakteristike `Modified` komponente Memo.
3. Promenite okvir za opis programa `ScratchPad` tako da piše: Version `1.05`. Takođe promenite karakteristiku `Title` u okviru za dijalog opcije projekta i karakteristiku `Caption` glavne forme. Napokon, dodali ste nove opcije, pa to korisnici treba i da znaju!

4. Kreirajte kursor po želji u okviru editora slika. Napišite program koji prikazuje kursor, kada je dugme na glavnoj formi pritisnuto.

5. **Dodatni zadatak**: Izmenite program `ScratchPad` iz treće vežbe, tako da prikazuje različite bitmape na statusnoj traci, u zavisnosti od toga da li je trenutna datoteka snimljena. (Savet: Pogledajte bitmape `led1on.bmp` i `led1off.bmp` u okviru direktorijuma `Borland Shared Files\Images\Buttons`.)

6. **Dodatni zadatak**: Izmenite program `ScratchPad` tako da korisnik može definisati gornju marginu, donju marginu, desnu marginu i levu marginu za štampanje teksta.

Dan 14

Napredno programiranje

Danas ćete raditi sa nekim naprednijim aspektima Windows programiranja u Delphi-ju. Konkretno, naučićete nešto o:

- Pravljenju sistema za kontekstno osetljivu pomoć,
- Obradi izuzetaka,
- Korišćenju Windows-ovog Registry-ja,
- Posebnoj obradi poruka.

Danas imate dosta posla pa je najbolje da krenemo odmah.

Pravljenje kontekstno osetljive pomoći

Ne tako davno, mogli ste uspešno isporučiti program koji u sebi ne sadrži sistem za kontekstno osetljivu pomoć. U stvari, za male programe i nije moguće napraviti - sistem za pomoć. Ipak, danas to ne bih mogao da preporučim za programe koji će se isporučivati na tržište. Korisnici su izuzetno zahtevni po pitanju mogućnosti programa. Sistemi za pomoć više nisu dodatna pogodnost, oni su postali obavezan deo svakog ozbiljnog programa.

Kontekstno osetljiva pomoć znači da kada korisnik pritisne taster F1, otvara se određena stranica u sistemu za pomoć i to u zavisnosti od trenutnog konteksta programa. Pogledajte, na primer, Delphi-jev IDE. Recimo, da na ekranu imate otvoren Project Options dijalog-prozor i da je aktivna Application stranica. Kada pritisnete F1 (ili taster

Help na dijalog-prozoru), pokreće se WinHelp i prikazuje se stranica u okviru sistema za pomoć, koja odgovara Application stranici Project Options dijalog-prozora.

Slično, kada pritisnete taster F1 u trenutku kada se Object Repository Options dijalog-prozor nalazi na ekranu, dobićete pomoć za taj dijalog-prozor. Sistem za konktekstno osetljivu pomoć takođe radi i sa stavkama menija. Ukoliko se postavite na određenu stavku menija i pritisnete F1, prikazaće se stranica sistema za pomoć koja se odnosi na tu stavku menija.

Da biste, u svojim programima, realizovali sistem za kontekstno osetljivu pomoć, morate uraditi sledeće:

1. Napravite fajl sa tekstom pomoći (engl. Help File) za svoj program.
2. Dodelite ime tog fajla `HelpFile` osobini `Application` objekta.
3. Postavite `HelpContext` osobinu svake forme i komponente za koju ste napravili stranicu za pomoć.
4. Napravite stavke Help menija, tako da Vaši korisnici mogu pristupiti sistemu za pomoć kroz meni.

Hajde da sada proučimo svaki od ovih koraka.

Pravljenje fajla sa tekstom pomoći

Pravljenje fajlova sa tekstom pomoći je zamorno. Ne mogu da kažem da mrzim to da radim, ali to nije jedan od poslova kojima se radujem. Ukoliko imate sreće, u Vašoj firmi postoji odeljenje za dokumentaciju koje će napraviti fajl sa tekstom pomoći za Vas.

Čak i neki programeri koji su imaju mali budžet za razvoj, imaju odeljenja za dokumentaciju koja za njih prave fajlove sa tekstom pomoći. Ovi programeri se često obraćaju piscima dokumentacije sa "Dragi", ili "Srce". Na primer, programer bi mogao da kaže sledeće piscu dokumentacije: "Dragi, da li bi mogao da staviš decu u krevet kada završiš sa tom stranicom?"

Bez obzira na to ko pravi fajl sa tekstom pomoći, mora postojati određena koordinacija između autora fajla sa tekstom pomoći i programera. Identifikatori konteksta, koji se nalaze u fajlu sa tekstom pomoći, moraju se poklopiti sa onima koji su navedeni u komponentama samog programa. Iako ovo nije težak posao, koordinacija je neophodna, kako bi svi znali o kojoj se stranici tačno radi.

Windows-ov fajl sa tekstom pomoći se može napraviti iz više zasebnih fajlova. Izvorni fajl Windows-ovog fajla sa tekstom pomoći se naziva tematski fajl (*topic file*). Tematski fajl je fajl u rich text formatu (`.rtf`) koji se sastoji od specijalnih kodova koji su razumljivi prevodiocu fajla sa tekstom pomoći. Ukoliko Vaš sistem za pomoć sadrži i grafiku, imaćete i jedan, ili više *grafičkih fajlova*. Grafika se u sistemu za pomoć može nalaziti u obliku bitmape (`.bmp`), ili Windows metafile-a (`.wmf`), kao i u nekim specijalnim tipovima grafičkih fajlova.

Konačno, morate imati i *projektni fajl* sistema za pomoć (`.hpj`). Projektni fajl sadrži uputstva prevodiocu pomoću kojih on spaja tematski fajl, grafičke fajlove i sve ostale fajlove koji su neophodni za dobijanje završnog fajla za pomoć. Ovaj fajl, takođe, sadrži i `[MAP]` odeljak u kome se nalaze identifikatori konteksta određenih tema u okviru fajla za pomoć. Kada napravite projektni fajl, potrebno ga je prevesti uz pomoć prevodioca za pomoć kao što je Microsoft Help Workshop (možete ga naći u `\Delphi 4\Help\`Tools direktorijumu). Help Workshop prevodi projektni fajl i proizvodi konačni fajl za pomoć (`.hlp`).

Iako možete napraviti fajl sa tekstom pomoći pomoću bilo kog programa za obradu teksta koji podržava rad sa `.rtf` fajlovima, preporučujem kupovinu posebnog programa za rad sa fajlovima za pomoć. Dobar program za rad sa fajlovima za pomoć Vam može sačuvati nerve. Neki od komercijalnih programa za rad sa fajlovima za pomoć su i ForeHelp kompanije Fore Front, Inc. (`http://www.ff.com`) i RoboHelp kompanije Blue Sky Software (`http://www.blue-sky.com`). Koristio sam ForeHelp i uglavnom mi se sviđa način na koji radi. Postoje, takođe, i shareware programi za rad sa fajlovima za pomoć. Jedan od njih je HelpScribble. Možete preuzeti ovaj program sa `http://www.ping.be/jg/`.

Identifikatori konteksta i `HelpContext` osobina

Bez obzira na način na koji pravite fajl za pomoć, morate imati broj konteksta koji će biti povezan sa svim glavnim temama u fajlu za pomoć. *Broj konteksta* koristi Windows-ov sistem za pomoć, `WinHelp32.exe`, da bi prikazao određenu stranicu iz fajla za pomoć.

Na primer, recimo da imate Options dijalog-prozor u svom programu. Kada korisnik pritisne F1, dok je ovaj dijalog-prozor aktivan, Vaš program prenosi identifikator konteksta tog dijalog-prozora WinHelp-u. WinHelp se pokreće i prikazuje stranicu iz fajla za pomoć koja objašnjava Options dijalog-prozor. Identifikator konteksta nije neophodan za svaku stranicu u fajlu za pomoć, ali biste trebali da imate identifikatore konteksta za svaku temu, dijalog-prozor i druge važnije komponente Vašeg programa.

Najveći broj komponenti (forme, stavke menija i kontrole) imaju osobinu koja se zove `HelpContext`. Ova osobina sadrži identifikator konteksta koji će biti prenesen WinHelp-u, ukoliko korisnik pritisne taster F1 u trenutku kada je ta komponenta aktivna. Podrazumevana vrednost `HelpContext` osobine je `0`. Ukoliko je vrednost `HelpContext` osobine `0` za neku komponentu, komponenta nasleđuje vrednost `HelpContext` osobine od svog roditeljskog prozora. Ovo Vam omogućava da postavite `HelpContext` osobinu za formu i da kasnije, bez obzira na to koja kontrola je aktivna, koristite identifikator konteksta forme, kada korisnik pritisne taster F1.

Možda ste primetili da `SpeedButton` i ToolButton komponente nemaju `HelpContext` osobinu. Pošto ove komponente nisu prozorske, ne mogu nikada primiti fokus. Sledi da se taster F1 ne može koristiti sa speed tasterima i tasterima na paleti sa alatkama.

U najmanju ruku, trebali biste da obezbedite kontekstno osetljivu pomoć za svaku formu u svom programu (bilo da je forma dijalog prozor, ili klasičan prozor). Na kraju krajeva, na Vama je da odlučite koji elementi Vašeg programa zaslužuju pomoć, a koji ne. Ukoliko idete u krajnost, bolje je da obezbedite više teksta za pomoć nego što je potrebno (ako je to uopšte moguće) nego manje.

Obezbeđivanje kontekstno osetljive pomoći

Obezbeđivanje kontekstno osetljive pomoći za Vaš Delphi program je relativno jednostavno. Kao što sam već rekao, najteži deo posla je pisanje samog fajla sa tekstom pomoći. Ostatak posla je, u odnosu na to, jednostavan.

Postavljanje fajla za pomoć

Bez obzira na to na koji način obezbeđujete kontekstno osetljivu pomoć, prvo morate reći Windows-u ime fajla za pomoć koji će Vaš program koristiti. Da biste to uradili, dodelite ime fajla za pomoć `HelpFile` osobini `Application` klase. To možete uraditi na jedan od dva načina. Lakši način je kroz Project Options dijalog prozor.

U danu 9 ste naučili kako se radi sa opcijama projekta. Objasnio sam činjenicu da Application stranica Project Options dijalog prozora ima polje koje se zove Help File i koje se koristi da bi se navelo ime fajla za pomoć, koji će program koristiti. Jednostavno unesite ime fajla za pomoć u ovo polje. VCL će sam dodeliti ime tog fajla `HelpFile` osobini i program će koristiti taj fajl svaki put kada se zatraži pomoć.

Možete postaviti ime fajla za pomoć i u toku izvršavanja programa. Ovo može biti neophodno ukoliko dozvolite svojim korisnicima da postave fajl za pomoć u proizvoljni direktorijum. Možete postaviti ime fajla za pomoć u Windows-ov Registry (Registry je objašnjen u odeljku "Korišćenje Registry-a") i dodeliti putanju do tog fajla u `HelpFile` osobinu `Application` objekta. Na primer, deo vaše procedure za obradu `OnCreate` događaja bi mogao da izgleda ovako:

```
var
  Filename : string;
begin
  Filename := GetHelpFileName; { user-defined function }
  Application.HelpFile := Filename;
```

Iako to nije uobičajeno, možete menjati sadržaj `HelpFile` osobine na nekoliko mesta u programu. Time možete, na primer, birati korišćenje jednog od više fajlova za pomoć. Pošto ste naveli ime fajla za pomoć, možete se upustiti u pravljenje sistema za pomoć.

Dodavanje podrške za taster F1

Da biste dodali podršku za taster F1 za sve Vaše forme i komponente, treba samo da postavite `HelpContext` osobinu na odgovarajući identifikator konteksta u fajlu za pomoć; odatle VCL preuzima stvar u svoje ruke. Proverite još jednom, da li ste upisali ime fajla za pomoć u `HelpFile` osobinu `Application` klase i da li sam fajl sadrži odgovarajuće identifikatore konteksta.

Dodavanje pristupa sistemu za pomoć kroz meni

Pored podrške za taster F1, većina programa koristi jednu, ili dve stavke koje se nalaze u meniju Help (a gde drugde?), kako bi pokrenuli WinHelp. Uobičajeno je da se u Help meniju nađe stavka Contents. Izborom ove opcije, prikazuje se sadržaj fajla za pomoć. Pored Contents stavke, neki programi imaju i Help Topics stavku u meniju Help. Izborom ove opcije, na ekranu se pojavljuje indeks tema koje se nalaze u fajlu za pomoć. (Indeks tema se generiše u toku procesa kreiranja fajla za pomoć.)

Da biste obezbedili ove i druge stavke Help menija, moraćete malo da programirate. (U svakom od ovih slučajeva radi se samo o jednoj liniji koda.) VCL sadrži metodu pod imenom `HelpCommand` koja se koristi za prikazivanje WinHelp-a u jednom od nekoliko mogućih režima. Ako želite da omogućite Help➥Contents stavku menija, kod treba da izgleda ovako:

```
procedure TForm1.Contents1Click(Sender: TObject);
begin
  Application.HelpCommand(HELP_FINDER, 0);
end;
```

`HelpCommand` metoda poziva WinHelp sa određenom komandom. (Pogledajte pomoć za Windows API i to temu `WinHelp`, kako biste videli listu mogućih komandi.) U ovom slučaju, poziva se WinHelp sa komandom `HELP_FINDER`. Ova komanda nalaže WinHelp-u da prikaže stranicu sa sadržajem fajla za pomoć, kao što je prikazano na slici 14.1. Poslednji parametar `HelpCommand` metode se koristi za prosleđivanje dodatnih podataka WinHelp-u. Ovaj parametar se ne koristi u kombinaciji sa `HELP_FINDER` komandom, tako da je postavljen na `0`.

Slika 14.1
Stranica sa sadržajem fajla za pomoć ScratchPad programa

NAPOMENA Da bi WinHelp mogao da prikaže stranicu sa sadržajem, morate imati fajl sa sadržajem za Vaš fajl za pomoć. Ovaj fajl ima ekstenziju .cnt i predstavlja tekstualni fajl u kome je opisano na koji način bi stranica sa sadržajem trebala da bude prikazana. Možete naći više informacija o fajlu sa sadržajem u fajlu za pomoć Microsoft Help Workshop programa. Help Workshop i njegov fajl za pomoć možete pronaći u `\Delphi 4\Help\Tools` direktorijumu.

NAPOMENA Dobar indeks tema za Vaš fajl za pomoć je od neprocenljive vrednosti. Kvalitet fajla za pomoć i kvalitet indeksa tema fajla za pomoć su direktno proporcionalni sa obimom tehničke podrške koju ćete morati da obezbedite. Nemojte prevideti ovu činjenicu.

Kontekstno osetljiva pomoć na zahtev korisnika

Ranije objašnjena dva načina za obezbeđivanje sistema za pomoć će Vam uglavnom biti dovoljna. U ostalim slučajevima ćete morati da direktno pozivate WinHelp i da mu prosleđujete određeni identifikator konteksta. Za ovakve slučajeve, VCL obezbeđuje `HelpContext` metodu. Ova metoda prihvata samo jedan parametar i on određuje identifikator konteksta stranice koju želite da vidite kada se pokrene WinHelp. Na primer, recimo da želite da vidite stranicu čiji je identifikator konteksta 99. Da biste pokrenuli WinHelp i prikazali tu stranicu uradite sledeće:

```
Application.HelpContext(99);
```

Prosleđivanjem određenog identifikatora konteksta možete naložiti WinHelp-u da prikaže bilo koju stranicu na korisnikov zahtev. To je ono što VCL radi za Vas kada postavite `HelpContext` osobinu određene komponente.

Korišćenje fajlova zaglavlja sa sistemom za pomoć

Postavljanje `HelpContext` osobina za sve forme i sve komponente za koje želite da omogućite kontekstno osetljivu pomoć će uglavnom zadovoljiti sve Vaše potrebe. Ukoliko, ipak, imate potrebu za pozivanjem specifičnih stranica za pomoć iz Vašeg

programa, razmislite o mogućnosti definisanja konstanti za identifikatore konteksta. Korišćenje imena je mnogo jednostavnije od pamćenja celobrojnih vrednosti za svaki identifikator konteksta.

U prethodnom odeljku, govorio sam o korišćenju `HelpContext` metode za pozivanje WinHelp-a sa određenim identifikatorom konteksta. Koristio sam, kao primer, indentifikator broj 99. Umesto korišćenja celobrojnih identifikatora, možete koristiti imena:

```
Application.HelpContext(IDH_FILEOPEN);
```

Očigledno je da je lakše zapamtiti string, nego celobrojnu vrednost. Simboli identifikatora kontekstno osetljive pomoći se mogu nalaziti u zasebnom fajlu zaglavlja, koji po potrebi možete uključivati u Vaš program (korišćenjem `{$I}` direktive prevodiocu). Listing 14.1 prikazuje tipičan fajl zaglavlja koji sadrži identifikatore kontekstno osetljive pomoći.

Listing 14.1: `HELP.INC`

```
const
  IDH_FILENEW         = 1;
  IDH_FILEOPEN        = 2;
  IDH_FILESAVE        = 3;
  IDH_FILESAVEAS      = 4;
  IDH_FILEPRINT       = 5;
  IDH_FILEPRINTSETUP  = 6;
  IDH_FILEEXIT        = 7;
  IDH_EDITUNDO        = 8;
  IDH_EDITCOPY        = 9;
  IDH_EDITCUT         = 10;
  IDH_EDITPASTE       = 11;
  IDH_EDITSELECTALL   = 12;
  IDH_EDITWORDWRAP    = 13;
  IDH_HELPABOUT       = 14;
```

Potrebno je da negde u izvornom kodu Vašeg programa dodate liniju koja će uključiti fajl sa identifikatorima:

```
{$I HELP.INC}
```

Sada možete koristiti imena indentifikatora umesto celobrojnih vrednosti.

Kako su fajlovi za pomoć organizovani, zavisi od alata koje koristite za pravljenje fajlova za pomoć. Većina programa za rad sa fajlovima za pomoć sadrži opciju pomoću koje možete napraviti neku vrstu fajla zaglavlja koji će sadržati imena identifikatora. Konkretan sadržaj tog fajla zavisi od toga da li ste fajl za pomoć pravili Vi, ili Vaš saradnik i od toga koji ste program za rad sa fajlovima za pomoć koristili. Ukoliko ne koristite program za rad sa fajlovima za pomoć, jednostavno sami unesite simbole.

Zaokruživanje sistema za rad sa konteksno osetljivom pomoći

Vreme je da svoje novo-stečeno znanje iskoristite u jednom praktičnom primeru. U ovom odeljku ćete dodati kontekstno osetljivu pomoć programu `ScratchPad` (znali ste da ćemo se vratiti na stari `ScratchPad`, zar ne?)

Izvorni kod iz ove knjige sadrži jednostavan fajl za pomoć koji treba koristiti u `ScratchPad` programu. Iskopirajte `Scratch.hlp` u svoj radni direktorijum, tako da Delphi može da ga pronađe kada bude generisali `ScratchPad` program.

Obezbeđivanje kontekstno osetljive pomoći zahteva oko 10 minuta posla. Krenite:

1. Učitajte `ScratchPad` projekat. Otvorite Application stranicu Project Options dijalog prozora i unesite `Scratch.hlp` u polje Help File. Kliknite OK da biste zatvorili dijalog prozor. (Proverite da li ste postavili `Scratch.hlp` fajl u direktorijum sa projektom.)
2. Prikažite glavnu formu programa `ScratchPad` u FormDesigner-u. Dva puta kliknite na MainMenu ikonu da biste aktivirali Menu Designer.
3. U Menu Designer-u, izaberite File➥New menu item. Pronađite `HelpContext` osobinu u Object Inspector-u i izmenite njenu vrednost u 1.
4. Ponovite prethodni korak za sve stavke menija. Koristite vrednosti iz listinga 14.1 da biste postavili `HelpContext` osobinu svake stavke.
5. Izaberite Help➥Contents. Postavite njenu `Enabled` osobinu na `True`. Zatvorite Menu Designer.
6. U Form Designer-u, iz glavnog menija programa `ScratchPad` izberite Help➥Contents. U Code Editor-u se prikazuje `HelpContentsClick` funkcija. Na mestu gde se nalazi kurzor, unesite sledeću liniju koda:

```
Application.HelpCommand(HELP_FINDER, 0);
```

7. Izmenite `HelpContext` osobinu glavne forme na `1000` (to je, u stvari, identifikator konteksta stranice sa sadržajem).

Pokrenite program i proverite da li radi na odogovarajući način. Ukoliko pritisnete taster F1 dok se kurzor nalazi u Memo prozoru, prikazaće se stranica sa sadržajem. Ukoliko se postavite na neku stavku menija i zatim pritisnete taster F1, prikazaće se stranica za pomoć koja je povezana sa tom stavkom. Takođe, proverite i da li Contents stavka iz menija radi na očekivani način.

SAVET Ukoliko prevedete i pokrenete `ScratchPad` program koji se nalazi u delu za dan 14 u izvornom kodu iz knjige, primetićete da sadrži i jednu osobinu koju ovde nismo pomenuli. Program ima režim za pružanje pomoći (engl. Help Mode) koji se koristi za dobijanje pomoći za određeni taster sa palete sa alatima, ili za određenu stavku menija. Da biste aktivirali režim za pružanje pomoći, kliknite na Help taster koji se nalazi na paleti sa alatkama. Oblik pokazivača miša se menja. Sada kliknite na bilo koji taster na paleti sa alatima, ili na bilo koju stavku menija. Prikazuje se stranica za pomoć koja je vezana za taj objekat. Režim za pružanje pomoći se automatski isključuje kada kliknete na neki objekat. Proučite izvorni kod, kako biste videli na koji način je režim za pružanje pomoći napravljen. Ova osobina zahteva posebnu obradu poruka koja je objašnjena nešto kasnije u ovom poglavlju.

Kontekstno osetljiva pomoć više nije luksuz. Bez obzira da li su Vaši korisnici Vaši saradnici, ili potpuno nepoznati ljudi, oni su, ipak, svi Vaši korisnici. Oni zahtevaju da programi sadrže određene mogućnosti, a kontekstno osetljiva pomoć je jedna od njih. Pošto je dodavanje kontekstno osetljive pomoći programima koji su pravljeni u Delphi-ju jednostavno, ne postoji ni jedan razlog da ona izostane iz Vaših programa.

Kontrola grešaka korišćenjem obrade izuzetaka

Čak i u najbolje dizajniranim programima, operacije koje se dešavaju bez kontrole programera mogu prouzrokovati problem. Korisnici prave greške. Na primer, korisnik može da unese pogrešnu vrednost u polje u bazi podataka, ili da otvori fajl koji ne odgovara Vašem programu. Bez obzira na tip greške, morate biti spremni da ih sprečite gde god i kad god je to moguće. Ne možete predvideti svaki korak svog korisnika, ali možete predvideti neke uobičajene greške i na vreme ih sprečiti.

Obrada izuzetaka je, u stvari, lepši način za kontrolisanje grešaka. Iako svaki program može obrađivati izuzetke, oni su od koristi uglavnom korisnicima komponenti i drugih Object Pascal klasa. Na primer, ukoliko koristite komponentu potrebno je da budete obavešteni da se sa njom desilo nešto nepredviđeno. Dobro napisana komponenta će generisati izuzetak u trenutku kada se desi nešto nepredviđeno. Tada Vi možete prihvatiti izuzetak i obraditi ga na željeni način (uglavnom tako što ćete prekinuti izvršavanje programa, ili dozvoliti korisniku da ispravi grešku i nastavi sa radom).

Verovatno nećete pisati mnogo koda za obradu izuzetaka u svakodenevnom programiranju. Obradu ćete uglavnom pisati za izuzetke koje VCL generiše kada se sa komponentom desi nešto nepredviđeno. Ukoliko se bavite pravljenjem komponenti, sigurno je da ćete često koristiti obradu izuzetaka.

NAPOMENA Ograničite korišćenje izuzetaka na slučajeve kada je greška toliko ozbiljna da ugrožava normalan nastavak rada programa. Najveći broj grešaka, koje se dešavaju u toku izvršavanja programa, se može otkloniti proverom parametara, proverom ispravnosti unetih podataka, ili korišćenjem tradicionalnih tehnika uklanjanja grešaka.

Puno objašnjenje obrade izuzetaka bi zahtevalo celo poglavlje, tako da ću se ograničiti na obradu izuzetaka koje generišu VCL komponente.

Ključne reči za obradu izuzetaka: `try`, `except`, `finally` i `raise`

Sintaksa obrade izusetaka nije posebno komplikovana. Postoje četiri ključne reči koje se koriste: `try`, `except`, `finally` i `raise`. `Try`, `except` i `finally` ključne reči se koriste prilikom obrade izuzetaka, dok se raise koristi za generisanje izuzetka. Skoncentrišimo se, sada, na ove ključne reči.

SINTAKSA

```
try with except
try
  TryStatements
except
  on TypeToCatch do begin
    ExceptStatements
  end;
end;
```

Ključna reč `try` obeležava početak `try` bloka. Naredbe u *`TryNaredbe`* se izvršavaju. Ukoliko se generiše bilo koji izuzetak prilikom izvršavanja TryNaredbi, izvršavaju se *`ExceptNaredbe`*. Ukoliko se ni jedan izuzetak ne generiše, *`ExceptNaredbe`* se ignorišu i program nastavlja sa izvršavanjem iza end naredbe. *`TipKojiTrebaPrihvatiti`* je jedna od VCL klasa izuzetaka. Ukoliko se *`TipKojiTrebaPrihvatiti`* ne navede, prihvataju se svi izuzeci, bez obzira na to kog su tipa.

SINTAKSA

```
try with finally
try
  TryStatements
finally
  FinallyStatements
end;
```

Ključna reč `try` obeležava početak `try` bloka. Izvršavaju se *`TryNaredbe`*. *`FinallyNaredbe`* se uvek izvršavaju, bez obzira da li je prilikom izvršavanja *`TryNaredbi`* generisan izuzetak, ili ne.

Pre nego što probam da objasnim `try` i `except` ključne reči, pogledajmo jednostavan primer.

U listingu 14.2 se nalazi kod iz procedure za obradu File➥Open događaja `Picture Viewer` programa koji ste napravili u danu 4. Procedura za obradu događaja je izmenjena tako da se koristi obrada izuzetaka. Setite se da ovaj kod pokušava da učita sliku (`.bmp`, `.wmf`, ili `.ico`). Ukoliko fajl koji korisnik izabere nije fajl sa slikom, VCL generiše izuzetak. Ukoliko se to desi, Vi morate prihvatiti izuzetak i prikazati poruku kojom ćete reći korisniku da izabrani fajl nije fajl sa slikom.

Listing 14.2: Primer obrade izuzetaka

```
01: procedure TMainForm.Open1Click(Sender: TObject);
02: var
03:   Child : TChild;
04: begin
05:   if OpenPictureDialog.Execute then begin
06:     Child := TChild.Create(Self);
07:     with Child.Image.Picture do begin
08:       try
09:         LoadFromFile(OpenPictureDialog.FileName);
10:         Child.ClientWidth := Width;
11:         Child.ClientHeight := Height;
12:       except
13:         Child.Close;
14:         Application.MessageBox(
15:           'This file is not a Windows image file.',
16:           'Picture Viewer Error', MB_ICONHAND or MB_OK);
17:         Exit;
18:       end;
19:     end;
20:     Child.Caption :=
21:       ExtractFileName(OpenPictureDialog.FileName);
22:     Child.Show;
23:   end;
24: end;
```

U ovom kodu možete videti `try` blok (linija 8) i except blok (linija 12). `Try` blok se koristi za definisanje koda koji bi mogao da izazove generisanje izuzetka. `Try` naredbe govore kompajleru sledeće: "Probaj ovo i vidi da li će raditi". Ukoliko kod radi, `except` blok se ignoriše i program nastavlja sa izvršavanjem. Ukoliko bilo koja od naredbi iz `try` bloka izazove generisanje izuzetka, izvršiće se naredbe iz `except` bloka. `Except` blok se mora nalaziti odmah ispod `try` bloka.

Bitno je shvatiti da čim neka od naredbi iz `try` bloka izazove generisanje izuzetka, izvršavanje programa se odmah prebacuje na `except` blok. U ovom primeru, poziv funkciji `LoadFromFile` (linija 9) bi mogao da generiše izuzetak. Ukoliko se izuzetak zaista generiše, izvršavanje programa se momentalno prebacuje na prvu liniju `except` bloka (na onu koja se nalazi ispod `except` ključne reči). U tom slučaju, naredbe iz linija 10 i 11 neće biti izvršene.

Generisanje izuzetaka

Kao što možete videti, except odeljak prihvata izuzetak koji je generisan negde u programu. Ovo, uglavnom, znači da je izuzetak generisan negde u VCL-u (ili u nezavisnoj komponenti, ukoliko ste instalirali neke od njih). Izuzetak se generiše korišćenjem ključne reči `raise`. Kod koji generiše izuzetak to čini za određenu klasu. Na primer, tipična `raise` naredba bi mogla da izgleda ovako:

```
if BadParameter then
  raise EInvalidArgument.Create('A bad parameter was passed');
```

`Raise` naredba, u ovom slučaju, generiše instancu klase za obradu izuzetaka pod imenom `EInvalidArgument`. Kada pišete sopstveni kod, možete generisati izuzetak koji je instanca jedne od VCL klasa za obradu izuzetaka, a možete napraviti i sopstvene klase za obradu izuzetaka. Recimo da ste odredili kod broj `111` za određeni tip greške i da imate realizovanu klasu za obradu izuzetaka pod imenom `EMyException`. U tom slučaju možete napisati raise naredbu na sledeći način:

```
raise EMyException.Create('Error 111');
```

Prevodilac pravi kopiju objekta koji se generiše i prosleđuje ga `except` bloku koji se nalazi iza `try` bloka (za nekoliko trenutaka ću se vratiti na tu temu).

Još nešto o `except`

Kao što sam rekao, u početku ćete uglavnom prihvatati izuzetke koje generiše VCL. Ukoliko se nešto nepredviđeno desi u VCL komponenti, VCL će generisati izuzetak. Ukoliko ne obradite izuzetak, VCL će ga samostalno obraditi na podrazumevani način. To, u najvećem broju slučajeva, znači da će biti prikazan prozor sa porukom o nastaloj grešci. Prihvatanjem ovih izuzetaka, možete odrediti na koji način će oni biti obrađeni, što je neuporedivo bolje od podrazumevanog načina obrade.

Pogledajte još jednom listing 14.2. Počevši od 12. linije možete videti sledeći kod:

```
except
  Child.Close;
  Application.MessageBox(
    'This file is not a Windows image file.',
    'Picture Viewer Error', MB_ICONHAND or MB_OK);
  Exit;
end;
```

Ključna reč `except` govori prevodiocu da želite da prihvatite svaki izuzetak, bez obzira na to kog je on tipa.

Pošto se iza `except` naredbe ne nalazi `case` struktura, kod u okviru `except` bloka će se izvršiti bez obzira na tip izuzetka. To je u redu, ukoliko ne znate koje sve tipove izuzetka određeni deo koda može generisati, ili ukoliko želite da prihvatite sve izuzetke, bez obzira na tip. U realnim programima ćete morati bliže da odredite na koji način želite da obradite određene tipove izuzetaka.

Vratimo se, ponovo, na listing 14.2. Kod u liniji 9 će izazvati generisanje izuzetka, ukoliko korisnik pokuša da otvori fajl koji nije grafički. Ukoliko se to desi, VCL generiše `EInvalidGraphic` izuzetak. Možete prihvatiti samo taj izuzetak i omogućiti da se ostali izuzeci obrade na podrazumevani način. Kod izgleda ovako:

```
except
  on EInvalidGraphic do begin
    Child.Close;
    Application.MessageBox(
      'This file is not a Windows image file.',
      'Picture Viewer Error', MB_ICONHAND or MB_OK);
    Exit;
  end;
end;
```

Primetite da ovaj kod prihvata samo `EInvalidGraphic` izuzetak. Ovo je bolji način za prihvatanje izuzetaka. Sada će svi izuzeci ovog tipa biti obrađeni, dok će se ostali izuzeci obrađivati na podrazumevani način.

Važno je razumeti da kada prihvatite izuzetak, program nastavlja sa izvršavanjem pošto se izvrše naredbe u `except` delu. Jedna od prednosti prihvatanja izuzetaka je i što možete ispraviti grešku i nastaviti sa izvršavanjem programa. Primetite `Exit` naredbu u prethodnom listingu. U ovom slučaju Vi ne želite da izvršite kod koji se nalazi iza except dela, tako da ćete napustiti proceduru posle obrade izuzetka.

VCL će automatski obraditi mnoge izuzetke, ali on ne može da predvidi svaku mogućnost. Izuzetak koji se ne obrađuje se zove neobrađeni izuzetak. Ukoliko dođe do generisanja neobrađenog izuzetka, Windows generiše poruku o grešci i prekida izvršavanje Vašeg programa. Pokušajte da predvidite koji se sve izuzeci mogu generisati u Vašem programu i učinite sve kako biste ih obradili.

Prihvatanje više tipova izuzetaka

Možete prihvatiti nekoliko tipova izuzetaka u okviru istog `except` bloka. Na primer, recimo da ste napisali kod koji poziva neke VCL metode i neke druge funkcije, koje bi mogle da prouzrokuju generisanje izuzetka. Ukoliko se generiše VCL izuzetak `EInvalidGraphic`, vi ćete želeti da ga obradite. Zatim, u Vašem kodu bi mogao da se generiše i `EMyOwnException` izuzetak (klasa koju ste napravili za obradu izuzetaka). Želite da obradite ova dva tipa izuzetka na različite načine. Na osnovu ovoga biste mogli da napišete sledeći kod:

```
try
  OneOfMyFunctions;
  Image.Picture.LoadFromFile('test.bmp');
  AnotherOfMyFunctions;
except
  on EInvalidGraphic do begin
    { do some stuff }
  end;
  on EMyOwnException do begin
    { do some stuff }
  end;
end;
```

Sada ste spremni da prihvatite i VCL izuzetak `EInvalidGraphic` i Vaš `EMyOwnException` izuzetak. Pošto treba da svaki izuzetak obradite na poseban način, morate ih i posebno prihvatiti.

Pravljenje sopstvene klase za obradu izuzetaka može biti jednostavno:

```
type
  EMyOwnException = class(Exception);
```

Najčešće ćete praviti klasu za obradu izuzetaka samo da biste mogli da jedan tip izuzetka razlikujete od drugih. Nije potrebno pisati nikakav dodatni kod.

Primetite da u prethodnom kodu samo prihvatate VCL izuzetak, ali da ne radite ništa sa instancom klase koja je prosleđena `except` bloku. U ovom slučaju, nije potrebno znati koje se informacije nalaze u `EInvalidGraphic` klasi, premda je jasno zbog čega je generisan izuzetak.

Da biste koristili informacije iz instance klase izuzetka koja je prosleđena except bloku, deklarišite promenljivu za klasu izuzetka u do naredbi. Na primer:

```
try
  Image.Picture.LoadFromFile('test.bmp');
except
  on E : EInvalidGraphic do begin
    { do something with E }
  end;
end;
```

U ovom primeru je promenljiva `E` deklarisana kao pokazivač na `EInvalidGraphic` klasu koja je prosleđena `except` bloku. Možete koristiti `E` da biste pristupili osobinama i metodama `EInvalidGraphic` klase. Zapamtite da je promenljiva `E` vidljiva samo u bloku u okviru kojeg je deklarisana. Trebali biste da pogledate Delphi-jev Help, ukoliko želite da saznate više o specifičnim VCL klasama za obradu izuzetaka, odnosno, o osobinama i metodama koje se nalaze u određenim klasama.

Sve VCL klase za obradu izuzetaka imaju osobinu pod imenom `Message`. Ova osobina sadrži opis izuzetka koji je generisan. Možete koristiti ovu osobinu da biste prikazali poruku korisniku:

```
except
  on E : EInvalidGraphic do begin
    { do some stuff }
    MessageDlg(E.Message, mtError, [mbOk], 0);
  end;
end;
```

Ponekad, VCL poruke i nisu dovoljno opisne. U takvim situacijama možete napraviti sopstvene poruke, ali, u principu, sadržaj Message osobine bi trebao da bude sasvim dovoljan.

Korišćenje ključne reči `finally`

Ključna reč `finally` definiše deo koda koji se izvršava bez obzira da li je izuzetak generisan, ili ne. Drugim rečima, kod u `finally` delu će biti izvršen i ako se događaj generiše i ako se ne generiše. Ukoliko koristite finally blok, ne možete koristiti `except` blok.

`Finally` blok se, uglavnom, koristi da bi se oslobodila sva dinamički alocirana memorija. Na primer:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  Buffer : Pointer;
begin
  Buffer := AllocMem(1024);
  try
    { Code here that might raise an exception. }
  finally
    FreeMem(Buffer);
  end;
end;
```

U ovom slučaju, memorija koje je alocirana za `Buffer` će biti propisno oslobođena bez obzira da li je, u okviru `try` bloka, izuzetak generisan, ili ne.

Prihvatanje neobrađenih izuzetaka na nivou programa

Vaš program može obrađivati izuzetke i na nivou programa. Klasa `TApplication` sadrži događaj pod imenom `OnException` koji će biti generisan svaki put kada se u Vašem programu generiše neobrađeni izuzetak. Odgovarajući na ovaj događaj, možete prihvatiti bilo koji izuzetak koji generiše Vaš program.

Ovaj događaj se generiše kada se generiše neobrađeni izuzetak. Svaki izuzetak koji obradite sa `try` i `except` je obrađen, tako da se `OnException` neće generisati za ove izuzetke.

`OnException` događaj prihvatate na isti način na koji ste ranije prihvatili `OnHint` događaj:

1. Dodajte dekleraciju metode u dekleraciju klase glavne forme:

   ```
   procedure MyOnException(Sender : TObject; E : Exception);
   ```

2. Dodajte telo metode u `implementation` celinu glavne forme:

   ```
   procedure TForm1.MyOnException(Sender : TObject; E : Exception);
   begin
    { Do whatever necessary to handle the exception here. }
   end;
   ```

3. Dodelite `OnMyExcept` metodu `OnException` događaju `Application` objekta:

```
Application.OnException := MyOnException;
```

Sada će Vaša `MyOnException` metoda biti automatski pozivana kada se generiše neobrađeni izuzetak.

Ukoliko ne obradite izuzetke na odgovarajući način, možete zablokirati svoj program, ili čak srušiti Windows. Uglavnom je najbolje prepustiti VCL-u i Windows-u obradu izuzetka, dokle god niste potpuno sigurni u to što radite.

Možete koristiti `ShowException` funkciju da biste prikazali prozor sa porukom koja opisuje grešku do koje je došlo. Ovu funkciju, najčešće, poziva podrazumevana procedura za obradu `OnException` događaja, ali je Vi možete ravnopravno koristiti za svoje potrebe. Jedan od parametara procedure za obradu OnException događaja je pokazivač na `Exception` objekat (`Exception` je VCL-ova osnovna klasa za obradu izuzetaka). Da biste prikazali poruku o grešci, prosledite `Exception` objekat `ShowException` funkciji:

```
procedure TForm1.MyOnException(Sender : TObject; E : Exception);
begin
  { Do whatever necessary to handle the exception here
  { then display the error message. }
  Application.ShowException(E);
end;
```

Sada će biti prikazana poruka o grešci na isti način kao i da je sam VCL radio sa neobrađenim izuzecima.

Kao što možete videti, obrada izuzetaka na nivou programa je napredna tehnika i ne biste trebali da je koristite, dokle god niste potpuno sigurni u to što radite.

Otkrivanje grešaka u programima u kojima se koristi obrada izuzetaka

Jednostavno gledajući, otkrivanje grešaka u programima u kojima se koristi obrada izuzetaka može biti veoma nezgodno. Svaki put kada se generiše neki događaj, program za otkrivanje grešaka (engl. debugger) prekida izvršavanje programa na početku `except` bloka, isto kao da je na toj liniji postavljena prekidna tačka. Ukoliko se `except` blok nalazi u Vašem kodu, tačka izvršavanja se prikazuje na isti način kao da je program zaustavljen zbog prekidne tačke. Sada možete pokreuti program klikom na Run taster, ili se kretati kroz program korak-po-korak.

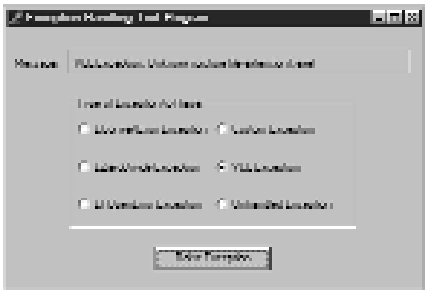
Možda nećete moći da nastavite sa izvršavanjem programa kada se generiše izuzetak. Da li ćete moći da nastavite proces otkrivanja grešaka, zavisi od toga šta se tačno desilo u Vašem programu.

Ponekad se `except` blok može nalaziti i u VCL kodu. To je slučaj sa VCL izuzecima koje niste obradili u svom kodu. Pod ovim okolnostima, CPU prozor će prikazati mesto gde je izvršavanje programa zaustavljeno.

Drugi aspekt otkrivanja grešaka u programima u kojima se koristi obrada izuzetaka je što se VCL poruka o grešci prikazuje i kada Vi obradite izuzetak. Ovo može biti zbunjujuće i može početi da Vas nervira, ukoliko Vas neko na to prethodno ne upozori. Da biste sprečili VCL da prikazuje poruke o greškama i da biste sprečili program za otkrivanje grešaka da prekida izvršavanje kada se generiše izuzetak, otvorite Language Exceptions stranicu Debugger Options dijalog prozora i poništite Stop on Delphi Exceptions izbor koji se nalazi na vrhu stranice. Kada poništite ovaj izbor, program za otkrivanje grešaka neće zaustavljati program kada se generiše VCL izuzetak.

Kao i sa drugim aspektima Delphi-ja i VCL-a, potrebno je neko vreme da biste dobro naučili da radite sa izuzecima. Zapamtite da obradu izuzetaka treba koristiti samo onda kada je to neophodno. Za zaštitu od sitnih grešaka, koristite tradicionalne tehnike.

Izvorni kod iz ove knjige sadrži program pod imenom `EHTest`. Ovaj program ilustruje generisanje i prihvatanje nekoliko tipova izuzetaka. Da bi ovaj program mogao ispravno da radi, morate poništiti Stop on Delphi Exceptions izbor. Možete preneti izvorni kod iz ove knjige sa `http://www.mcp.com/info`. Slika 14.2 prikazuje izvršavanje programa `EHTest`.



Slika 14.2
Program EHTest u toku izvršavanja

Korišćenje Registry-a

Nekada su Windows programi koristili konfiguracione (`.ini`) fajlove da bi upisali informacije koje su specifične i bitne samo za njihovo izvršavanje. Glavni konfiguracioni fajl je, kao što znate, `WIN.INI`. Programi su mogli da upisuju informacije bitne za ceo sistem u `WIN.INI`, a informacije bitne samo za njih u sopstvene `.INI` fajlove. Ovakav pristup ima nekoliko prednosti i nekoliko mana.

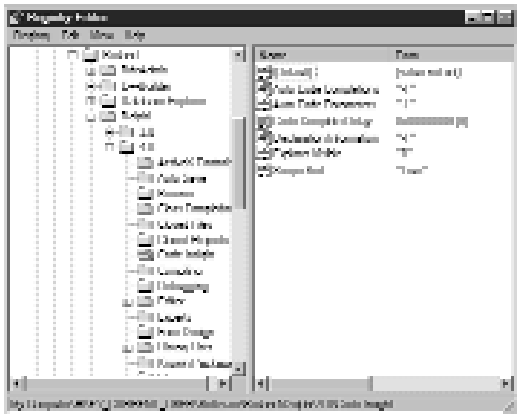
Neko (pretpostavljam, pametniji od mene) je odlučio da bi Registry baza trebala da bude novo mesto gde će programi upisivati svoje konfiguracije. Bez obzira da li je to dobro, ili loše rešenje, korišćenje `.ini` fajlova je zastarelo i danas se koristi isključivo Registry baza.

Izraz *Registry* je skraćenica od Windows Registration Database. Registry sadrži široki skup informacija o konfiguraciji Windows-a. Skoro svaka opcija u Windows-u je upisana u Registry. Pored informacija o sistemu, u Registry-ju možete pronaći informacije koje su specifične za svaki instalirani program. Vrsta upisanih informacija u potpunosti zavisi od samog programa, ali se tu mogu naći i podaci o veličini i poziciji prozora, lista fajlova koji su poslednji otvarani, direktorijum iz koga je otvoren poslednji fajl i tako dalje. Mogućnosti su neograničene.

Windows 95 i Windows NT se isporučuju sa programom koji se zove Registry Editor (`REGEDIT.EXE`) i njega možete koristiti za pregled i izmenu Registry baze.

Budite pažljivi prilikom izmene podataka u Registry-ju. Ove izmene direktno utiču na rad programa, ali i na rad samo Windows-a!

Na slici 14.3 je prikazan Registry Editor u kome se nalaze Delphi Code Insight opcije.



Slika 14.3
Windows-ov Registry Editor

Kao što se može videti sa slike 14.3, struktura Registry-ja je hijerarhijska. Zapisima u Registry-ju možete pristupati na potpuno isti način na koji pristupate direktorijumima i fajlovima na Vašem hard disku.

Registry ključevi

Zapis u Registry-ju se zove *ključ*. Ključ može biti povezan sa direktorijumom na Vašem hard disku. Da biste pristupili određenom ključu, morate ga otvoriti. Pošto otvorite ključ, iz njega možete čitati podatke, ili ih upisivati. Pogledajte sliku 14.3. Ključ koji je trenutno prikazan je:

```
My Computer\HKEY_CURRENT_USER\Software\Borland\Delphi\4.0\Code
Insight
```

Ne možete videti svaku granu Registry-ja, ali, ukoliko pogledate statusnu liniju Registry Editor-a, možete videti ime trenutno prikazanog ključa. Takođe, primetite da `Delphi\4.0` ključ ima nekoliko *podključeva*. Možete napraviti koliko god želite ključeva i podključeva za Vaš program.

Svaki ključ može upisati podatke u svoja polja. Svaki ključ ima polje pod imenom (`Default`). Podrazumevana vrednost se obično ne koristi, pošto ćete gotovo uvek želeti da napravite posebna polja za određene ključeve. Ako pogledate sliku 14.3, primetićete da `Code Insight` ključ ima sledeća polja:

```
Auto Code Completions
Auto Code Parameters
Code Complete Delay
Declaration Information
Explorer Visible
Scope Sort
```

Ako ste obratili pažnju, primetili ste da ova polja odgovaraju Code Insight opcijama na Code Insight strani Enviroment Options dijalog prozora. Svako polje sadrži neku vrednost. Možete pročitati ovu vrednost iz Registry-a, ili je možete promeniti.

Tipovi podataka u Registry-ju

Registry može da radi sa nekoliko različitih tipova podataka. Osnovni tipovi podataka su: binarni podaci, stringovi i celobrojni podaci. *Binarni tip podataka* se može koristiti za smeštanje bilo kakvih binarnih podataka. Možete, na primer, smestiti niz celih brojeva u binarni podatak. Najverovatnije je da nećete direktno koristiti binarni tip podataka.

U najvećem broju slučajeva ćete se baviti pisanjem i čitanjem stringova i celobrojnih vrednosti. Kao što možete videti sa slike 14.3, numerički podaci se, takođe, mogu zapisati u obliku stringova. Na Vama je da odlučite kako ćete smeštati podatke u Registry.

Do sada smo rekli šta sve možete da radite sa Registry-jem, ali ne i *kako* to da uradite. Sledeće što ćemo videti je upravo to.

Klasa `TRegistry`

Windows-ov API sadrži nekoliko funkcija koje se koriste za manipulisanje Registry bazom. Neke od ovih funkcija su: `RegCreateKey`, `RegOpenKey`, `RegQueryValue`, `RegSetValue`, `RegDeleteKey`. Rad sa Registry-jem na API nivou može biti prilično težak. Zahvalan sam (a i Vi ćete biti) ljudima iz Borland-a koji su napraviti VCL klasu `TRegistry` i u nju uključili sve operacije za rad sa Registry-jem.

Ova klasa sadrži sve što je potrebno za uspešno pisanje i čitanje podataka iz Registry-ja. Pre nego što objasnim kako se koristi `TRegistry`, obratimo pažnju na osobine i metode ove klase.

Osobine klase `TRegistry`

Klasa `TRegistry` ima samo četiri osobine. Osobina `CurrentKey` sadrži celobrojnu vrednost koja određuje trenutno aktivni ključ. Kada pozovete neku od metoda klase `TRegistry`, ona radi sa aktivnim ključem. Vrednost osobine `CurrentKey` se automatski postavlja u trenutku kada otvorite ključ. Vrednost ove osobine možete pročitati, ali je vrednost tog podatka čisto teorijska.

Vrednosti `RootKey` i `CurrentPath` osobina zajednički sačinjavaju ime trenutno aktivnog ključa. `CurrentKey` osobina sadrži tekstualni opis putanje do aktivnog ključa, ali bez vrednosti `RootKey` osobine. Za primer, uzmimo ovaj ključ:

```
\HKEY_CURRENT_USER\Software\Borland\Delphi\4.0\Code Insight
```

U ovom slučaju, `\HKEY_CURRENT_USER` se nalazi u `RootKey` osobini, dok se ostatak imena ključa, `Software\Borland\Delphi\4.0\Code Insight`, nalazi u `CurrentPath` osobini.

Podrazumevana vrednost `RootKey` osobine je `\HKEY_CURRENT_USER`. Tu biste trebali da smeštate podatke koji su bitni za Vaš program, tako da se najčešće vrednost ove osobine neće menjati. Ukoliko je potrebno da promenite vrednost RootKey osobine, jednostavno joj dodelite novu vrednost. Primetite da moguće vrednosti RootKey osobine nisu obični stringovi već vrednosti koje su unapred definisane u samom Windows-u. Druge moguće vrednosti su: `HKEY_CLASSES_ROOT`, `HKEY_LOCAL_MACHINE`, `HKEY_USERS`, `HKEY_CURRENT_CONFIG` i `HKEY_DYN_DATA`.

`LazyWrite` osobina određuje način na koji program upisuje vrednost trenutno aktivnog ključa u `Registry` bazu. Ukoliko je vrednost `LazyWrite` osobine `True`, program nastavlja sa izvršavanjem u onom trenutku kada zatvorite ključ. Drugim rečima, program samo započinje proces upisivanja ključa i nastavlja dalje. Ukoliko je vrednost `LazyWrite` osobine `False`, program nastavlja sa izvršavanjem tek onda kada se proces upisivanja ključa završi. Podrazumevana vrednost ove osobine je `True` i ne biste trebali da je menjate, dokle god ne radite sa podacima koje morate upisati u `Registry` bazu, pre nego što Vaš program nastavi sa radom.

Metode klase `TRegistry`

Klasa `TRegistry` ima nekoliko metoda koje ćete koristiti prilikom upisivanja i čitanja iz Registry baze. U tabeli 14.1 se nalaze primarne metode i njihovi opisi.

Tabela 14.1: Primarne metode klase `Tregistry`

Metoda	Opis
`CloseKey`	Zatvara ključ i upisuje podatke u njega. Trebalo bi da zatvorite ključ čim završite rad sa njim, ali da biste to uradili, ne morate pozivati `CloseKey`, pošto destruktor klase `TRegistry` automatski zat vara ključ.
`CreateKey`	Kreira ključ, ali ga ne otvara. Koristite `OpenKey`, ukoliko želite da kreirate ključ i da odmah počnete sa upisivanjem podataka.
`DeleteKey`	Briše određeni ključ. Možete obrisati bilo koji ključ. Da biste obrisali aktivni ključ, prosledite prazan string `DeleteKey` metodi.
`GetKeyNames`	Vraća imena svih podključeva datog ključa u obliku `TString` objeka ta. Možete koristiti ovu metodu, ako želite da obradite sve podključeve datog ključa.
`GetValueNames`	Vraća imena svih polja koja su smeštena u dati ključ. Koristite ovu meto du, ako želite da obradite sva polja iz datog ključa.
`KeyExists`	Vraća `True`, ukoliko dati ključ postoji, ili `False`, ukoliko on ne pos toji. Koristite ovu metodu, pre nego što probate da pročitate vrednost ključa, kako biste saznali da li on uopšte postoji.
`LoadKey`	Učitava ključ koji je prethodno snimljen na disk. Pogledajte Delphi-jev Help da biste dobili više podataka o ovoj metodi.
`OpenKey`	Otvara određeni ključ. Ukoliko ključ ne postoji, vrednost `CanCreate` parametra određuje da li će ključ biti automatski kreiran. Koristite ovu metodu pre nego `CreateKey` ukoliko želite da kreirate ključ i da odmah upišete podatke u njega. `OpenKey` kreira ključ i automatski ga otvara, dok `CreateKey` kreira ključ, ali ga ne otvara.
`ReadBinaryData`	Čita binarne podatke iz određenog polja u ključu.
`ReadBool`	Čita logičku vrednost iz određenog polja u ključu.
`ReadDateTime`	Čita podatke o datumu i vremenu iz određenog polja u ključu. Povratna vrednost je instanca klase `TDateTime`. Da biste pročitali samo podatak o datumu, koristite `ReadDate`. Da biste pročitali samo podatak o vremenu, korisitite `ReadTime`.
`ReadFloat`	Čita vrednost smeštenu u formatu sa pokretnim zarezom iz određenog polja u ključu.
`ReadInteger`	Čita celobrojnu vrednost iz određenog polja u ključu.
`ReadString`	Čita string iz određenog polja u ključu.

nastavlja se

Tabela 14.1: Primarne metode klase `Tregistry` *nastavak*

Metoda	Opis
`SaveKey`	Snima dati ključ na disk, tako da on kasnije može biti učitan korišćenjem `LoadKey` metode. Ne biste trebali da snimate više od 2KB (2048 baj tova) podatka pomoću ove metode.
`ValueExists`	Vraća `True`, ukoliko dato polje postoji.
`WriteBinaryData`	Upisuje binarne podatke u dati ključ. Koristite ovu metodu za smeštanje nizova i drugih tipova binarnih podataka.
`WriteBool`	Upisuje logički podatak u dato polje. Vrednost tog podatka se konvertu je u ceo broj i tek posle toga se upisuje u vrednost u ključu.
`WriteDateTime`	Upisuje `TDateTime` objekat u dato polje. Da biste upisali samo podatak o datumu, koristite `WriteDate`. Da biste upisali samo podatak o vremenu, koristite `WriteTime`. `TDateTime` objekat se pre upisivanja konvertuje u binarni podatak.
`WriteFloat`	Upisuje podatak u pokretnom zarezu u dato polje posle konverzije u binarni podatak.
`WriteInteger`	Upisuje celobrojni podatak u dato polje.
`WriteString`	Upisuje string u dato polje.

Iako je puno metoda navedeno u tabeli 14.1, mnoge od njih rade istu operaciju, ali sa drugačijim tipovima podataka. Kada naučite kako da koristite jednu od ovih metoda, znaćete kako da koristite i ostale. Primetite da neke od ovih metoda konvertuju podatke u binarni oblik, pa ih tek onda upisuju u Registry.

Korišćenje klase `TRegistry`

Korišćenje klase `TRegistry` je prilično jednostavno. Uglavnom će se sav Vaš posao oko Registry-ja sastojati u izvršavanju sledeća četiri koraka:

1. Kreiranje instance klase `TRegistry`.
2. Kreiranje, ukoliko je potrebno, i otvaranje ključa, korišćenjem `OpenKey` metode.
3. Čitanje i upisivanje podataka korišćenjem jedne, ili više `Read`, ili Write metoda.
4. Oslobađanje instance `TRegistry` klase.

Pre nego što budete mogli da korisite klasu `TRegistry`, morate dodati `Registry` celinu u uses deo celine Vaše forme.

Sledeći kod ilustruje ove korake:

```
procedure TForm1.FormCreate(Sender: TObject);
var
  Reg       : TRegistry;
  KeyGood   : Boolean;
  Top       : Integer;
  Left      : Integer;
  Width     : Integer;
  Height    : Integer;
begin
  Reg := TRegistry.Create;
  try
    KeyGood := Reg.OpenKey(
      'Software\SAMS\Delphi 4 in 21 Days', False);
    if not KeyGood then begin
      Top := Reg.ReadInteger('Top');
      Left := Reg.ReadInteger('Left');
      Width := Reg.ReadInteger('Width');
      Height := Reg.ReadInteger('Height');
      SetBounds(Left, Top, Width, Height);
    end;
  finally
    Reg.Free;
  end;
end;
```

Ovaj kod otvara ključ i iz njega čita podatke o koordinatama vrha i leve ivice prozora, kao i podatke o visini i širini forme. Zatim se poziva `SetBounds` funkcija koja pomera prozor, ili menja njegovu veličinu. Primetite da se povratna vrednost funkcije OpenKey smešta u logičku promenljivu. `OpenKey` vraća `True`, ukoliko je ključ uspešno otvoren i False, ukoliko nije. Ukoliko ključ jeste uspešno otvoren, iz njega se iščitavaju vrednosti pojedinačih polja. Uvek biste trebali da ispitate vrednost koju vraća `OpenKey` metoda. Primetite da prethodni kod koristi `try...finally` konstrukciju, kako bi osigurao da se vrednost `Reg` promenljive uspešno oslobodi, pre nego što funkcija završi sa radom.

VCL će generisati izuzetak ukoliko čitanje podataka, ili njihov upis u `Registry` ne uspe. Ukoliko pokušate da pročitate vrednost iz neotvorenog ključa, dobićete izuzetak. Budite spremni da obradite ovaj izuzetak, ili proverite povratnu vrednost funkcije `OpenKey` pre čitanja, ili upisivanja podataka u ključ.

Konačno, primetite da se u prethodnom kodu, ključ nigde eksplicitno ne zatvara. Ukoliko zaboravite da zatvorite ključ, u trenutku brisanja `Reg` objekta će biti pozvan destruktor i ključ će biti zatvoren. Sledi da eksplicitno korišćenje `CloseKey` metode nije neophodno.

`OpenKey` funkcija automatski dodaje vrednost `RootKey` osobine (podrazumevano, `HKEY_CURRENT_USER`) na početak stringa koji joj se prenosi kao parametar. To znači da ne morate navoditi koren prilikom otvaranja ključa.

Upisivanje podataka u Registry

Do sada smo govorili samo o čitanju podataka iz Registry-ja. Možda Vam je to izgledalo neobično, premda sam govorio o čitanju, pre nego što sam Vam pokazao kako da upišete podatke, ali to zaista nije važno, pošto je upisivanje podataka u Registry takođe vrlo jednostavno:

```
procedure TForm1.FormDestroy(Sender: TObject);
var
  Reg       : TRegistry;
begin
  Reg := TRegistry.Create;
  try
    Reg.OpenKey(
      'Software\SAMS\Delphi 4 in 21 Days', True);
    Reg.WriteInteger('Top', Top);
    Reg.WriteInteger('Left', Left);
    Reg.WriteInteger('Width', Width);
    Reg.WriteInteger('Height', Height);
  finally
    Reg.Free;
  end;
end;
```

Ovaj kod jednostavno otvara ključ i upisuje vrednosti `Top`, `Left`, `Width` i `Height` osobina forme u ključ korišćenjem `WriteInteger` metode. Primetite da je poslednji parametar koji se prenosi `OpenKey` metodi `True`. To znači da ključ treba da bude kreiran, ukoliko se već ne nalazi u bazi. Ukoliko podatke upisujete na ovaj način, nikada nećete morati da pozivate `CreateKey` metodu.

To je praktično sve što je potrebno znati da bi se moglo čitati i pisati u `Registry` bazu. Druge metode za čitanje i pisanje podataka su u stvari samo varijacije metoda koje smo već koristili.

Korišćenje Registry-ja za smeštanje podataka

Listing 14.3 sadrži glavnu celinu programa pod imenom `RegTest` koji koristi `Registry` za smeštanje bitnih podataka. Ovaj program smešta nekoliko vrednosti u `Registry`: poslednju veličinu i poziciju prozora, poslednji fajl sa kojim se radilo, status prozora (normalan, minimizovan, ili maksimizovan), poslednji korišćeni direktorijum, poslednji filter koji je korišćen prilikom otvaranja fajla i datum i vreme kada je program poslednji put pokrenut. Da biste izbrisali ključeve koje kreira `RegTest` program, dva puta kliknite na `DeleteKey` taster, koji se nalazi na glavnoj formi programa. (pogledajte sliku 14.4 koja je prikazana nešto kasnije u ovom poglavlju). Ovaj program se nalazi i u izvornom kodu iz knjige.

Listing 14.3: RegTestU.pas

```
unit RegTestU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs,  Menus, StdCtrls, ExtCtrls, Registry;

type
  TForm1 = class(TForm)
    Panel1: TPanel;
    Label1: TLabel;
    DeleteKey: TButton;
    Panel2: TPanel;
    Label2: TLabel;
    Label3: TLabel;
    TimeLabel: TLabel;
    DateLabel: TLabel;
    MainMenu: TMainMenu;
    File1: TMenuItem;
    FileOpen: TMenuItem;
    FileExit: TMenuItem;
    OpenDialog: TopenDialog;
    procedure FormCreate(Sender: TObject);
    procedure FileOpenClick(Sender: TObject);
    procedure FormDestroy(Sender: TObject);
    procedure FileExitClick(Sender: TObject);
    procedure DeleteKeyClick(Sender: TObject);
  private
    { Private declarations }
    KeyDeleted : Boolean;
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}

procedure TForm1.FormCreate(Sender: TObject);
var
  Reg       : TRegistry;
  KeyGood   : Boolean;
  DT        : TDateTime;
  Top       : Integer;
```

nastavlja se

Listing 14.3: RegTestU.pas *nastavak*

```
    Left      : Integer;
    Width     : Integer;
    Height    : Integer;
  begin
    { Initialize the KeyDeleted variable to False. This
    { variable is used if the user deletes the key from
    { the program. See the MainFormClose function. }
    KeyDeleted := False;

    { Create a TRegistry object to access the registry. }
    Reg := TRegistry.Create;

    try
      { Open the key. }
      KeyGood := Reg.OpenKey(
        'Software\SAMS\Delphi 4 in 21 Days', False);

      { See if the key is open. If not, then this is the first
      { time the program has been run so there's nothing to do. }
      { If the key is good then read all of the data items
      { pertinent to application startup. }
      if KeyGood then begin
        Top := Reg.ReadInteger('Top');
        Left := Reg.ReadInteger('Left');
        Width := Reg.ReadInteger('Width');
        Height := Reg.ReadInteger('Height');
        SetBounds(Left, Top, Width, Height);
        WindowState :=
          TWindowState(Reg.ReadInteger('WindowState'));

        { The TDateTime class is a handy item to have around
        { if you are doing date and time operations. }
        DT := Reg.ReadDate('Date and Time');
        DateLabel.Caption := DateToStr(DT);
        TimeLabel.Caption := TimeToStr(DT);
      end;
    finally
      Reg.Free;
    end;
  end;

  procedure TForm1.FileOpenClick(Sender: TObject);
  var
    Reg      : TRegistry;
  begin
    { This function displays the File Open dialog but
    { doesn't actually open a file. The last path, filter,
    { and filename are written to the registry when the
    { user presses OK. }
```

Listing 14.3: RegTestU.pas

```
    { Create a TRegistry object to access the registry. }
    Reg := TRegistry.Create;

    try
      { Open the key. }
      Reg.OpenKey('Software\SAMS\Delphi 4 in 21 Days', True);

      { Read the values. Check if the value exists first. }
      if Reg.ValueExists('LastDir') then
        OpenDialog.InitialDir := Reg.ReadString('LastDir');
      if Reg.ValueExists('LastFile') then
        OpenDialog.FileName := Reg.ReadString('LastFile');
      if Reg.ValueExists('FilterIndex') then
        OpenDialog.FilterIndex := Reg.ReadInteger('FilterIndex');

      { Display the File Open dialog. If the user presses OK then
      { save the path, filename, and filter to the registry. }
      if OpenDialog.Execute then begin
        Reg.WriteString('LastDir',
          ExtractFilePath(OpenDialog.FileName));
        Reg.WriteString('LastFile',
          ExtractFileName(OpenDialog.FileName));
        Reg.WriteInteger
          ('FilterIndex', OpenDialog.FilterIndex);
      end;
    finally
      Reg.Free;
    end;
  end;

  procedure TForm1.FormDestroy(Sender: TObject);
  var
    Reg       : TRegistry;
  begin
    { If the user pressed the button to the key then
    { we don't want to write out the information. }
    if KeyDeleted then
      Exit;

    { Create a TRegistry object to access the registry. }
    Reg := TRegistry.Create;

    try
      { Open the key. }
      Reg.OpenKey(
        'Software\SAMS\Delphi 4 in 21 Days', True);

      { Write out the values. }
```

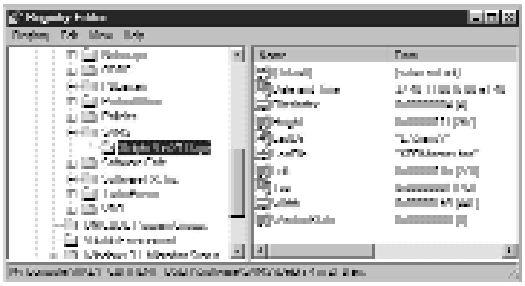
nastavlja se

Listing 14.3: `RegTestU.pas` *nastavak*

```
      Reg.WriteInteger('WindowState', Ord(WindowState));
      if WindowState <> wsMaximized then begin
        Reg.WriteInteger('Top', Top);
        Reg.WriteInteger('Left', Left);
        Reg.WriteInteger('Width', Width);
        Reg.WriteInteger('Height', Height);
      end;
      Reg.WriteDate('Date and Time', Now);
    finally
      Reg.Free;
    end;
  end;

  procedure TForm1.FileExitClick(Sender: TObject);
  begin
    Close;
  end;

  procedure TForm1.DeleteKeyClick(Sender: TObject);
  var
    Reg       : TRegistry;
  begin
    { The user pressed the Key button so delete the key. }
    { Set a flag so that we don't re-create the key when
    { the program closes. }
    Reg := TRegistry.Create;
    try
      Reg.DeleteKey('Software\SAMS');
      KeyDeleted := True;
    finally
      Reg.Free;
    end;
  end;

  end.
```

Proučavanjem listinga ovog programa i njegovim pokretanjem možete naučiti dosta o korišćenju Registry-ja u svojim programima. Slika 14.4 prikazuje `RegTest` program u fazi izvršavanja. Slika 14.5 prikazuje ključ u Registry-ju koji je kreirao ovaj program.

Slika 14.4
Program `RegTest` u fazi izvršavanja



Slika 14.5
Registry Editor u kome je otvoren ključ koji je napravio program `RegTest`

NAPOMENA Klasa `TRegIniFile` je specijalizovana klasa za rad sa `Registry` bazom i može se koristiti u programima koji prelaze sa korišćenja .ini fajlova na korišćenje Registry-ja. Da biste naučili kako se radi sa ovom klasom, pogledajte `TRegIniFile` stavku u Delphi-jevom Help-u.

NAPOMENA Iako neka pravila nalažu da treba koristiti `Registry` za smeštanje podataka koji su bitni za program, Vi i dalje možete koristiti .ini fajlove. Jedna prednost, kod korišćenja `.ini` fajlova, je što i manje iskusni korisnici mogu menjati ove podatke korišćenjem običnog editora teksta (istini za volju, to ponekad ume da bude i mana). Da bi rad sa `.ini` fajlovima u Delphi programima bio jednostavniji, VCL sadrži klasu pod imenom `TIniFile`. Obavezno pregledajte ovu klasu ako nameravate da koristite `.ini` fajlove u svojim programima.

Posebna obrada poruka

U danu 11 sam govorio o Windows-ovim porukama u okviru diskusije o WinSight programu. U Delphi-ju se, u najvećem broju slučajeva, obrada poruka vrši kroz korišćenje događaja, tako da ne predstavlja poseban problem. Kao što sam rekao i ranije, događaj se obično generiše kao odgovor na poruku koju Windows šalje programu. Ipak, postoje slučajevi kada biste želeli da sami vršite obradu poruka. Postoje barem dve situacije u kojima biste želeli da obrađujete poruke nezavisno od Delphi-jevog sistema poruka:

- Obrada Windows-ovih poruka koje VCL ne obrađuje,
- Obrada korisnički-definisanih poruka.

Samostalna obrada poruka zahteva nešto više programiranja. To je ono što ćete naučiti u ovom odeljku.

Detaljnije upoznavanje sa Windows-ovim porukama

Na koji način se Windows-ove poruke šalju? Neke poruke šalje sam Windows da bi naložio prozoru da nešto uradi, ili da bi obavestio prozor da se nešto desilo. U drugim slučajevima, poruke šalje programer, ili biblioteka koju on koristi, u ovom slučaju VCL. Bez obzira na to ko šalje poruku, možete biti sigurni da puno poruka proleti kroz sistem u svakoj milisekundi.

Tipovi poruka

Postoje dva osnovna tipa poruka:

- Komandne poruke - pokreću neku akciju, bilo u samom Windows-u, ili u određenoj kontroli.
- Notifikacione poruke - šalje ih sam Windows i obaveštavaju Vas da se nešto dogodilo.

Da bismo pokazali razliku između ova dva tipa poruka, skoncentrišimo se, na trenutak, na klasičnu Edit kontrolu. Klasična Edit kontrola ima skoro 80 komandnih i oko 20 notifikacionih poruka. Da li ste iznenađeni? To je istina, i ne zaboravite da je Edit kontrola samo jedna od Windows-ovih kontrola kojih ima puno.

Može se reći da sam Vas lagao u danu 5 kada sam govorio o događajima (dobro, ne baš lagao). Tada sam rekao da postoji preko 200 poruka koje Windows može da pošalje programu, što je u principu tačno. Ali, kada uzmete u obzir i poruke koje su specifične za kontrole, kao i poruke koje se mogu porslati glavnim prozorima, taj broj se povećava na 700. Nabrojane su komandne poruke dok notifikacione nisu. Verujem da ćete posle ovoga imati više poštovanja prema VCL-u.

Programer koji pravi Windows program u C-u mora da šalje mnogo poruka, kako bi izvršio neku operaciju. Na primer, da biste dobili dužinu trenutno izabranog teksta u Edit kontroli morate uraditi sledeće:

```
int start;
int end;
long result = SendMessage(hWndEdit, EM_GETSEL, 0, 0);
start = LOWORD(result);
end = HIWORD(result);
int length = end - start;
```

Na ovaj način C programeri provode svoje dane. Za razliku od gornjeg pristupa problemu, VCL radi na nešto drugačiji način:

```
Length := Edit.SelLength;
```

Koji način Vam se više sviđa? Bez obzira na to ko šalje poruku, komandne poruke koriste i programeri i sam Windows.

Notifikacione poruke, sa druge strane, šalje samo Windows. Pomoću notifikacionih poruka Vas Windows obaveštava da se nešto dogodilo. Na primer, poruka `EN_CHANGE` se šalje svaki put kada se sadržaj Edit kontrole promeni. U okviru `Edit`, `MaskEdit` i `Memo` VCL komponenti je realizovan `OnChange` događaj, koji se generiše svaki put kada se sadržaj neke od ovih kontrola promeni. Programeri koji koriste tradicionalan način za pravljenje Windows poruka moraju da presreću ove poruke i da ih obrađuju - a to nas dovodi do naše sledeće teme: parametri poruka.

WPARAM, LPARAM i razbijanje poruke

Svaka Windows-ova poruka ima dva parametra: `WPARAM` (skraćenica od word parameter) i `LPARAM` (skraćenica od long parameter).

U 32-bitnom Windows-u, `WPARAM` i LPARAM su 32-bitne vrednosti. U 16-bitnom Windows-u, WPARAM je 16-bitna vrednost (Word) dok je `LPARAM` 32-bitna vrednost (LongInt). To je bila istorijska lekcija o `WPARAM` i `LPARAM` imenima. Sada ćemo se vratiti na program...

Ova dva parametra se mogu porediti sa parametrima koji se šalju funkciji. Kada program primi Windows-ovu poruku, potrebno je analizirati ove parametre da bi se dobila informacija koja je posebna za svaku poruku. Na primer, program prima `WM_LBUTTONDOWN` notifikacionu poruku, kada se pritisne levi taster miša, dok se pokazivač miša nalazi na radnoj površini prozora. U slučaju `WM_LBUTTONDOWN` poruke, `WPARAM` sadrži specijalan kod koji pokazuje da li je još neki taster miša bio pritisnut i, takođe, koji su tasteri na tastaturi u tom trenutku bili pritisnuti. `LPARAM` sadrži koordinate kurzora miša u trenutku kada je pritisnut taster. X-koordinata se nalazi u nižoj reči a Y-koordinata u višoj reči `LPARAM` parametra.

Da biste dobili ove informacije, morate razbiti poruku i videti šta se unutra nalazi. Kod za razbijanje `WM_LBUTTONDOWN` poruke može biti ovakav:

```
procedure MessageCracker(wParam, lParam : Integer);
var
  Shift, X, Y : Integer;
begin
  Shift := wParam;
  X := LOWORD(lParam);
  Y := HIWORD(lParam);
  { Code that does something with Shift, X, and Y. }
end;
```

Ovo je krajnje jednostavan primer. Ipak, pokazuje šta se sve dešava u trenutku kada Windows program obrađuje poruku.

NAPOMENA Pošto se X i Y koordinate nalaze u istoj promenljivoj (`lParam`), iz nje se moraju pojedinačno vaditi. Pri tome se koriste `HIWORD` i `LOWORD`. U svetu C/C++ programiranja, `HIWORD` i `LOWORD` su Windows makroi. U Object Pascal-u, `LOWORD` je tip koji je isti kao i Word, dok je HIWORD funkcija koja vadi levih 16 bitova iz 32-bitne promenljive. Kada se `LOWORD` koristi kao u prethodnom primeru, vadi desnih 16 bitova.

Biće Vam drago kada shvatite da VCL razbija poruke za svaki VCL događaj. Na primer, ukoliko napravite proceduru za obradu OnMouseDown događaja, Delphi, u Vašem kodu, generiše funkciju koja izgleda ovako:

```
procedure TForm1.FormMouseDown(Sender: TObject; Button: TMouseButton;
  Shift: TShiftState; X, Y: Integer);
begin
  if Button = mbLeft then begin
    { Put your code here. }
  end;
end;
```

Kao što možete videti, procedura za obradu događaja koju je generisao Delphi sadrži informacije koje su Vama potrebne. VCL će razbiti `WPARAM` i `LPARAM` umesto Vas i proslediće Vam ih u delovima, kako biste Vi mogli lakše da ih koristite. Ista stvar se dešava sa svakom porukom za koju VCL generiše događaj.

NAPOMENA U toku razbijanja poruka vezanih za taster miša VCL ide jedan korak dalje. Windows može da generiše tri različite poruke, kada se klikne na taster miša: jednu za levi taster, jednu za srednji taster (ukoliko postoji) i jednu za desni taster miša. VCL obrađuje sve tri poruke u okviru istog događaja. `TMouseButton` parametar procedure za obradu `OnMouseDown` događaja Vam govori koji od tri tastera je pritisnut.

Pošto ste razumeli prethodni primer, hajde da se pozabavimo slanjem poruka.

Dva načina za slanje poruka

Windows API sadrži dve funkcije za slanje poruka: `PostMessage` i `SendMessage`. `PostMessage` funkcija postavlja poruku u Windows-ov red za čekanje i vraća kontrolu programu. Ova funkcija jednostavno prenosi poruku Windows-u i dozvoljava programu iz koga je pozvana da nastavi sa radom. `PostMessage` vraća 1, ukoliko je uspela da prenese poruku i `0`, ukoliko je pokušaj prenošenja poruke propao. (Praktično, jedini razlog zbog kojeg bi `PostMessage` vratila `0` je pokušaj da se poruka pošalje nepostojećem prozoru, ili prozoru koji iz bilo kog razloga ne prima poruke).

`SendMessage` funkcija, sa druge strane, šalje poruku Windows-u i čeka dok se poruka ne obradi. Tek pošto se poruka obradi, kontrola se vraća programu iz kog je `SendMessage` pozvana. Povratna vrednost ove funkcije zavisi od tipa poslate poruke. Ponekad je jedini razlog zbog kojeg treba koristiti `SendMessage`, umesto `PostMessage`, baš povratna vrednost poruke.

NAPOMENA Razlika između situacija u kojima ćete koristiti `PostMessage` i onih u kojima ćete koristiti `SendMessage` je jedva primetna. Na primer, zbog vremena koje je potrebno Windows-u da obradi poruku, korišćenje `SendMessage` u nekim situacijama ne bi dalo pravi rezultat. Tada biste morali da koristite `PostMessage`. Takođe, korišćenje `PostMessage` u nekim situacijama, može biti pogrešno, i tada ćete morati da koristite `SendMessage`. Iako ovo pitanje nije nešto što bi sada trebalo da Vas muči, nemojte ga zanemariti.

Možete koristiti i `PostMessage` i `SendMessage` u svojim Delphi programima. Na primer, da biste sebi poslali poruku, uradite sledeće:

```
PostMessage(Handle, WM_QUIT, 0, 0);
```

I kod `PostMessage` i kod SendMessage, prvi parametar je uvek pokazivač na prozor kome šaljete poruku. U ovom slučaju šaljete poruku glavnoj formi programa (pod uslovom da je ovaj kod napisan u celini glavne forme programa).

Pored funkcija koje obezbeđuje Windows API, VCL dodaje metodu pod imenom `Perform`, koju možete koristiti za slanje poruka bilo kom VCL prozoru. Perform zaobilazi Windows-ov sistem poruka i šalje poruku direktno do procedure za obradu poruka datog VCL prozora. Perform ekvivalent gornjeg primera je:

```
Perform(WM_QUIT, 0, 0);
```

Možete koristiti bilo koju od pomenute tri funkcije za slanje poruka drugim programima, ili drugim prozorima u Vašem programu.

Obrada događaja

Već ste naučili kako se obrađuju VCL događaji. Ipak, kratko podsećanje neće smetati. Kada određena komponenta primi poruku od Windows-a, proverava da li je toj poruci dodeljena procedura za obradu događaja. Ukoliko jeste, VCL će je pozvati i izvršiti. Ukoliko nije, poruka će biti obrađena na podrazumevani način. Možete obraditi one poruke koje želite, a ostale jednostavno ignorisati.

Ono što se dešava u proceduri za obradu događaja zavisi od mnogo faktora: od same poruke koja se obrađuje, od toga kako će Vaš program odgovoriti na poruku, od toga da li ćete promeniti pristiglu poruku, ili ćete koristiti proceduru za obradu događaju samo radi obaveštenja da se nešto desilo. Kako budete dublje zalazili u Windows programiranje, videćete da postoje, praktično, hiljade stvari koje možete uraditi da biste obradili događaj.

Uglavnom ćete koristiti procedure za obradu događaja kao obaveštenje da se određeni događaj desio. Uzimite, na primer, `OnMouseDown` događaj. Ukoliko obradite `OnMouseDown` događaj, Vi zapravo tražite da budete obavešteni kada korisnik klikne na komponentu pomoću miša (zapamtite da su i forme komponente). Verovatno nećete menjati parametre poruke - vi samo želite da znate da se to desilo. Veliki broj Vaših procedura za obradu događaja će se koristiti samo u cilju obaveštavanja.

Ponekad ćete, ipak, želeti da izmenite jedan, ili više parametara poruke, pre nego što poruku prosledite dalje. Na primer, recimo da želite da se sadržaj određene `Edit` kontrole ispisuje samo velikim slovima. (Naravno, možete, postaviti `CharCase` osobinu `Edit` kontrole na `ecUpperCase`, ali tu mogućnost ćemo sada zanemariti.) Da biste preveli sve karaktere u velika slova, u svojoj proceduri za obradu `OnKeyPress` događaja biste mogli da uradite nešto ovako:

```
procedure TForm1.Edit1KeyPress(Sender: TObject; var Key: Char);
begin
  if Key >= 'a' then
    Dec(Key, 32);
end;
```

U ovom slučaju Vi zapravo menjate jedan deo poruke, pre nego što ona stigne do VCL-a, gde će se konačno obraditi. Baš zbog ovakvih potreba, parametri se procedurama za obradu događaja prenose po referenci.

Delphi Vam ne dozvoljava da menjate parametre koje ne biste smeli, tako što ih prenosi po vrednosti, a ne po referenci. To, takođe, važi i za sve poruke čijom izmenom parametara ne biste postigli nikakav efekat. Na primer, `OnMouseDown` događaj služi samo kao obaveštenje. Nema smisla menjati `Button`, `Shift`, `X`, ili `Y` parametre, tako da su svi preneseni po vrednosti.

Da li ćete menjati jedan, ili više parametara, u potpunosti zavisi od same poruke i od toga šta Vi želite da uradite sa tom porukom. U principu, često ćete dolaziti u situacije kada je izmena odgovarajuće poruke neophodna, kako bi Windows reagovao na pravi način. U ostalim slučajevima, nećete menjati parametre, samo ćete ih analizirati da biste saznali šta se dešava sa Vašim programom. Pošto su mogućnosti nebrojene, moraću da napustim dalje objašnjavanje parametara poruka i da Vam za domaći zadatak dam da se sami pozabavite njima.

Obrada ostalih Windows-ovih poruka

Sigurno je da ćete nekad imati potrebu da obradite Windows-ovu poruku za koju VCL ne obezbeđuje događaj. Kada se nađete u takvoj situaciji, želećete da znate kako se obrađuju takve poruke. O tome ćemo govoriti u ovom odeljku.

VCL obezbeđuje događaje za najčešće korišćene Windows-ove poruke. Očigledno, VCL ne može obezbediti događaje za svih 700 poruka. Teorija 80/20 kaže da, između ostalog, 20 posto ljudi uradi 80 posto posla. Isto verovatno važi i za Windows-ove poruke. VCL obezbeđuje podršku za 20 posto poruka, i tih 20 posto ćete Vi koristiti u 80 posto slučajeva. Ipak, postoji mnogo poruka za koje VCL ne obezbeđuje događaje i Vi treba da znate kako treba obrađivati te poruke.

Biće Vam drago kada čujete da možete obrađivati bilo koju Windows-ovu poruku. Potrebno je samo da znate kako. Pošto naučite osnove, obrada bilo koje poruke će biti samo varijacija na temu. Mehanizam koji koristi Delphi prilikom obrade poruka za koje nije obezbeđen VCL događaj je ključna reč `message`. Ova ključna reč se koristi za povezivanje određene Windows-ove poruke i neke metode u Vašem kodu. Kada Vaš prozor primi poruku, poziva se ta metoda. Liči na događaje, zar ne? Sasvim sigurno, postoje neke sličnosti.

Obrada poruka

Obrada poruka na ovom nivou je vrlo jednostavna:

1. Dodajte dekleraciju metode za obradu poruke u dekleraciju klase.
2. Dodajte definiciju procedure za obradu poruke u `implementation` deo.

Sledi primer dekleracije procedure koja obrađuje `WM_ERASEBKGND` poruku:

```
procedure WmEraseBkgnd(var Msg : TWMEraseBkgnd); message WM_ERASE-
BKGND;
```

Primetite da se `message` ključna reč nalazi na kraju dekleracije metode. Iza ključne reči `message` se nalazi ime Windows-ove poruke koju procedura treba da obradi. Primetite da je parametar metode `TWMEraseBkgnd` struktura. VCL sadrži strukture za razbijanje poruka za skoro svaku Windows-ovu poruku. Ime strukture je isto kao i ime Windows-ove poruke, pri čemu je na početku dodato slovo `T` i izbačen je znak za podvlačenje.

Kao što možete videti, Windows-ova poruka `WM_ERASEBKGND` se prevodi u strukturu pod imenom `TWMEraseBkgnd`. Ova struktura se prenosi pravo do procedure za obradu poruke (više o ovome u sledećem odeljku). Samu metodu možete nazvati proizvoljnim imenom, ali je dobra ideja korišćenje tradicionalne forme koja je prikazana na prethodnom listingu.

Strukture za razbijanje poruka su definisane u VCL-ovoj celini pod imenom `Messages.pas`. Pregledajte `Messages` celinu, ukoliko Vam je potrebna informacija o određenoj strukturi za razbijanje poruke.

Da bi Vam prethodna priča bila jasna, trebali biste da vidite kompletnu dekleraciju klase koja će obraditi posebnu poruku. Listing 14.4 prikazuje tipičnu dekleraciju klase glavne forme koja koristi obradu posebnih poruka.

Listing 14.4: `Message.h`

```
TMainForm = class(TForm)
    ShowTBoix: TButton;
    GroupBox1: TGroupBox;
    Hatched: TCheckBox;
    LetVCLHandle: TCheckBox;
    Instructions: TButton;
    procedure InstructionsClick(Sender: TObject);
    procedure ShowTBoixClick(Sender: TObject);
    procedure HatchedClick(Sender: TObject);
  private
    { Private declarations }
    procedure WmNCHitTest(var Msg : TWMNCHitTest);
      message WM_NCHITTEST;
    procedure WmEraseBkgnd(var Msg : TWMEraseBkgnd);
      message WM_ERASEBKGND;
    procedure WmGetMinMaxInfo(var Msg : TWMGetMinMaxInfo);
      message WM_GETMINMAXINFO;
    procedure MyMessage(var Msg : TMessage);
      message My_Message;
  public
    { Public declarations }
  end;
```

Primetite da su metode za dekleraciju procedura za obradu poruka smeštene u private deo. Postavio sam message ključne reči u linije koje se nalaze neposredno ispod dekleracije procedura zbog boljeg preloma linija. Vi ćete, naravno, ključnu reč `message` staviti neposredno posle dekleracije procedure (u istoj liniji). Na kraju krajeva, prevodilac ne pravi razliku između ova dva načina zapisivanja dekleracija, tako da možete sami da izaberete način koji Vam se više sviđa. Ne dajte da Vas zbuni dekleracija `WMyMessage` metode. Ovo je procedura za obradu korisnički-definisane poruke. O tome ću govoriti malo kasnije.

Procedure za obradu poruka

Procedura za obradu poruka je metoda koje se poziva svaki put kada program primi poruku na koju ona odgovara. Procedura za obradu poruka prima jedan parametar: strukturu za razbijanje poruka, koju sam pomenuo ranije. Procedura za obradu `WM_ERASEBKGND` poruke bi trebala da izgleda ovako:

```
procedure TMainForm.WmEraseBkgnd(var Msg: TWMEraseBkgnd);
begin
  { Your message handling code here. }
end;
```

Kao što sam rekao, struktura za razbijanje poruke sadrži sve parametre koji su potrebni za uspešnu obradu poruke. Struktura za razbijanje `WM_ERASEBKGND` poruke izgleda ovako:

```
TWMEraseBkgnd = record
  Msg: Cardinal;
  DC: HDC;
  Unused: Longint;
  Result: Longint;
end;
```

Sve strukture za razbijanje poruka imaju dva zajednička člana: `Msg` i `Result`. Član `Msg` sadrži poruku koja je poslata i njega koristi sam VCL. Vi ne biste trebali da obraćate pažnju na njega.

Član `Result` je, sa druge strane, veoma bitan. Koristi se za postavljanje povratne vrednosti poruke koju obrađujete. Povratna vrednost se razlikuje od poruke do poruke. Na primer, povratna vrednost za poruku `WM_ERASEBKGND` je `True` (ne-nula), ukoliko želite da obrišete pozadinu pre crtanja, ili `False` (nula), ukoliko ne želite da obrišete pozadinu. (Pogledajte Win32 API Help da biste odredili kojeg je tipa povratna vrednost određene poruke). Ukoliko je potrebno, možete postaviti vrednost člana `Result`:

```
procedure TMainForm.WmEraseBkgnd(var Msg: TWMEraseBkgnd);
begin
  { Do some stuff. }
  Message.Result := 0;
end;
```

Ova dva člana su zajednička za sve strukture za razbijanje poruka. Ostali članovi se razlikuju od poruke do poruke.

Ponekad će biti potrebno da pozovete podrazumevanu proceduru za obradu poruke, pošto izvršite samostalnu obradu. Tada je potrebno pozvati `DefaultHandler`. Na primer, možda biste želeli da obrišete pozadinu samo u nekim slučajevima, dok u drugim slučajevima ona treba da ostane onakva kakva jeste. Ukoliko Vi ne obrišete pozadinu, treba da dozvolite VCL-u da je obriše na podrazumevani način. Dakle, potrebno je da uradite ovo:

```
procedure TMainForm.WmEraseBkgnd(var Msg: TWMEraseBkgnd);
begin
  if LetVCLHandle.Checked then begin
    DefaultHandler(Msg);
    Exit;
  end;
  { Do some other drawing. }
  Msg.Result := 1;
end;
```

U durgim situacijama biste koristili `DefaultHandler` da biste izvršili neku podrazumevanu operaciju. Da li ćete DefaultHandler pozvati pre, ili posle samostalne obrade zavisi od toga šta želite da uradite.

Korisnički-definisane poruke

Pored klasičnih Windows-ovih poruka, sam Windows Vam dozvoljava da definišete sopstvene poruke.

NOVI TERMIN Korisnički-definisana poruka je, u stvari, privatna poruka koju možete slati sebi, ili drugim prozorima u Vašem programu.

Slanje i prihvatanje korisnički-definisanih poruka je praktično isto kao i kod klasičnih Windows-ovih poruka. Ipak, pre nego što budete mogli da radite sa porukom, moraćete da je definišete. Korisnički-definisanu poruku možete definisati na sledeći način:

```
const
  My_Message = WM_USER + 1;
```

Ovaj kod deklariše korisnički-definisanu poruku pod imenom `My_Message`.

NAPOMENA Simbol `WM_USER` je poseban simbol koji označava početak niza brojeva koji se mogu koristiti za korisnički-definisane poruke. Možete koristiti bilo koji broj do WM_USER+31,743. Ja, obično, koristim nešto u opsegu `WM_USER+1` do WM_USER+100. Sam broj koji ćete koristiti nije bitan, ali pazite da ne dodelite dvema porukama isti broj.

Ukoliko ponovo pogledate listing 14.4, primetićete dekleraciju korisnički-definisane poruke. Pošto ste definisali samu poruku, možete deklarisati proceduru za obradu te poruke i to na sledeći način:

```
procedure MyMessage(var Msg : TMessage); message My_Message;
```

Primetite da je tip strukture za razbijanje poruke koja se prenosi ovoj proceduri `TMessage`. Ovo je osnovni tip strukture za razbijanje poruka. Definisan je na sledeći način:

```
TMessage = record
  Msg: Cardinal;
  WParam: Longint;
  LParam: Longint;
  Result: Longint);
end;
```

NAPOMENA Prava dekleracija `TMessage` izgleda malo drugačije ali, suštinski, prikazana struktura je korektna.

Kada pošaljete korisnički definisanu poruku možete, u okviru `WParam` i `LParam` članova, poslati proizvoljne vrednosti. Recimo da želite da pošaljete korisnički-definisanu poruku, koja će obavestiti da je došlo do greške sa kodom 124 u 1019-oj iteraciji petlje. Poziv `Perform` metode bi izgledao ovako:

```
Res := MainForm.Perform(MyMessage, 124, 1019);
```

Dobro, poruka je sada definisana i poslata. Sada biste trebali da napišete kod koji će je obraditi. Procedura za obradu MYMESSAGE poruke bi mogla da izgleda ovako:

```
procedure TMainForm.WmMyMessage(var Msg: TMessage);
var
  S : string;
begin
  S := Format('Error #%d occurred on iteration number %d.',
    [Msg.WParam, Msg.LParam]);
  MessageDlg('Error Message', mtError, [mbOk], 0);
  Msg.Result := 1;
end;
```

Povratna vrednost od Perform će biti vrednost Result člana TMessage strukture. Na Vama je da odlučite da li će Vaša poruka slati parametre i da li će Vaša procedura za obradu poruke vraćati rezultat.

Izvorni kod iz knjige sadrži program pod imenom MsgTest, koji demonstrira obradu Windows-ovih poruka za koje VCL ne obezbeđuje događaje. Program, takođe, demonstrira i obradu korisnički-definisanih poruka. U ovom programu se nalazi i realizacija jednog od programerskih trikova koji se koristi u Windows-u: prozor se može pomerati po ekranu jednostavnim klikom na bilo koji deo korisničke površine i pomeranjem.

Zaključak

Danas ste obradili dosta gradiva. Počeli ste sa pregledom konktekstno-osetljive pomoći i načinom na koji se ona koristi. Zapamtite, pravljenje kontekstno-osetljive pomoći ne mora biti lako, ali je to nešto što biste svakako trebali da uradite. Posle toga ste se bavili obradom izuzetaka i načinom na koji se obrađuju VCL izuzeci. Takođe ste dobili i lep pregled Windows-ovog Registry-ja. Registry je nešto što treba koristiti za skladištenje podataka, koji su bitni za sam program. Poznavanjem rada Registry-ja možete napraviti puno detalja zbog kojih će Vaši korisnici biti zadovoljni. Konačno, završili smo ovaj dan pričom o obradi poruka za koje VCL ne obezbeđuje događaje. Sve u svemu, bio je ovo dugačak, ali i zahvalan dan.

Radionica

Radionica sadrži test pitanja koja Vam pomažu da učvrstite svoje razumevanje izložene materije i vežbe koje Vam pomažu da steknete iskustvo u onome što ste naučili. Možete pronaći odgovore na test pitanja u Dodatku A “Odgovori na test pitanja”.

Pitanja i odgovori

P **Pravljenje fajlova sa tekstom pomoći, korišćenjem programa za obradu teksta je zamorno. Šta predlažete kao zamenu?**

O Nabavite komercijalni, ili shareware program za rad sa fajlovima za pomoć. Ovi programi se brinu o onim stvarima koje Vas nerviraju, dok pravite fajlove sa tekstom pomoći. Pravljenje ovih fajlova je prava muka za većinu ljudi, ali korišćenjem dobrog programa za rad sa fajlovima za pomoć možete sebi olakšati posao. Takođe, primetite da sve više i više proizvođača koristi HTML sisteme za pomoć. Postoji mogućnost da HTML zameni klasični Windows-ov sistem za pomoć.

P **Da li moram da stavljam identifikatore konteksta u moj fajl sa tekstom pomoći.**

O To nije obavezno. Možete naterati Vaše korisnike da pristupaju sistemu za pomoć samo korišćenjem menija. Ipak, nije moguće napraviti kontekstno osetljiv sistem za pomoć bez korišćenja identifikatora konteksta.

P **Zašto bih trebao da se zamaram sa obradom izuzetaka?**

O Obradom izuzetaka možete imati bliži uvid u to šta se dešava kada u Vašem programu dođe do greške.

P **Prihvatio sam VCL izuzetak. Kako mogu da ispišem poruku koju ispisuje VCL u trenutku kada generiše izuzetak?**

O Pozovite `Application.ShowException` i VCL će prikazati poruku o grešci.

P **Da li moram da koristim Registry, kako bih skladištio podatke koji su bitni za moj program?**

O Ne. Možete koristiti `.ini` fajlove. Ipak, danas je `Registry` mesto gde bi programi trebali da upisuju ovakve podatke. Pomoću klase `TRegistry`, korišćenje Registry-ja je vrlo jednostavno, tako da ne biste trebali da izbegavate ovu pogodnost.

P **Dobijam izuzetak svaki put kada pokušam da napravim ključ i iskoristim `WriteString` za upisivanje podataka u ključ. Šta nije u redu?**

O Verovatno koristite `CreateKey` za kreiranje ključeva. CreateKey kreira ključ, ali ga ne otvara. Korisite `OpenKey` umesto `CreateKey` da biste kreirali i otvorili ključ.

P **Šta je korisnički-definisana poruka?**

O Korisnički-definisana poruka je poruka koju Vi definišete da biste je koristili u svom programu. U tome je i razlika u odnosu na klasične Windows-ove poruke koje su definisane na globalnom nivou.

P Šta bih trebao da uradim da bih izvršio podrazumevanu obradu određene Windows-ove poruke?

O Pozovite `DefaultHandler` metodu:

```
DefaultHandler(Msg);
```

Kviz

1. Kako se postavlja ime fajla sa tekstom pomoći koji će koristiti Vaš program?
2. Kako se pravi podrška za taster F1 u određenoj formi, ili dijalog-prozoru?
3. Koju metodu treba pozvati da bi se prikazao indeks tema koje se nalaze u Vašem fajlu sa tekstom pomoći.
4. Koje tipove objekata može generisati izuzetak?
5. Da li je dozvoljeno imati više od jednog `except` bloka u okviru istog `try` bloka?
6. Kako se generiše izuzetak?
7. Koja je podrazumevana vrednost `RootKey` osobine `TRegistry` klase?
8. Da li morate pozvati CloseKey kada završite rad sa ključem?
9. Koja je razlika između SendMessage i `PostMessage`?
10. Kako se zove VCL metoda pomoću koje se poruka šalje direktno do komponente?

Vežbe

1. Ispitajte raspoloživost programa za rad sa fajlovima za pomoć. Iako Vam ovo deluje kao čudan zadatak, može biti vrlo isplativ.
2. Napravite novi projekat. Dodajte neke komponente na glavnu formu. Dajte svakoj komponenti poseban `HelpContext` broj.
3. Dodelite fajl sa tekstom pomoći (bilo koji) Vašem projektu. Ukoliko imate program za rad sa fajlovima za pomoć, napravite jednostavan fajl sa tekstom pomoći koji će koristiti Vaš program. Pokrenite program i pritisnite F1, kada komponenta postane aktivna.
4. Izmenite `ScratchPad` program, tako da on koristi `Registry`. Sačuvajte ime i putanju do poslednjeg fajla sa kojim se radilo.
5. Izmenite `ScratchPad` program, tako da on koristi ime i putanju do fajla iz Registry-ja prilikom aktiviranja File Open i File Save dijalog prozora.

6. Napišite program koji sam sebi šalje korisnički-definisanu poruku, kada se klikne na taster. Prikažite prozor sa porukom kojim ćete potvrditi da je poruka primljena.
7. Dodajte proceduru za obradu Windows-ove `WM_MOVE` poruke u program iz vežbe 6. Kada se prozor pomeri, generišite zvučni signal i prikažite nove koordinate na formi.
8. Posebna vežba: Izmenite PictureViewer program iz dana 4 da biste prihvatili izuzetke koji se generišu kada korisnik pokuša da otvori fajl koji nije grafički.

Dan 15

COM i ActiveX

OLE, ActiveX, DCOM, VCL, CORBA, MTS..., očigledno je da danas, u softverskoj industriji, postoji veliki broj pojmova vezanih za arhitekturu komponenti. Objasniću jedan broj ovih pojmova dok ću ostale samo pomenuti. Objasniću šta ovi pojmovi znače i na taj način ću pokušati da rasvetlim svet COM i ActiveX tehnologija koji je, često, zbunjujući. Specifično, pokrivene su sledeće teme:

- Šta je COM?
- Kreiranje COM objekata
- Delphi-jev editor biblioteke tipova (Type Library Editor)
- Kreiranje ActiveX kontrola
- Kreiranje aktivnih formi
- Razvoj ActiveX kontrola

Lagao bih kada bih rekao da su COM, ActiveX i OLE tehnologije lake za razumevanje. Nisu. U početku mogu delovati veoma zbunjujuće. Ne mogu se detaljno objasniti u okviru samo jednog poglavlja. Moj cilj je da Vas snabdem dovoljnom količinom informacija da biste mogli da razumete pojmove koje ćete viđati u tekstu ovih dana. Takođe, proćićete kroz dobar trening iz oblasti kreiranja COM i ActiveX kontrola. Srećom, Delphi će za Vas odraditi veliki deo posla komunicirajući sa API (engl. Application Programming Interface) funkcijama.

Razumevanje COM

Ne mogu se razmatrati OLE i ActiveX tehnologije bez razmatranja COM-a, što je skraćenica od Component Object Model.

NOVI TERMIN COM (*Component Object Model*) je Microsoft-ova specifikacija za kreiranje i implementiranje komponenti koje se mogu ponovo koristiti.

"Komponente? Mislio sam da Delphi koristi VCL komponente." Svakako, VCL komponente su najkorisniji skup komponenti koji ćete koristiti pri radu sa Delphi-jem. Ipak, one nisu jedina mogućnost. U toku ovog dana, imaćete sve jasniju sliku o tome kako se COM i ActiveX mogu koristiti u Delphi-ju.

COM je osnova i za OLE i za ActiveX. Analogni pojam može predstavljati `TObject` klasa u VCL-u. Sve klase u VCL-u su obavezno izvedene iz klase `TObject`. Izvedene klase automatski poprimaju sve osobine i sve metode klase `TObject`. Kasnije, one dodaju svoje osobine i metode, kako bi omogućile dodatnu funkcionalnost. Slično, OLE i ActiveX su sagrađeni na osnovama koje čini COM. COM je osnova za sve OLE i ActiveX objekte.

Kao komponentna arhitektura, COM ima dve velike pogodnosti:

- Kreiranje COM objekata ne zavisi od programskog jezika. (COM objekti se mogu pisati u više različitih programskih jezika)
- COM objekat se može koristiti u svim razvojnim okruženjima pod Windows-om među kojima su Delphi, C++Builder, Visual C++, Visual Basic, PowerBuilder, Visual dBASE kao i mnogi drugi.

NAPOMENA Najveća mana COM-a je što je tesno vezan za WinTel (Windows / Intel) platformu. Dakle, iako možete koristiti COM objekat u više različitih razvojnih okruženja pod Windows operativnim sistemom, to ne mora da znači da ćete moći da ga koristite i u razvojnom okruženju pod UNIX-om. Nedavno, Microsoft je pokušao da prilagodi COM objekte i ne-Windows platformama. Ostaje da se vidi da li je ovaj pokušaj bio uspešan. Ovo poglavlje obrađuje COM i ActiveX objekte na onom nivou na kom oni postoje u programskim okruženjima pod Win32.

Možete koristiti veliki broj različitih programskih jezika i okruženja za pisanje COM objekata. COM objekte možete praviti u Delphi-ju, C++Builder-u, Visual C++-u, Visual Basic-u i u verovatno još nekoliko razvojnih okruženja. Kada je COM objekat napravljen, može se koristiti u verovatno još većem broju razvojnih okruženja. COM objekat napravljen u Delphi-u može koristiti Visual Basic programer, C++Builder programer, ili čak, Visual dBASE, ili PowerBuilder programer.

COM objekat se, obično, nalazi u okviru DLL-a. DLL datoteka može imati ekstenziju `.DLL`, ili .OCX. Jedan fajl (DLL, ili OCX) može sadržati jedan COM objekat, ili više njih.

Terminologija COM-a

COM je prepun nejasnih termina. Odeljak koji sledi objašnjava neke od termina koji se koriste u COM-u i na koji način se mnogi delovi COM-a uklapaju u jednu celinu. Svi ovi delovi su povezani među sobom, tako da ćete morati da pročitate ceo odeljak kako biste razumeli celinu.

COM objekat

NOVI TERMIN COM *objekat* je deo binarnog koda koji obavlja određenu operaciju.

COM objekat otkriva pojedine svoje metode, kako bi omogućio programima da pristupe njegovim funkcijama. Te metode su dostupne kroz COM interfejse. COM objekat može imati samo jedan, ali i više interfejsa. Za programera, COM objekat funkcioniše slično kao i Object Pascal klase.

COM Interfejsi

Pristup COM objektima se vrši kroz njihove interfejse.

NOVI TERMIN COM *interfejs* je sredstvo pomoću kojeg korisnik COM objekta pristupa funkcijama tog objekta.

COM interfejs se *koristi* za pristup COM objektu, odnosno, omogućava korišćenje COM objekta. Interfejs, u stvari, opisuje šta sve COM objekat može da ponudi. COM objekat može imati samo jedan interfejs, ali može imati i nekoliko. Specijalno, jedan COM interfejs može implementirati više COM interfejs-a.

Uobičajeno je da COM intefejsi počinju sa slovom I. Window Shell, na primer, implementira interfejse koji se zovu: `IShellLink`, `IShellFolder`, `IShellExtInit`. Iako možete koristi kakva god želite imena, početno slovo I univerzalno i momentalno govori drugim programerima da klasa predstavlja COM objekat.

COM intefejsi-ma upravlja sam Windows preko njihovih interfejs identifikatora (engl. Interface Indentificators (IIDs)). IID je numerička vrednost koja se nalazi u okviru podataka samog COM objekta. Interfejs je jedinstveno određen preko IID-a.

COM Klase

NOVI TERMIN COM *klasa* (takodje poznata kao Coklasa) je klasa koja sadži jedan, ili više COM interfejsa.

Ne možete koristiti COM interfejs direktno. Umesto toga, možete pristupiti interfejsu kroz Coklasu. CoKlasa sadrži deo koda koji kreira traženi interfejs i vraća pointer na njega. COM klase su jedinstveno određene uz pomoć identifikatora klase (engl. Class Identifiers (CLSIDs)). CLSID je kao i IID, numerički podatak.

GUID-ovi

COM objekti moraju biti registrovani u samom Windows-u. Tada IID-ovi i CLSID-ovi dolaze do izražaja. CLSID i IID su u stvari različita imena za istu, osnovnu strukturu podataka: jedinstveni globalni identifikator (engl. Globally Unique Identifier (GUID)).

NOVI TERMIN *GUID* je jedinstvena 128-bitna (16-bajtna) vrednost.

GUID-ovi se kreiraju pomoću specijalne funkcije `CoCreateGUID` koja se nalazi u COM biblioteci. Ova funkcija kreira GUID koji je (praktično sigurno) jedinstven. `CoCreateGUID` koristi kombinaciju informacija dobijenih od Vašeg računara, slučajnih brojeva i trenutnog sistemskog vremena za kreiranje GUID-a. Iako je teorijski moguće da funkcija `CoCreateGUID` kreira dva ista GUID-a, to je praktično malo verovatno (pre bi se moglo reći statistički nemoguće).

Srećom, Delphi programeri ne moraju da obraćaju pažnju na kreiranje GUID-ova. Delphi automatski kreira GUID kada vi kreirate novi objekat za automatizaciju, COM objekat, ActiveX kontrolu, ili aktivnu formu. GUID-ovi su u Delphi-ju definisani u okviru TGUID strukture. `TGUID` je definisana u fajlu `System.pas` na sledeći način:

```
TGUID = record
  D1: Integer;
  D2 : Word;
  D3 : Word;
  D4 : array[0..7] of Byte;
end;
```

Kada kreirate novi COM objekat, Delphi automatski kreira GUID. Na primer, pogledajte GUID za probni COM objekat koji sam ja napravio:

```
Class_Test: TGUID = '{F34107A1-ECCF-11D1-B47A- 0040052A81F8}';
```

Pošto Delphi automatski izvršava sve zadatke u vezi sa GUID-ovima, nećete morati da obraćate mnogo pažnje na njih. Ipak, često ćete se sretati sa GUID-ovima kada budete kreirali COM objekte (uključujući i ActiveX kontrole).

SAVET Ako je neophodno da sami kreirate GUID, pritisnite Ctrl+Shift+G u editoru. Delphi će automatski kreirati GUID i ubaciti ga u vaš kod na mesto gde se trenutno nalazi kurzor.

Biblioteke tipova

COM objekti često koriste biblioteku tipova.

NOVI TERMIN *Biblioteka tipova* je specijalni fajl u kome se nalaze informacije vezane za COM objekte. Ove informacije se sastoje od: liste osobina, metoda, interfejsa struktura i drugih elemenata koji se nalaze u kontroli. Biblioteka tipova takođe obezbeđuje informacije o tipu podatka svake osobine i o povratnoj vrednosti i listi parametara svake metode.

Ova informacija sadrži tipove podataka u objektu, metode i osobine objekta, podatke o verziji, interfejse u objektu i tako dalje. Biblioteke tipova mogu postojati kao resursi u okviru COM objekta, ili kao samostalni fajlovi. Ovi fajlovi imaju ekstenziju `.TLB`. Biblioteka tipova je neophodna drugim programerima ukoliko žele da koriste Vaš COM objekat prilikom razvoja. Biblioteka tipova jednog objekta sadrži više informacija o samom objektu nego što biste mogli dobiti prostim čitanjem interfejsa objekta. Delphi-jevo razvojno okruženje, na primer, koristi informacije iz biblioteka tipova da bi postavilo ActiveX kontrolu na paletu sa komponentama. Korisnici COM objekta mogu pregledom biblioteke tipova da tačno saznaju koje metode i interfejse objekat sadrži.

DCOM

Distribuirani COM (engl. Distributed COM (DCOM)) je podskup COM tehnologije koji omogućava korišćenje COM objekata kroz lokalnu mrežu, ili preko Interneta. DCOM proširuje COM da bi omogućio mehanizme koji su neophodni da bi se COM objekat mogao koristiti u mrežnom okruženju. Detaljni pregled DCOM-a prevazilazi nivo ove knjige, ali imajte na umu da je DCOM preovlađujuća tehnologija u nekim aspektima mrežnog programiranja.

CORBA (Common Object Request Broker Architecture) je tehnologija koja je konkurent DCOM-u. CORBA je nezavisna od radnog okruženja što je čini privlačnijom od DCOM-a u mnogim primenama. Takođe, CORBA je podržana od strane velikog broja softverskih kuća (za razliku od DCOM-a koji je specifičan za Microsoft). Srećom, Delphi Vam daje mogućnost da kreirate i DCOM i CORBA objekte.

Brojanje referenci

Svaki COM objekat ima brojač referenci. *Brojač referenci*, prirodno, sadrži broj procesa koji trenutno koriste taj COM objekat. *Proces* je bilo koji program, ili DLL koji koristi COM objekat. Zbog toga što više procesa u istom trenutku može koristiti COM objekat, brojanje referenci se koristi da bi se odredilo da li je neophodno da se dati COM objekat nalazi u memoriji.

Kada se COM objekat kreira, njegov brojač referenci se postavi na 1. Brojač referenci se povećava za 1 svaki put kada se neki proces priključi na COM objekat. Kada se proces isključi, brojač referenci se smanji za 1. Kada brojač referenci dostigne vrednost 0, COM objekat se uklanja iz memorije.

`IUnknown` interfejs

Svi interfejsi COM objekata potiču iz baznog interfejsa koji se zove `IUnknown`. Tabela 15.1 prikazuje metode interfejsa `IUnknown`.

Tabela 15.1: Metode intefejsa `IUnknown`

Metod	Opis
`QueryInterface`	Postavlja upit nad interfejsom da bi dobio listu podržanih interfejsa.
`AddRef`	Povećava brojač referenci za 1.
`Release`	Smanjuje brojač referenci za 1. Kada brojač referenci dostigne vrednost 0, objekat se uklanja iz memorije.

Pominjem `IUnknown` uglavno iz istorijskih razloga. Za razliku od drugih programera, Delphi programeri ne moraju mnogo da obraćaju pažnju na `IUnknown`. Delphi obavlja sve poslove oko upravljanja brojačem referenci i oslobađanja memorije za COM objekat. Delphi, takođe, radi sa COM objektima na taj način, da je detaljno poznavanje `IUnknown` nepotrebno.

Kreiranje COM objekta

Da bismo oživeli prethodnu priču, hajde da kreiramo jedan COM objekat. Ovaj COM objekat će biti vrlo jednostavan, ali dovoljan da bi se razumeo način na koji se kreiraju COM objekti u okviru Delphi-a. COM objekti koje ćete Vi kreirati, nikada neće imati ovakve osobine:

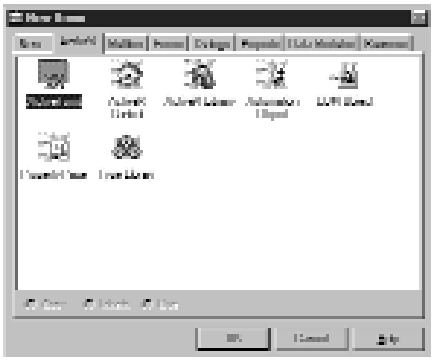
Tip	Ime	Opis
osobina	`x`	Prvi broj koji se množi
osobina	`y`	Drugi broj koji se množi
metod	`DoIt`	Metod koji množi dva broja i vraća rezultat.

Sledeći pasusi detaljnije opisuju proces kreiranja COM objekta.

Kreiranje ActiveX biblioteke

Prvi korak u kreiranju COM objekta je kreiranje DLL-a koji će sadržati njegov kod. Delphi koristi izraz "ActiveX biblioteka" (engl. ActiveX Library) kao sinonim za sve projekte za kreiranje COM biblioteka. Objašnjenje nije precizno, ali je približno tačno. Izvršite sledeće operacije kako biste kreirali ActiveX biblioteku:

1. Zatvorite sve projekte. Izaberite File➥New iz glavnog menija kako biste aktivirali Object Repository.
2. Kliknite na ActiveX jezičak kako biste videli ActiveX stranicu (pogledajte sliku 15.1). Dva puta kliknite na ActiveX Library ikonu.



Slika 15.1
ActiveX stranica prozora Object Repositor.

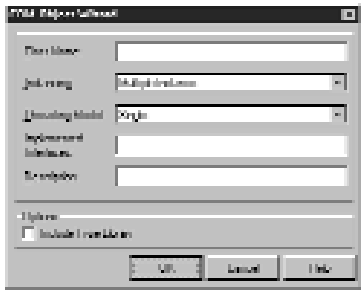
3. Izaberite File➥Save i sačuvajte projekat kao `ComTest`.

To je sve za sada. Delphi kreira DLL projekat i čeka Vaš sledeći korak.

Kreiranje samog objekta

Sledeći korak je kreiranje samog objekta. Ovaj korak je relativno jednostavan. Izvršite sledeće operacije:

1. Izaberite File➥New iz Delphi-jevog glavnog menija. Prikazuje se Object Repository prozor. Kliknite na ActiveX stranicu.
2. Dva puta kliknite na COM Object ikonu. Delphi prikazuje Com Object Wizard, kao što je prikazano na slici 15.2.



Slika 15.2
Com Object Wizard

COM Object Wizard

Za trenutak, obratite pažnju na COM Object Wizard. Class Name polje se koristi za navođenje imena klase Vašeg COM objekta. Na ovom mestu upišite ime klase, ali nemojte ga započeti sa T, kao što činite sa Delphi-jevim klasama, niti sa `I` koje je uobičajeno za interfejse. Delphi će automatski kreirati imena za klasu i interfejs.

Instancing polje se koristi da bi se kontrolisao broj instanci COM objekta koje su u upotrebi. Izbor se sastoji iz Internal, Single Instance i Multiple Instance. Pogledajte

odeljak "Com Object Wizard" u Delphi-jevom Help-u da biste videli opis ovih opcija (možete kliknuti na Help taster na prozor Com Object Wizard, kako biste automatski prikazali ispravnu stranicu).

Threading Model polje se koristi da bi se odredilo na koji način aplikacije mogu pozvati Vaš COM objekat. Izbor sadrži Single, Apartment, Free, ili Both. Takođe, pogledajte odgovarajuću stranicu u Delphi-jevom Help-u.

U polje Implemented Interface upisujete ime interfejsa koji će Vaš objekat realizovati. Ako imate interfejs koji se zove `IMyFileIO` i želite da koristite taj interfejs sa Vašim novim COM objektom, upišite `IMyFileIO` u ovo polje.

U Description polje se upisuje opis COM objekta. Opis može biti proizvoljan i preporučljivo je da ga upišete.

Kada je Include Type Library opcija uključena, Delphi će kreirati biblioteku tipova za COM objekat. Kreiranje biblioteke tipova omogućava drugim aplikacijama da koriste Vaš COM objekat.

U redu, vratimo se na posao:

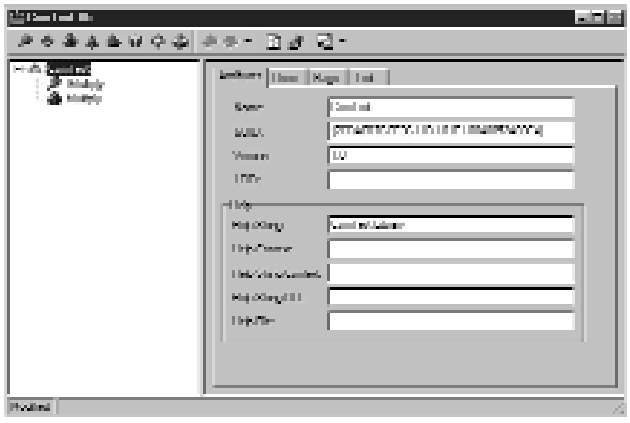
3. Unesite `Multiply` u polje Class Name.

4. Unesite `Test COM Object` u Description polje.

5. Uključite opciju Include Type Library. Ostala polja u prozoru ne morate menjati.

6. Kliknite taster OK kako biste zatvorili prozor.

Kada kliknete OK taster, Delphi kreira jedinicu (Unit) za klasu COM objekta i prikazuje editor biblioteke tipova (Type Library Editor), kao što je prikazano na slici 15.3. Pre nego što nastavimo, potrebno je da prokomentarišem editor biblioteke tipova.



Slika 15.3
Editor biblioteke tipova (Type Library Editor)

Editor biblioteke tipova (Type Library Editor)

Editor biblioteke tipova se koristi za manipulaciju sa bibliotekom tipova. On Vam omogućava da dodajete i izbacujete interfejse, dodajete osobine i metode intefejsima, izbacujete elemente iz interfejsa i kreirate neke druge COM elemente kao što su nabrajanja (enumerations), strukture i koklase. Pomoću editor biblioteke tipova se lako dodaju elementi u biblioteku tipova. Naučićete kako se dodaju elementi u sledećem pasusu, kada budete dodavali osobine i metode COM objektu.

Sa leve strane editora biblioteke tipova se nalazi pano sa objektima (Object Pane). Pano sa objektima sadrži stablo (Tree View Control). Na početku stabla se nalazi sama biblioteka tipova. Ispod nje se nalaze elementi koje ona sadrži. Na slici 15.3 se mogu videti dva elementa: `IMultiply` interfejs i `Multiply` koklasa.

Sa desne strane editora biblioteke tipova se nalazi pano sa informacijama (Information pane). Ovaj pano prikazuje informacije o objektu koji je trenutno izabran u panou sa objektima. Informacije u ovom panovu se razlikuju od tipa do tipa objekta koji je trenutno izabran. Stranica Attributes prikazuje ime biblioteke, njen GUID, verziju, fajl za pomoć (help file) i tako dalje.

Da li se sećate da sam ranije rekao da Delphi programeri ne moraju mnogo da brinu o GUID-ovima? COM objekat koji ste upravo kreirali ima GUID, kao i sama biblioteka tipova. Delphi automatski kreira ove GUID-ove za Vas. Kao što sam rekao, sretaćete se sa GUID-ovima dok budete radili sa COM objektima, ali nećete morati da obraćate pažnju na njihovo kreiranje.

Kada je izabrana stavka biblioteke tipova, pano sa informacijama prikazuje jezičak (tab) sa nazivom Uses. Kada kliknete na ovaj jezičak, videćete biblioteke tipova na kojima se bazira ova biblioteka tipova. Skoro u svim slučajevima, u ovoj listi će se nalaziti OLE Automation Library ali će se nalaziti i drugi. Tačan spisak biblioteka na kojima se zasniva biblioteka tipova zavisi od tipa i kompleksnosti COM objekta.

Stranica Text prikazuje definicije iz biblioteke tipova u IDL sintaksi. IDL je vrsta skript jezika (engl. scripting language) koja se koristi za kreiranje binarne biblioteke tipova. Ne biste trebali da menjate ni jedan podatak na ovoj stranici, dokle god niste sigurni šta u stvari radite. Ipak, možete koristiti Text stranicu kao referencu. Ona je, verovatno, od veće pomoći iskusnim programerima nego početnicima.

Ostale stranice mogu biti prikazane u zavisnosti od toga koji je tip objekta izabran. Za kompletne informacije, pročitajte "Type Library Editor" odeljak u Delphi-jevom Help-u.

Kako budete napredovali kroz ovo poglavlje, saznaćete više o radi sa editorom biblioteke tipova. A sada, vratimo se na kreiranje samog COM objekta.

Dodavanje osobina i metoda COM objektu

Pre nego što krenete dalje, trebali biste da ponovo sačuvate projekat. Vi to niste primetili, ali Delphi je kreirao novu jedinicu kada ste kreirali COM objekat u prethodnom koraku. Izaberite File/Save All iz glavnog menija i sačuvajte jedinicu kao `MultiplyU`.

Sada ste spremni da učinite COM objekat korisnim. Zapamtite, ovaj COM objekat je vrlo jednostavan, tako da neće uraditi mnogo, ali će uraditi makar *nešto*.

Dodavanje osobina

Prvo, potrebno je dodati osobine COM objektu. To se radi na sledeći način:

1. Kliknite na `IMultiply` stavku u panou sa objektima u editoru biblioteke tipova. Primetite da pano sa informacijama pokazuje ime interfejsa, GUID i verziju. Primetite, takođe, da Parent Interface polje pokazuje da je osnova `IMultiply`, u stvari, `IUnknown`. Ako se prisetite, ranije sam rekao da je `IUnknown` baza iz koje su izvedeni svi interfejsi. Delphi automatski podrazumeva da je baza `IUnknown`. Možete promeniti bazni interfejs, ukoliko želite, tako što ćete izabrati neki drugi iz liste raspoloživih interfejsa. Ostali interfejsi u listi su izvedeni iz `IUnknown`, ili iz nekog od njihovih baznih interfejsa.
2. Pritisnite desni taster i izaberite New➥Property iz konteksnog menija. Editor biblioteke tipova automatski dodaje dve stavke na pano sa objektima ispod `IMultiply` interfejsa. Kurzor se nalazi u režimu za izmene tako da možete uneti ime nove osobine.
3. Nazovite osobinu imenom X i pritisnite taster Enter. Imena obe stavke se menjaju u `X`. Postoje dve stavke za svaku osobinu zato što se podrazumeva da su one predviđene i za čitanje i za pisanje. COM zahteva `Get` metod da bi pročitao osobinu i `Put` metod da bi je zapisao. Kliknite na bilo koju od dve osobine obeležene sa `X`. Zapazite Invoke Kind polje u panou sa informacijama kada budete izabirali prvo jednu, pa zatim i drugu osobinu obeleženu sa `X`. Primetite da se polje menja iz Property Set u Property Get.
4. Primetite da polje Type u panou za informacije prikazuje `Integer`. To je tip podatka koji će se sadržati u ovoj osobini. Pošto je to ono što nam treba, nemojte ništa menjati.
5. Kreirajte još jednu osobinu, ali ovaj put na drugačiji način. Uočite New Property taster na galeriji editora biblioteke tipova. Kliknite na strelicu pored New Property tastera. Izaberite Read➥Write iz liste tipova osobina. Editor biblioteke tipova kreira novu osobinu. Nazovite je Y. Možete prihvatiti podrazumevani tip podatka `Integer` i za ovu osobinu. Dok vi dodajete nove elemente Delphi generiše kod u jedinici projekta.

Dodavanje metoda

Sledeće što treba da uradite je da dodate metod. Izvršite sledeće operacije:

1. Izaberite `IMultiply` objekat iz panoa sa objektima i kliknite na New Method taster na galeriji editora biblioteke tipova.
2. Nazovite metod imenom `DoIt`. Primetite da polje Invoke Kind prikazuje Function (umesto Property Get, ili Property Set).

   Zatim, morate postaviti parametre metoda. Metod će imati sledeću sintaksu:

   ```
   function DoIt : Integer;
   ```

3. Kliknite na Parameters jezičak u panou za informacije. Promenite Return Type u `Integer` (izaberite Integer iz combo-liste). Ovaj metod ne prima nikakve parametre, tako da možete ostaviti Parameters listu praznom. Pošto ste definisali povratni tip, kliknite na Attributes jezičak da biste prikazali Attributes stranicu. Ovaj korak nije neophodan, ali se koristi da biste se vratili tamo odakle ste i krenuli.
4. Kliknite Refresh Implementation taster na galeriji editora biblioteke tipova.

Sada ste dodali dve osobine i metod. Vreme je da vidimo šta je Delphi radio za to vreme. Listing 15.1 prikazuje kako jedinica klase izgleda kada se izvrše sve operacija do ovog nivoa. (Ne brinite ukoliko Vaša jedinica ne izgleda isto kao i ona na listingu 15.1. Moja verzija Delphi-ja je možda dodala kod u nešto drugačijem redosledu nego Vaša).

Listing 15.1: `MultiplyU` jedinica posle dodavanja osobina i metode

```
unit MultiplyU;

interface

uses
  Windows, ActiveX, ComObj, ComTest_TLB;

type
  TMultiply = class(TTypedComObject, IMultiply)
  protected
    function DoIt: Integer; stdcall;
    function Get_X: Integer; stdcall;
    function Get_Y: Integer; stdcall;
    procedure Set_X(Value: Integer); stdcall;
    procedure Set_Y(Value: Integer); stdcall;
    {Declare IMultiply methods here}
  end;

implementation

uses ComServ;
```

nastavlja se

Listing 15.1: `MultiplyU` jedinica posle dodavanja osobina i metode *nastavak*

```
function TMultiply.DoIt: Integer;
begin

end;

function TMultiply.Get_X: Integer;
begin

end;

function TMultiply.Get_Y: Integer;
begin

end;

procedure TMultiply.Set_X(Value: Integer);
begin

end;

procedure TMultiply.Set_Y(Value: Integer);
begin

end;

initialization
  TTypedComObjectFactory.Create(ComServer, TMultiply, Class_Multiply,
    ciMultiInstance, tmSingle);
end.
```

Ovo je osnova COM objekta. Primetite da je `TMultiply` klasa izvedena i iz `TTypedComObject` i iz `IMultiply`. (C++ programere će ovo podsetiti na višestruko nasleđivanje. To nije baš višestruko nasleđivanje, ali je u neku ruku slično). Vi još uvek niste videli `IMultiply` klasu, ali ćete je videti malo kasnije. Morate popuniti ovu osnovu da bi COM objekat mogao nešto da radi. To je sledećete što ćete uraditi.

Dodavanje koda

Sada ćete dodati kod `TMultiply` klasi kako bi COM objekat mogao da radi. Izvršite sledeće operacije (pogledajte listing 15.2, ukoliko je potrebno):

1. Otvorite fajl `MultiplyU.pas`. Dodajte sledeće linije u deklaraciju klase `TMultiply` neposredno iznad ključne reči `protected`:

```
  private
    FX : Integer;
    FY : Integer;
```

Ovo su deklaracije podataka koje će čuvati vrednosti osobina `X` i `Y`.

2. Spustite se do `implementation` dela i pronađite `Get_X` metod (koristite Code Explorer, ukoliko želite). Unesite ovu liniju koda u metod:

```
Result := FX;
```

3. Pronađite `Get_Y` metod i dodajte ovu liniju:

```
Result := FY;
```

4. Pronađite `DoIt` metod i dodajte ovu liniju:

```
Result := FX * FY;
```

Ova linija koda množi vrednosti iz `FX` i `FY` i vraća rezultat.

5. Spustite se još niže. Dodajte ovu liniju koda u

```
Set_X metod:
```

```
FX := Value;
```

6. Pronađite `Set_Y` metod i dodajte ovu liniju:

```
FY := Value;
```

To je sve što Vam je potrebno. Vaš kod bi trebao da izgleda kao onaj u listingu 15.2.

Listing 15.2: Kompletna `MultiplyU` celina

```
unit MultiplyU;

interface

uses
  Windows, ActiveX, ComObj, ComTest_TLB;

type
  TMultiply = class(TTypedComObject, IMultiply)
  private
    FX : Integer;
    FY : Integer;
  protected
    function DoIt: Integer; stdcall;
    function Get_X: Integer; stdcall;
    function Get_Y: Integer; stdcall;
    procedure Set_X(Value: Integer); stdcall;
    procedure Set_Y(Value: Integer); stdcall;
    {Declare IMultiply methods here}
  end;
```

nastavlja se

Listing 15.2: Kompletna `MultiplyU` celina *nastavak*

```
implementation

uses ComServ;
ss
function TMultiply.DoIt: Integer;
begin
  Result := FX * FY;
end;

function TMultiply.Get_X: Integer;
begin
  Result := FX;
end;

function TMultiply.Get_Y: Integer;
begin
  Result := FY;
end;

procedure TMultiply.Set_X(Value: Integer);
begin
  FX := Value;
end;

procedure TMultiply.Set_Y(Value: Integer);
begin
  FY := Value;
end;

initialization
  TTypedComObjectFactory.Create(ComServer, TMultiply, Class_Multiply,
    ciMultiInstance, tmSingle);
end.
```

Dok ste Vi pravili MultiplyU celinu, Delphi je bio zauzet pravljenjem biblioteke tipova i celine koja će sadržati kod biblioteke tipova. Ova celina ima isto ime kao i projekat sa dodatkom _TLB. Ovaj projekat se zove ComTest. Dakle, puno ime celine biblioteke tipova je ComTest_TLB.pas. Listing 15.3 pokazuje kako u ovom trenutku celina izgleda. Zapamtite da Vaša celina ne mora izgledati tačno kao ona u listingu 15.3.

Listing 15.3: Celina `ComTest_TLB.pas`

```
unit ComTest_TLB;

//
************************************************************************
//
// WARNING
//
// -------
//
// The types declared in this file were generated from data read from
a //
// Type Library. If this type library is explicitly or indirectly
(via  //
// another type library referring to this type library) reimported,
or  //
// the 'Refresh' command of the Type Library Editor activated while
//
// editing the Type Library, the contents of this file will be
//
// regenerated and all manual modifications will be lost.
//
//
************************************************************************
//
// PASTLWTR : $Revision:   1.11.1.55  $
// File generated on 6/8/98 7:16:51 PM from Type Library described
below.

//
************************************************************************
//
// Type Lib: D:\Borland\D4\Bin\ComTest.tlb
// IID\LCID: {7CDAFB76-FF36-11D1-81F1-0040052A83C4}\0
// Helpfile:
// HelpString: ComTest Library
// Version:    1.0
//
************************************************************************
//

interface

uses Windows, ActiveX, Classes, Graphics, OleCtrls, StdVCL;
Listing 15.3. continued
//
************************************************************************
//
// GUIDS declared in the TypeLibrary. Following prefix-
es are used:      //
```

nastavlja se

Listing 15.3: Celina `ComTest_TLB.pas` *nastavak*

```
//   Type Libraries     : LIBID_xxxx
//
//   CoClasses          : CLASS_xxxx
//
//   DISPInterfaces     : DIID_xxxx
//
//   Non-DISP interfaces: IID_xxxx
//
//
************************************************************************
//
const
  LIBID_ComTest: TGUID = '{7CDAFB76-FF36-11D1-81F1-0040052A83C4}';
  IID_IMultiply: TGUID = '{7CDAFB77-FF36-11D1-81F1-0040052A83C4}';
  CLASS_Multiply: TGUID = '{7CDAFB79-FF36-11D1-81F1-0040052A83C4}';
type

//
************************************************************************
//
// Forward declaration of interfaces defined in Type Library
//
//
************************************************************************
//
  IMultiply = interface;

//
************************************************************************
//
// Declaration of CoClasses defined in Type Library
//
// (NOTE: Here we map each CoClass to its Default Interface)
//
//
************************************************************************
//
  Multiply = IMultiply;

//
************************************************************************
//
// Interface: IMultiply
// Flags:     (0)
// GUID:      {7CDAFB77-FF36-11D1-81F1-0040052A83C4}
//
************************************************************************
//
```

```
  IMultiply = interface(IUnknown)
    ['{7CDAFB77-FF36-11D1-81F1-0040052A83C4}']
    function Get_X: Integer; stdcall;
    procedure Set_X(Value: Integer); stdcall;
    function Get_Y: Integer; stdcall;
    procedure Set_Y(Value: Integer); stdcall;
    function DoIt: Integer; stdcall;
  end;

  CoMultiply = class
    class function Create: IMultiply;
    class function CreateRemote(const MachineName: string):
IMultiply;
  end;

implementation

uses ComObj;

class function CoMultiply.Create: IMultiply;
begin
  Result := CreateComObject(CLASS_Multiply) as IMultiply;
end;

class function CoMultiply.CreateRemote(const MachineName: string):
➥IMultiply;
begin
  Result := CreateRemoteComObject(MachineName, CLASS_Multiply) as
➥IMultiply;
end;

end.
```

Primetite da ova celina sadrži deklaraciju `IMultiply` interfejsa. Kao što možete videti, `IMultiply` je izvedena iz `IUnknown`. Primetite takođe da ova celina sadrži i koklasu `Multiply`.

Važno je da razumete da se ova celina ponovo generiše svaki put kada prevedete projekat ActiveX biblioteke. Celina se ponovo generiše iz biblioteke tipova. Pročitajte upozorenje na počtku celine. Komentari Vam ukazuju na to da će sve Vaše izmene u ovoj celini biti izbrisane kada se COM objekat bude generisao sledeći put. Dakle, nije baš dobro unositi bilo kakve izmene u izvorni kod biblioteke tipova.

Generisanje i registracija COM objekta

Sada ste spremni za prvo prevođenje Vašeg projekta ActiveX biblioteke. Ovaj korak će izvršiti prevođenje COM objekta i generisanje DLL fajla u kome se COM objekat nalazi. Posle generisanja COM objekta, možete ga registrovati i to na sledeći način:

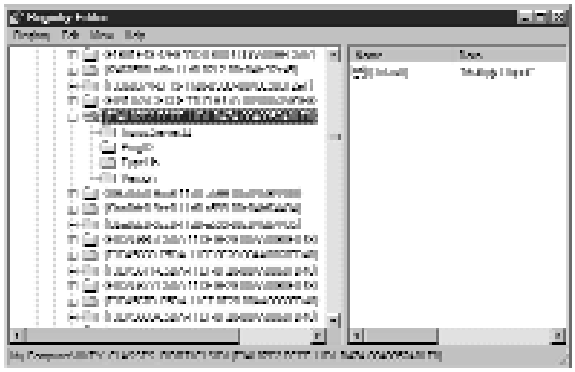
1. Izaberite Projects➥Build ComTest iz glavnog menija. Delphi generiše DLL fajl koji sadrži COM objekat.
2. Izaberite Run➥Register ActiveX Server iz glavnog menija. Ovaj korak registruje COM objekat unutar Windows-a. Ukoliko ovaj korak ne uspe, kada pokušate da pristupite COM objektu, dobićete poruku o grešci koja glasi: "Class not registred".

Delphi registruje DLL fajl COM objekta unutar Windows-a. Posle registracije Delphi prikazuje poruku kao što je ona na slici 15.4.

Slika 15.4
Delphi obaveštava da je COM objekat uspešno registrovan

Kada Windows registruje COM objekat, dodaje informaciju o objektu u Registry bazu. Slika 15.5 pokazuje stavku u Registry bazi koja je nastala kada je COM objekat registrovan u Windows-u.

Slika 15.5
Ključ u Registry bazi koji je nastao kada je registrovan COM objekat



NAPOMENA Delphi se isporučuje sa pomoćnim programom pod imenom `TREGSVR.EXE` koji omogućuje registrovanje ActiveX kontrole iz komandne linije. Da bi ste registrovali kontrolu koja se zove `MYSTUFF.OCX`, pokrenuli biste TREGSVR sa komandrnog prompta na sledeći način:

```
tregsvr mystuff.ocx
```

Da biste izbacili ActiveX kontrolu iz Registry baze koristite prekidač -u na sledeći način:

```
tregsvr -u mystuff.ocx
```

Ponekad je ovo jednostavniji način nego učitavanje ActiveX projekta u Delphi i registrovanje, ili izbacivanje kontrole iz samog razvojnog okruženja.

NAPOMENA U ovom vežbanju sam Vam pokazao kako se kreira COM objekat. Takođe ste možda koristili i objekat za automatizaciju (engl. automation object). Objekat za automatizaciju je izveden iz `IDispatch` umesto iz `IUnknown`. `IDispatch` obezbeđuje dodatnu funkcionalnost koja je neophodna da bi se COM objekat ponašao kao automatizacioni server (engl. automation server) (objekat koji može da kontroliše jedan program iz drugog).

Vaš COM objekat je sada spreman za upotrebu.

Generisanje programa koji koristi COM objekat

COM objekat Vam ne pruža mnogo koristi ako ne možete da ga koristite. U ovom koraku ćete kreirati program koji koristi COM objekat koji ste upravo napravili. Pratite sledeće korake:

1. Kreirajte nov program. Postavite po jednu `Button` i `Label` komponentu na formu. Sačuvajte projekat pod imenom `ComApp` i celinu glavne forme pod imenom `ComAppU`.

2. Pređite u editor koda i pronađite `uses` listu u celini glavne forme. Dodajte sledeće celine u `uses` listu:

```
ComObj
ComTest_TLB
```

   Ovaj korak garantuje da ćete moći da prevedete kod koji referencira COM objekat.

3. Dva puta kliknite na Button komponentu da biste kreirali `OnClick` hendler. Izmenite `OnClick` hendler tako da izgleda kao sledeći kod:

```
procedure TForm1.Button1Click(Sender: TObject);
var
  Mult : IMultiply;
  Res  : Integer;
begin
  Mult := CreateComObject(CLASS_Multiply) as IMultiply;
  if Assigned(Mult) then begin
    Mult.Set_X (20);
    Mult.Set_Y (60);
    Res := Mult.DoIt;
    Label1.Caption := IntToStr(Res);
  end;
end;
```

Ovaj kod prvo dekrariše pokazivač na `IMultiply` interfejs pod nazivom `Mult` i celobrojnu promenljivu koja će prihvatiti rezultat. Zatim se poziva `CreateComObject` funkcija sa parametrom `CLASS_Multiply`. CLASS_Multiply je konstanta koja sadrži GUID klase COM objekta (pogledajte listing 15.3).

Povratna vrednost funkcije `CreateComObject` je dodeljena pointeru `Mult`. Primetite da ja koristim as operator da bih prebacio povratnu vrednost u tip pokazivača na `IMultiply`. `CreateComObject`, inače, vraća pokazivač na `IUnknown`, pa as operator konvertuje pokazivač na `IUnknown` u pokazivač na `IMultiply`.

Pošto je COM objekat kreiran, dodeljujem vrednosti `X` i `Y` osobinama. Posle toga, pozivam `DoIt` metodu COM objekta i prikazujem rezultat u `Label` komponenti.

NAPOMENA U realnim programima napisao bih prethodnu proceduru drugačije. Na primer:

```
procedure TForm1.Button1Click(Sender: TObject);
begin
  with CreateComObject(CLASS_Multiply) as IMultiply do
begin
    Set_X(20);
    Set_Y(60);
    Label1.Caption := IntToStr(Res);
  end;
end;
```

Napisao sam proceduru na taj način da bih ilustrovao svaki korak.

Pokrenite program. Kada kliknete na Button komponentu, tekst u Label komponenti će glasiti: "1200" (proizvod `20 * 60`). To je to! Vaš COM objekat radi. Ovaj COM objekat se može koristiti iz Visual Basic-a, Visual C++-a, C++Builder-a, ili bilo kojeg drugog razvojnog okruženja koje podržava COM.

Objašnjenje ActiveX-a

ActiveX je relativno nov naziv za tehnologiju koja je prisutna već neko vreme. Originalno ime za ActiveX kontrole je *OCX kontrole*. Termin OCX se još uvek ponegde koristi. Fajl sa ActiveX kontrolom obično ima ekstenziju DLL ili OCX.

ActiveX kontrola je, u suštini, prerušeni COM objekat. Osnovna razlika između ActiveX kontrola i COM objekata je što ActiveX kontrole imaju interfejs koji se može prilagoditi sopstvenim potrebama. ActiveX kontrole, takođe, sadrže kod koji im omogućava da budu distribuirane preko Web-a, ili preko mreže. ActiveX je podskup COM-a, tako da sve što ste naučili o COM objektima u prvom delu ovog poglavlja važi i za ActiveX kontrole.

Korišćenje tuđih ActiveX kontrola

Instaliranje i korišćenje tuđih ActiveX kontrola nije komplikovano. Sve što treba da uradite je da uvezete ActiveX kontrolu u razvojno okruženje i koristite je. Da bi ste videli na koji način se to radi uradićemo jednu vežbu. Ova vežba zahteva instaliran Microsoft Internet Explorer na vašem računaru. Ukoliko nemate instaliran Internet Explorer, preskočite ovu vežbu. (Nećete ništa propustiti zato što ću Vam pokazati

kako da instalirate ActiveX kontrolu koju ćete sami napraviti u odeljku "Generiši, registruj i instaliraj kontrolu.") Izvršite sledeće operacije:

1. Izaberite Component➥Import ActiveX Control iz glavnog menija. Prikazaće se Import ActiveX dijalog-prozor.
2. Spuštajte se na dole kroz listu instaliranih komponenti dok ne pronađete `Microsoft Internet Controls` (ime zavisi od verzije Internet Explorer-a koja je instalirana na Vašem računaru). Izaberite tu stavku. Slika 15.6 prikazuje Import ActiveX dijalog-prozor posle ove operacije.

Slika 15.6
Import ActiveX dijalog-prozor

Primetite listu Class names na sredini prozora. Ta lista sadrži spisak ActiveX kontrola u izabranom fajlu (u ovom slučaju, `SHDOCVW.DLL`).

3. Palette page polje pokazuje `ActiveX`. Na ovoj stranici će se instalirati nove komponente. Kliknite na Palette page polje i upišite `ActiveXTest`.
4. Nemojte menjati Unit dir name i Search path polja i kliknite na Install taster. Pojavljuje se Install dijalog-prozor sa pitanjem u koju kolekciju želimo da instaliramo kontrole. (Svaka kontrola se, bez obzira da li je VCL, ili ActiveX, mora nalaziti u kolekciji).
5. Kliknite na Into new package jezičak. Unesite `MSIE` u polje File name i `Internet Explorer Package` u Description polje.
6. Kliknite OK taster. Delphi kreira novu kolekciju sa imenom `MSIE.dpk` i poziva Vas da generišete i instalirate kolekciju. Kliknite na taster Yes da biste instalirali kolekciju. Pošto je kolekcija generisana Delphi prikazuje poruku kojom Vam govori da su komponente postavljene na paletu komponenti (Component palette). Kliknite na taster Yes da biste zatvorili prozor za porukom.

7. Pronađite ActiveXText jezičak u paleti komponenti. Na toj stranici ćete videti dve ili tri kontrole (opet, u zavisnosti od verzije Internet Explorer-a). Komponente su spremne za upotrebu.

Eksperimentišite sa novo-instaliranim kontrolama da biste videli na koji način rade. Verovatno nećete daleko stići bez dokumentacije ali ste makar videli na koji način se instaliraju ActiveX komponente. (Za detaljnija objašnjenja u vezi sa korišćenjem Internet Explorera u formi ActiveX komponente, pogledajte odeljak "Korišćenje Internet Explorera u formi ActiveX komponente" u dodatnom danu u delu: "Generisanje internet programa").

Morate imati licencu (design-time license) da biste mogli da koristite instalirane ActiveX kontrole. Licenca se nalazi u fajlu sa `.LIC` ekstenzijom. U nekim slučajevima ćete moći da uvezete ActiveX kontrolu u Delphi-jevu paletu komponenti i bez licence, ali ćete dobiti poruku o grešci kada budete pokušali da je postavite na formu.

Da biste sklonili Internet Explorer kontrole sa palete komponenti, izaberite Component➥Install Packages iz glavnog menija. Pronađite Internet Explorer Package u Design Packages listi i kliknite Remove taster. ActiveXTest jezičak je uklonjen iz palete komponenti.

Isporučivanje programa koji koristi ActiveX kontrole zahteva posebnu pažnju. Isporučivanje programa koji koristi ActiveX kontrolu je pokriveno u bonus danu u delu: "Isporučivanje internet programa".

Kreiranje novih ActiveX kontrola

Postoje dva načina za kreiranje ActiveX kontrola u Delphi-ju:

- iz postojeće VCL komponente,
- od početka, koristeći aktivne forme.

U ovom odeljku, kreiraćete ActiveX kontrolu koristeći oba načina.

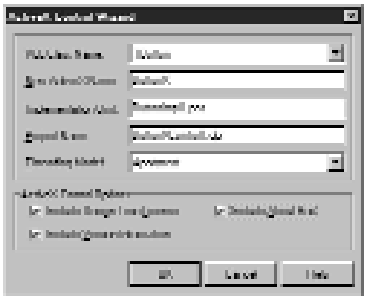
Kreiranje ActiveX kontrole iz postojeće VCL komponente

Kreiranje ActiveX kontrole iz postojeće VCL komponente je jednostavno. Pošto ste kreirali komponentu, možete je prevesti u ActiveX kontrolu praktično automatski. Do sada još nije bilo reči o kreiranju komponenti, pa ovde neće biti detalja po tom pitanju (ta tema je pokrivena u danu 20, "Kreiranje komponenti"). Ono što ćete sada raditi jeste kreiranje ActiveX kontrole iz jedne od kontrola iz VCL biblioteke koju isporučuje Borland.

Generišite ActiveX projekat sa ActiveX Control Wizard-om

Prvi korak je generisanje ActiveX projekta. Delphi će odraditi najveći deo posla za Vas. Sve što je potrebno da uradite je da popunite nekoliko polja u ActiveX Control Wizard-u.

1. Izaberite File➥Close All da biste zatvorili sve projekte. Zatim izaberite File➥New iz glavnog menija. Prikazuje se Object Repository.
2. Kliknite na ActiveX stranicu a zatim dva puta kliknite na ActiveX Control ikonu. Prikazuje se ActiveX Control Wizard. (pogledajte sliku 15.7)



Slika 15.7
ActiveX Control Wizard

3. Izaberite `TButton` iz combo-liste sa imenima klasa VCL Class Name. Sledeća četiri polja su automatski popunjena podrazumevanim vrednostima. Pošto je ovo samo test, ne morate menjati ove podrazumevane vrednosti. Imena ovih polja su jasna sama po sebi, tako da nije potrebno objašnjavati svako od njih.
4. Threading Model je postavljen na Apartment. Nemojte menjati ovu opciju. Ostali moguće vrednosti polja Threading Model su: Single, Free i Both. Pogledajte Delphi-jev Help da biste imali više informacija.
5. Izaberite Include Design-Time Licence opciju. Kada je ova opcija izabrana, Delphi će kreirati licencu za kontrolu. Ona sprečava druge programere da koriste Vašu kontrolu, dokle god ne nabave licencu.
6. Izaberite opciju Include Version Information. Pomoću nje ste u mogućnosti da dodate informacije o verziji kontrole u samu kontrolu uz pomoć Project Options dijalog-prozora.
7. Izaberite Include About Box opciju. Kada je ova opcija izabrana, Delphi automatski kreira dijalog-prozor sa obaveštenjima (engl. About Box) za kontrolu. Kliknite taster OK da biste zatvorili ActiveX Control Wizard.

Delphi će kreirati projektni fajl (`ButtonXControl1.bpr`) i tri celine za ovaj projekat. Prva celina je vezana za `TButtonX` klasu (`ButtonXImp1.pas`). Druga celina je biblioteka tipova za kontrolu i zove se `ButtonXControl1_TLB.pas`. Ovaj fajl sadrži informacije koje su neophodne da bi Delphi mogao da kreira biblioteku tipova za kontrolu. Treći fajl `About1.pas` je celina dijalog-prozora sa obaveštenjima

(About Box). Ukoliko želite da promenite ovaj dijalog-prozor, sada je pravi trenutak da to učinite. Pošto ovaj dijalog-prozor predstavlja običnu Delphi-jevu formu, možete ga promeniti na koji god način želite.

NAPOMENA Da biste mogli da koristite Vašu ActiveX kontrolu u Visual Basic-u, morate uključiti podatke o verziji kontrole (Version info)

Generisanje, registrovanje i instaliranje kontrole

Pošto nećete ručno menjati kontrolu, možete odmah preći na generisanje i registrovanje kontrole. To je isti proces kroz koji ste već prošli kada ste ranije registrovali COM objekat. Ipak, neophodan je jedan korak više, pošto ActiveX kontrole imaju interfejs koji možete menjati onda kada kontrolu postavljate na neku formu (engl. design-time interface). Pokušajte sledeće:

1. Izaberite Project➥Build ButtonXControl1 iz glavnog menija. Delphi generiše ActiveX projekat.
2. Izaberite Run➥Register ActiveX Server iz glavnog menija. ActiveX kontrola je sada registrovana i Delphi prikazuje poruku koja govori da je OCX registrovan (ActiveX projekti imaju ekstenziju `.OCX`). Kliknite na taster OK kako biste uklonili poruku.
3. Izaberite Component➥Import ActiveX Control iz glavnog menija. Izaberite `ButtonXControl1 Library (Version 1.0)` iz liste instaliranih komponenti (da niste izvršili korak 2, ne biste videli ovu komponentu u listi). Ime klase tastera `TButtonX` se prikazuje u Class names listi.
4. Postavite Pallete Page polje na ActiveX. Kliknite taster Install za nastavak.
5. Prikazuje se Install dijalog-prozor. Instaliraćete ikonu u Delphi-jevu korisničku kolekciju `DCLUSR40.BPL`. Polje File name bi već trebalo da bude postavljeno na ovu kolekciju. Ako nije, izaberite kolekciju koristeći combo-listu. U polju za opis (engl. Description field) sada stoji Delphi User's Components. Kliknite na OK taster da biste instalirali kontrolu.
6. Kliknite na Yes taster u poruci koja Vas obaveštava o generisanju i instalaciji `DCLUSR40.BPL`. Kliknite taster OK kada se pojavi dijalog-prozor koji zahteva da potvrdite instalaciju. Kontrola je sada instalirana.

Testirajte ActiveX kontrolu

Sada možete testirati Vašu novu ActiveX kontrolu. Prvo, kreirajte novi projekat.

NAPOMENA Kada budete kreirali novi projekat, Delphi će od Vas tražiti da sačuvate fajl sa kolekcijom (`DCLUSR40.DPK`) i projekat ActiveX kontrole. Da li ćete sačuvati ove fajlove, zavisi od Vas. Namera je bila da kreirate jednostavnu ActiveX kontrolu tako da ne postoji potreba za snimanjem ovih fajlova. Ukoliko ipak želite da kasnije proučite ove fajlove, sačuvajte ih.

Izvršite sledeće operacije:

1. Pronađite ActiveX jezičak na paleti komponenti (Component palette).
2. Poslednja kontrola u listi će biti `ButtonX` kontrola. Izaberite je.
3. Postavite `ButtonX` kontrolu na formu. Primetite da ButtonX kontrola nema podrazumevani naslov kao što ga ima VCL kontrola.
4. Promenite `Caption` osobinu u `Test`. Caption osobina se menja kao i kod VCL Button komponente.

   Uočite listu osobina u Object Inspector-u. One su praktično iste kao i osobine VCL Button komponente (na kraju krajeva, ActiveX kontrola je kreirana iz VCL `TButton` kontrole), ali ćete primetiti da Value kolona izgleda nešto drugačije. Zapamtite, ovo je ActiveX kontrola i predviđeno je da se koristi u svim okruženjima koja podržavaju ActiveX. Iz tog razloga, neke osobine su prikazane generički.
5. Dva puta kliknite na taster i videćete da se ništa ne dešava. Za razliku od VCL kontrola, ActiveX kontrola nema mogućnost da automatski kreira proceduru događaja (Event handler) kada dva puta kliknete na nju. Zbog toga, pređite u Events prozor i dva puta kliknite na Value kolonu, neposredno pored `OnClick` događaja. Kreira se procedura događaja. Unesite sledeći kod:

```
MessageDlg('Hej, kontrola radi!, mtInformation, [mbOK], 0);
```

6. Pokrenite program i testirajte kontrolu kako biste se uverili da radi. Zatim, zatvorite program.
7. Aktivirajte formu i kliknite desnim tasterom na taster. Izaberite About iz kontekstnog menija. Prikazuje se dijalog-prozor sa informacijama o kontroli (About box). Ovaj dijalog prozor nije završen, ali vi to možete uraditi ukoliko želite.

NAPOMENA Ideja koja stoji iza ovog procesa je kreiranje ActiveX komponente iz VCL komponente koja već funkcioniše. Uglavnom, nećete morati da menjate kod ActiveX kontrole. Međutim, Vi možete menjati kod koji je Delphi izgenerisao, ukoliko to želite. Budite oprezni jer pri ponovnom generisanju ActiveX komponente iz VCL komponente, nestaće sve izmene koje ste uneli.

NAPOMENA Možete kreirati ActiveX kontrole isključivo iz kontrola koje su izvedene iz TWinControl klase, ili iz neke od klasa koje su izvedene iz nje. Lista VCL kontrola iz kojih možete generisati ActiveX kontrole sadrži sve komponente koje zadovoljavaju ovaj kriterijum.

Brisanje ActiveX kontrole iz Registry-ja

Pošto ste isprobali Vašu ActiveX kontrolu trebalo bi da je obrišete iz Registry-ja da ne bi bespotrebno zauzimala prostor. Da biste to učinili, postupite na sledeći način:

1. Izaberite Components➥Import ActiveX Control iz glavnog menija.
2. Izaberite ActiveX kontrolu iz liste instaliranih i kliknite na taster Remove.

3. Kliknite na taster Yes u dijalog-prozoru koji traži od Vas da potvrdite operaciju. Delphi će izbaciti ActiveX kontrolu iz Registry-ja.

Takođe, možete otvoriti ActiveX projekat (ako ste ga prethodno sačuvali) i izabrati Run➥Unregister ActiveX Server iz glavnog menija.

Ukoliko ni na jedan od ova dva načina ne uspete da izbrišete kontrolu, pokrenite Registry Editor i obrišite ključ vezan za tu kontrolu. Koristite opciju za pretraživanje Registry Editor-a kako biste pronašli kotrolu prema imenu, ili prema GUID-u. Naravno, budite vrlo pažljivi prilikom ručnog menjanja Registry-a.

Kreiranje aktivnih formi

Kreiranje aktivnih formi je skoro isto toliko jednostavno kao i kreiranje ActiveX komponenti iz već postojećih VCL komponenti. Vi, naravno, možete kreirati kompleksne ActiveX kontrole koje će imati puno komponenti na jednoj jedinoj formi. Aktivna forma se može koristiti (iako se to možda ne može zaključiti iz imena) za kreiranje jednostavnih ActiveX kontrola praktično "od nule". Drugim rečima, aktivne forme ne služe samo kreiranju veselih formi sa puno sličica. One služe i kreiranju jednostavnih formi za jednokratnu upotrebu.

U ovom odeljku ćete kreirati aktivnu formu. Ona će sadržati dve kontrole za unos (Edit), statičnu tekstualnu kontrolu (Label) i jedan taster (Button). Funkcija pritiska na ovaj taster je da uzme vrednosti iz dve kontrole za unos, pomnoži ih i prikaže rezultat u statičnoj tekstualnoj kontroli. Da, znam da množenje dva broja ne zahteva mnogo veštine, ali moj cilj je da Vam pokažem kako se kreira aktivna forma sa najmanjom mogućom količinom koda. Na taj način ćete moći da posvetite više pažnje samom procesu kreiranja aktivne forme bez potrebe da se zamarate pisanjem koda.

Kreirajte aktivnu formu

Kreiranje aktivne forme je neverovatno jednostavno. Pokušajte na ovaj način:

1. Zatvorite sve projekte i izaberite File➥New iz glavnog menija. Prikazuje se Object Repository.
2. Dva puta kliknite na ActiveForm ikonu. Prikazuje se ActiveForm Wizard. Ovaj dijalog-prozor je isti kao i dijalog-prozor ActiveX Control Wizard-a, s tom razlikom što je ovde blokirano polje VCL Class Name (ovde ne možemo koristiti VCL komponente).
3. Unesite `MyFormX` u polje New ActiveX Name.
4. Izmenite sadržaj polja Implementation Unit Field u `MyFormImpl.pas`.
5. Promenite polje Project Name u `MyFormProj.dpr`.
6. Ostavite Thread Model na `Apartment`. Izaberite opciju Include Version Information.

7. Kliknite na taster OK da biste nastavili.

Delphi kreira neophodne celine i prikazuje formu.

Kreirajte formu

U ovom trenutku, aktivna forma je isto što i obična forma. Možete dodati kontrole na formu, dodati kod i odgovarati na događaje, kao što činite sa formom koja pripada nekom programu. Jedina je razlika u tome što se naslovna traka (engl. title bar) ne pojavljuje na samoj kontroli. Ona je tu samo u trenutku kreiranja forme.

U ovom odeljku ćete dodati komponente i kod kako biste načinili aktivnu formu funkcionalnom. Kako budete pratili ovaj odeljak, pomoći će Vam slika 15.8 koja je prikazana kasnije i koja prikazuje završenu formu. Ovde ću Vam dati osnovna uputstva i pustiću Vas da sami dovršite formu. Uradite sledeće:

1. Postavite veličinu forme na (približno) 175 (visina) i 275 (širina).
2. Dodajte `Edit` komponentu u gornji centralni deo forme (pogledajte sliku 15.8). Izmenite `ime` komponente u `Num1Edit`, njenu `Text` osobinu postavite na `0` i njenu širinu postavite na 50 (širina, u principu, nije bitna). Izmenite osobinu `AxBorderStyle` u `afbRaised`.
3. Kliknite na `Edit` komponentu i iskopirajte je u Clipboard. Zatim vratite (Paste) komponentu iz Clipboard-a. Postavite drugu komponentu ispod prve i promenite njenu `Name` osobinu tako da glasi `Num2Edit`.
4. Postavite `Label` komponentu ispod ove dve Edit komponente. U ovoj komponenti će se ispisati rezultat. Promenite `Name` osobinu u `ResultLbl` i `Caption` osobinu u `0`.
5. Postavite `Button` komponentu na formu sa desne strane `Edit` komponenti. Promenite njenu `Name` osobinu u `GoButton` i njenu `Caption` osobinu u `Go!`.
6. Dva puta kliknite na Button komponentu i unesite sledeći kod u `OnClick` proceduru događaja:

```
procedure TMyFormX.GoButtonClick(Sender: TObject);
begin
  try
    ResultLbl.Caption := IntToStr(
      StrToInt(Num1Edit.Text) * StrToInt(Num2Edit.Text));
  except
    on EConvertError do
      MessageDlg('Oops! You entered an invalid value.',
        mtError, [mbOK], 0);
  end;
end;
```

Ovaj kod jednostavno uzima vrednosti iz dve Edit kompnente, množi ih i postavlja rezultat u `Caption` osobinu `Label` komponente. Kod za obradu izuzetaka (engl. Exception handling) prikazuje poruku o grešci ukoliko korisnik unese pogrešne vrednosti. `EConvertError` izuzetak nastaje kada se ne može izvršiti konverzija iz tekstualne u celobrojnu vrednost (na primer, kada jedna od Edit kontrola sadrži i znake, a ne samo cifre).

7. Dodaje još potrebnih Label komponenti kako bi forma izgledala kao ona na slici 15.8.
8. Izaberite View➥Type Library iz glavnog menija. U stranici sa informacijama (Information page), izmenite sadržaj polja Help String u `My Test ActiveForm Library`. Ovaj tekst će se prikazati u Import ActiveX dijalog-prozoru kada budete instalirali aktivnu formu.
9. Sačuvajte projekat. Prihvatite ponuđena imena fajlova (vi ste ih zadali u ActiveForm Wizard-u). Slika 15.8 prikazuje završenu formu.

Slika 15.8
Završena aktivna forma

Generišite, registrujte i uvezite aktivnu formu

Sada možete da generišete, registrujete i uvezete aktivnu formu. Kada je forma generisana postaje ista kao i bilo koja druga ActiveX kontrola. Neću ponovo objašnjavati svaki korak, pošto ste već nekoliko puta radili isto. Postupite ovako:

1. Izaberite Project➥Build MyFormProj iz glavnog menija.
2 Kada se projekat izgeneriše, izaberite Run➥Register ActiveX Server iz glavnog menija.
3. Izaberite Component➥Import ActiveX Control iz glavnog menija. Instalirajte `My Test ActiveForm Library (Version 1)` u `DCLUSR40` kolekciju. Instalirajte je na ActiveX stranicu, ili na bilo koju drugu.

Aktivna forma je sada instalirana kao ActiveX kontrola.

Isprobajte aktivnu formu

Sada je vreme da isprobate aktivnu formu. Ovo će biti prilično jednostavno:

1. Kreirajte nov program.

2. Kliknite na ActiveX jezičak na paleti komponenti i izaberite `MyFormX` taster (onaj sa Delphi-jevom ikonom).
3. Postavite `MyFormX` kontrolu na formu.
4. Pokrenite program i istestirajte ActiveX kontrolu.

I, to je sve. Sa Delphi-jem se jednostavno kreiraju odlične ActiveX kontrole. Jednostavno, ne postoji bolje okruženje za kreiranje ActiveX kontrola od Delphi-ja.

Izmena podrazumevane ikone u paleti komponenti

Sasvim sigurno ćete želeti da promenite ikonu ActiveX komponente. Zamenićete podrazumevanu Delphi-jevu ikonu sa nekom koju ćete sami napraviti. Izmena ikone zahteva sledeće:

1. Kreirajte binarni fajl sa resursima (`.RES`) pomoću Image Editor-a.
2. Kreirajte 24x24 bit mapu. Dajte joj numeričko ime (na primer `2`).
3. Povežite fajl sa resursima sa ActiveX projektom uz pomoć `$R` direktive (Povezivanje resursa je objašnjeno u danu 8, "Kreiranje programa u Delphi-ju" i biće detaljnije objašnjeno u danu 20, "Kreiranje komponenti").
4. Izmenite rutinu za kreiranje ActiveX klase u njenoj celini (`.PAS` fajl aktivne forme). Tipičan izgled rutine za kreiranje ActiveX klase izgleda kao (nalazi se u `initializatation` delu na dnu celine):

```
TActiveFormFactory.Create(
    ComServer,
    TActiveFormControl,
    TMyFormX,
    Class_MyFormX,
    1,                    { Change this number. }
    '',
    OLEMISC_SIMPLEFRAME or OLEMISC_ACTSLIKELABEL,
    tmApartment);
```

   Primetite liniju koju sam obeležio komentarom. Ovaj parametar `TActiveFormFactory.Create` rutine je broj resursa bit mape koju želite da prikažete kao ikonu u paleti komponenti. Ako ste novu bit mapu sačuvali pod brojem `2`, zamenite broj `1` u ovom delu koda sa `2`.
5. Ponovo izgenerišite, ponovo registrujte, uvezite i instalirajte aktivnu formu. Nova ikona bi sada trebala da se vidi u paleti komponenti.

Takođe, mogli ste izmeniti `.RES` fajl aktivne forme i prilagoditi bit mapu pod brojem `1` Vašim potrebama.

Korišćenje ActiveX kontrola i aktivnih formi na Web-u

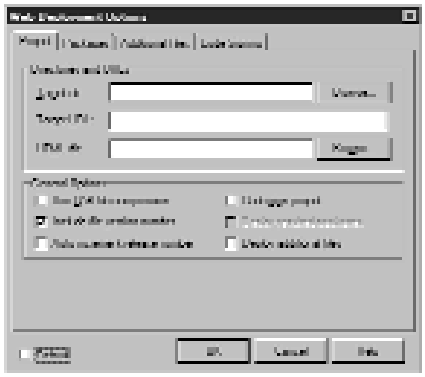
Jedna od odličnih osobina aktivnih formi je da ih možete koristiti na Web stranama. Da biste mogli da koristite aktivnu formu na Web strani, morate koristiti Web Deploy opciju. Korišćenje Web Deploy opcije zahteva Web Server tako da Vam ne mogu prikazati ceo proces. Ipak, mogu Vam prikazati pojedine delove ovog procesa. Kada izaberete Web Deploy opciju Delphi će izvršiti dve operacije:

- Generisaće ActiveX kontrolu i iskopiraće fajl u Web Deploy odredišni direktorijum.
- Kreiraće HTML fajl koji sadrži kod neophodan za učitavanje ActiveX kontrole.

Lokacija fajlova je određena u Web Deployment opcijama. Pogledajmo kako to izgleda.

Web Deployment opcije

Pre korišćenja Web Deploy opcije, morate podesiti Web Deployment opcije. Izaberite Project➥Web Deployment Options iz glavnog menija. Web Deployment Options dijalog-prozor se prikazuje, kao što je prikazano na slici 15.9.



Slika 15.9
Web Deployment Options dijalog-prozor

Na dnu Web Deployment Options dijalog-prozora se nalazi izbor Default. Izberite ga ukoliko želite da opcije koje sada postavite važe i za sve buduće projekte. U najvećem broju slučajeva, koristićete Vaše kontrole na istoj Web strani pa ćete verovatno hteti da postavite podrazumevane opcije tako da sve bude podešeno baš onako kako Vi želite.

Project stranica: Postavljanje direktorijuma i URL-ova

Deo za postavljanje direktorijuma i URL-ova je mesto gde određujete ciljnu lokaciju za Vašu ActiveX kontrolu. Target dir polje predstavlja mesto gde će Delphi iskopirati Vašu ActiveX kontrolu pošto je izgeneriše. Sadržaj ovog polja mora biti direktorijum - to ne sme biti URL.

Ako ste, kao i ja, ograničeni u pristupu direktorijumu gde se nalazi Web sajt, morate precizno naglasiti u koji lokalni direktorijum želite da Delphi smešta generisane ActiveX kontrole. Kasnije ćete, uz pomoć softvera za Web izdavaštvo, postaviti Vaše fajlove na Web sajt.

Target URL polje se koristi da bi se naglasilo na kojoj će se Web stranici nalaziti ActiveX kontrole na Web serveru. Ovaj podatak koristi sam Delphi kada generiše HTML fajl za prikaz kontrole. Na primer, HTML fajl koji je Delphi kreirao za moje potrebe se nalazi u listingu 15.4. (Nekoliko linija je moralo biti izdeljeno zato što su bile preduge za prikaz u tekstu).

Listing 15.4: HTML kod koji je generisao Delphi za ActiveX fajl

```
<HTML>
<H1> Delphi 4 ActiveX Test Page </H1><p>
You should see your Delphi 4 forms or controls
embedded in the form below.
<HR><center><P>
<OBJECT
  classid="clsid:52FB5B97-EDA3-11D1-B47B-0040052A81F8"
  codebase="http://www.home.turbopower.com/~kentr/test/MyFormProj.cab
    #version=1,0,0,0"
  width=275
  height=175
  align=center
  hspace=0
  vspace=0
>
</OBJECT>
</HTML>
```

Primetite URL u codebase izrazu. Ovo je putanja koju sam ja naveo u Target URL polju u Web Deployment Options dijalog-prozoru. Usput, možete korirati ceo `OBJECT` tag iz HTML fajla koji je generisao Delphi direktno u HTML izvorni kod vaše Web strane, kada budete spremni da zvanično koristite Vaš ActiveX kod.

Ime HTML fajla koji kreira Delphi je isti kao i ime projekta sa ekstenzijom `.htm`.

HTML dir polje Web Deployment Options dijalog-prozora se koristi da bi se naglasilo gde Delphi treba da postavi HTML kod koji izgeneriše (pogledajte listing 15.4). Kao i kod Target dir polja, ukoliko nemate direktan pristup Vašim Web direktorijumima, moraćete da naznačite ime lokalnog direktorijuma i da kasnije postavite HTML fajl na Vaš Web sajt.

Project stranica: Odeljak General Options

U ovom odeljku je potrebno da precizirate globalne opcije. Use CAB file compression polje određuje da li će se ActiveX fajl komprimovati. Komprimovanje ActiveX fajla smanjuje njegovu veličinu i čini prenošenje kontrole sa Web-a dosta bržim. Na primer, koristio sam CAB kompresiju za aktivnu formu kreiranu ranije i veličina ActiveX fajla je spala sa 312KB na 204KB u CAB formatu. Windows automatski dekomprimuje i registruje ActiveX komponentu tako da ne postoji razlog zbog kojeg ne biste koristili CAB komprimovanje.

Include file version number izbor označava da li će Delphi uključiti tag sa brojem verzije u `codebase` izraz (pogledajte listing 15.4). Tag sa brojem verzije nije obavezan. Obratite pažnju na činjenicu da neki Web browser-i neće hteti da učitaju ActiveX kontrolu ukoliko je tag sa brojem verzije prisutan (na primer, Netscape Navigator sa ActiveX dodatkom).

Auto Increment release number izbor označava Delphi-ju da automatski inkrementira informaciju o broju verzije svaki put kada budete koristili Vašu ActiveX kontrolu.

Code sign project opcija igra važnu ulogu prilikom korišćenja ActiveX komponenti. Ukoliko se ova opcija uključi, Delphi će izvršiti obeležavanje kontrole. Obeležavanje je proces priključivanja binarnog potpisa fajlu u kome se nalazi ActiveX kontrola. Ovaj potpis, između ostalog, identifikuje kompaniju koja je kreirala ActiveX kontrolu.

Obeležavanje je bitno pošto Internet Explorer očekuje da ActiveX kontrole budu obeležene. Ukoliko je sigurnosni nivo (engl. Security level) Internet Explorer-a postavljen na Medium, ili High, komponente koje nisu obeležene neće biti učitane. Pojednostavljeno, ukoliko želite da Vašu ActiveX kontrolu mogu da koriste i drugi, obeležite je.

Stranica Code Signing dijalog-prozora Web Deployment Options sadrži informacije koje su potrebne da bi se kotrola obeležila. Delphi ne podržava kreiranje credentials fajlova, ili privatnih ključeva koji su potrebni za obeležavanje fajlova. Da biste dobili credentials fajl i privatni ključ morate kontaktirati Microsoft. Da biste dobili više informacija, pretražite Microsoft Web sajt po ključevima “Digital Signing” i “Certificate Authority”.

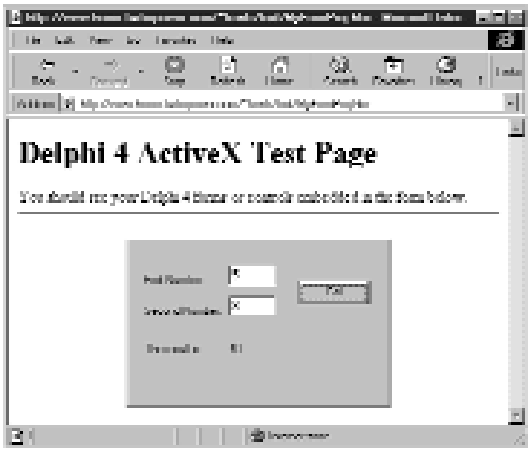
Deploy required packages i Deploy additional files opcije se koriste ukoliko ste generisali ActiveX kontrolu zajedno sa dodatnim paketima, ili ako postoje fajlovi koji se moraju isporučiti zajedno sa kontrolom. Ukoliko izaberete bilo koju od ovih opcija, moraćete da navedete pakete ili fajlove na Packages i Additional Files stranicama Web Deployment Options dijalog-prozora.

Kada god ste u nedoumici pritisnite taster Help na Web Deployment Options dijalog-prozoru. Delphi-jev Help objašnjava svaku stranicu ovog dijalog-prozora.

Postavljanje kontrole na Web

Pošto ste postavili sve opcije, spremni ste da postavite Vašu ActiveX kontrolu. Da biste to uradili, jednostavno izaberite Project➥Web Deploy iz Delphi-jevog glavnog menija. Delphi će izgenerisati ActiveX kontrolu i postaviti na osnovu parametara iz Web Deployment Options dijalog-prozora. Ako ste izabrali korišćenje CAB kompresije, Delphi će takođe komprimovati ActiveX kontrolu u CAB fajl. Zapamtite, ukoliko nemate direktan pristup direktorijumu na Web serveru, moraćete da, uz pomoć posebnog softvera, isporučite Vaše HTML i ActiveX fajlove na Web sajt pre nego što testirate ActiveX kontrolu.

Postavljanje kontrole je jednostavan zadatak. Postavljanje parametara u Web Deployment Options dijalog-prozoru je teži deo posla. Slika 15.10 prikazuje aktivnu kontrolu koju smo ranije kreirali kako se izvršava na Web strani.



Slika 15.10
Aktivna kontrola koja se izvršava na Web strani

NAPOMENA ActiveX kontrole nemaju nikakvih ograničenja po pitanju sigurnosti. Budi pažljivi prilikom preuzimanja ActiveX kontrola iz nepoznatih, ili neproverenih izvora. Kada se ActiveX kontrola preuzme, ona ima pristup celokupnom sistemu. Budite isto tako pažljivi i prilikom pisanja ActiveX kontrola. Uverite se da Vaša ActiveX kontrola neće učiniti ništa loše na nekom drugom računaru.

Zaključak

Neću Vas lagati - postoji mnogo više stvari koje je potrebno naučiti o COM-u i ActiveX-u nego što je ovde prikazano. Nisam govorio o OLE-u. OLE je, kao i ActiveX, podskup COM-a. OLE pruža podlogu COM-u kako bi dozvolio programima da se povežu i ugradio OLE Automation servere u container programe. Ipak, danas ste naučili dosta toga o COM-u i ActiveX-u. Najvažnije je da ste saznali kako da kreirate COM objekte, ActiveX kontrole i aktivne forme. Takođe ste naučili i nešto o postavljanju na Web i kako se ta tehnika koristi prilikom postavljanja Vaših ActiveX kontrola na Web stranu.

Radionica

Radionica sadrži test pitanja koja Vam pomažu da učvrstite svoje razumevanje izložene materije i vežbe koje Vam pomažu da steknete iskustvo u onome što ste naučili. Možete pronaći odgovore na test pitanja u Dodatku A "Odgovori na test pitanja".

Pitanja i odgovori

P **Da li je neophodno da detaljno razumem rad COM-a da bih mogao da pravim ActiveX kontrole u Delphi-ju?**

O Iako je razumevanje COM-a od velike pomoći, ono nije od presudnog značaja za pravljenje ActiveX kontrola u Delphi-ju. Deplhi omogućava jednostavno kreiranje ActiveX kontrola i bez detaljnog poznavanja COM-a.

P **Šta je biblioteka tipova?**

O Biblioteka tipova je binarni fajl koji opisuje interfejse, tipove podataka, metode i klase u COM biblioteci (uključujući i ActiveX).

P **Da li su OLE i COM isto?**

O Ne. COM je osnova na kojoj je izgrađen OLE. OLE je mnogo složeniji i savršeniji od COM-a. Iako OLE ima više mogućnosti, dobro je izbegavati ga, ukoliko je to moguće.

P **Primetio sam da su na mom računaru registrovane neke interesantne ActiveX kontrole, pa sam ih instalirao u Dephi-jevu kolekciju komponenti. One se pojavljuju na paleti sa komponentama, ali kada pokušam da ih koristim dobijam poruku o grešci vezanu za licencu. Zbog čega se to dešava?**

O Ukratko, nije Vam dozvoljeno da koristite te komponente u razvojnim okruženjima (na primer u Delphi-ju). Svaki korisnik ActiveX-a mora isporučiti i registrovati svoje kontrole na svakom računaru na kome će se one koristiti. Da bi onemogućili nelegalno korišćenje komponenti, proizvođači zahtevaju licencu da bi se kontrola mogla koristiti u drugim programima. Kada kupite ActiveX kontrolu od proizvođača, dobićete i licencu.

P **Kreirao sam ActiveX kontrolu i postavio sam je na formu. Program je radio kako treba, ali ukoliko sada pokušam da pokrenem program dobijam izuzetak koji glasi: "`Class not recognized`.". Zbog čega?**

O Svaka ActiveX kontrola mora biti registrovana na računaru na kome će se koristiti. Možda ste nehotice izbrisali ActiveX kontrolu iz Registry-a nakon što ste je registrovali. Da biste je ponovo registrovali, otvorite ActiveX projekat i izaberite Run➥Register ActiveX Server iz glavnog menija. Takođe, možete registrovati OCX fajl pomoću `TREGSVR.EXE` programa.

P Kreirao sam i instalirao aktivnu formu i sve je bilo u redu. Kasnije, hteo sam da izmenim aktivnu formu. Nisam mogao da prevedem projekat aktivne forme zato što se stalno pojavljivala greška: "`Could not create output file MyProj.OCX`". Šta nije u redu?

O Morate da izbacite aktivnu formu iz Delphi-jeve palete sa komponentama pre nego što pokušate da ponovo generišete kontrolu. Kada je kontrola instalirana u Delphi-ju, njegovo okruženje učitava njen OCX fajl tako da on ne može biti prepisan.

P Isporučivanje na Web me zbunjuje. Postoji previše opcija. Da li sam ja jedini koji to ne razume?

O Svakako da niste. Pošto nekoliko puta budete koristili opciju isporučivanja na Web videćete da nije toliko komplikovano kao što možda izgleda na prvi pogled.

P Imam problema prilikom izvršavanja ActiveX komponente na Web stranici. Dobijam grešku od Internet Explorer-a kada god pokušam da učitam stranicu. Greška glasi: "`You current settings prohibit ActiveX controls`". U čemu je problem?

O Vaša ActiveX kontrola nije obeležena. ActiveX kontrola mora biti obeležena pre nego što je Internet Explorer preuzme ukoliko su sigurnosne opcije postavljenje na Medium ili High.

Kviz

1. Koji je osnovni interfejs za sve COM interfejse?
2. Šta je GUID?
3. Šta se dešava kada brojač referenci COM objekta dostigne vrednost 0?
4. Kako se zove Delphi-jev alat za rad sa bibliotekama tipova?
5. Kako se kreiraju GUID-ovi prilikom pisanja COM objekata u Delphi-ju?
6. Šta je potrebno da izaberete iz Object Repository-ja kada želite da kreirate ActiveX komponentu iz VCL komponente?
7. Da li možete da koristite ActiveX kontrole koje ste kreirali u Delphi-ju u okviru Visual Basic okruženja?
8. Kada ste generisali i registrovali ActiveX kontrolu šta je potrebno uraditi da bi se je postaviti na Delphi-jevu paletu sa komponentama?
9. Kako biste izbrisali ActiveX kontrolu koju ste sami napravili iz Registry-a?
10. Da li možete da koristite ActiveX kontrole koje ste kreirali u Delphi-ju na Web strani?

Vežbe

1. Samostalno napravite jednostavan COM objekat iz početka. Nije važno šta COM objekat radi već je bitno da prođete kroz sve faze kreiranja objekta.
2. Napišite Delphi program koji koristi COM objekat koji ste napravili u prvoj vežbi. (Nemojte zaboraviti da registrujete COM objekat)
3. Napravite ActiveX kontrolu iz VCL `TEdit` komponente. Instalirajte komponentu na Delphi-jevu paletu sa komponentama i iskoristite je na formi.
4. Ako imate pristup Visual Basic-u, pokušajte da iskoristite ActiveX komponentu koju ste kreirali u trećoj vežbi.
5. Napravite aktivnu formu proizvoljnog izgleda.
6. Ako imate Web stranu, postavite aktivnu formu koju ste kreirali u petoj vežbi i prikažite je.
7. **Posebna vežba:** Izmenite aktivnu formu koju ste kreirali u petoj vežbi tako da prikazuje drugačiju ikonu na Delphi-jevoj paleti sa komponentama, umesto podrazumevane.

Dan 16

Arhitektura baza podataka u Delphi-ju

Danas ćete početi sa učenjem programiranja baza podataka u Delphi-ju. Ukoliko ste početnik u programiranju baza podataka, u prvom trenutku ćete se osetiti pregaženim. Danas ću pokušati da razbijem konfuziju tako što ću Vam prikazati jasnu sliku lavirinta poznatog kao programiranje baza podataka. Prvo, daću Vam pregled arhitekture baza podataka u Delphi-ju. Zatim, pogledaćemo neke komponente za rad sa bazama podataka

Da ne bude zabune: programiranje baza podataka je komplikovano. Daću Vam pregled programiranja baza podataka koji je na vrlo visokom nivou, ali se neću truditi da pokrijem svaki detalj.

Nisu sve ideje i komponente, koje su prikazane u ovom pogravlju, primenljive u svakoj verziji Delphi-ja. Profesionalna verzija (engl. Professional Edition) Delphi-ja ima više mogućnosti za programiranje baza podataka od standardne verzije (engl. Standard Edition). Klijent/server verzija (engl. Client/Server Edition), opet, ima više mogućnosti i od standardne i od profesionalne verzije.

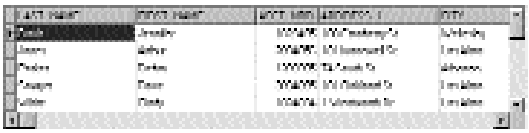
Osnove baza podataka

Programiranje baza podataka dolazi sa čitavom šumom novih pojmova: *BDE*, klijent (*client*), server, *ODBC*, **alias**, *SQL*, upit (*query*), stored procedura i tako dalje. Dobra vest je ta da nisu svi tako strašni kada naučite neke osnove. Kao prvo, recimo nešto o bazama podataka. Kada čujete reči: baza podataka, verovatno da je prvo na šta pomislite skup podataka koji se nalaze u tabeli. Tabela verovatno sadrži polja kao što su: ime, prezime i broj telefona. Ova polja su popunjena podacima i čine jedan zapis u fajlu baze podataka.

Ako je to ono što vidite kada pomislite na baze podataka, znajte da niste daleko, ali da niste baš ni sasvim u pravu. Pojam baza podataka se koristi za opisivanje celokupnog sistema za kreiranje podataka i njihovo održavanje. Tačno je da baza podataka nekada može biti toliko jednostavna kao i prosta tabela. Sa druge strane, baze podataka koje se koriste za rešavanje realnih problema se mogu sastojati od nekoliko, ili čak stotina tabela sa hiljadama, ili milionima zapisa. Te tabele mogu imati jedan, ili više indeksa. Kompletno klijent/server rešenje može sadržati puno upita i stored procedura. (Ne brinite, kasnije ću objasniti neke od ovih pojmova). Kao što vidite, baza podataka je mnogo više od jednostavne tabele sa podacima.

Kada već govorimo o tabelama, hajde da vidimo neke osnovne stvari vezane za tabele. Tabela se sastoji od najmanje dva dela: polja i zapisa. Polja su pojedinačne kategorije podataka u tabeli. Na primer, tabela koja sadrži telefonski imenik će imati polje "prezime", polje "ime", polje "adresa", polje "broj telefona" i tako dalje. Polja se takođe nazivaju i kolonama tabele. Zapis je, sa druge strane, komplet podataka o jednoj osobi: ime i prezime, adresa i tako dalje. Zapisi se takođe nazivaju i vrstama tabele.

Baza podataka je naravno samo kolekcija podataka, ali tabele sa podacima se često prikazuju u tabelarnoj formi. Zaglavlja na vrhu prikazuju imena polja. Svaka vrsta u tabeli sadrži kompletan zapis. Slika 16.1 prikazuje takvu tabelu sa podacima prikazanu u grid (ili tabelarnoj) formi.



Slika 16.1
Tipična tabela sa podacima

NOVI TERMIN Pokazivač na aktivni zapis u okviru baze podataka se zove kurzor (engl. cursor).

Kurzor pokazuje na zapis iz kojeg će se pročitati podaci, ukoliko se to zatraži i u kojem će podaci biti izmenjeni ukoliko se vrednost bilo kog polja promeni. Kurzor se pomera kada se korisnik kreće kroz bazu podataka, ubacuje zapise, briše zapise i tako dalje.

NAPOMENA Kada kažem da je kurzor pokazivač, ne mislim na pokazivač u smislu Object Pascal-a. U stvari, mislim da je on indikator trenutno aktivne pozicije u tabeli.

NOVI TERMIN Kolekcija podataka koju vraća baza podataka se zove *dataset*.

Dataset može biti više od jednostavnog skupa podataka iz jedne tabele. Dataset može biti rezultat upita koji sadrži podatke skupljene iz više tabela. Na primer, recimo da Vi imate bazu podataka sa imenima i adresama Vaših klijenata, njihovih porudžbina i detaljima o svakoj porudžbini. Sada ste, recimo, zatražili detalje o poslednjih 10 porudžbina od kompanije X. Možete primiti dataset koji sadrži infor-

macije iz tabele klijenata, tabele porudžbina i iz tabele detalja o porudžbinama. Iako podaci dolaze iz nekoliko izvora, prikazaće Vam se kao jedan dataset.

Lokalne baze podataka

Najjednostavniji tip baza podataka je lokalna baza podataka. *Lokalna baza podataka* je baza podataka koja se nalazi na jednom računaru. Zamislite da imate program koji treba da upiše niz imena i adresa. Mogli biste da kreirate lokalnu bazu podataka da biste upisali podatke. Ova baza podataka bi, verovatno, bila sačinjena od samo jedne tabele. Ovoj tabeli pristupa samo Vaš program. Niko drugi joj ne može pristupiti. Sve izmene podataka se upisuju direktno u bazu podataka. Paradox, dBASE i Access su tipični primeri lokalnih baza podataka.

Klijent/server baze podataka

Drugi način na koji se mogu implementirati *baze podataka* je *klijent/server* način. Sama baza podataka se nalazi na fajl serveru (engl. file server) i on upravlja njom (to je server deo). Jedan, ili više korisnika (*klijenti*) ima pristup bazi podataka. Korisnici ovog tipa baza podataka se obično nalaze u mreži (engl. network). Pošto su korisnici nezavisni jedan od drugoga, u istom trenutku više od jednog korisnika može pokušati da pristupi bazi podataka. Ovo nije problem za klijent/server baze podataka zato što server zna kako da reši problem istovremenog prisupa bazi podataka.

Korisnici klijent/server baza podataka gotovo nikada ne pristupaju bazi podataka direktno. Umesto toga, pristupaju bazi podataka kroz programe na lokalnim računarima. Ovi programi, koji se zovu klijent programi (engl. *client applications*), brinu o tome da korisnici poštuju određena pravila i da ne rade sa bazom podataka ono što ne bi trebali da rade. Zadatak je klijent programa da spreči korisnika da pokuša da uradi nešto što bi oštetilo bazu podataka.

Serveri za baze podataka:

Dok još govorimo o klijent/server bazama podataka, hajde da kažemo nešto i o serverima za baze podataka. Serveri za baze podataka postoje u nekoliko oblika. Najpopularniji su proizvodi firmi: InterBase (Borland-ova kompanija), Oracle, Sybase, Informix i Microsoft. Kada kompanija kupi jedan od ovih servera za baze podataka, kupuje i licencu koja dozvoljava maksimalnom broju korisnika da pristupi serveru za baze podataka. Ove licence su zovu pozicije (engl. seats). Zamislimo da je kompanija kupila InterBase i licencu za 50 pozicija. Ako kompanija naraste i ukaže se potreba da 75 korisnika ima pristup bazi podataka, moraće da kupi licencu za dodatnih 25 pozicija. Drugi način na koji se prodaju klijent/server baze podataka je po konekciji (engl. per connection). Kompanija kupuje licencu za 50 istovremenih konekcija. Ta kompanija može imati i 1000 korisnika baze podataka, ali samo 50 može biti povezano na server u isto vreme. Tržište servera za baze podataka je veliki posao, nema sumnje.

Single-tier, Two-tier i Multitier arhitektura baza podataka

Lokalne baze podataka se obično nazivaju single-tier bazama podataka. *Single-tier baza podataka* je baza podataka kod koje se sve promene, kao što su izmena podataka, ubacivanje, ili brisanje zapisa, dešavaju momentalno. Program ima direktnu vezu sa bazom podataka.

Kod *two-tier baza podataka*, klijent program komunicira sa serverom za baze podataka kroz drajvere (engl. database drivers). Server za baze podataka preuzima odgovornost za manipulisanje konekcijama, dok je klijent program odgovoran za korektnost podataka koji se upisuju u bazu podataka. Veliki teret je stavljen na klijent program kako bi se osigurao integritet baze podataka.

Kod *multitier klijent/server arhitekture*, klijent program komunicira sa jednim, ili više servera za programe koji komuniciraju sa serverom za baze podataka. Ovi programi srednjeg nivoa se nazivaju serverima za programe zato što obrađuju zahteve klijent programa. Jedan server za programe može služiti raspodeli podataka (uslužujući i odgovarajući na zahteve klijenta) dok drugi može upravljati sigurnosnim sistemom.

Klijent programi se izvršavaju na lokalnim računarima. Serveri za programe se obično izvršavaju na serveru. Sama baza se može, takođe, nalaziti na posebnom serveru. Ideja multitier arhitekture je u tome da omogući klijent programima da budu vrlo mali i to tako što serveri za programe odrađuju najveći deo posla. Ovaj način rada Vam dozvoljava da pišete takozvane sitne klijent programe (engl. thin-client applications).

Sledeći razlog za korišćenje multitier arhitekture je upravljanje programerskim resursima. Klijent programe mogu pisati i manje iskusni programeri zato što klijent programi komuniciraju sa serverima za programe koji kontrolišu pristup samoj bazi podataka. Servere za programe mogu pisati iskusniji programeri koji poznaju pravila po kojima funkcioniše pristup bazama podataka. Gledano na drugi način to znači da servere za programe pišu iskusniji programeri čiji je posao da zaštite podatke od mogućeg oštećenja koje mogu izazvati loše napisani klijent programi.

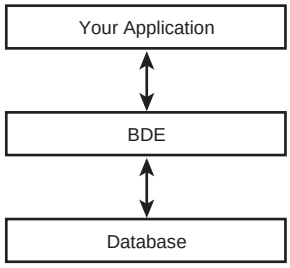
Iako ima izuzetaka, najveći broj lokalnih baza podataka koristi single-tier arhitekturu. Klijent/server baze podataka koriste ili two-tier, ili multitier arhitekturu.

Na koji način ovo utiče na Vas? Najveći broj programa koje budete pisali u Delphi-ju za korišćenje klijent/server baza podataka će biti klijent programi. Iako možete biti jedan od programera kojima će biti poveren zadatak pisanja server programa, uglavno možete računati na to da ćete uglavnom pisati klijent programe. Kao aplikativni programer, ne možete direktno komunicirati sa serverima za baze podataka. Pogledajmo kako program pisan u Delphi-ju komunicira sa bazom podataka.

Borland Database Engine (BDE)

Da bi omogućio pristup lokalnim i klijent/server bazama podataka, Delphi koristi Borland Database Engine (BDE). BDE je kolekcija DLL-ova i uslužnih programa koji omogućavaju pristup raznim bazama podataka.

Da biste komunicirali sa klijent/server bazama podataka morate imati klijent/server verziju Delphi-ja (engl. Delphi Client/Server Edition). Ova verzija Delphi-ja se isporučuje sa SQL Links drajverima koje BDE koristi za komunikaciju sa klijent/server bazama podataka. Slika 16.2 prikazuje odnos između Vašeg programa, BDE-a i baze podataka.



Slika 16.2
Vaš program, BDE i baza podataka

28671602.eps
31286-7
p2/v4

BDE drajveri

Prirodno, formati baza podataka i njhovi API-ji (engl. Application Programming Interface) se razlikuju od slučaja do slučaja. Iz tog razloga, BDE se isporučuje sa skupom drajvera koji omogućuju Vašim programima da komuniciraju sa nekoliko različitih tipova baza podataka. Ovi drajveri prevode komande bazama podataka visokog nivoa (kao što su open i post) u komande specifične za određeni tip baze

podataka. Ovakav način rada dozvoljava Vašim programima da pristupaju bazama podataka bez potrebe da znaju na koji način te baze podataka funkcionišu.

U zavisnosti od toga koju verziju Delphi-ja posedujete, na Vašem računaru će se nalaziti određeni skup drajvera. Sve verzije Delphi-ja dolaze sa drajverom koji omogućava pristup Paradox i dBASE bazama podataka. Ovaj drajver, koji se zove `STANDARD`, omogućava sve što je potrebno da biste radili sa ovim tipovima lokalnih baza podataka.

Klijent/server verzija Delphi-ja uključuje drajvere za vezivanje na baze podataka koje se nalaze na Sybase, Oracle, Informix, InterBase i drugim serverima.

BDE alias-i

BDE koristi alias za pristup određenoj bazi podataka. Ovo je jedan od onih termina koji vas može zbuniti. Termini *alias* i *baza podataka* se često ravnopravno upotrebljavaju kada se govori o BDE-u.

NOVI TERMIN BDE *alias* predstavlja skup parametara koji opisuju način za vezivanje na bazu podataka.

Na kraju krajeva, alias nije ništa posebno. U najjednostavnijoj formi, alias govori BDE-u koji drajver da upotrebi i gde se na disku nalaze fajlovi sa bazom podataka. Ovo je primer kako se postavljaju alias-i za pristup lokalnim bazama podataka. U ostalim slučajevima, kao što su alias-i za pristup klijent/server bazama podataka, alias-i takođe sadrže i druge informacije kao što su: maksimalna veličina BLOB polja, maksimalni broj polja, način otvaranja baze podataka, ili korisnikovo ime. Pošto ste kreirali alias za Vašu bazu podataka, možete ga upotrebiti da biste izabrali bazu podataka u svojim Delphi programima. Kasnije, u odeljku "Kreiranje BDE alias-a" reći ću Vam kako da kreirate alias-e za Vaše baze podataka.

Baze podataka koje su ugrađene u Delphi

Dok smo kod alias-a, pogledajmo alias-e koji se već nalaze na Vašem računaru. Da biste videli postojeće alias-e, izvršite sledeće operacije:

1. Pokrenite Delphi, ili kreirajte nov program ukoliko je Delphi već pokrenut.
2. Pomerite se na Data Access stranicu u paleti sa komponentama, izaberite Table komponentu i postavite je na formu.
3. Kliknite na `DatabaseName` osobinu u Object Inspector-u, pa zatim kliknite na strelicu da biste ostvorili listu alias-a.

Posle ovih operacija, videćete listu baza podataka koje su Vam na raspolaganju. Barem jedan od njih bi trebao da bude `DBDEMOS` alias. Ovaj alias je postavio sam Delphi prilikom instalacije. Izaberite `DBDEMOS` bazu podataka iz liste.

NAPOMENA Lista baza podataka koju vidite zavisi od nekoliko činilaca. Prvo, zavisi od toga da li imate standardnu, profesionalnu, ili klijent/server verziju Delphi-ja. Takođe zavisi od toga da li ste instalirali Local InterBase. Konačno, bitno je da li imate instaliran C++Builder, ili neki drugi Borland-ov proizvod (na primer Visual dBASE ili IntraBuilder). U tom slučaju videćete dodatne baze podataka.

Dok smo još ovde, izaberite `TableName` osobinu i pogledajte listu tabela koje su na raspolaganju. Tabele koje vidite se nalaze u okviru izabrane baze podataka (izabranog alias-a). Izaberite neki drugi alias u `DatabaseName` osobini. Pogledajte ponovo listu tabela i videćete razliku.

SQL Links

Klijent/server verzija Delphi-ja dolazi sa SQL Links dodatkom za BDE. SQL Links je kolekcija dodatnih drajvera za BDE. Ovi drajveri omogućuju Delphi programima da se vežu na klijent/server baze podataka kao što su Oracle, InterBase, Informix, SyBase i Microsoft. Detalji o isporučivanju SQL Links drajvera se takođe nalaze u fajlu `DEPLOY.TXT`.

Local Interbase:

Standardna i profesionalna verzija Delphi-ja dolaze sa kopijom InterBase-a za jednog korisnika (engl. single-user). Local InterBase je upravo ono što i samo ime kaže: verzija InterBase-a koja radi sa lokalnim bazama podataka. Sa druge strane, klijent/server verzija InterBase-a, je puna (klijent/server) verzija InterBase-a. Glavni razlog zbog kojeg se InterBase isporučuje sa Delphi-jem je što možete praviti programe koji rade sa lokalnim bazama podataka, dok kasnije možete preći na rad klijent/server bazama podataka bez izmena koda. Dakle, imate mogućnost da razvijate svoje klijent/server programersko umeće bez potrebe da trošite novac na klijent/server baze podataka.

Ako pokušate da pristupite Local InterBase tabeli u toku razvoja programa, ili tokom izvršavanja, moraćete da unesete korisničko ime i lozinku. Local InterBase administrator je podešen za korisnika `SYSDBA` sa lozinkom `masterkey`. Možete koristiti ove podatke, ili možete pokrenuti InterBase Server Manager pomoćni program i dodati sebe kao novog korisnika InterBase sistema.

Delphi-jeve komponente za rad sa bazama podataka

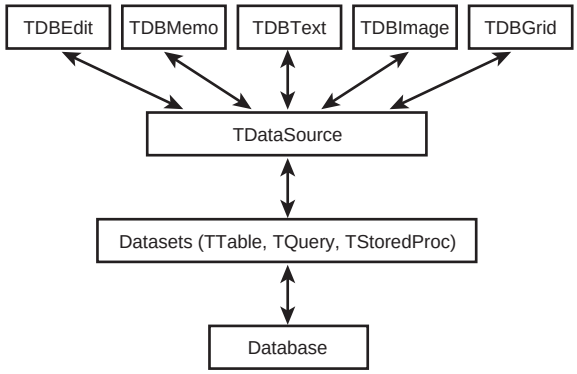
Prethodni odeljak nije baš materijal koji bi vas naterao da provedete celu noć okrećući stranice. Ipak, bitno je da razumete kako se svi elementi baze podataka uklapaju u jednu celinu. Sa tim znanjem, možete usmeriti svoju pažnju na komponente za rad sa bazama podataka koje se nalaze u VCL-u i na način na koji treba koristiti te komponente za pravljenje programa za rad sa bazama podataka. Prvo,

daću Vam kratak pregled VCL komponenata za rad sa bazama podataka a onda ćemo usmeriti pažnju na pojedinačne klase i komponente.

VCL komponente za rad sa bazama podataka spadaju u jednu od dve kategorije: vizuelne i ne-vizuelne komponente. Jednostavno, ne-vizuelne komponente za pristup bazama podataka omogućavaju mehanizam pomoću kojeg stižete do samih podataka a vizuelne komponente za rad podacima Vam omogućavaju da pregledate i menjate podatke. Komponente za pristup bazama podataka su izvedene iz TDataSet klase i to su: `TTable`, `TQuery` i `TStoredProc`. Vizuelne komponente za rad sa podacima su, na primer: `TDBEdit`, `TDBListBox`, `TDBGrid`, `TDBNavigator` i druge. Ove komponente funkcionišu slično kao i standardne edit, listbox i grid komponente, s tom razlikom što su vezane za određenu tabelu, ili polje u tabeli. Izmenom podataka u nekoj od kontrola za rad sa podacima zapravo menjate i samu bazu podataka.

Sve VCL komponente za rad sa bazama podataka možemo nazvati komponentama za rad sa podacima. Koristim izraz komponente za pristup bazama podataka za ne-vizuelne komponente na Data Access stranici palete sa komponentama, dok termin komponente za rad sa podacima koristim za vizuelne komponente koje se nalaze na Data Controls stranici palete sa komponentama.

Ove dve grupe komponenti ne mogu komunicirati direktno. Umesto toga, `TDataSource` komponenta predstavlja vezu između `TDataSet` komponenti i vizuelnih komponenti za rad sa podacima. Ova relacija je prikazana na slici 16.3.



Slika 16.3
Arhitektura VCL komponenti za rad sa bazama podataka

28671603.eps
31286-7
p2/v4

Bavićete se detaljnije ovim komponentama, ali prvo biste trebali da odradite kratku vežbu koja će ilustrovati relaciju opisanu u ovom odeljku. Pokrenite Delphi, ili kreirajte novi program ukoliko je Delphi već pokrenut. Uradite sledeće:

1. Postavite `Table` komponentu na formu.
2. Pronađite `DatabaseName` osobinu u Object Inspector-u i izaberite `DBDEMOS` bazu podataka.
3. Pronađite `TableName` osobinu i izaberite `ANIMALS.DBF` tabelu.
4. Postavite `DataSource` komponentu na formu i postavite njenu `DataSet` osobinu na `Table1` (izaberite `Table1` iz padajuće liste). Izvor podataka (engl. `Data Source`) je sada vezan za dataset.
5. Postavite `DBGrid` komponentu na formu i postavite njenu `DataSource` osobinu na `DataSource1`. Sada ste povezali mrežu (engl. grid) za izvor podataka i, indirektno, za dataset (u stvari tabelu).
6. Sada kliknite na `Table` komponentu na Vašoj formi da biste je izabrali. Postavite njenu `Active` osobinu na `True`. Sada imate podatke u tabeli.

To je bilo jednostavno, ali još uvek niste završili. Primetite, usput, da možete koristiti skrolere na mreži, čak i za vreme kreiranja programa. U redu, još nekoliko koraka:

1. Postavite `DBImage` komponentu na formu i postavite njenu DataSource osobinu na `DataSource1`. Takođe, postavite njeno `DataField` osobinu na `BMP` (`BMP` je ime polja u `ANIMALS.DBF` tabeli koje sadrži sliku životinje). Pogledajte, to je riba! Promenite veličinu DBImage komponente kako bi odgovarala veličini slike koja se prikazuje u njoj.
2. Postavite `DBNavigator` komponentu na formu i postavite njenu `DataSource` osobinu na `DataSource1`.

Pokrenite program. Kliknite na bilo koji taster na `DBNavigator` komponenti. Kada kliknete na Next Record taster, pokazivač zapisa u `DBTable` se menja kao i slika u `DBImage` komponenti. Sve to bez pisanja i jedne linije koda!

Komponente za pristup bazi podataka se koriste da bi ste se povezali sa određenom bazom podataka, ili sa određenom tabelom u bazi podataka. `Table` komponenta se koristi za pristup tabeli. Ovo je najjednostavniji način za pristup podacima iz tabele.

Korišćenje `Query` komponente je način za pristup bazi podataka korišćenjem komandi strukturiranog jezika za postavljanje upita (engl. Structured Query Language, SQL). SQL je moćniji način za pristupanje tabelama, ali je i komplikovaniji. Koristićete ili `Table`, ili `Query` komponentu za pristup tabeli, ali ne obe. Sledeća komponenta je `StoredProc` koja Vam omogućava pristup bazama podataka kroz takozvane stored procedure. *Stored procedura* je kolekcija naredbi za rad sa bazama podataka koje izvršavaju jednu, ili više operacija nad bazom podataka. Stored procedure se obično koriste za nizove komandi za rad sa bazama podataka koje se često upotrebljavaju.

`TDataSet` Klasa

`TDataSet` klasa je osnovna klasa za `TTable`, `TQuery` i `TStoredProc`. Mnoge osobine, metodi i događaji koje koriste ove klase su zapravo definisane u `TDataSet`. Zbog toliko karakteristika koje su nasleđene iz `TDataSet` izlistaću primarne osobine, metode i događaje koji su karakteristični za svaku izvedenu klasu.

Tabela 16.1 prikazuje najčešće korišćene osobine `TDataSet` klase. Tabela 16.2 prikazuje primarne metode, dok tabela 16.3 prikazuje primarne događaje.

Tabela 16.1: Primarne osobine klase `TDataSet`

Osobina	Opis
`Active`	Otvara dataset ukoliko je `True` i zatvara ga ukoliko je `False`.
`AutoCalcFields`	Utvrđuje kada su proračunata polja zaista proračunata.
`Bof`	Vraća `True` ukoliko je kurzor na prvom zapisu u dataset-u, ili False ukoliko nije.
`CachedUpdates`	Kada je `True`, sve izmene se čuvaju u kešu na klijent računaru, dok se ne izvrši cela transakcija. Kada je `False`, sve izmene nad bazom se dešavaju polje-po-polje.
`CanModify`	Određuje da li korisnik može menjati dataset.
`DataSource`	`DataSource` komponenta vezana za ovaj dataset.
`DatabaseName`	Ime baze podataka koja se trenutno koristi.
`Eof`	Vraća `True` ukoliko se kurzor nalazi na kraju fajla, ili `False` ukoliko se ne nalazi.
`FieldCount`	Broj polja u dataset-u. Pošto dataset može biti dinamički (rezultat upita, na primer), broj polja se može menjati od jednog zahteva do drugog.
`Fields`	Niz `TField` objekata koji sadrži informacije o poljima u dataset-u.
`FieldValues`	Vraća vrednost nekog polja u trenutnom zapisu. Vrednost se prikazuje u formi Variant tipa podatka.
`Filter`	Izraz koji određuje koji zapisi će se naći u dataset-u.

`Filtered`	Kada je `True`, dataset se filtrira, ili na osnovu `Filter` osobine, ili na osnovu `OnFilterRecord` događaja. Kada je `False`, vraća se ceo dataset.
`FilterOptions`	Određuje kako se filteri primenjuju.
`Found`	Ispituje da li je operacija pretraživanja uspela.
`Handle`	Predstavlja BDE kurzor na datastet. Koristi se samo kada se zahtevaju direktni pozivi BDE-u.
`Modified`	Pokazuje da li je aktivni zapis menjan, ili ne.
`RecNo`	Sadrži broj trenutnog zapisa u dataset-u.
`RecordCount`	Vraća ukupan broj zapisa u dataset-u.
`State` `dsInserti`	Vraća trenutno stanje dataset-a (`dsEdit`, `dsBrowse`, tako dalje).
`UpdateObject`	Određuje `TUpdateObject` komponentu koja se koristi za keširanje izmena.
`UpdatePending`	Kada je `True`, bafer za keširanje izmena sadrži izmene koje se još nisu odrazile na dataset.

Tabela 16.2: Primarne metode `TDataSet` klase

Metoda	Opis
`Append`	Kreira prazan zapis i dodaje ga na kraj dataset-a.
`AppendRecord`	Dodaje prazan zapis na kraj dataset-a sa zadatim podacima i izvršava izmenu.
`ApplyUpdates`	Nalaže bazi podataka da izvrši sve izmene koje se nalaze u kešu. Izmene se ne upisuju dokle god se ne pozove `CommitUpdates` metoda.
`Cancel`	Odustajanje od izmena u trenutnom zapisu, ukoliko one već nisu upisane.
`CancelUpdates`	Odustajanje od svih izmena koje se nalaze u kešu.
`ClearFields`	Prazni sadržaj svih polja u aktivnom zapisu.
`CommitUpdates`	Nalaže bazi podataka da izvrši sve izmene i očisti keš za izmene.
`Close`	Zatvara dataset.
`Delete`	Briše aktivni zapis.
`DisableControls`	Zabranjuje unos u sve kontrole za podatke koje su vezane za taj dataset.
`Edit`	Dozvoljava izmene aktivnog zapisa.
`EnableControls`	Omogućava unos u sve kontrole za podatke koje su vezane za taj dataset.
`FetchAll`	Preuzima sve zapise koji se nalaze između kurzora i kraja dataset-a i lokalno ih zapisuje.

nastavlja se

`FieldByName`	Vraća `TField` pokazivač na polje sa određenim imenom.	*nastavak*
`FindFirst`	Vraća prvi zapis koji zadovoljava kriterijum pretraživanja.	
`FindNext`	Vraća sledeći zapis koji zadovoljava kriterijum pretraživanja.	
`FindLast`	Vraća poslednji zapis koji zadovoljava kriterijum pretraživanja.	
`FindPrior`	Vraća prethodni zapis koji zadovoljava kriterijum pretraživanja.	
`First`	Postavlja kurzor na prvi zapis u dataset-u.	
`FreeBookmark`	Briše poslednji marker koji je postavljen sa `GetBookMark` i oslobađa memoriju koja je bila alocirana za taj marker.	
`GetBookMark`	Postavlja marker na trenutno aktivni zapis.	
`GetFieldNames`	Vraća listu imena polja u dataset-u.	
`GotoBookmark`	Postavlja kurzor na polje određeno datim markerom.	
`Insert`	Dodaje novi zapis i postavlja dataset u stanje za izmene.	
`InsertRecord`	Dodaje zapis u dataset sa datim podacima i izvršava izmenu.	
`Last`	Postavlja kurzor na poslednje polje u dataset-u.	
`Locate`	Pretražuje dataset prema određenom zapisu.	
`Lookup`	Traži zapis na najbrži mogući način i vraća podatke koji se nalaze u zapisu.	

Tabela 16.2: Primarne metode `TDataSet` klase

Metoda	Opis
`Post`	Zapisuje izmenjene podatke u bazu podataka, ili u keš za izmene.
`Prior`	Pomera kurzor na prethodni zapis.
`Refresh`	Osvežava podatke u dataset-u koristeći bazu podataka.
`RevertRecord`	Kada se koristi keš za izmene ova metoda ignoriše izmene koje su izvršene u aktivnom zapisu a koje još nisu upisane u bazu podataka.
`SetFields`	Postavlja vrednost svim poljima u zapisu.
`UpdateStatus`	Vraća trenutno stanje izmena kada se koristi keš za izmene.

Tabela 16.3: Događaji klase `TDataSet`

Događaj	Opis
`AfterCancel`	Generiše se pošto se odustane od izmena.
`AfterClose`	Generiše se kada se dataset zatvori.
`AfterDelete`	Generiše se kada se zapis obriše iz dataset-a.
`AfterEdit`	Generiše se kada se izvrši izmena zapisa.
`AfterInsert`	Generiše se kada se ubaci novo polje.
`AfterOpen`	Generiše se kada se dataset otvori.
`AfterPost`	Generiše se kada se izmene upišu u bazu podataka.
`BeforeCancel`	Generiše se pre nego što se odustane od izmena.

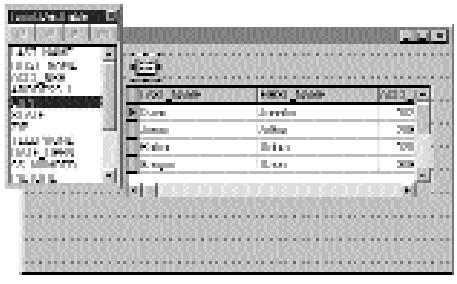
`BeforeClose`	Generiše se pre zatvaranja dataset-a.
`BeforeDelete`	Generiše se pre brisanja zapisa iz dataset-a.
`BeforeEdit`	Generiše se pre nego što se zapis izmeni.
`BeforeInsert`	Generiše se pre ubacivanja novog polja.
`BeforeOpen`	Generiše se pre otvoraranja dataset-a.
`BeforePost`	Generiše se pre nego što se izmene upišu u bazu podataka.
`OnCalcFields`	Generiše se kada se izvrše proračuni nad proračunatim poljima.
`OnDeleteError`	Generiše se ako je došlo do greške prilikom brisanja zapisa.
`OnEditError`	Generiše se ako je došlo do greške prilikom izmene zapisa.
`OnFilterRecord`	Generiše se kad god se pristupi novom polju, ako je `Filter` postavljen na `True`.
`OnNewRecord`	Generiše se kada se doda novi zapis u dataset.
`OnPostError`	Generiše se kada dođe do greške prilikom upisa izmena u bazu podataka.
`OnUpdateError`	Generiše se kada dođe do greške prilikom upisa podataka iz keša za izmene u bazu podataka.
`OnUpdateRecord`	Generiše se kada se izmene iz keša upišu u bazu podataka.

Editor polja (The Fields Editor)

Bilo koji potomak klase `TDataSet` (`TTable`, `TQuery`, ili `TStoredProc`) dozvoljava pristup editoru polja u toku pisanja programa. Editor polja Vam dozvoljava da izaberete polja koja želite da uključite u dataset.

Da biste pozvali editor polja, kliknite desnim tasterom na `Table`, `Query`, ili `StoredProc` komponentu na Vašoj formi i izaberite Fields Editor iz konteksnog menija. Prikazuje se editor polja. Na početku editor polja je prazan, pri čemu su sva polja uključena u dataset.

Možete dodati koliko god želite polja u dataset, birajući Add fields iz konteksnog menija editora polja. Takođe, možete kreirati nova polja u tabeli, birajući New field iz konteksnog menija. Slika 16.4 prikazuje izgled editora polja posle dodavanja polja.



Slika 16.4
Editor polja

Pošto ste dodali polja u dataset, možete kliknuti na bilo koje polje da biste izmenili njegove osobine. Osobine se prikazuju u Object Inspector-u koji Vam dozvoljava da izmenite format prikaza, ograničenja, tekst za prikaz i druge osobine polja.

Keširane izmene

Keširane izmene Vam dozvoljavaju da odredite kada će se izmene upisati u bazu podataka. Keširane izmene kontroliše osobina `CachedUpdates`. Kada je keširanje izmena dozvoljeno, izmene izvršene u zapisima se ne upisuju direktno u bazu podataka. Umesto toga, izmene se upisuju u keš koji se nalazi na lokalnom računaru. Zapisi se drže u kešu, dokle god ne pozovete ApplyRecords metod. Da biste zanemarili sve izmene u kešu, pozovite `CancelUpdates` metodu. Zanemarujete izmene izvršene u aktivnom zapisu pozivajući metod `RevertRecord`.

Kada se keširanje ne koristi (`CachedUpdates` je `False`), sve izmene izvršene u zapisu se upisuju u bazu podataka kada kurzor promeni mesto. Ovo je dobro za lokalne baze podataka, ali nije preporučljivo za klijent/server baze podataka iz više razloga. Uglavnom ćete čuti kako ljudi govore da je protok podataka kroz mrežu glavni razlog za korišćenje keširanja izmena. Iako je tačno da keširanje izmena pomaže prilikom smanjivanja protoka podataka kroz mrežu, vrednost keširanja izmena je daleko veća. Dozvolite mi da bliže objasnim.

Mnoge klijent/server baze podataka vraćaju rezultate upita koji se mogu samo čitati (engl. read-only). Jedna od prednosti keširanja izmena je i što korisnik može raditi sa lokalnom kopijom dataset-a, izmeniti je, ukoliko je to potrebno i zatim odjednom upisati sve izmene u bazu podataka. Ovo je moguće zato što server za baze podataka radi sa izmenama, ubacivanjima i brisanjima zapisa iz dataset-a koji se mogu samo čitati. Lokalna baza podataka mora zaključati zapise kada se oni aktivno menjaju. Kada je zapis zaključan ostali korisnici baze podataka mu ne mogu pristupiti. Korišćenjem keširanja izmena bitno se smanjuje vreme koje zapis mora provesti zaključan.

Sledeća prednost keširanja izmena je što korisnik može vršiti više izmena u dataset-u i na kraju ih, ili upisati (*commit*, apply), ili zanemariti (*rollback*, cancel). Ovo je mač sa dve oštrice, jer zbog eventualne greške na serveru, u trenutku kada se izmene upisuju u bazu podataka, sve izmene mogu biti izgubljene.

Jedna mana keširanja izmena je što više korisnika može raditi sa istim zapisom u isto vreme. Tada nastaje trka da bi se videlo ko će prvi dobiti izmenjen zapis. U praksi, problem se ublažava raznim tehnikama ugrađenim u klijent programe koje proveravaju da li je došlo do višestrukog menjanja podataka u zapisu. Na primer, ako Jovan pokuša da upiše izmene u zapis, baza podataka i/ili klijent program će ga obavestiti da je Marija izmenila zapis, pošto ga je Jovan primio iz baze podataka. Jovan će morati da osveži svoju kopiju dataset-a da bi video da li treba da unese izmene u zapis.

Table komponenta

Table komponenta koja je predstavljena TTable klasom omogućava najbrži i najjednostavniji način za pristup tabelama. Tabele su više nego dovoljne za mnoge singletier programe za rad sa bazama podataka. Često ćete koristiti Table komponentu kada budete trebali da radite sa lokalnom bazom podataka i Query komponentu kada budete trebali da radite sa serverima za baze podataka baziranima na SQL-u.

TTable klasa ima dosta osobina i metoda koje su dodate na one iz TDataSet klase. Tabela 16.4 prikazuje primarne osobine klase TTable, dok Tabela 16.5 prikazuje primarne metode ove klase. Zapamtite, prikazane osobine i metode su specifične za TTable klasu i u ovim tabelama se ne nalaze osobine i metode nasleđene iz TDataSet.

Za najveći broj osobina i metoda možemo reći da su prilično jasne. Pod time se podrazumeva da uglavnom možete saznati šta osobina, ili metod rade prostim čitanjem imena. Ne treba mnogo vremena da bi se shvatilo da LockTable metoda zaključava tabelu za specifične poslove koje obavlja program, a da je UnlockTable ponovo otključava. Takođe, ne morate imati IQ od 150 da biste pogodili šta rade CreateTable, DeleteTable i RenameTable metode. Zbog te činjenice, neću posebno objašnjavati svaki aspekt, svaku osobinu i svaki metod koji se nalazi u tabelama. Umesto toga, prelazimo na neke interesantnije aspekte Table komponente.

Tabela 16.4: Primarne osobine klase TTable

Osobina	Opis
Exlusive	Zaključava lokalnu tabelu, tako da samo aktuelni program može da je koristi.
IndexDefs	Sadrži informacije o ideksima tabele.
IndexFieldCount	Broj indeksa koji sačinjavaju trenutni ključ.
IndexFieldNames	Koristi se da bi se trenutni ključ podesio da radi sa indeksima na data polja.
IndexFields	Koristi se da bi se dobila informacija o poljima u indeksu.
KeyFieldCount	Broj polja koja se koriste prilikom pretraživanja po parcijalnom ključu.
MasterFields	Polje, ili polja koja treba da povežu glavnu (engl. master table) i tabelu sa detaljima (engl. detail table).
MasterSource	Tabela koju treba koristiti kao glavnu kada se ova tabela koristi kao detaljna.
ReadOnly	Određuje da li je ova tabela samo za čitanje.
TableName	Ime tabele iz baze podataka.
TableType	Tip tabele (Paradox, dBASE, ili ASCII).

nastavlja se

Tabela 16.5: Primarne metode klase `TTable` *nastavak*

Metoda	Opis
`AddIndex`	Kreira novi indeks za tabelu.
`ApplyRange`	Postavlja dataset u režim korišćenja samo određenog opsega zapisa. Samo zapisi koji se nalaze u tom opsegu (koji se određuje pomoću `SetRangeStart` i `SetRangeEnd`) su na raspolaganju za pregledanje i izmene.
`BatchMove`	Premešta zapise iz dataset-a u tabelu.
`CancelRange`	Vraća dataset u režim korišćenja svih zapisa.
`CreateTable`	Ponovo kreira tabelu koristeći nove informacije.
`DeleteIndex`	Uklanja sekundarni indeks.
`DeleteTable`	Briše tabelu.
`EmptyTable`	Briše sve zapise iz tabele.
`GetIndexNames`	Vraća listu svih indeksa u tabeli.
`GotoKey`	Pomera kurzor na zapis koji je određen trenutnim ključem.
`GotoNearest`	Pomera kurzor na zapis koji je najsličniji trenutnom ključu.
`LockTable`	Zaključava tabelu tako da drugi programi ne mogu da je koriste.
`RenameTable`	Menja ime tabele.
`SetKey`	Omogućava da postavite ključeve za dataset.

Tabela 16.5: Primarne metode klase `TTable`

Metoda	Opis
`SetRange`	Postavlja početak i kraj opsega i postavlja dataset u režim korišćenja samo tog opsega zapisa. Korišćenje ovog metoda proizvodi isti rezultat kao i korišćenje `SetRangeStart`, `SetRangeEnd` i `ApplyRange` metoda.
`SetRangeEnd`	Postavlja kraj opsega.
`SetRangeBegin`	Postavlja početak opsega.
`UnlockTable`	Otključava tabelu koja je prethodno zaključana korišćenjem metode `LockTable`.

Kao što ste videli, `DatabaseName` osobina se koristi da bi se izabrao BDE alias. Za lokalne baze podataka, radije ćete uneti ime direktorijuma u kome se nalaze fajlovi sa bazama podataka, nego što ćete birati alias-e. `TableName` osobina će tada sadržati listu tabela u tom direktorijumu.

Filteri

Zajednička potreba programa za rad sa bazama podataka je filtriranje tabela. Pre nego što detaljno opišem filtere, želeo bih da objasnim za šta se filteri, prvenstveno,

koriste na lokalnim bazama podataka. Filteri se retko koriste kod klijent/server baza podataka. Umesto njih, koriste se SQL upiti koji daju isti rezultat kao i filteri kod lokalnih baza podataka.

Zašto koristiti filtere? Zamislite da imate tabelu sa više hiljada zapisa, ali da trenutno želite da radite samo sa malim podskupom svih tih zapisa. Recimo da imate bazu podataka koja sadrži imena i adrese svih korisnika računara širom sveta. Vaša kompanija prodaje ove podatke drugim kompanijama koje šalju cirkularna pisma.

Ja sam Vas pozvao i želim da naručim podatke od Vaše kompanije, ali želim samo podatke o korisnicima koji žive u Srbiji. Mogli biste da isfiltrirate svoju tabelu po imenu zemlje i da dobijete podatke samo za korisnike koji žive u Srbiji. Ili, zamislite da su Vas pozvali ljudi iz Borland-a koji od Vas traže podatke o korisnicima koji žive u Srbiji, ali čije je primarno zanimanje programiranje. U tom slučaju isfiltrirali biste tabelu po imenu zemlje i po korisnikovim interesovanjima i dobili biste tražene podatke.

Korišćenje filtera u `Table` komponenti. Sa filterima se, u okviru `Table` komponente radi na jedan od dva načina: kroz `Filter` osobinu, ili kroz `OnFilterRecord` događaj. Pre nego što ovo objasnim, reći ću nešto o `Filtered` osobini. Ova osobina određuje da li je tabela filtrirana. Ukoliko je `Filtered True`, tabela će biti filtrirana prema filteru koji je trenutno u upotrebi (bilo da je postavljen pomoću `Filter` osobine, bilo pomoću rezultata `OnFilterRecord` događaja). Ukoliko je `Filtered False`, sadržaj `Filter` osobine se ignoriše i događaj `OnFilterRecord` se nikada neće generisati.

`Filter` osobina se satoji od imena polja, logičkog operatora i od vrednosti. Filter može da izgleda, na primer, ovako:

```
FirstName = 'Bob'
```

Ovaj izraz, u suštini, govori sledeće: "Daj mi sve zapise u kojima je vrednost polja `FirstName` Bob". Filteri, takođe, mogu da sadrže ključne reči `AND`, `OR` i `NOT`:

```
CustNo = 1384 AND ShipDate < '1/1/94'
```

Imena polja i logičkih operatora se mogu pisati kako malim, tako i velikim slovima (engl. no-case-sensitive). Sledeća dva izraza za filtriranje su identrični:

```
CustName = 'TurboPower' and ShipDate < '1/1/94'
CUSTNAME = 'TurboPower' AND SHIPDATE < '1/1/94'
```

Kada se pretražuje tekst, `FilterOptions` osobina određuje da li će se prilikom pretraživanja obraćati pažnja na velika i mala slova.

Sledeći operatori se mogu koristiti u izrazima za filtriranje:

Operator	Upotreba
<	Manje od
>	Veće od

=	Jednako
<>	Različito
>=	Veće ili jednako
<=	Manje ili jednako
( )	Koristi se za promenu redosleda izvršavanja naredbi
[ ]	Koristi se za označavanje imena polja koja sadrže prazne znake (space-ove).
AND, OR, NOT	Logički operatori

Filtriranje kroz `Filter` osobinu. Ranije sam rekao da postoje dva načina za filtriranje tabela. Jedan način je korišćenjem `Filter` osobine. Da biste koristili taj način, sve što treba da učinite je da dodelite izraz za filtriranje `Filter` osobini kroz Object Inspector u toku pravljenja programa, ili da dodelite string vrednost ovoj osobini u toku izvršavanja programa. Naravno, morate i da dodelite osobini `Filtered` vrednost `True`.

Da biste videli o čemu pričam, uradite sledeću vežbu. Prvo, postavite osnovne komponente:

1. Postavite `Table` kompnonentu, `DataSource` komponentu i `DBGrid` komponentu na formu.
2. Kliknite na `Table` komponentu i promenite njenu `DatabaseName` osobinu u `DBDEMOS`, njenu `TableName` osobinu u `ORDERS.DB` i njenu `Active` osobinu na `True`.
3. Kliknite na `DataSource` komponentu i postavite njenu `DataSet` osobinu na `Table1`.
4. Kliknite na `DBGrid` komponentu i izmenite njenu `DataSource` osobinu na `DataSource1`. Proizvoljno postavite njenu veličinu.

U ovom trenutku biste trebali da imate mrežu sa podacima. Sada možete da počnete sa filtriranjem baze.

5. Unesite sledeće u Value kolonu pored `Filter` osobine:

```
CustNo = 1384
```

6. Postavite `Filtered` osobinu na `True`.

Sada bi tabela trebala da prikazuje samo narudžbine mušterije broj 1384. Provedite malo vremena kroz eksperimentisanje sa izrazom za filtriranje i obratite pažnju na promene koje se dešavaju svaki put kada promenite izraz za filtriranje. Pokušajte sledeće:

```
CustNo = 1510
CustNo = 1384 and ShipDate < '1/1/94'
```

```
CustNo = 1384 and ShipDate > '1/1/94'
OrderNo > 1100 and OrderNo < 1125
```

Sada vršite izmene filtera u toku pravljenja programa, ali ćete češće praviti izmene dinamički, u toku izvršavanja. U ovom slučaju je to jednostavno kao:

```
Table1.Filter := 'CustNo = 1510';
```

Ukoliko je `Filtered` postavljeno na `True`, ali je `Filter` osobina prazna, biće vraćen ceo dataset, kao da tabela nije ni filtrirana.

Filtriranje pomoću `OnFilterRecord` događaja. Drugi način za filtriranje tabele je pomoću `OnFilterRecord` događaja. Da biste generisali kod za obradu ovog događaja, dva puta kliknite na Value kolonu pored `OnFilterRecord` događaja. Delphi će kreirati kod za obradu događaja. Sada možete napisati kod za filtriranje tabele. Izvedimo prvi primer filtriranja iz postojećeg (`CustNo = 1384`) i filtrirajmo koristeći `OnFilterRecord` događaj umesto `Filter` osobine:

```
procedure TForm1.Table1FilterRecord(DataSet: TDataSet;
  var Accept: Boolean);
var
  Value : Integer;
begin
  Value := Table1.FieldByName('CustNo').Value;
  Accept := (Value = 1384);
end;
```

Kod je podeljen na dve linije da bi bio čitljiviji. Ključni element je parametar `Accept`. `OnFilterRecord` događaj se generiše za svaki zapis u tabeli. Postavite parametar `Accept` na `True` za svaki zapis koji želite da prikažete. Prethodni kod postavlja `Accept` na `True` za svaki zapis kod kojeg je vrednost polja `CustNo 1384`. Ranije sam Vam dao primere izraza za filtriranje. Prva dva filtera bi izgledala ovako, ukoliko biste koristili `OnFilterRecord` događaj umesto `Filter` osobine:

```
Accept := Table1.FieldByName('CustNo').Value = 1510;
Accept := (Table1.FieldByName('CustNo').Value = 1384) and
  (Table1.FieldByName('ShipDate').AsDateTime < StrToDate('1/1/94'));
```

Siguran sam da sada mislite kako je ovo komplikovanije. U pravu ste. Korišćenje `OnFilterRecord` događaja zahteva više rada, ali je moćnije od filtriranja korišćenjem samo `Filter` osobine.

Korišćenje `FilterOptions` osobine. `FilterOptions` osobina određuje na koji način će se primeniti filter. Ova osobina je skup koji može sadržati ili `foCaseInsensitive`, ili `foNoPartialCompare`, ili oba. Podrazumeva se da je vrednost ove osobine prazna. To znači da će se prilikom pretraživanja obraćati pažnja na velika i mala slova kao i da će se vršiti parcijalno pretraživanje. Kada je opcija parcijalnog pretraživanja uključena, postavljanje filtera kao što je `LastName := 'M*'` vraća dataset koji sadrži sve zapise u kojima vrednost `LastName` polja počinje sa *M*.

Pronalaženje zapisa

Možete pretraživati tabelu na nekoliko različitih načina. U stvari, ovaj odeljak se odnosi na sve `TDataSet` naslednike, ne samo na `TTable`.

Kao i filtriranje, i pretraživanje se kod klijent/server baza podataka obavlja preko SQL upita. Pronalaženje zapisa korišćenjem metoda klase `TTable` se uglavnom koristi pri radu sa lokalnim bazama podataka.

Da biste pretraživali filtriran dataset možete koristiti `FindFirst`, `FindNext`, `FindPrior` i `FindLast` metode. Korišćenje ovih metoda je najbolji način za pretraživanje filtriranog dataset-a jer se dataset ponovo filtrira svaki put kada se pozove neka od ovih metoda. To znači da zapisi koji prethodno nisu prošli kroz filter, a u međuvremenu su promenjeni, mogu postati deo dataset-a, pre nego što se izvrši pretraživanje.

Drugi način za pretraživanje tabele je korišćenje `FindKey` i `GotoKey` metoda. Ove metode zahtevaju indeks. `FindKey` metoda pretražuje polje, ili polja sa primarnim indeksom u potrazi za određenom vrednošću. Ukoliko postoji sekundarni indeks, polje sa njim će se koristiti za pretraživanje. Sledeći primer postavlja sekundarni indeks, a zatim traži mušteriju broj 1384:

```
Table1.IndexName := 'CustNo';
if not Table1.FindKey([1384]) then
  MessageBox(Handle, 'Record Not Found', 'Message', MB_OK);
```

Treći način za pretraživanje tabele uključuje korišćenje `Locate` i `Lookup` metoda. Jedna od prednosti ovih metoda je što one ne zahtevaju da tabela bude indeksirana. Ove metode se razlikuju po dva aspekta. Prvi je da `Locate` metoda koristi najbrži mogući način za pretraživanje tabele. Ukoliko je tabela indeksirana, `Locate` će korisititi indeks.

Drugi aspekt po kome se ove metode razlikuju je da će `Lookup` metoda takođe vratiti vrednosti polja koja ste naveli u `ResultFields` parametru pre pozivanja `Lookup` metode. Obe metode Vam dozvoljavaju da navedete polje, ili polja koja ćete pretraživati i vrednost koju tražite. Sledeći primer demonstrira upotrebu `Locate` metode:

```
var
  Options : TLocateOptions;
begin
  Options := [loPartialKey];
  if not Table1.Locate('CustNo', '1384', Options) then
    MessageBox(Handle, 'Record Not Found', 'Message', MB_OK);
end;
```

Ukoliko je zapis pronađen, `Locate` vraća `True` i kurzor se pomera na taj zapis.

Master/Detail tabele

Postavljanje Master/Detail relacije korišćenjem Delphi-jeve `Table` komponente je jednostavno. Dozvolite mi da Vam objasnim pojam Master/Detail relacije i da Vam pokažem kako se ona postavlja. Recimo da imate tabelu koja se zove `CUSTOMER` i koja sadrži informacije o Vašim mušterijama. Tabela će verovatno biti indeksirana po polju `CustNo`.

Pretpostavimo dalje, da imate tabelu koja se zove `ORDERS` i koja sadrži listu svih narudžbina vaših mušterija. Logično, i ova tabela će imati polje `CustNo`. Recimo, sada, da želite da pregledate tabelu koja sadrži sve Vaše mušterije. Zar ne bi bilo lepo kada biste mogli da vidite i sve narudžbine dok pregledate mušterije? Master/Detail relacija Vam omogućava baš to. Izvršite sledeće operacije, kako biste dobro razumeli smisao Master/Detail tabela:

1. Kreirajte nov program. Postavite `Table` komponentu na formu. Postavite njene osobine na ovaj način:

   ```
   Name              Master
   DatabaseName      DBDEMOS
   TableName         customer.db
   ```

2. Postavite `DataSource` komponentu na formu i postavite njenu `DataSet` odobinu na `Master`.

3. Sada postavite drugu `Table` komponentu i promenite njenu `Name` osobinu u `Details`. Videćete i ostale osobine ove komponente za koji minut.

4. Postavite drugu `DataSource` komponentu. Postavite njenu `DataSet` osobinu na `Details`.

5. Kliknite na `Details Table` komponentu. Izmenite njene osobine na sledeći način:

   ```
   DatabaseName      DBDEMOS
   TableName         orders.db
   MasterSource      DataSource1
   ```

6. Kliknite na ellipsis taster pored `MasterField` osobine. Prikazuje se Field Link Designer dijalog-prozor.

7. Na vrhu Field Link Designer dijalog-prozora se nalazi kombo lista koja se zove Available Indexes. Izaberite `CustNo` indeks iz kombo liste.

8. Sada i Detail Fields lista i Master Fields lista imaju `CustNo` stavku. Izaberite `CustNo` iz obe liste i klinknite na Add taster da biste napravili relaciju. Joined Fields lista pokazuje da su dve tabele povezane preko svojih `CustNo` polja.

9. Kliknite OK da biste zatvorili Field Link Designer dijalog-prozor.
10. Postavite dve `DBGrid` komponente na formu i povežite jednu sa `DataSource1`, a drugu sa `DataSource2`.
11. Postavite `Active` osobinu obe tabele na `True`. `Master` tabela će prikazivati sve mušterije dok će `Details` tabela prikazivati narudžbine svake mušterije.

Vi ste sada postavili relaciju između glavne (master) i tabele sa detaljima (detail). Ova relacija je povezala dve tabele kroz zajedničko polje: `CustNo`. Da biste tačno razumeli šta ovo znači, pokrenite program i pomerajte se zapis po zapis kroz glavnu tabelu. Pošto ste izabrali mušteriju u glavnoj tabeli, videćete njegove narudžbine u tabeli sa detaljima.

`Query` komponenta

`Query` komponenta je uobičajeni način za pristup podacima u klijent/server bazama podataka. Sledeći odeljak opisuje primarne osobine i metode `TQuery` klase.

SAVET Query komponenta nema `TableName` osobinu kao što je ima `Table` komponenta. To znači da ne možete videti listu tabele aktivne baze podataka u toku pravljenja programa. Da biste videli listu tabela možete uraditi jednu od dve operacije. Prvo, možete privremeno postaviti Table komponentu na formu, postaviti njenu `DatabaseName` osobinu i zatim videti listu tabela kroz `TableName` osobinu. Takođe možete izabrati Query komponentu, kliknuti na nju desnim tasterom i izabrati Explore iz konteksnog menija. Ovo će Vas odvesti ili u SQL Explorer (klijent/server verzija), ili u BDE Administrator (standardna, ili profesionalna verzija). Možete koristiti bilo koji od ova dva alata da biste videli listu tabela u bazi podataka.

`SQL` osobina

`SQL` osobina je `TStringList` objekat koji sadrži SQL izraze koji treba da se izvrše. Možete postaviti vrednost `SQL` osobine u toku pravljenja programa kroz Object Inspector, ili u toku izvršavanja programa kroz kod.

Da biste postaviti vrednost u toku pravljenja programa, kliknite na ellipsis taster pored `SQL` osobine u Object Inspector-u. Prikazuje se String List Editor dijalog prozor u koji možete uneti jedan, ili više SQL izraza.

SAVET Zapamtite da String List Editor dijalog-prozor ima osobinu koja Vam dozvoljava da menjate listu stringova u Delphi-jevom editoru koda.

Pre nego što počnete da dodajete linije u SQL osobinu u toku izvršavanja programa, obrišite prethodni sadržaj - na primer:

```
Query1.SQL.Clear;
Query1.SQL.Add('select * from country');
```

Radite sa SQL osobinom kao da je string, a ne lista stringova. Ako ne obrišete prethodni sadržaj pre dodavanja stringa, prethodni SQL izrazi će ostati u listi. Gotovo sigurno će doći do greške kada budete pokušali sa izvršite SQL izraze.

Izvršavanje SQL izraza

Izraze u `SQL` osobini možete izvršiti korišćenjem Open, ili `ExecSQL` metoda. Ukoliko koristite SQL izraze koji sadrže `SELECT` naredbu, koristite Open za izvršavanje izraza. Ukoliko koristite `INSERT`, `UPDATE`, ili `DELETE` naredbe, potrebno je da koristite `ExecSQL` metodu za izvršavanje izraza. Sledeći primer postavlja `SQL` osobinu a zatim poziva Open metodu:

```
Query1.SQL.Clear;
Query1.SQL.Add('select * from country');
Query1.Open;
```

`SELECT` naredba SQL-a vraća određena polja iz baze podataka. Zvezdica govori serveru za baze podataka da vrati sve zapise iz tabele. Prethodni primer, dakle, vraća celu tabelu country iz aktivne baze podataka. Da biste vratili samo određena polja, koristite kod sličan sledećem:

```
Query1.SQL.Clear;
Query1.SQL.Add('select Name, Capital from country');
Query1.Open;
```

Postavljanje osobine `Active` na `True` proizvodi isti efekat kao i pozivanje `Open` metode.

`DELETE` naredba SQL-a briše zapise iz dataset-a. Da biste obrisali zapis iz dataset-a, koristite kod sličan sledećem:

```
Query1.SQL.Clear;
Query1.SQL.Add('delete from country where name = 'Royland');
Query1.ExecSQL;
```

Primetite da se koristi `ExecSQL` metoda umesto `Open`. Kao što sam ranije rekao, kada upit koristi `INSERT`, `UPDATE`, ili `DELETE` naredbe, potrebno je koristiti `ExecSQL`.

`INSERT` komanda ubacuje zapis u dataset:

```
Query1.SQL.Add('insert into country');
Query1.SQL.Add('(Name, Capital)');
Query1.SQL.Add('values ("Royland", "Royville")');
Query1.ExecSQL;
```

Primetite dvostruke navodnike u prethodnom kodu. Sintaksa SQL-a ne bi trebala da se meša sa sintaksom ObjectPascal-a. SQL Vam dozvoljava da koristite bilo jednostruke, bilo dvostruke navodnike oko imena vrednosti. Možete koristiti bilo koje, ali ako koristite jednostruke budite sigurni da ste ih duplirali. Ova dva izraza su ispravna:

```
Query1.SQL.Add('values ("Royland", "Royville")');
Query1.SQL.Add('values (''Royland'', ''Royville'')')
```

Osvežavanje dataset-a korišćenjem `UPDATE` naredbe izgleda ovako:

```
Query1.SQL.Clear;
Query1.SQL.Add('update country');
Query1.SQL.Add('set Capital = ''Royburg''');
Query1.SQL.Add('where Name = "Royland"');
Query1.ExecSQL;
```

Iako mi nije namera da Vas učim SQL-u, mislio sam da će par primera biti korisno za početak.

Korišćenje parametara u SQL izrazima

SQL izrazi koriste parametre zbog fleksibilnosti. Parametar u SQL izrazu je sličan ObjectPascal promenljivoj. Parametru u SQL izrazu prethode dve tačke. Uzmite za primer sledeći SQL izraz:

```
select * from country where name = :Param1
```

Parametar u prethodnom izrazu se zove `Param1`. Kada se ovaj izraz izvršava vrednost parametra `Param1` u `Params` osobini zamenjuje ime parametra:

```
Query1.SQL.Add('select * from country where Name = :Param1');
Query1.ParamByName('Param1').AsString := 'Brazil';
Query1.Open;
```

Možete postavljati vrednosti parametara u toku pravljenja programa kroz Parameters dijalog-prozor, ali uglavnom ćete menjati vrednosti parametara u toku izvršavanja programa (što je naravno i smisao korišćenja parametara). Obratite pažnju na prethodni kod u kome se koristi `ParamByName` metoda za postavljanje vrednosti parametra `Param1`. Ovo je verovatno najjednostavniji način za postavljanje vrednosti parametara. Postoji i drugi način:

```
Query1.Params[0].AsString := 'Brazil';
```

Ovde se koristi `Items` osobina `TParam` klase za postavljanje vrednosti parametra. Pristup parametru kroz indeks je podložniji greškama neko pristup kroz ime, zato što morate da pamtite redosled Vaših parametara. U najvećem broju slučajeva ćete koristiti `ParamByName`.

Ne mogu se svi delovi SQL naredbi parametarizovati. Na primer, većina SQL servera ne dozvoljava da se umesto imena tabele koristi parametar. Pogledajte sledeći izraz:

```
select * from :TableName
```

Ovaj izraz proizvodi SQl grešku zato što ne možete koristiti parametar na mestu imena tabele.

StoredProc komponenta

StoredProc komponenta reprezentuje stored proceduru na serveru za baze podataka. Stored procedura je skup SQL izraza koji se izvršavaju kao zaseban program. Stored procedure su zasebni programi koji se izvršavaju nasuprot baze podataka i u kojima se mogu nalaziti često korišćeni izrazi za rad sa bazama podataka. Ovo olakšava rad programerima zato što ne moraju da ponavljaju iste linije koda svaki put kada žele da izvrše istu operaciju. Sve što treba da urade je da pozovu stored proceduru sa servera.

Ovo, takođe, rezultuje u manjim klijent programima zato što se u njima ne mora nalaziti nepotreban kod. Sledeća funkcija stored procedura je održavanje integriteta podataka. Stored procedura može da omogući, ili zabrani izmene baze podataka na osnovu rezultata provere podataka.

Kao i kod SQL upita, neke stored procedure dozvoljavaju upotrebu parametara, dok druge zabranjuju. Sve što treba da uradite kod stored procedura koje nemaju parametre, je da postavite ime procedure i da je izvršite:

```
StoredProc1.StoredProcName := 'DO_IT';
StoredProc1.Prepare;
StoredProc1.ExecProc;
```

Primetite da se Prepare metod poziva prvi, kako bi pripremio stored proceduru za izvršavanje. Zatim se poziva ExecProc metod da bi se stored procedura izvršila.

Stored procedure koje sadrže parametre zahtevaju da se parametri prvo postave, pa da se zatim izvrše same procedure:

```
StoredProc1.StoredProcName := 'ADD_EMP_PROJ';
StoredProc1.ParamByName('EMP_NO').Value := 12;
StoredProc1.ParamByName('PROJ_ID').Value := 'VBASE';
StoredProc1.Prepare;
StoredProc1.ExecProc;
```

Usput, ako imate profesionalnu, ili klijent/server verziju Delphi-ja možete sami probati prethodni kod na sledeći način:

1. Postavite StoredProc komponentu na formu i postavite njenu DatabaseName osobinu na IBLOCAL.
2. Postavite taster na formu i dva puta kliknite na njega kako biste kreirali proceduru za obradu događaja OnClick.
3. Unesite prethodni kod.
4. Postavite Table komponentu na formu, postavite njenu DatabaseName osobinu na IBLOCAL i njenu TableName osobinu na EMPLOYEE_PROJECT. Postavite DBGrid i DataSource komponente na formu i povežite ih sa tabelom. Postavite

`Active` osobinu tabele na `True`. Na ovaj način možete videti izmene koje su izvršene nad tabelom.

5. Dodajte sledeću liniju koda na kraj koda iz koraka 3:

```
Table1.Refresh;
```

Sada pokrenite program. Kada kliknete na taster, dodaje se novi zapis u tabelu sa employee ID brojem 12 i project ID sa vrednošću `VBASE`. Zatvorite program. Sada izmenite kod tako da employee ID postane 10 i ponovo pokrenite program. Sada ćete dobiti poruku o grešci od stored procedure koja Vam govori da je employee ID neispravan. Dobijate ovu poruku o grešci zato što `ADD_EMP_PROJ` stored procedura proverava podatke, a vrednost 10 nije ispravna za ovu bazu podataka.

Možete videti stored proceduru koristeći Explore iz konteksnog menija. Stored procedura pod nazivom `ADD_EMP_PROJ` izgleda ovako:

```
CREATE PROCEDURE ADD_EMP_PROJ (
  EMP_NO SMALLINT,
  PROJ_ID CHAR(5)
)  AS
BEGIN
 BEGIN
 INSERT INTO employee_project (emp_no, proj_id) VALUES
(:emp_no,             :proj_id);
 WHEN SQLCODE -530 DO
  EXCEPTION unknown_emp_id;
 END
 SUSPEND;
END
```

Naravno, ne biste trebali da menjate stored proceduru, dokle god niste potpuno sigurni u to što radite.

`UpdateSQL` komponenta

`UpdateSQL` komponenta omogućava vršenje izmena nad read-only dataset-om, ukoliko se koristi keširanje izmena. Uobičajeno je da se read-only dataset ne može menjati. Kada se koristi keširanje izmena, read-only baza podataka se može menjati i te izmene se mogu upisati u nju.

Mnoge klijent/server baze podataka sadrže podrazumevane operacije koje se izvršavaju kada se izmene upišu u keš za izmene. `UpdateSQL` komponenta Vam dozvoljava da definišete sopstvene SQL izraze koji će se izvršiti kada je potrebno ubaciti, izmeniti, ili obrisati zapis u read-only dataset-u. Na primer, možete postaviti podrazumevane vrednosti za određena polja u dataset-u, koristeći `UpdateSQL` komponentu.

DeleteSQL osobina Vam omogućava da definišete sopstveni SQL upit koji će izvršavati kada se izmene iz keša za izmene upišu u bazu podataka i ukoliko keš za izmene sadrži obrisane zapise. Analogno, InsertSQL dozvoljava definisanje SQL upita koji će se izvršavati kada se zapisi ubacuju u dataset i kada se izmene iz keša za izmene upišu u bazu podataka. ModifySQL osobina se koristi za definisanje SQL upita koji će se pozivati kada se zapisi izmene i kada se izmene iz keša za izmene upišu u bazu podataka.

DataSource komponenta

DataSource komponenta obezbeđuje mehanizam za povezivanje dataset komponenti (Table, Query, ili StoredProc) za vizuelne komponente koje prikazuju podatke (DBGrid, DBEdit, DBListBox i tako dalje). Glavna funkcija DataSource komponente je da učini lakšim proces izmene Vašeg programa. Sve komponente za rad sa podacima, koje se nalaze na formi, su vezane za DataSource, koji je vezan za dataset.

Pošto komponente za rad sa podacima nisu direktno vezane za dataset, možete jednostavno menjati dataset bez potrebe da, pritom, menjate kod. Da biste promenili dataset iz Table u Query, na primer, sve što treba da učinite je da promenite DataSet osobinu komponente DataSource. Nema potrebe da menjate nijednu od osobina komponenti za rad sa podacima.

TDataSource ima nekoliko osobina. Kao što ste već videli, DataSet osobina se koristi za povezivanje sa dataset-om. Enabled osobina određuje da li će komponente, koje su vezane za ovaj DataSource, prikazivati podatke. Kada je Enabled True, podaci će se prikazivati. Kada je Enabled False, komponente ostaju prazne.

Metode TDataSource klase su uglavnom beznačajne, tako da se neću detaljno baviti njima. OnDataChange događaj se generiše kada se aktivni zapis promeni, ili kada se kurzor pomeri na neki drugi zapis. OnStateChange događaj nastaje kada se promeni režim rada dataset-a (kada korisnik prebaci dataset iz režima izmena u režim pregleda, na primer).

Session komponenta

Session komponenta upravlja određenim periodom, ili načinom korišćenja baze podataka. Svaki put kada pokrene program koji radi sa bazama podataka, BDE postavlja globalni TSession objekat koji se zove Session. Možete koristiti Session da biste pristupili trenutnom načinu rada sa bazom podataka. Ne morate da pravite sopstvene TSession objekte dokle god ne pravite višenitni (engl. multithread) program. Pošto to, uglavnom, nećete raditi, globalni TSession objekat je sve što Vam treba.

`TSession` ima nekoliko, posebno interesantih, metoda. `AddAlias` i `AddStandardAlias` metode se koriste za kreiranje BDE alias-a u toku izvršavanja programa. Verovatno ćete kreirati alias-e u toku izvršavanja programa, kada budete isporučivali svoj program. Kreiranje BDE alias-a je detaljnije objašnjeno u odeljku "Kreiranje BDE alias-a".

`GetAliasNames` i `GetDatabaseNames` metodi se koriste za dobijanje liste baza podataka. Ovo je korisno kada želite da dozvolite svojim korisnicima izbor baze podataka iz liste. Na primer, mogli biste da stavite imena baza podataka u kombo-listu:

```
Session.GetDatabaseNames(DBNamesComboBox.Items);
```

U ovom slučaju, `Items` osobina kombo-liste `DBNamesComboBox` se popunjava listom imena baza podataka. `GetTableNames` i `GetStoredProcNames` metode se koriste na isti način.

`Database` komponenta

`Database` komponenta Vam omogućava pristup specifičnim operacijama sa bazama podataka. Za neke programe Vam ona neće biti potrebna. Postoje, ipak, neke operacije koje zahtevaju `Database` komponentu. Ove operacije su obrađene u sledećim odeljcima.

Zadržavanje veza sa bazama podataka

`KeepConnections` osobina se koristi za kontrolu rada sa vezama (engl. connections) sa bazama podataka, kada je dataset zatvoren. Ukoliko je `KeepConnections` `False`, veza sa bazom podataka će biti prekinuta kada se dataset zatvori. Ovo zahteva ponovno prijavljivanje kada se dataset ponovo otvori. Nije toliko bitno to što je ponovno logovanje dosadno (a sigurno da jeste), već što zahteva određeno vreme. Pri tome se ne misli na vreme koje je Vama potrebno da otkucate korisničko ime i lozinku, već na procesorsko i mrežno vreme koje je potrebno da bi se veza uspostavila (čak, iako je ovaj proces automatizovan). Ukoliko ne želite da se prijavljujete svaki put kada Vam je dataset potreban, postavite `KeepConnections` na `True`.

Kontrola prijavljivanja

Jedan od razloga za korišćenje `Database` komponente je kontrola operacija prijavljivanja. Postoje dva načina za kontrolu prijavljivanja. Jedan je postavljanje `LoginPrompt` osobine na `False` i eksplicitno podešavanje parametara prijavljivanja. To možete uraditi pre otvaranja dataset-a:

```
Database1.Params.Values['user name'] := 'SYSDBA';
Database1.Params.Values['password'] := 'masterkey';
```

Prethodni kod postavlja korisničko ime i lozinku za povezivanje sa Local InterBase-om.

NAPOMENA Trebali biste da budete veoma pažljivi prilikom kodiranja informacija o lozinkama u Vašem programu. Naravno, iz sigurnosnih razloga. Prozori za prijavljivanje se koriste sa razlogom. Nemojte ih ignorisati, dokle god ne budete imali jak razlog za to.

Hajde da odemo korak dalje. Pretpostavimo da imate formu sa `Database` i `Table` komponentama. Recimo da želite da kreirate vezu sa bazom podataka i otvorite tabelu bez prozora za prijavljivanje. Sledi kod:

```
Database1.AliasName := 'IBLOCAL';
Database1.DatabaseName := 'MyDatabase';
Database1.Params.Values['user name'] := 'SYSDBA';
Database1.Params.Values['password'] := 'masterkey';
Table1.DatabaseName := Database1.DatabaseName;
Table1.TableName := 'CUSTOMER';
Table1.Open;
```

Ovaj kod prvo postavlja `Alias` osobinu Database komponente na `IBLOCAL` da bi ste se vezali na Local InterBase. Zatim se `DatabaseName` osobina postavlja na proizvoljnu vrednost. Možete koristiti koje god želite ime za bazu podataka. Zatim se postavljaju parametri veze sa bazom podataka (korisničko ime i lozinka). Posle toga se postavlja `DatabaseName` osobina `Table` komponente na vrednost `DatabaseName` osobine `Database` komponente, koja povezuje tabelu sa bazom podataka. Konačno, postavlja se `TableName` osobina za tabelu i ona se otvara.

Drugi način za vezivanje je preko `OnLogin` događaja. Ovaj događaj se generiše uvek kada se zahtevaju parametri veze. Da biste generisali ovaj događaj morate biti sigurni da ste postavili `LoginPrompt` osobinu na `True`. Posle toga, možete kreirati proceduru za kontrolu `OnLogin` događaja. Izgledaće ovako:

```
procedure TForm1.Database1Login(Database: TDatabase;
  LoginParams: TStrings);
begin
  LoginParams.Values['user name'] := 'SYSDBA';
  LoginParams.Values['password'] := 'masterkey';
end;
```

Da li Vam ovaj kod izgleda poznato? To je praktično isti kod kao i onaj koji ste koristili kada ste direktno postavljali parametre vezivanja za bazu podataka. Obično nećete morati da kodirate korisničko ime i lozinku (ili, barem ne lozinku) i verovatno ćete tu informaciju dobijati iz spoljnog izvora, kao što je Edit komponenta, konfiguracioni fajl, ili Windows Registry.

Kontrola transakcija

Sledeći razlog za korišćenje `Database` komponente je kontrola transakcija. Uobičajeno je da sam BDE vrši ovu kontrolu. Međutim, ponekad postoji potreba za kontrolisanjem ovog procesa. U tom slučaju možete koristiti metode za kontrolu transakcija `Database` komponente.

Transakcija je kolekcija izmena dataset-a. Izmene mogu biti: izmene izvršene nad zapisima, brisanje zapisa, ubacivanje zapisa i druge. Transakciju započinjete pozivom `StartTransaction` metode. Sve izmene koje izvršite na dataset-om se zadržavaju dokle god ne pozovete `Commit` metodu. Kada pozovete `Commit`, sve izmene iz transakcije se upisuju u bazu podataka. Ukoliko želite da ignorišete ove izmene, koristite `Rollback` metodu. Nivo izolacije transakcije se kontroliše putem vrednosti postavljene u `TransIsolation` osobinu. (Za više informacija o nivoima izolacije transakcija pogledajte `TransIsolation` temu u Delphi-jevom Help-u.)

Sve izmene u transakciji se tretiraju zajednički. To znači da se sve izmene upisuju kada pozovete `Commit`. Kada pozovete `Rollback`, sve izmene se ignorišu. To takođe znači, da ukoliko dođe do greške prilikom upisivanja izmena, nijedna od izmena u transakciji neće biti upisana u bazu podataka.

`BatchMove` komponenta

`BatchMove` komponenta se koristi za kopiranje zapisa iz jednog dataset-a u drugi. `Source` osobina određuje izvorni dataset, dok `Destination` osobina određuje odredišni dataset.

Postavljanje `Mapping` osobine je neophodno ukoliko izvorni i odredišni dataset nemaju identična polja. `Mapping` je `TStringList` osobina. Da biste odredili mapiranje, izmenite listu stringova i dodajte sledeća mapiranja:

```
FirstName = FName
LastName = LName
Notes = Comments
```

Stringovi za mapiranje koriste znak jednakosti, a ne Pascal-ov operator dodele (`:=`).

Polje na levoj strani znaka jednakosti pripada odredišnom dataset-u, dok polje na desnoj strani pripada izvornom dataset-u. Postavljanje mapinga na gornji način govori klasi `TBatchMove` sledeće: "Ova dva dataset-a nisu ista, pa kopiraj podatke iz `FName` kolone izvornog dataset-a u `FirstName` kolonu odredišnog dataset-a". Ako Vaši izvorni i odredišni dataset-ovi nisu identični i ako ne uspete da postavite mapiranje, `BatchMove` neće raditi kako treba.

`Execute` metod je taj koji pokreće `BatchMove` proces. Da biste koristili `TBatchMove`, potrebno je da postavite `Source`, `Destination` i `Mode` osobine i da pozovete `Execute` metod. Možete postavljati `Source` i `Destination` osobine u toku pravljenja, ili u toku izvršavanja programa. Sledeći kod kreira kopiju tabele:

```
DestTable.TableName := 'copy.db';
BatchMove1.Destination := DestTable;
BatchMove1.Source := SourceTable;
BatchMove1.Mode := batCopy;
BatchMove1.Execute;
```

Mode osobina određuje koliko zapisa će biti kopirano u odredišni dataset. Tabela 16.6 prikazuje sve moguće vrednost `Mode` osobine i njihova značenja.

Tabela 16.6: Vrednosti osobine `Mode`

Vrednost	Opis
`batAppend`	Dodaje zapise iz izvornog na kraj odredišnog dataset-a.
`batAppendUpdate`	Kombinacija vrednosti `batAppend` i `batUpdate`. Ako isti zapis već postoji u odredišnom dataset-u, menja se. Ukoliko isti zapis ne postoji, dodaje se novi.
`batCopy`	Kreira novu tabelu i kopira sve zapise iz izvorne u tu, novu, tabelu.
`batDelete`	Briše zapise iz odredišnog dataset-a, koji već postoje u izvornom dataset-u. Odredišni dataset mora biti indeksiran.
`batUpdate`	Zamenjuje zapise u odredišnom dataset-u sa zapisima iz izvornog dataset-a, koji imaju iste vrednosti ključa.

Budi pažljivi pri korišćenju `batCopy`. Pozivanje `Execute` metoda izaziva brisanje svih postojećih tabela i zamenu njihovih sadržaja sa sadržajem izvorne tabele.

`Field` komponenta

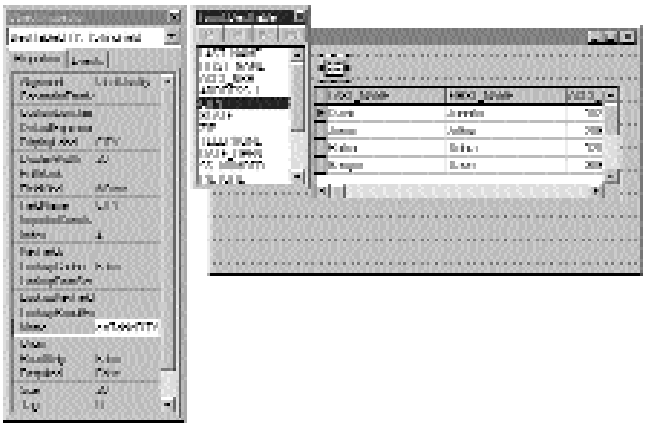
`TField` klasa predstavlja polje u bazi podataka. Kroz `TField` možete postaviti atribute polja. Ovi atributi su: tip podatka (string, ceo broj, realan broj i tako dalje), veličinu polja, indeks, osobinu da je polje proračunato, ili ne, osobinu da je polje neophodno, ili ne i tako dalje. Takođe možete pristupiti podacima u polju kroz osobine kao što su: `AsString`, `AsVariant`, ili `AsInteger`.

`TField` je osnovna klasa za druge, specifičnije, klase. Naslednici `TField` su: `TStringField`, `TIntegerField`, `TSmallIntField`, `TWordField`, `TFloatField`, `TCurrencyField`, `TBCDField`, `TBooleanField`, `TDateTimeField`, `TDateField`, `TTimeField`, `TBlobField`, `TBytesField`, `TVarBytesField`, `TMemoField` i `TGraphicField`.

Ove, izvedene klase, malo proširuju osnovnu klasu, kako bi dodale određenu funkcionalnost. Na primer, numeričke klase polja imaju `DisplayFormat` osobinu koja određuje kako će se broj prikazivati i `EditFormat` osobinu koja određuje kako će se broj prikazivati prilikom izmene. Svaki naslednik `TField` klase odgovara specifičnom tipu polja u bazi podataka. `TIntegerField` se koristi pri radu sa poljima koja su tipa integer, `TDateTimeField` se koristi pri radu sa poljima koja su tipa date, ili time (ili date/time), `TBlobField` se koristi pri radu sa velikim binarnim objektima (engl. `Binary Large Objects`) i tako dalje.

Možete pristupati osobinama klase `TField` za vreme pravljenja programa kroz Fields Editor. Pošto dodate polja, možete kliknuti na polje u Fields Editor-u i njegove

osobine će se prikazati u Object Inspector-u. Slika 16.5 prikazuje Fields Editor i Object Inspector u toku izmene polja.



Slika 16.5
Fields Editor i Object Inspector

Osobina i metoda `TField` klase ima puno, tako da ih neću navoditi ovde. Umesto toga, provešću Vas kroz primere najčešće upotrebe `TField` klase i klasa izvedenih iz nje.

Pristupanje poljima

Pre nego što budete mogli da pročitate, ili postavite vrednost podatka u polju, morate, na neki način, locirati polje. Postoje barem tri načina da se ovo uradi:

- Kroz pokazivač na ime polja
- Kroz `Fields` osobinu `TDataSet-a`
- Kroz `FieldByName` metodu `TDataSet-a`

Pristup polju kroz pokazivač na njegovo ime je verovatno najslabije korišćen način pristupa. Može se koristiti samo ukoliko ste dodali polja Vašem projektu kroz Fields Editor. Kada dodate polja kroz Fields Editor, Delphi kreira pokazivač na svako polje, kombinujući ime tabele sa imenom polja. Ako imate tabelu `Table1` i string polje `FirstName`, Delphi će kreirati `TStringField` pokazivač `Table1FirstName`. Vi možete koristiti ovaj pokazivač da biste pristupili polju:

```
Table1FirstName.Value := 'Per';
```

Problem kod ovakvog pristupa je što Vam neće uvek biti potrebno da dodajete polja kroz Fields Editor.

`Fields` osobina omogućava drugačiji način pristupa polju - na osnovu pozicije. Ukoliko znate da je prvo polje u tabeli `LastName` možete uraditi ovo:

```
Edit1.Text := Table1.Fields[0].Value;
```

Problem kod ovakvog pristupa je što morate znati tačan raspored polja u tabeli.

Od tri pomenuta načina za pristupanje poljima, najčešće korišćen i najsigurniji način je putem `FieldByName` metode. Korišćenjem `FieldByName`, morate znati samo ime polja da biste mogli da mu pristupite:

```
Table1.FieldByName('LastName').AsString := Edit1.Text;
```

`FieldByName` vraća pokazivač na `TField`. Da bi Vam bilo jasnije, prelomiću prethodnu liniju koda:

```
var
  Field : TField;
begin
  Field := Table1.FieldByName('LastName');
  Field.AsString := Edit1.Text;
end;
```

U najvećem broju slučajeva `FieldByName` je najbolji izbor. Možda ćete se zapitati, koji zapis je promenjen kada je izvršen prethodni kod. Sve navedene metode pristupa poljima rade sa trenutnim zapisom.

Vraćanje i postavljanje vrednosti polja

Pošto ste obezbedili pokazivač na određeno polje, možete mu promeniti vrednost korišćenjem Value osobine, ili korišćenjem neke od `As` osobina (pod `As` se podrazumevaju `AsString`, `AsInteger`, `AsDateTime`, `AsBoolean` i tako dalje). Ove osobine vrše konverzije iz jednog tipa podatka u drugi. Naravno, ne možete uvek biti sigurni da se konverzija može izvršiti. Na primer, ukoliko pokušate da konvertujete string polje sa sadržajem `Smith` u celobrojnu vrednost, generisaće se izuzetak.

Postavljanje vrednosti polja je jednostavno ukoliko poznajete `FieldByName`:

```
Table1.Edit;
Table1.FieldByName('LastName').AsString := Edit1.Text;
Table1.Post;
```

Prvo se poziva `Edit` metoda kojom se tabela postavlja u režim izmena. Ukoliko poziv `Edit` metode ne uspe, generisaće se izuzetak kada pokušate da izmenite vrednost nekog polja. Pošto je tabela postavljena u režim izmena, postavlja se vrednost polja. U ovom slučaju sam koristio `AsString` osobinu umesto `Value`. Kod string polja to ne pravi razliku. Konačno, poziva se `Post` metoda da bi se izmene upisale u bazu podataka (ili u keš za izmene ukoliko je osobina `CachedUpdates` postavljena na `True`). To je sve. Vraćanje vrednosti polja je takođe jednostavno:

```
var
 AcctNo : Integer;
begin
  AcctNo := Table1.FieldByName('ACCT_NBR').Value;
```

```
  { More code here. }
end;
```

Događaji klase `TField`

Bitni događaji klase `TField` su `OnChange` i `OnValidate`. `OnChange` događaj se generiše svaki put kada se promeni vrednost polja. Ovo se dešava kada se izmene upišu u bazu podataka. Možete koristiti ovaj događaj ukoliko želite da znate kada je došlo do promene vrednosti polja.

`OnValidate` događaj se generiše neposredno pre upisivanja izmena u bazu podataka. Ukoliko imate komponentu za rad sa podacima na formi koja je povezana sa poljem, onda će ta komponenta vršiti proveru podataka. Ukoliko, ipak, postavljate vre-dnosti polja kroz kod, želećete da samostalno vršite proveru podataka kroz proceduru za upravljanje `OnValidate` događajem. Ovaj događaj je malo čudan premda Vam ne prenosi nikakav parametar pomoću kojeg biste mogli da prihvatite, ili odbijete izmenu sadržaja polja. Umesto toga, potrebno je da generišete izuzetak ukoliko podaci nisu korektni:

```
procedure TForm1.Table1ACCT_NBRValidate(Sender: TField);
begin
  if Sender.AsInteger < 3000 then
    raise EDBEditError.Create('Bad Account Number.');
end;
```

Kada generišete izuzetak, postupak upisivanja podataka u bazu podataka se prekida.

Da biste kreirali proceduru za obradu izuzetka u toku pravljenja programa morate koristiti Fields Editor da biste dodali polja u dataset. Kada to uradite, možete izabrati polje u Fields Editor-u i dva puta klinknuti na ime događaja u Object Inspector-u. Dakle, isto kao i za bilo koji drugi događaj.

Komponente za klijent/server baze podataka

Klijent/server verzija Delphi-ja se isporučuje sa tri dodatne komponente za pristup podacima koje omogućavaju kreiranje multitier sistema za rad sa bazama podataka. (Da se podsetimo, multiter sistem za rad sa bazama podataka je onaj kod koga klijent program komunicira sa serverom za programe, a ovaj komunicira sa serverom za baze podataka.) Multitier komponente su `TRemoteServer`, `TProvider` i `TClientDataSet`.

`TRemoteServer` komponenta se koristi u klijent programu da bi uspostavila vezu sa jednim, ili više servera za programe. `TProvider` komponentu koristi server za programe i ona predstavlja vezu između servera za baze podataka i klijent programa. `TClientDataSet` komponenta se koristi u klijent programima da bi bio moguć pristup provajderu na serveru za programe. Detaljni opis mogućnosti ovih komponenti izlazi iz okvira ove knjige.

Kreiranje BDE alias-a

Možete se baviti programiranjem baza podataka na tom nivou da nikada ne kreirate ni jedan BDE alias. Jednostavne baze podataka su dobre, ali pre, ili kasnije ćete morati da kreirate alias za svoju bazu podataka. Kada isporučite program napisan u Delphi-ju, moraćete da kreirate nekoliko alias-a i na računaru Vašeg korisnika. Postoji više načina za kreiranje alias-a:

- Kroz BDE Administrator pomoćni program iz Delphi programske grupe
- Kroz program Database Desktop
- Kroz SQL Explorer (samo u klijent/server verziji)
- Kroz kod, u toku izvršavanja programa

Da biste kreirali alias, morate uraditi jednu od dve stvari. Ili ćete naterati korisnike da koriste BDE Administrator, ili ćete kreirati alias-e kroz kod. Očigledno, kreiranje alias-a kroz kod je bolje rešenje. (ne oslanjajte se na korisnike, čak ni kod najjednostavnijih operacija.) Prvo ću Vam pokazati kako da koristite BDE Administrator prilikom kreiranja alias-a. Zatim ću Vam pokazati kako se kreiraju alias-i kroz kod.

Kreiranje alias-a uz pomoć BDE Administrator-a

Dok pravite svoj program, morate kreirati jedan, ili više BDE alias-a. To se najlakše radi korišćenjem uslužnih programa koji se isporučuju uz Delphi. Neophodni koraci za kreiranje alias-a korišćenjem BDE Administrator-a, ili SQL Explorer-a su identični. Iz tog razloga ću pokazati samo kako se kreira alias uz pomoć BDE Administrator-a.

Za početak, pretpostavimo da želite da napravite program za mailing liste. Prvi stvar koju treba da uradite je kreiranje alias-a za Vašu bazu podataka. Možete kreirati alias na nekoliko načina, ali verovatno je najlakše pomoću BDE Administratora. Izvršite sledeće operacije:

1. Pokrenite BDE Administrator (izaberite Delphi-jevu grupu iz Windows-ovog Start menija, a zatim i BDE Administrator ikonu). BDE Administrator se prikazuje na ekranu zajedno sa listom alias-a koji su instalirani na računaru.
2. Izaberite Object/New iz menija BDE Administrator-a (budite sigurni da je izabran Databases jezičak). Prikazuje se New Database Alias dijalog-prozor i pita Vas koji će se drajver koristiti za novi alias.
3. Kreiraćete bazu podataka koristeći Standard drajver i, premda je STANDARD već izabran, jednostavno kliknite na OK taster. Sada BDE Administrator izgleda kao na slici 16.6.

Slika 16.6
BDE Administrator kreira novi alias za bazu podataka

4. BDE Administrator čeka da unesete ime za novi alias. Unesite `MyDatabase` i pritisnite Enter.

U ovom trenutku je potrebno da unesete nekoliko informacija u Definition prozor. Tip je već postavljen na `STANDARD`, tako da nemate više šta da unosite. `DEFAULT DRIVER` polje je postavljeno na `PARADOX`, a to je baš ono što Vam treba, tako da ni tu nisu potrebne nikakve intervencije (ostale mogućnosti su: `dBASE`, `FOXPRO` i `ASCIIDRV`). Takođe, ne morate menjati vrednost polja `ENABLE BCD`. Jedina informacije koju morate uneti je putanja na disku gde će se nalaziti Vaši fajlovi sa bazom podataka:

1. Kliknite na `PATH` polje, pa, ili unesite putanju, ili je izaberite iz dijalog-prozora koji se aktivira pritiskom na ellipses taster.
2. Zatvorite BDE Administrator i kliknite na taster Yes, kada Vas bude pitao da li želite da sačuvate izmene. To je to. Kreirali ste BDE alias.

Vratite se nazad u Delphi i postavite `Table` komponentu na formu. Proverite da li se ime Vašeg alias-a pojavljuje u `DatabaseName` osobini u Object Inspector-u. Ukoliko ste sve uradili kako treba, videćete ga zajedno sa ostalim alias-ima. Vaša baza podataka još uvek nema ni jednu tabelu, ali to je u redu. O tome ćete se pobrinuti kasnije.

Kreiranje alias-a kroz kod

Da biste izbegli nepotrebnu konfuziju kod korisnika verovatno ćete želeti da kreirate sve potrebne alias-e kada se program startuje po prvi put. Srećom, kreiranje alias-a kroz kod je jednostavno. Ovo je kod pomoću kojeg se kreira lokalni Paradox alias sa imenom WayCool:

```
CreateDirectory('C:', nil);
Session.AddStandardAlias('WayCool', 'C:', '');
```

To je to? Da, to je to. Naravno, potrebno je da proverite da li su alias-i i direktorijumi postavljeni kako treba, ali u principu, to je to.

Ovaj primer koristi `AddStandardAlias` metodu da bi kreirao `STANDARD` tip alias-a. Da biste kreirali alias-e za druge servere za baze podataka, koristite `AddAlias` metodu.

Zaključak

Puno je informacija koje je potrebno zapamtiti. Najbolji način da utvrdite svoje znanje je da provedete puno vremena eksperimentišući. Uzmite neke jednostavne baze podataka i probajte da postavite filtere na tabele, izvršite neke SQL naredbe i pregleda baze podataka kroz BDE Administrator, ili SQL Explorer. U ovom trenutku ne biste trebali da razmišljate o pisanju ozbiljnih programa za rad sa bazama podataka. Dovoljno je da provedete neko vreme koristeći razne komponente, kako biste osetili na koji način BDE i VCL komponente sarađuju.

Radionica

Radionica sadrži test pitanja koja Vam pomažu da učvrstite svoje razumevanje izložene materije i vežbe koje Vam pomažu da steknete iskustvo u onome što ste naučili. Možete pronaći odgovore na test pitanja u Dodatku A "Odgovori na test pitanja".

Pitanja i odgovori

P **Kada isporučujem program za rad sa bazama podataka, pisan u Delphi-ju, mogu li jednostavno da iskopiram potrebne BDE fajlove na računar mog korisnika?**

O Ne, morate ispoštovati Borland-ove zahteve koji se nalaze u fajlu `DEPLOY.TXT`. Uopšteno, zahteva se korišćenje instalacionog programa koji je odobrio Borland. Ovo je neophodno prilikom instaliranja bilo kog programa koji koristi BDE.

P **Zašto je potrebno koristiti `DataSource` komponentu? Zašto kontrole za rad sa podacima i kontrole za pristup ne mogu da komuniciraju direktno?**

O Korišćenje `DataSource` komponente kao veze Vam olakšava posao ukoliko, kasnije, treba da promenite dataset-ove. Umesto menjanja `DataSet` osobine velikog broja komponenti, treba samo da promenite `DataSet` osobinu `DataSource` komponente. Recimo da treba da promenite Vaš dataset iz `TTable` u `TQuery` (najveća moguća promena). Izmena će se odmah reflektovati na sve Vaše komponente, pošto `DataSource` odrađuje najveći deo posla.

P **Kada koristiti `TTable`, a kada `TQuery`?**

O U najvećem broju slučajeve ćete, kada budete radili sa lokalnim bazama podataka (Paradox, ili dBASE), koristiti `TTable`, a TQuery kada budete radili sa klijent/server bazama podataka. Na kraju krajeva, sami treba da odlučite koju komponentu ćete koristiti i kada.

P **Čemu služi Local InterBase?**

O Local InterBase Vam omogućava da pravite program za rad sa lokalnim bazama podataka koji ćete, kasnije, lako konvertovati u program za rad sa klijent/server bazama podataka.

P **Da li moram da kreiram `TSession` za moj program?**

O U najvećem broju slučajeva, ne. Podrazumevani `TSession` se automatski kreira za svaki program za rad sa bazama podataka. Možete koristiti ovaj objekat, koji se zove `Session`, kada je god potrebno da pristupute osobinama i metodama klase `TSession`. Jedini slučaj kada biste trebali da kreirate poseban `TSession` objekat je kada pravite višenitni program za rad sa bazama podataka.

P **Koristim lokalnu bazu podataka u mom programu. Da li moram da radim sa keširanjem izmena?**

O Uopšteno govoreći, ne. Keširanje izmena je mnogo važnije pri radu sa klijent/server bazama podataka.

Kviz

1. Šta je lokalna baza podataka?
2. Koja je uloga BDE-a?
3. Da li su dataset i tabela jedno te isto? Ako nisu, objasnite razlike.
4. Navedite jednu prednost korišćenja keširanja izmena.
5. Šta je stored procedura?
6. Koja je uloga `SQL` osobine `TQuery` komponente?
7. Navedite jedan razlog zbog kojeg biste želeli da koristite sopstveni `TDatabase` objekat, umesto podrazumevanog.
8. Zašto biste želeli da održavate vezu između udaljene baze podataka i Vašeg programa, iako je trenutno ne koristite?
9. Šta radi `TBatchMove` komponenta?
10. Šta je BDE alias?

Vežbe

1. Opišite na koji način zajednički funkcionišu: Vaš program, BDE i baza podataka.
2. Postavite `DataSource`, `Table` i `DBGrid` komponente na formu. Povežite komponente. Izaberite bazu podataka i tabelu za `Table` komponentu. Postavite `Active` osobinu tabele na `True`. Pregledajte podatke koji se nalaze u mreži.

3. Izmenite `TableName` osobinu tabele nekoliko puta da biste pregledali različite tabele. (Savet: postavite `Active` osobinu na `False`, pre promene `TableName` osobine.)
4. Postavite drugu `Table` komponentu na formu kreiranu u vežbi 2. Izaberite bazu podataka i tabelu. Postavite `Active` osobinu na `True`. Sada promenite `DataSet` osobinu `DataSource` komponente sa `Table1` na `Table2` (druga tabela). Šta se dešava sa `DBGrid` komponentom?
5. Kreirajte novi BDE alias na svom računaru.
6. **Posebna vežba**: Kreirajte tabelu pod BDE alias-om, koga ste kreirali u vežbi 5 i popunite je sa podacima.

Dan 17

Pravljenje formi za rad sa bazama podataka

Posle prilično neuzbudljivog pregleda Delphi-jeve arhitekture baza podataka možete početi sa interesantnijim zadacima, kao što je generisanje programa za rad sa bazama podataka. Prvo što treba da naučite je kako da pravite forme za rad sa bazama podataka, tako da će to biti današnja tema.

Naučićete kako da pravite forme za rad sa bazama podataka, koristeći Delphi-jev Database Form Wizard. Takođe ćete naučiti kako da samostalno pravite forme za rad sa bazama podataka. Pred kraj današnjeg dana naučićete i nešto o korišćenju Delphi-jevih komponenti za rad sa podacima. To su komponente koje prikazuju podatke iz baze podataka i koje Vam omogućavaju da izmenite te podatke. Možete ih pronaći na Data Controls jezičku palete sa komponentama. Ove komponente se *često nazivaju* komponentama koje zavise od podataka. Ja ću ih u ovom poglavlju jednostavno zvati: komponente za rad sa podacima. Počnimo.

Database Form Wizard

Delphi-jev Database Form Wizard omogućava brzo i jednostavno pravljenje formi za rad sa bazama podataka. Koristeći ovaj Wizard, možete praviti formu za rad sa bazama podataka od početka do kraja. Ne morate postavljati nikakve komponente za rad sa bazama podataka na formu. Jednostavno pokrenite Wizard i prepustite mu ostali deo posla.

NAPOMENA Ni jedan automatizovan proces nije dovoljno dobar da bi mogao da zadovolji sve i svačije potrebe. Neću reći da su forme koje pravi Database Form Wizard sve što će Vam ikada trebati, ili da će Database Form Wizard odraditi ceo posao za Vas. Sve što Database Form Wizard može da uradi je započinjanje procesa pravljenja forme za rad sa bazama podataka. Pošto je početni deo završen, možete nastaviti rad na formi, prilagođavajući je Vašim potrebama.

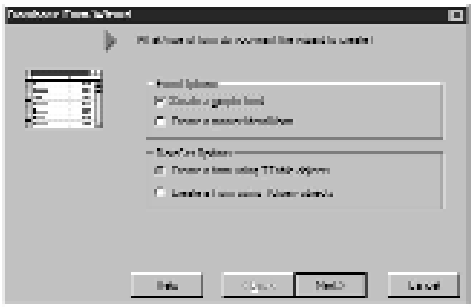
Database Form Wizard omogućava kreiranje kako jednostavnih, tako i master/detail formi. Omogućava izbor između dva moguća dataset-a (`TTable`, ili `TQuery`). Dozvoljava Vam da izaberete bazu podataka i tabelu kao i da izaberete polja koja želite da prikažete na formi. Daje Vam i mogućnost izbora izgleda forme. Pošto opremite Database Form Wizard sa neophodnim podacima, on kreira novu formu.

Da biste pokrenuli Database Form Wizard, izaberite Database➥Form Wizard iz Delphi-jevog glavnog menija. Database Form Wizard možete pokrenuti i sa Business stranice Object Repository-a.

Prvo ću Vam pokazati kako da napravite jednostavnu formu, a zatim ću govoriti o master/detail formama.

Pravljenje jednostavne forme korišćenjem Database Form Wizard-a

Kada se Database Form Wizard pokrene, na ekranu se dobija dijalog-prozor prikazan na slici 7.1.



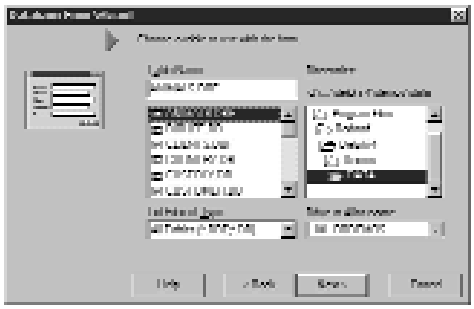
Slika 7.1
Prva strana Database Form Wizard-a

Ova strana Database Form Wizard-a od Vas traži da izaberete tip baze podataka koju želite da kreirate i tip dataset-a koji želite da koristite. Form Sections odeljak Vam pruža izbor između jednostavne (sa jednim dataset-om), ili master/detail forme.

Juče sam rekao nešto o master/detail tabelama, u odeljku "Master/Detail tabele". Govoriću više o njima kasnije, u odeljku "Pravljenje master/detail forme". Prva strana Database Form Wizard-a Vam dozvoljava da izberete da li želite da radite sa `TTable`, ili sa `TQuery` tipom dataset-a. Za ovu vežbu je sasvim dovoljan podrazumevani izbor koji omogućava rad sa jednostavnom formom i `TTable` dataset-om tako da možete da kliknete na taster Next da biste se pomerili na sledeću stranu.

SAVET U nekim slučajevima ćete možda želeti da koristite podatake iz više tabela, ali da pritom ne koristite master/detail relaciju. Database Form Wizard Vam dozvoljava izbor samo jedne tabele. Ono što možete da uradite jeste da pokrenete Database Form Wizard i izaberete prvu tabelu. Forma će biti napravljena. Izmenite ime `Table` i `DataSource` komponenti u nešto smišljeno. Sada ponovo pokrenite Database Form Wizard i izaberite drugu tabelu. Kada se forma prikaže izaberite sve komponente za rad sa bazama podataka sa forme i iskopiraje ih u clipboard. Vratite se nazad na prvu formu, napravite malo slobodnog mesta na njoj i vratite komponente iz clipboard-a na formu. Sada uklonite drugu formu iz projekta i to je to.

Sledeća stranica od Vas traži da izaberete tabelu iz koje ćete koristiti podatke. Drive i Alias name kombo-liste Vam omogućavaju izbor baze podataka na isti način kao što biste ručno podesili `DatabaseName` osobinu dataset-a u toku pravljenja programa. Takođe Vam dozvoljava izbor direktorijuma iz kojeg ćete birati tabele. Za ovu vežbu, izaberite `DBDEMOS` alias. U Table Name listi će se pojaviti spisak svih tabela u tom alias-u. Izaberite `ANIMALS.DBF` tabelu. Database Form Wizard sada izgleda kao na slici 17.2. Kliknite na taster Next da biste prešli na sledeću stranu.



Slika 17.2
Izbor tabele iz Database Form Wizard-a

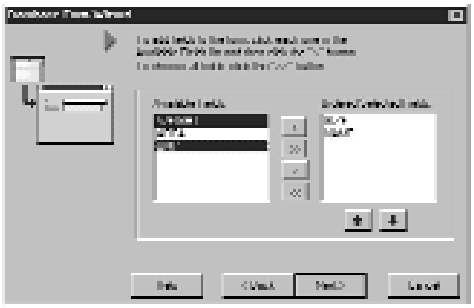
Treća strana Database Form Wizard-a Vam dozvoljava da izaberete polja iz tabele koja želite da se pojave na formi. Available Fields lista sa leve strane prikazuje polja koja se nalaze u tabeli koju ste izabrali. Ordered Selected Fields lista sa desne strane sadrži polja koja želite da se nađu na Vašoj novoj formi. Između ove dve liste se nalaze četiri tastera koji služe za dodavanje i brisanje polja iz Ordered Selected Fields liste.

Da biste dodali novo polje, kliknite na ime polja u Available Fields listi i kliknite na > taster. Na ovaj način možete dodati sva polja koja želite. Takođe možete izabrati i više polja i dodati ih korišćenjem tastera >. Da biste dodali sva polja od jednom, kliknite na >> taster.

SAVET Možete dva puta kliknuti na ime polja da biste ga dodali u Ordered Selected Fields listu, ili ponovo klinknuti dva puta na njega da biste ga izbacili iz ove liste.

Pošto se sva potrebna polja nalaze u Ordered Selected Fields listi, možete izmeniti njihov redosled drag and drop tehnikom, ili kliktanjem na tastere sa strelicama na gore i na dole, koji se nalaze ispod liste. Slika 17.3. prikazuje ovu stranicu Wizard-a.

Za sada, dodajte sva polja u Ordered Selected Fields listu i kliknite na taster Next za prelazak na sledeću stranu.

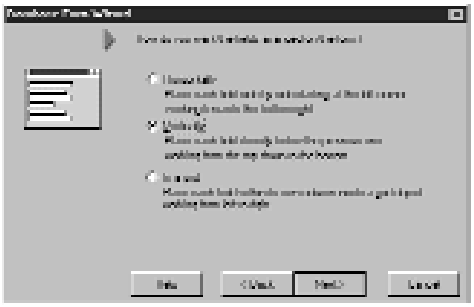


Slika 17.3
Treća strana Database Form Wizard-a - dodavanje polja

Sledeća strana Database Form Wizard-a od Vas traži da navedete na koji način želite da se razmeste komponente za rad sa podacima i to za svako izabrano polje. Imate tri mogućnosti na raspolaganju:

- Horizontalno (Horizontally)
- Vertikalno (Vertically)
- Raspoređivanje u mrežu (In a grid)

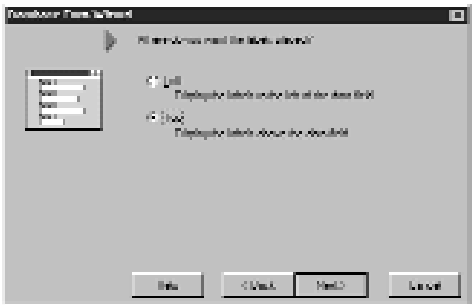
Kliknite na svaki od tri izbora i gledajte sliku koja se nalazi sa leve strane. Slika se menja svaki put kada izaberete drugu opciju, kako biste videli izgled izabranog načina raspoređivanja komponenti. Izaberite Vertically i kliknite na Next taster za nastavak. Slika 17.4 prikazuje ovu stranu Wizard-a.



Slika 17.4
Izbor rasporeda komponenti u Database Form Wizard-u

NAPOMENA Kada Delphi pravi novu formu, bira komponente koje najviše odgovaraju tipu podatka polja koje će komponenta predstavljati. Na primer, za teksualno polje će biti izabrana `DBEdit` komponenta, za memo polje `DBMemo` i za BLOB (engl. Binary Large Object (veliki binarni objekat)) polje `DBImage` komponenta. Možda ćete želeti da promenite formu, kako biste dobili tačno one komponente koje želite za određena polja.

Sledeća strana Vas pita gde želite da postavite labele za polja u odnosu na svaku komponentu. Možete postaviti labele, ili sa leve strane, ili odgore. Primetite da se i ovde menja izgled dijalog-prozora, kako birate drugačiju opciju. Za sada izaberite drugu opciju da biste postavili labele iznad komponenti. Slika 17.5. prikazuje ovu stranu Database Form Wizard-a.



Slika 17.5
Ova strana Vam omogućava da postavite labele

NAPOMENA Strana prikazana na slici 17.5 se prikazuje samo ukoliko ste izbrali opciju za vertikalno raspoređivanje komponenti (Vertically) na prethodnoj strani Database Form Wizard-a.

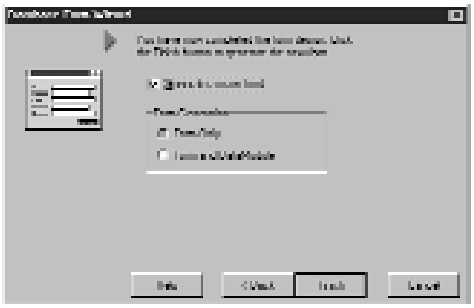
Poslednja strana Database Form Wizard-a od Vas traži da načinite dva izbora:

4 možete izabrati postavljanje novokreirane forme za rad sa bazama podataka na mesto glavne forme programa. Ukoliko je to ono što želite, postavite izbor na Generate a main form. Ukoliko ne želite, ukinite izbor.

NAPOMENA Ukoliko izaberete postavljanje novokreirane forme na mesto glavne forme programa, obratite pažnju na činjenicu da postojeća glavna forma i dalje ostaje u projektu. Moraćete da je izbacite iz projekta, ukoliko ne želite više da radite sa njom. Da biste izbacili staru glavnu formu, kliknite na Remove from Project taster i izbacite jedinicu glavne forme (podarazumeva se da je njeno ime Unit1.pas).

4 možete izabrati samo kreiranje forme, ili kreiranje i forme i modula za podatke. Ukoliko izaberete prvu mogućnost (Form Only), forma će sadržati sve komponente za rad sa podacima, `DataSet` komponentu (`Table`, ili `Query`) i `DataSource` komponentu. Ukoliko izaberete drugu mogućnost (Form and DataModule), `DataSet` i `DataSource` komponente će biti sklonjene sa forme i postavljene u poseban modul za podatke.

Govoriću više o modulima za podatke sutra, u odeljku "Rad sa modulima za podatke". Za sada, izaberite Generate a main form i postavite Form Generation opciju na Form Only. Slika 17.6 prikazuje poslednju stranu Database Form Wizard-a.



Slika 17.6
Poslednja strana Database Form Wizard-a

Kliknite na Finish taster i forma će biti napravljena. Završena forma izgleda kao i forma na slici 17.7.

Slika 17.7
Završena forma

Primetite da forma na slici 17.7 sadrži komponentu za podatke za svako polje koje je izabrano i labelu za svaku komponentu. Labele su prikazane velikim slovima zbog toga što su imena polja na isti taj način upisana u tabelu. Primetite da se na vrhu forme nalazi `DCNavigator` komponenta koja Vam omogućava kretanje kroz zapise tabele.

Nova forma na delbnm

Sada ste spremni da pritisnete Run taster i da vidite na koji način forma radi. Kada se program pokrene, prikazuje se prvi zapis u dataset-u. Koristite tastere na `DBNavigator` kontroli za kretanje kroz dataset. Zapamtite da u ovom trenutku radite sa stvarnim podacima. Možete menjati podatke prostom izmenom teksta u nekoj od komponenti. Da biste sačuvali izmene u bazu podataka, kliknite na Post taster na `DBNavigator-u` (prikazan je simbolom za tačno). Da biste odbacili sve izmene nad ovim zapisom, kliknite na Cancel taster na `DBNavigator-u` (X taster). Izmene će biti sačuvane kada pritisnete Post taster, ili kada se pomerite na neki drugi zapis.

Zatvorite program i vratite se u Delphi-jev IDE. Provedite neko vreme u istraživanju komponenti koje se nalaze na formi i njihovih osobina. Kasnije će biti reči o nekima od tih komponenti, a za sada samo ekspetimentišite.

Ukoliko primetite da često pravite istu, ili sličnu formu za rad sa bazama podataka, razmislite o postavljanju te forme na Object Repository. Forme koje se nalaze na Object Repository-ju se mogu ponovo koristiti uz samo nekoliko pokreta mišom.

Pravljenje master/detail forme

Sledeće što je potrebno da naučite je kako da napravite master/detail formu korišćenjem Database Form Wizard-a. Kada pravite master/detail formu, prvih nekoliko strana Wizard-a su slične onima koje ste videli kada ste pravili jednostavnu formu. Drugačije su samo labele koje opisuju ponuđene opcije. Prvo, izvršite navedene operacije. Posle toga ću objasniti razlike pri radu sa Database Form Wizard-om prilikom pravljenja master/detail forme.

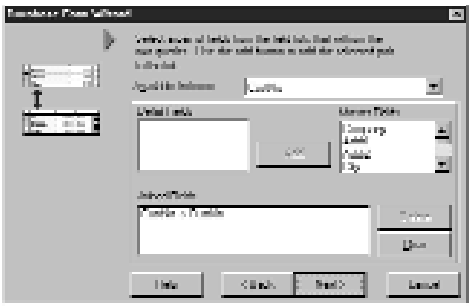
1. Pokrenite Database Form Wizard. Na prvoj strani izaberite opciju kreiranja master/detail forme i opciju korišćenja `TTable` komponente za sve tabele. Kliknite na taster Next.
2. Sledeći korak je izbor master tabele. Izaberite prvo `DBDEMOS` alias, pa zatim i `CUSTOMER.DB` tabelu. Kliknite na taster Next.
3. Izaberite samo `CustNo` i `Company` polja i dodajte ih u Ordered Selected Fields listu. Kliknite na taster Next.
4. Izaberite Horizontally opciju da biste rasporedili komponente horizontalno. Kliknite na taster Next.
5. Sada izaberite detail tabelu. Izaberite `ORDERS.DB` tabelu, a zatim kliknite na taster Next.
6. Izaberite `CustNo`, `OrderNo`, `SaleDate` i `AmountPaid` polja i dodajte ih u Ordered Selected Fields listu. Kliknite na taster Next.
7. Izaberite opciju postavljanja komponenti u mrežu i kliknite na taster Next.

Sada vidite stranu koju niste ranije videli u Database Form Wizard-u. Na ovoj strani birate polje preko koga će se povezati dve tabele.

1. Kliknite na Available Indexes kombo-listu koja se nalazi na vrhu strane i izaberite `CustNo` polje. `CustNo` polje sadrži sekundarni indeks, tako da ćete ga koristiti za povezivanje tabela.
2. Izaberite `CustNo` i u Detail Fields i u Master Fields listi. Pošto ste to uradili, Add taster koji se nalazi između dve liste postaje dostupan.
3. Kliknite na Add taster da biste dodali `CustNo` polje u Joined Fields listu koja se nalazi na dnu strane.

NAPOMENA Ukoliko ste obratili pažnju, primetili ste da ova strana Database Form Wizard-a izgleda skoro isto kao i Link Field Designer koji ste videli u danu 16, "Arhitektura baza podataka u Delphi-ju", kada ste ručno pravili master/detail formu. I jedan i drugi dijalog-prozor obavljaju istu operaciju.

Slika 17.8 prikazuje Database Form Wizard posle povezivanja tabela.

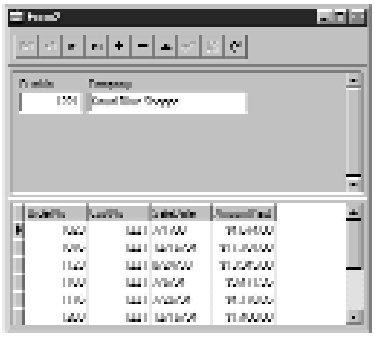


Slika 17.8
Dve tabele povezane preko CustNo polja

NAPOMENA Strana prikazana na slici 17.8 je nešto drugačija, ukoliko koristite `TQuery` umesto `TTable` komponentu kao dataset. Konkretno, Available Indexes kombo-lista nije prisutna kada se koriste TQuery komponente. Razlog za to je što se veza ostvaruje kroz SQL naredbe, a tada se indeksi ne koriste na isti način kao kod `TTable` komponenti.

4. Kliknite na taster Next da biste se pomerili na sledeću stranicu Database Form Wizard-a. Pošto su na ovoj strani već postavljenje opcije koje će Vam trebati, kliknite na taster Finish.

Sada Delphi pravi formu i spremni ste da je isprobate. Kliknite na taster Run da biste pokrenuli program. Koristite DBNavigator da biste se pomerali kroz zapise. Primetite da se ime mušterije prikazuje u gornjem delu forme, a da se njegove narudžbine prikazuju u tabeli na dnu forme. Slika 17.9 prikazuje master/detail program u toku izvršavanja.



Slika 17.9
Vaša nova master/detail forma na delu

SAVET Ukoliko želite da koristite SQL naredbe za pravljenje master/detail relacije, pokrenite ponovo Database Form Wizard i ovaj put izaberite TQuery komponente za dataset-ove. Pošto Delphi napravi formu, kliknite na svaku Query komponenetu i proučite njenu SQL osobinu.

Ručno pravlje formi za rad sa bazama podataka

Database Form Wizard je dobar alat, kada je potrebno za kratko vreme napraviti jednostavnu formu. Posebno je pogodan za test programe. Neće proći mnogo vremena, a Vama će zatrebati mogućnost ručnog pravljenja formi za rad sa bazama podataka. Kada steknete iskustvo, radije ćete ručno praviti forme nego uz pomoć Database Form Wizard-a.

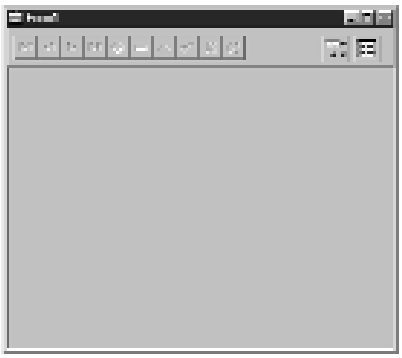
Ručno pravljenje formi za rad sa bazama podataka nije komplikovano. Potrebno je da postavite jedan, ili više dataset-ova (`Table`, ili `Query` tipa) na formu, po jednu `DataSource` komponentu za svaki dataset i po jednu komponentu za svako polje u dataset-u koje želite da vidite. Zvuči lako? I jeste.

Hajde da sada generišemo formu koja će biti slična onoj koju ste napravili u odeljku "Pravljenje jednostavne forme korišćenjem Database Form Wizard-a". Neophodno je da prvo postavite bazne komponente i da, kasnije, dodate komponente za rad sa podacima. Izvršite sledeće operacije:

1. Pokrenite novi program. Postavite `Panel` komponentu na formu i postavite njenu `Align` osobinu na `alTop`. Obrišite naslov (caption).
2. Postavite `ScrollBox` komponentu ispod panela (možete je pronaći u Additional jezičku na paleti sa komponentama). Postavite `Align` opciju na `alClient`.
3. Postavite `Table` komponentu na formu. Možete je postaviti gde god želite, ali preporučujem da je postavite sa desne strane panela. Izmenite njenu `DatabaseName` osobinu u `DBDEMOS` i `TableName` osobinu u `ANIMALS.DBF`.
4. Postavite `DataSource` komponentu pored `Table` komponente. Izmenite njenu `DataSet` osobinu u `Table1`.
5. Postavite `DBNavigator` na panel i to negde pri vrhu forme. Postavite njegovu `DataSource` osobinu na `DataSource1`.

SAVET Možete postaviti `DataSource` osobinu dvostrukim klikom na vrednost koja se nalazi pored osobine u Object Inspector-u. Delphi će prikazivati raspoložive `DataSource` komponente posle svakog dvostrukog klika.

Vaša forma sada izgleda kao i ona na slici 17.10.



Slika 17.10
Forma u početnoj fazi izrade

Sada je potrebno da postavite kontrole za rad sa podacima, kako biste mogli da vidite podatke iz tabele. Verovatno ćete želeti da, dok radite ovu vežbu, pogledate sliku 17.7, kako biste videli krajnji rezultat. Uradite sledeće:

1. Postavite `DBEdit` komponentu uz levu ivicu glavnog dela forme. Izmenite Name osobinu u `NameEdit`. Izmenite `DataSource` osobinu u `DataSource1`. Takođe, izmenite `DataField` osobinu u `NAME` (u stvari, izaberite `NAME` iz liste).
2. Postavite `Label` komponentu iznad `DBEdit` komponente (koristite standardnu `Label` komponentu koja se nalazi na Standard strani palete sa komponentama, a ne `DBText` komponentu). Izmenite njenu `Caption` osobinu u Name.
3. Ponovite korake 1 i 2 i dodajte `DBEdit` i `Label` komponente za `SIZE`, `WEIGHT` i `AREA` polja. Pazite da sadržaj `Name` i `DataField` osobina bude isti kao i imena polja.

   U sledeća dva koraka ćete odstupiti od slike 17.7 i postaviti komponentu za `BMP` polje. Ona će se nalaziti sa desne strane komponenti koje ste upravo postavili. Logičnije je da se ona nađe tu, nego ispod ostalih komponenti.
4. Postavite `DBImage` komponentu na formu i to sa desne strane u odnosu na `DBEdit` komponente. Izmenite `Name` osobinu u `BMPImage`. Takođe, izmenite `DataSource` osobinu u `DataSource1` i `DataField` osobinu u `BMP`.
5. Postavite `Label` komponentu iznad `DBImage` i izmenite njenu `Caption` osobinu u `Picture` (ime polja je `BMP`, ali je izraz prikladniji).

Vaša forma sada izgleda kao ona na slici 17.11.

Slika 17.11
Forma sa svim potrebnim komponentama

Sada je potrebno da postavite veličinu `DBImage` komponente. Iako ne možete biti sigurni da će svaka slika u svakoj bazi podataka biti iste veličine, u ovom slučaju će biti tako. Mogli biste nagađati pravu veličinu, ali je mnogo lakše postaviti veličinu komponente onda kada se u njoj već nalazi slika. Kliknite na `Table` komponentu i postavite njenu osobinu na `True`. U `DBImage` kontroli će se naći slika vezana za prvi zapis u tabeli. Sada možete postaviti veličinu `DBImage` kontrole na potrebnu veličinu.

Skoro ste završili sa pravljenjem forme, ali postoji još nešto što biste trebali da uradite. Da bi ova forma bila ista kao i ona koju ste napravili pomoću Database Form Wizard-a, potrebno je da otvorite tabelu neposredno pošto se kreira forma:

1. Prvo, postavite `Active` osobinu `Table` komponente na `False` (pre par minuta ste je postavili na `True`).
2. Izaberite samu formu u Object Inspector-u. (Da biste izabrali formu, kliknite na panel, pa pritisnite taster Esc. Formu, takođe, možete izabrati iz Component Selector-a koji se nalazi na vrhu Object Inspector-a).
3. Prebacite se na Events stranicu i napravite proceduru za rad sa `OnCreate` događajem. Unesite sledeću liniju koda u proceduru:

```
Table1.Open
```

To je to. Vaša forma je gotova. Kliknite na Run taster da biste je testirali. Ponašaće se isto kao i forma koju ste napravili sa Database Form Wizard-om.

Pogled iz blizine na komponente za rad sa podacima

Sada je na redu pogled iz blizine na komponente za rad sa podacima. Daću Vam kratak opis svake komponente sa naglaskom na osobine i metode svake od njih. Većina ovih komponenti je nasleđena iz standardnih komponenti, tako da sa njima imaju dosta sličnih osobina. Govoriću samo o osobinama koje su specifične za verziju za rad sa podacima svake komponente.

Zajedničke osobine komponenti za rad sa podacima

Sve komponente za rad sa podacima imaju zajedničkih osobina. Na primer, sve komponente imaju `DataSource` osobinu. Ova osobina se koristi za povezivanje komponente i izvora podataka koji je vezan za dataset. U prethodnih par dana ste dosta koristili `DataSource` osobinu, tako da biste trebali da imate ideju kako ona radi.

Većina kontrola za rad sa podacima ima `DataField` osobinu. Ovu osobinu koristite da biste vezali kontrolu za određeno polje u dataset-u. Videli ste kako se koristi `DataField` osobina, kada ste ručno pravili formu za rad sa bazama podataka. Kada povežete komponentu za određeno polje u dataset-u, sadržaj tog polja se prikazuje direktno u komponenti. U slučaju `Table` dataset-ova (i `Query` datasetova, ukoliko je `RequestLive` osobina postavljena na `True`), izmena podataka koji se nalaze u kontroli znači i izmenu podataka u samoj bazi podataka.

Većina kontrola za rad sa podacima sadrži i `Field` osobinu. Koristićete `Field` osobinu za pristup sadržaju komponente kroz kod. Na primer, da biste izmenili sadržaj DBEdit komponente možete uraditi sledeće:

```
NameEdit.Field.AsString := 'Clown Fish';
```

Kroz `Field` osobinu, možete promeniti i sadržaj drugih TField komponenti.

Možete koristiti `ReadOnly` osobinu da biste korisnicima zabranili izmenu podataka u komponenti koja inače to dozvoljava (`DBEdit`, `DBGrid` i druge).

`DBGrid` komponenta

`DBGrid` komponenta prikazuje sadržaj dataset-a u tabelarnom obliku. Jedna od najvažnijih osobina ove komponente je `Columns` osobina. Ova osobina Vam dozvoljava da promenite broj i redosled kolona koje će se videti. Korišćenjem Columns Editor-a možete dodavati i izbacivati kolone, kao i menjati njihov redosled.

Da biste aktivirali Columns Editor, kliknite desnim tasterom na DBGrid komponentu i izaberite `Columns` Editor iz konteksnog menija. Takođe, možete ga aktivirati i pritiskom na ellipsis taster pored `Columns` osobine u Object Inspector-u. Pretpostavimo da dataset sadrži puno polje, ali da Vi želite da vidite samo nekoliko u okviru `DBGrid` komponente. Korišćenjem Columns Editor-a možete sakriti polja koja ne želite da budu prikazana.

Nemojte mešati `Columns` osobinu `DBGrid` komponente (sa kojom se radi pomoću Colums Editor-a) i `FieldDefs` osobinu Table komponente (sa kojom se radi pomoću Fields Editor-a). `FieldDefs` osobina `Table` komponente određuje koja će se polja nalaziti u samom dataset-u. Sa druge strane, `Columns` osobina DBGrid komponente samo određuje koja polja će se videti.

`DefaultDrawing` osobina određuje da li sama VCL biblioteka iscrtava ćelije u `DBGrid` komponenti, ili se o iscrtavanju brine programer.

Ukoliko je `DafaultDrawing` `False`, morate odgovoriti na `OnDrawColumnCell` i `OnDrawDataCell` događaje da biste obezbedili iscrtavanje ćelija.

`Options` osobina Vam omogućava da postavite opcije za prikazivanje i ponašanje DBGrid komponente. Koristeće ovu osobinu, možete da izbacite naslove kolona, dozvolite, ili zabranite promenu veličine kolona, uključite, ili isključite prikazivanje linija kolona i vrsta i tako dalje.

`TitleFont` osobina određuje kojim fontom će se ispisivati imena kolona. Da biste postavili font kojim se ispisuje sadržaj ćelija, koristite `Font` osobinu.

`DBGrid` ima samo dve javne metode. Kada sami iscrtavate sadržaj, možete pozvati `DefaultDrawColumnCell` i `DefaultDrawDataCell` metode da biste naredili VCL-u da iscrta ćeliju umesto Vas. To je korisno ukoliko želite da kontrolišete iscrtavanje samo nekih ćelija, dok Vam je za ostale dovoljno podrazumevano iscrtavanje.

`DBGrid` komponenta ima nekoliko događaja od kojih je većina vezana za izmenu ćelija i kretanje kroz dataset. Neću navoditi događaje na ovom mestu, pošto se iz njihovih imena može lako zaključiti koju funkciju obavljaju.

`DBNavigator` komponenta

`DBNavigator` komponenta omogućava korisniku da se kreće kroz dataset zapis po zapis. Ova komponenta obezbeđuje tastere za pomeranje na prvi zapis, sledeći zapis, prethodni zapis, poslednji zapis, kao i tastere za dodavanje zapisa, brisanje zapisa, izmenu zapisa, odustajanje od izmena, upisivanje izmena i osvežavanje prikaza. Ova komponenta uglavnom radi automatski, tako da ćete je uglavnom samo postaviti na formu, povezati za `DataSource` i zaboraviti na nju.

Kada je `ConfirmDelete` osobina postavljena na `True`, svaki pritisak na taster za brisanje zapisa (Delete) dovodi do prikazivanja dijalog-prozora u kome se traži potvrda brisanja. Možete postaviti `Hints` osobinu na `True`, kako biste omogućili prikazivanje balončića sa opisom svakog tastera na `DBNavigator-u`. `VisibleButtons` osobina Vam omogućava da odredite koji će tasteri biti aktivni na `DBNavigator-u`. Možete dodavati i izbacivati tastere i u toku pravljenja programa i u toku izvršavanja.

`DBNavigator` ima samo jednu bitnu metodu i jedan događaj. Možete koristiti `BtnClick` metodu za simulaciju pritiska levog tastera miša na ovoj komponenti. Možete koristiti `OnClick` događaj za detekciju pritiska levog tastera miša na ovoj komponenti. Retko ćete koristiti `OnClick` događaj, pošto `DBNavigator` već zna šta treba da uradi kada se klikne na određeni taster.

DBText komponenta

DBText komponenta je verzija za rad sa podacima klasične Label komponente. Omogućava prikaz podatka iz polja, bez mogućnosti da korisnik izmeni taj podatak. Ova komponenta nema osobine koje su specifične za rad sa bazama podataka, kao ni metode, ni događaje koji su zajednički za sve komponente za rad sa podacima.

DBEdit komponenta

DBEdit komponenta predstavlja kontrolu za izmenu podataka koja je povezana za određeno polje u dataset-u. DBEdit komponentu ste koristili kada ste ručno pravili formu za rad sa bazama podataka. Kao ni DBText, ni DBEdit komponenta nema osobine koje su specifične za rad sa bazama podataka, kao ni metode, ni događaje koji su zajednički za sve komponente za rad sa podacima.

DBMemo komponenta

DBMemo je verzija za rad sa podacima klasične Memo komponente. Možete koristiti ovu komponentu za prikazivanje podataka iz polja koja sadrže veću količinu teksta. AutoDisplay osobina kontroliše da li se podaci iz dataset-a prikazuju automatski, tj. kada se kurzor pomeri na neki drugi zapis.

Kada je AutoDisplay True podaci se prikazuju automatski. Kada je AutoDisplay False, morate kliknuti dva puta na DBMemo, kako biste prikazali podatke (ili pritisnuti taster Enter kada kontrola ima fokus). Da biste naredili prikazivanje podataka kroz kod koristite LoadMemo metodu. Naravno, korišćenje ove metode ima smisla samo ako je AutoDisplay postavljen na False.

DBImage komponenta

DBImage komponenta se koristi za prikazivanje BLOB podataka koji su zapisani u formatu slike. Možete promeniti sliku tako što ćete je vratiti iz Clipboard-a, ili korišćenjem Picture osobine, da biste učitali sliku sa diska. Sledeći deo koda menja sliku u toku izvršavanja programa:

```
DBImage1.Picture.LoadFromFile('peregrine.bmp');
```

Osobine DBImage komponente određuju način na koji se slika prikazuje. Sledi njihov opis:

- AutoDisplay osobina radi na način koji je opisan za DBMemo komponentu.
- LoadPicture metoda se može koristiti za prikazivanje slika u slučaju da je AutoDisplay osobina postavljena na False.
- Picture osobina omogućava pristup samoj slici i radi na istu način kao i kod standardne Image komponente.

- `Center` osobina određuje da li će se slika centrirati u okviru `DBImage` prozora.
- Ukoliko je slika veća od veličine `DBImage` prozora, `Stretch` osobina određuje da li će biti smanjena da bi mogla cela da bude prikazana, ili će se prikazati u originalnoj veličini. Ukoliko je `Stretch False`, deo slike će biti odsečen, ukoliko je slika veća od veličine `DBImage` prozora.
- `QuickDraw` osobine određuje da li će se paleta boja pridružiti slici u toku prikazivanja. Ukoliko je `QuickDraw True`, paleta se ne pridružuje. Ukoliko je `QuickDraw False`, paleta se pridružuje i iscrtana slika je kvalitetnija, ali je brzina iscrtavanja nešto manja.

Pomenimo `CutToClipboard`, `CopyToClipboard` i `PasteFromClipboard` metode `DBImage` komponente. Ove metode rade baš to.

DBListBox i DBComboBox komponente

`DBListBox` je u mnogome isto što i standardna ListBox komponeneta. Jedina razlika je u tome što se stavka koju izabere korisnik direktno upisuje u polje u dataset-u (koje se određuje pomoću `DataField` osobine). Da biste dodelili listu mogućih stavki, postavite je pomoću Items osobine, isto kao i kod `ListBox` komponente. Važno je razumeti da se ova lista mogućih stavki ne nalazi u bazi podataka (za to se koristi `DBLookupListBox`).

`DBComboBox` radi na potpuno isti način kao i `DBListBox`, pri čemu postoje standardne razlike između lista i kombo-lista.

DBCheckBox komponenta

Koristićete `DBCheckBox` komponentu prvenstveno za prikazivanje sadržaja logičkih polja (True/False, Yes/No, On/Off). Postavite string za proveru stanja u `ValueChecked` osobinu. Na primer:

```
DBCheckBox1.ValueChecked := 'On';
```

U ovom slučaju, ukoliko je vrednost podatka u polju On, `DBCheckBox` će biti izabran. Ukoliko vrednost podatka u polju nije On, DBCheckBox neće biti izabran. Možete postaviti više od jednog podatka za proveru stanja u ValueChecked osobinu. Na primer:

```
DBCheckBox1.ValueChecked := 'On;Yes;True';
```

Ukoliko je vrednost podatka u polju jednaka bilo kojoj od navedenih, `DBCheckBox` će biti izabran. `ValueUnchecked` osobina radi na isti način, samo što `DBCheckBox` neće biti izabran ukoliko je vrednost podatka u polju bilo koja od navedenih. Ukoliko postavite i `ValueChecked` i `ValueUnchecked` osobine, `DBCheckBox` će biti zatamnjen (engl. grayed), ukoliko se vrednost podatka u polju ne nalazi ni u `ValueChecked`, ni u `ValueUnchecked`.

DBRadioGroup komponenta

DBRadioGroup komponenta radi na sličan način kao i DBListBox i DBComboBox komponente. Navedite stavke za grupu i kada se neka od njih izbere, njen tekst (vrednost) se upisuje u određeno polje u bazi podataka.

Values osobina sadrži trenutnu vrednost polja u bazi podataka. Možete koristiti Values osobinu da biste zamenili string koji se prikazuje u grupi sa nekim drugim. Na primer, možete imati DBRadioGroup komponentu sa opcijama koje se zovu: Yearly, Quarterly i Monthly. U Vašoj bazi podataka se umesto punih imena mogu nalaziti kodovi kao što su Y, Q i M. U ovom slučaju biste trebali da postavite Value osobinu (listu stringova) na:

```
Y
Q
M
```

Kada se izabere neka od opcija, umesto punog imena, u bazu podataka će se upisati jednoslovni kod. Ukoliko je Values osobina prazna, u bazu podataka će se upisati tekst (vrednost) izabrane opcije.

DBLookupListBox i DBLookupComboBox komponente

DBLookupListBox komponenta Vam omogućava prikaz liste podataka iz određenog polja. Za razliku od DBListBox, ovu listu podataka ne unosite Vi, nego se ona uzima iz posebnog dataset-a. Postavite DataSource i DataField osobine na željeni dataset i na polje u koje će se upisivati izbor. Postavite ListSource i ListField osobine na polje odakle će se sakupljati podaci koji će se naći u listi.

DBLookupComboBox radi na isti način kao i DBLookupListBox, uz jedan dodatak. Kod DBLookupComboBox komponente se koriste DropDownAlign, DropDownRows i DropDownWidth osobine da bi se odredio izgled padajuće liste.

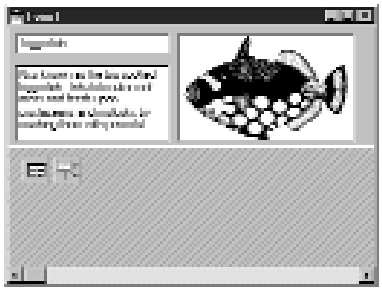
DBRichEdit komponenta

DBRichEdit komponenta Vam omogućava pregled i izmenu memo polja iz dataset-a koje sadrži rich tekst. AutoDisplay i LoadMemo metode ove komponente rade na potpuno isti način kao i kod DBMemo komponente.

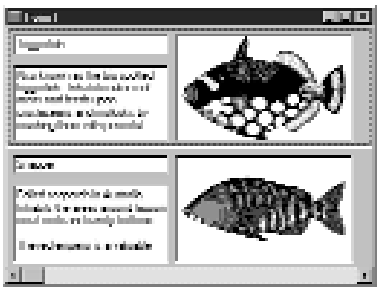
DBCtrlGrid komponenta

DBCtrlGrid je komponenta koja Vam dozvoljava da napravite posebnu komponentu za tabelarni prikaz podataka uz mogućnost korišćenja skrolera. Možete postaviti bilo koju komponentu za rad sa podacima na mesto prve ćelije DBCtrlGrid komponente i Delphi će duplicirati te komponente za svaki zapis u dataset-u.

Ilustracija će Vam pomoći da ovo bolje razumete. Na slici 17.12 se nalazi forma na kojoj postoju `DBCtrlGrid` komponenta koja je postavljena tako da popunjava celi korisnu površinu forme. `DBCtrlGrid` sadrži `DBEdit`, `DBMemo` i `DBImage`. Sve te komponente su postavljene na prvu ćeliju `DBCtrlGrid` komponente. Druga ćelija je zatamnjena, što znači da na nju ne možete postavljati komponente. Slika 17.13 prikazuje istu formu u momentu izvršavanja programa.



Slika 17.12 Forma sa DBCtrlGrid komponentom za vreme pravljenja programa



Slika 17.13 Ista ta forma u momentu izvršavanja programa

`DBCtrlGrid` komponenta sadrži nekoliko bitnih osobina. Možete koristiti Orientation osobinu da biste odredili sa koje strane će se nalaziti skroler i kako će se komponenta ponašati kada se klikne na skroler (`DBCtrlGrid` na slikama 17.12 i 17.13 ima Orientation osobinu postavljenu na `goHorizontal`). Možete koristiti `PanelWidth` i `PanelHeight` osobine da biste postavili širinu i visinu ćelije u tabeli. `RowCount` osobina određuje koliko zapisa treba prikazivati istovremeno.

`DBCtrlGrid` komponenta sadrži i jedan događaj, `OnPaintPanel`. Ovaj događaj se generiše svaki put kada neka od ćelija u tabeli treba da bude iscrtana. Možete odgovoriti na ovaj događaj da biste crtali na pozadini. Ovo se odnosi samo na pozadinu. Komponente koje se nalaze na DBCtrlGrid se automatski iscrtavaju i Vi o tome ne treba da brinete.

Ostale komponente za rad sa podacima

`DBChart` je komponenta za grafički prikaz podataka i isporučuje se sa profesionalnom i klijent/server verzijom Delphi-ja. Ova osobina nije samo moćna već i vrlo složena. Ne mogu Vam sada objasniti sve mogućnosti ove komponente, tako da ćete morati da eksperimentišete i da je sami istražite.

Klijent/server verzija Delphi-ja sadrži i Decision Cube jezičak u paleti sa komponentama, na kome se nalazi još šest komponenti. Ove komponente omogućavaju kompleksne analize podataka, kao što je kros tabulacija tabela i grafova. Ne bi bilo naročito produktivno objašnjavati na ovom mestu osobine ovih komponenti. Ipak, želim da znate da se te komponente nalaze u klijent/server verziji Delphi-ja.

Zaključak

Pa, to je sve što se tiče baza podataka. Šalim se, tek smo počeli. Koristite Database Form Wizard za brzo pravljenje jednostavnih formi za rad sa bazama podataka. Komplikovanije forme za rad sa bazama podataka pravite ručno, kako biste imali bolju kontrolu nad njima. Kojim god putem da krenete, morate potrošiti određeno vreme da biste dobro savladali pravljenje formi za rad sa bazama podataka. Ne možete postati eksperti preko noći. Sa druge strane, to nije ni naročito teško. Nastavite sa radom i vrlo brzo ćete savladati baze podataka. Veoma je važno da znate koje su Vam sve komponente za rad sa podacima na raspolaganju u toku pravljenja programa.

Radionica

Radionica sadrži test pitanja koja Vam pomažu da učvrstite svoje razumevanje izložene materije i vežbe koje Vam pomažu da steknete iskustvo u onome što ste naučili. Možete pronaći odgovore na test pitanja u Dodatku A "Odgovori na test pitanja".

Pitanja i odgovori

P **Sviđa mi se način na koji Database Form Wizard štedi moje vreme, ali mi se ne dopada način na koji on raspoređuje komponente na formi. Šta predlažete?**

O Pokrenite Database Form Wizard da biste kreirali komponente za sva polja koja želite da vidite na formi. Onda ručno promenite raspored komponenti, kako bi forma zadovoljila Vaše potrebe. Database Form Wizard pomaže samo na početku. Kasnije možete sa formom raditi šta god želite.

P **Da li je bitno da li pravim formu koja će koristiti `TTable`, ili `TQuery`, pri radu sa Database Form Wizard-om?**

O To zavisi od tipa baze podataka sa kojom želite da radite. Ako radite sa lokalnim bazama podataka kao što su Paradox i dBASE, trebali biste da koristite `TTable`. Ukoliko radite sa klijent/server bazama podataka, koristite `TQuery`.

P **Da li postoji ograničenje broja `DataSource`, `Table` i `Query` komponenti koje se mogu nalaziti na formi?**

O Ne postoji ograničenje. Sa druge strane postoji praktično ograničenje koje zavisi od toga sa koliko dataset-ova možete raditi u isto vreme.

P **Primetio sam da `DBEdit` komponenta nema `Text` osobinu kao klasična `Edit` komponenta. Zbog čega?**

O `DBEdit` komponenta nema `Text` osobinu zato što se njen sadržaj uzima iz dataset-a. Da biste pristupili sadržaju `DBEdit` komponente, koristite `GetFieldByName` metodu dataset-a i `Value` osobinu `TField-a`.

P **Da li moram koristiti `DBNavigator` da bi moji korisnici mogli da se kreću kroz zapise baze podataka?**

O Ne, možete napraviti sopstvene tastere i napisati kod pomoću kojeg se kurzor pomera, ukoliko se klikne na neki od tih tastera. Ipak, korišćenjem `DBNavigator-a` problem postaje jednostavniji.

P **Moja baza podataka ima nekoliko velikih slika u BLOB poljima. Čini mi se da je proces prikazivanja slika veoma spor. Da li je moguće ubrzati prikazivanje slika?**

O Postavite `QuickDraw` osobinu na `True`. Vaše slike će se prikazivati brže, ali možda neće izgledati tako lepo.

Kviz

1. Koji je najbrži i najjednostavniji način za pravljenje forme za rad sa bazama podataka?
2. Na koji način možete da kontrolišete raspored i broj kolona koje će se prikazivati u `DBGrid` komponenti?
3. Koje komponente Vam omogućavaju da prikažete dataset u tabelarnom obliku?
4. Kako možete dodati, ili ukloniti taster sa `DBNavigator-a`?
5. Koje komponente ćete koristiti za prikaz slika koje se nalaze u BLOB poljima?
6. Koje osobine su zajedničke za sve komponente za rad sa podacima?
7. Koja osobina se koristi za definisanje polja za koje će se vezati određena komponenta?
8. Da li možete ponovo rasporediti kolone u `DBGrid` komponenti?

9. Koja komponenta se koristi za prikaz i izmenu tekst polja u bazi podataka?
10. Šta znači BLOB?

Vežbe

1. Napravite novi program koji prikazuje sadržaj `VENDORS.DB` tabele (nalazi se u `DBDEMOS` bazi podataka).
2. Izmenite program koji ste napravili u vežbi 1, tako da se prikazuju samo polja `VendorName`, `City`, `State` i `Phone`.
3. Za korisnike profesionalne i klijent/server verzije Delphi-ja: Napravite master/detail formu iz `IBLOCAL` baze podataka (ukoliko je potrebno, instalirajte Local InterBase sa Delphi-jevog CD-a). Koristite `EMPLOYEE` tabelu kao master i `EMPLOYEE_PROJECT` tabelu kao detail. (Savet: koristite `EMP_NO` polje za povezivanje tabela).
4. Napravite formu za rad sa bazama podataka koja će sadržati `DBCtrlGrid` komponentu za pregled tabele. Koristite bilo koju tabelu i uključite bilo koja polja.
5. Ručno napravite formu za rad sa bazama podataka (bez korišćenja Database Form Wizard-a), koristeći bilo koju tabelu i bilo koja polja. Dodajte `DBNavigator` komponentu za kretanje kroz tabelu.
6. Uklonite sve tastere osim First, Next, Prior i Last sa `DBNavigator` komponente koju ste koristili u vežbi 5.

Dan 18

Pravljenje programa za rad sa bazama podataka

Ovo poglavlje se bavi pravljenjem programa za rad sa bazama podataka. Ipak, ja Vam ne mogu reći, u jednom kratkom poglavlju, sve što je potrebno znati da bi se pisali programi za rad sa bazama podataka. Ono što mogu je da Vam ukažem na neke bitne principe sa kojima ćete se sretati, dok budete pisali programe.

Pošto već znate kako se prave forme za rad sa bazama podataka, najveći deo današnjeg dana ćete provesti radeći sa aspektima nižeg nivoa programiranja baza podataka. Počećete sa kreiranjem i popunjavanjem baze podataka kroz kod da biste kasnije radili sa modulima za podatke (engl. data modules). Pred kraj dana, upoznaćete se sa generisanjem izveštaja korišćenjem QuickReport komponente. Konačno, da završavamo pričom o isporučivanju programa za rad sa bazama podataka.

Programiranje baza podataka bez vizuelnih komponenti

Do sada, radili ste skoro isključivo sa Delphi-jevim komponentama pri radu sa bazama podataka. Taj aspekt programiranja baza podataka je lak i zabavan, ali pre, ili kasnije ćete morati da se bavite i pravim programiranjem baza podataka. Programi, koje ste do sada pravili, su pretežno služili jednostavnom pregledu i izmeni podataka. U ovom odeljku, naučićete kako da izvršite pojedine operacije sa bazama podataka kroz kod.

SAVET Pomenuću samo operacije sa bazama podataka koje su izvodljive kroz `TTable` klasu. Vi, u svom kodu, možete koristiti i SQL da biste izvršili pojedine operacije, ali Vam danas neću govoriti o tom aspektu programiranja baza podataka.

Čitanje iz baze podataka

Ovaj odeljak je kratak zato što čitanje iz postojeće baze podataka nije teško. U ovom odeljku ćemo koristiti `CUSTOMER.DB` tabelu i iz njenih podataka ćemo kreirati tekstualni fajl sa zarezima. Nećete upisivati u fajl baš svako polje iz tabele, ali većinu hoćete.

Prvi korak se sastoji u kreiranju `TTable` objekta. Sledeći korak je njegovo vezivanje za tabelu u okviru baze podataka. Kod izgleda ovako:

```
var
  Table : TTable;
begin
  Table := TTable.Create(Self);
  Table.DatabaseName := 'DBDEMOS';
  Table.TableName := 'Customer.db';
end;
```

Ovaj kod radi istu stvar koju radi i Delphi kada postavite `DatabaseName` i `TableName` osobine Table komponente u toku pravljenja programa. Sada je potrebno pročitati svaki zapis iz baze podataka i uspisati ga u teksualni fajl. Prvo ću Vam pokazati osnovnu strukturu bez stvarnog koda koji je potreban da bi se sadržaj polja upisao u tekstualni fajl. Posle toga ću biti konkretniji po pitanju koda. Osnovna struktura izgleda ovako:

```
Table.Active := True;
while not Table.Eof do begin

  { Code here to read some fields from the }
  { current record and write them to a text file. }

  Table.Next;
end;
Table.Free;
```

Ovaj kod ima jednosmernu strukturu. Prvo se otvara tabela, postavljanjem `Active` osobine na `True` (može se, takođe, pozvati `Open` metoda). Zatim se u okviru `while` petlje iščitavaju zapisi iz tabele. Pošto se zapis upiše u tekstualni fajl, poziva se metoda `Next` koja pomera kurzor na sledeće polje. Uslovni izraz `while` petlje proverava `Eof` osobinu i prekida petlju kada se dođe do kraja tabele. Konačno, kada se svi zapisi prepišu u tekstualni fajl, `TTable` objekat se briše iz memorije.

Ne morate eksplicitno brisati `TTable` objekat iz memorije. Kada u svom kodu kreirate `TTable` objekat, obično mu prosleđujete formin `Self` pointer. Na primer:

```
Table := TTable.Create(Self);
```

Ovaj deo koda proglašava formu vlasnikom tabele. Kada se forma obriše, VCL automatski briše `TTable` objekat iz memorije. U slučaju glavne forme programa, ovo se dešava kada se program zatvori. Iako niste obavezni da brišete `TTable` objekat, dobra je ideja da obrišete svaki objekat, kada Vam on postane nepotreban.

Naravno, potrebno je da pročitate podatak iz svakog polja i da ga upišete u tekstualni fajl. Da biste to uradili, koristićete `FieldByName` metodu i `AsString` osobinu `TField` objekta. Ovaj postupak sam ukratko objasnio u danu 16 "Arhitektura baza podataka u Delphi-ju", u odeljku "Pristupanje poljima". Prvo polje iz `CUSTOMER.DB` tabele koje treba da prepišete je polje `CustNo`. Čitanje podatka iz polja se vrši na sledeći način:

```
var
  S : string;
begin
  S := Table.FieldByName('CustNo').AsString + ',';
```

Primetite da se na kraj pročitanog podatka dodaje zarez, kako bi se razdvojili podaci iz različitih polja. Ovaj kod ćete prepisati za svako polje iz kojeg želite da prepišete podatke. Celokupni proces je prikazan u listinzima 18.1 i 18.2. Ovi listinzi prikazuju program pod nazivom `MakeText` koji se može naći u kodu koji je isporučen uz knjigu. Ovaj jednostavan program koristi tabelu `CUSTOMER.DB` i kreira teksualni fajl sa zarezima pod imenom `CUSTOMER.TXT`. Listing 18.1 prikazuje celinu glavne forme (`MakeTxtU.pas`). Forma sadrži samo taster i `Memo` kontrolu. Pregledajte ove listinge, pa ćemo kasnije objasniti na koji način program radi.

Listing 18.1: `MakeTxtU.pas`

```
unit MakeTxtU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
  StdCtrls, DbTables;

type
  TForm1 = class(TForm)
    CreateBtn: TButton;
    Memo: TMemo;
    procedure CreateBtnClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

implementation

{$R *.DFM}
```

nastavlja se

Listing 18.1: MakeTxtU.pas *nastavak*

```
procedure TForm1.CreateBtnClick(Sender: TObject);
var
  Table : TTable;
  S     : string;
begin
  { Create the Table and assign a database and Table name. }
  Table := TTable.Create(Self);
  Table.DatabaseName := 'DBDEMOS';
  Table.TableName := 'Customer.db';

  { Change to the busy cursor. }
  Screen.Cursor := crHourGlass;

  { We can use a Memo to show the progress as well as to    }
  { save the file to disk. First clear the memo of any text.}
  Memo.Lines.Clear;

  { Open the Table. }
  Table.Active := True;
  CreateBtn.Enabled := False;

  { Loop through the records, writing each one to the memo. }
  while not Table.Eof do begin

    { Get the first field and add it to the string S, }
    { followed by the delimiter.                      }
    S := Table.FieldByName('CustNo').AsString + ',';

    { Repeat for all the fields we want. }
    S := S + Table.FieldByName('Company').AsString + ',';
    S := S + Table.FieldByName('Addr1').AsString + ',';
    S := S + Table.FieldByName('Addr2').AsString + ',';
    S := S + Table.FieldByName('City').AsString + ',';
    S := S + Table.FieldByName('State').AsString + ',';
    S := S + Table.FieldByName('Zip').AsString + ',';
    S := S + Table.FieldByName('Phone').AsString + ',';
    S := S + Table.FieldByName('FAX').AsString;

    { Add the string to the Memo. }
    Memo.Lines.Add(S);

    { Move to the next record. }
    Table.Next;
  end;

  { Write the file to disk. }
  Memo.Lines.SaveToFile('customer.txt');
```

```
  { Turn the button back on and reset the cursor. }
  CreateBtn.Enabled := True;
  Screen.Cursor := crDefault;
  Table.Free;
end;

end.
```

Celokupni proces se izvršava u okviru `CreateBtnClick` metode. Kada kliknete na Create File taster, program iščitava podatke iz tabele i stavlja ih u `Memo` komponentu. Prvo se iščitava sadržaj polja `CustNo` i upisuje se u string, posle čega se dodaje zarez. Posle toga, svako sledeće polje se dodaje na kraj stringa i ponovo se dodaje zarez. Pošto se iščitaju svi podaci iz zapisa, string se, pomoću Add metode, dodaje u Memo komponentu. Kada se dođe do kraja tabele, sadržaj Memo komponente se upisuje u teksualni fajl na disku.

Koristio sam `Memo` komponentu iz dva razloga. Prvo, prikazivanjem podataka u Memo komponenti imate uvid u to koje podatke program pravi. Drugo, `Lines` osobina Memo komponente (skup `TString` objekata) omogućava jednostavan način za uspisivanje podataka u tekstualni fajl na disku. Slika 18.1 prikazuje `MakeText` program, pošto se tekstualni fajl upiše na disk.

Slika 18.1 `MakeText` program u fazi izvršavanj

Pravljenje baze podataka kroz kod

Najveći deo informacija koje budete pročitali, a vezane su za Delphi će Vam sugerisati pravljenje baze podataka korišćenjem programa kao što je Database Desktop. To je u redu ukoliko Vaš program koristi samo tabele koje su napravljene ranije. Ali ukoliko sam program treba da pravi tabele, takav pristup nije moguć. Morate da pronađete način kako da napravite tabelu kroz kod. Na primer, ukoliko Vaš program dozvoljava korisnicima da postave imena polja u tabeli, nećete moći da znate ta imena dokle god ih sam korisnik ne unese.

Pravljenje tabele kroz kod zahteva sledeće korake:

1. Napravite BDE alias za bazu podataka.
2. Kreirajte `TTable` objekat.
3. Dodajte definiciju polja u `FieldDefs` osobinu.
4. Dodajte definiciju indeksa u `IndexDefs` osobinu (naravno, ukoliko Vaša tabela sadrži indekse).
5. Napravite tabelu uz pomoć `CreateTable` metode.

Krenimo korak po korak, kako biste tačno znali šta se dešava .

Kreiranje BDE alias-a i TTable objekta

Vi ste već jednom izvršili prvu operaciju u danu 16 kada sam govorio o pravljenju BDE alias-a, a `TTable` objekat ste kreirali u prethodnom odeljku. Hajde da ponovimo ove dve operacije:

```
var
  Table : TTable;
begin
  š Create the alias. }
  CreateDirectory('c:', nil);
  Session.AddStandardAlias('MyDatabase', 'c:', 'PARADOX');
  Session.SaveConfigFile;

  { Create the table. }
  Table := TTable.Create(Self);
  Table.DatabaseName := 'MyDatabase';
  Table.TableName := 'MyTable.db';
end;
```

Ovaj kod pravi BDE alias za bazu podataka pod imenom `MyDatabase` u direktorijumu C:. Posle toga se kreira `TTable` objekat i `DatabaseName` osobina se postavlja na upravo napravljen alias. Konačno se `TableName` osobina postavlja na ime nove tabele koju ćete upravo kreirati. U ovom trenutku je kreiran `TTable` objekat, ali se tabela još uvek ne nalazi na disku.

Kreiranje definicije polja

Sledeći korak je definisanje polja koja će postojati u tabeli. Kao što verovatno pogađate, definicija polja opisuje svako polje. Definicija polja sadrži ime polja, njegov tip, njegovu veličinu (ukoliko je potrebno) i podatak da li je neophodno da u svakom zapisu polje bude ispunjeno (tzv. zahtevano polje). Tip polja može biti jedna od `TFieldType` vrednosti.

Tabela 18.1: Mogući tipovi polja u tabeli

Tip polja	Opis
`tfUnknown`	Nepoznato ili nedefinisano polje
`ftString`	Karakter ili string polje
`tfSmallint`	16-bitno celobrojno polje
`ftInteger`	32-bitno celobrojno polje
`ftWord`	16-bitno neoznačeno celobrojno polje
`ftBoolean`	Logičko polje
`ftFloat`	Numeričko polje sa pokretnim zarezom
`ftCurrency`	Polje sa podacima o novcu
`ftBCD`	Binarno-kodirano decimalno polje
`ftDate`	Datumsko polje
`ftTime`	Vremensko polje
`ftDateTime`	I datumsko i vremensko polje
`ftBytes`	Polje sa fiksnim brojem bajtova (binarni zapis)
`ftVarBytes`	Polje sa promenljivim brojem bajtova (binarni zapis)
`ftAutoInc`	32-bitno celobrojno brojačko polje koje se automatski inkrementira
`ftBlob`	Polje za zapisivanje velikih binarnih objekata
`ftMemo`	Tekstualno memo polje
`ftGraphic`	Polje za zapisivanje bit-mapiranih slika
`ftFmtMemo`	Formatirano memo polje
`ftParadoxOle`	Paradox-ovo OLE polje
`ftDBaseOle`	dBASE-ovo OLE polje
`ftTypedBinary`	Tipizirano binarno polje

Definicija polja se kreira korišćenjem Add metode `TFieldDefs` klase. `FieldDefs` osobina `TTable` objekta je `TFieldDefs` objekat. Dakle, dodavanje novog tekstualnog polja u bazu podataka izgleda ovako:

```
Table.FieldDefs.Add('Customer', ftString, 30, False);
```

Ovaj kod dodaje tekstualno polje pod nazivom `Customer` koje je veličine 30 znakova. Polje nije zahtevano, pošto je poslednji parametar `Add` metode `False`. Dodajte koliko god želite novih polja, postavljajući odgovarajući tip polja i veličinu.

Kreiranje definicije indeksa

Ukoliko će Vaša tabela biti indeksirana, morate dodati jednu, ili više definicija indeksa. Dodavanje definicije indeksa je slično dodavanju definicije polja. `IndexDefs` osobina `TTable` objekta sadrži definicije indeksa. Da biste dodali novu definiciju indeksa pozovite `Add` metodu `TIndexDefs` objekta:

```
Table.IndexDefs.Add('', 'CustNo', [ixPrimary]);
```

Prvi parametar `Add` metode određuje ime indeksa. Ukoliko pravite primarni indeks (kao u ovom primeru), ne morate unositi ime indeksa. Drugim parametrom se određuje po kom polju, ili po kojim poljima će se vršiti indeksiranje. Ukoliko imate više od jednog polja u indeksu, razdvojite ih korišćenjem tačka-zarez simbola (;). Na primer:

```
Table.IndexDefs.Add('', 'Room;Time', [ixPrimary]);
```

Poslednji parametar `Add` metode određuje tip indeksa. Ovaj parametar je u stvari `TIndexOption` skup vrednosti. Može se postaviti više različitih vrednosti. Na primer, indeks može biti primaran i može se sortirati u opadajućem redosledu. U tom slučaju, poziv `Add` metode može da izgleda ovako:

```
Table.IndexDefs.Add('', 'CustNo', [ixPrimary, ixDescending]);
```

Kreiranje tabele

Pošto ste dodali sve definicije polja i indeksa, možete kreirati samu tabelu. Do sada ste definisali sastav tabele, ali niste je zaista i kreirali. Kreiranje tabele je najlakši korak:

```
Table.CreateTable;
```

`CreateTable` metod kreira tabelu na disku. `CreateTable` koristi `FieldDefs` i `IndexDefs` osobine i na osnovu njih kreira strukturu tabele.

Pošto je tabela kreirana, možete je popunjavati na bilo koji od raspoloživih načina. Možete popunjavati tabelu kroz kod, ili dati Vašim korisnicima mogućnost unosa podataka kroz formu.

Kod koji se isporučuje uz ovu knjigu sadrži program pod imenom `MakeTabl`. Program popunjava tabelu sa podacima iz `CUSTOMER.TXT` fajla koji ste kreirali programom `MakeText`. Listing 18.2 sadrži celinu glavne forme (`MakeTblU.pas`). Primetite da uses lista ove celine sadrži `DBGrids`, `DBTables` i `DB` celine.

LISTING 18.2: `MakeTblU.pas`

```
unit MakeTblU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, Grids, DBGrids, DbTables, Db;

type
  TForm2 = class(TForm)
    DBGrid: TDBGrid;
    CreateBtn: TButton;
    FillBtn: TButton;
```

```
    procedure CreateBtnClick(Sender: TObject);
    procedure FillBtnClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
continues

Listing 18.2. continued
var
  Form2: TForm2;

implementation

{$R *.DFM}

procedure TForm2.CreateBtnClick(Sender: TObject);
const
  Dir : PChar = 'c:';
var
  Table : TTable;
begin
{ Create a BDE alias, but only if it doesn't already exist. }
  if not Session.IsAlias('MyDatabase') then begin
    CreateDirectory(Dir, nil);
    try
      Session.AddStandardAlias('MyDatabase', Dir, 'PARADOX');
      Session.SaveConfigFile;
    except
      MessageDlg('Error Creating Alias', mtError, [mbOk], 0);
      Exit;
    end;
  end;

  { Now create the Table. }
  Screen.Cursor := crHourGlass;
  Table := TTable.Create(Self);
  try
    Table.DatabaseName := 'MyDatabase';
    Table.TableName := 'MyTable.db';

    { Add the field definitions for each field. }
    Table.FieldDefs.Add('CustNo', ftFloat, 0, True);
    Table.FieldDefs.Add('Customer', ftString, 30, False);
    Table.FieldDefs.Add('Addr1', ftString, 30, False);
    Table.FieldDefs.Add('Addr2', ftString, 30, False);
    Table.FieldDefs.Add('City', ftString, 15, False);
    Table.FieldDefs.Add('State', ftString, 20, False);
    Table.FieldDefs.Add('Zip', ftString, 10, False);
```

nastavlja se

LISTING 18.2: MakeTblU.pas *nastavak*

```
      Table.FieldDefs.Add('Phone', ftString, 15, False);
      Table.FieldDefs.Add('Fax', ftString, 15, False);

      { Add an index definition for the primary key. }
      Table.IndexDefs.Add('', 'CustNo', [ixPrimary]);

      { Everything is set up, so create the Table. }
      Table.CreateTable;
    except
      MessageDlg('Error Creating Table', mtError, [mbOk], 0);
      Screen.Cursor := crDefault;
      Table.Free;
      Exit;
    end;

    { All done, so let the user know. }
    Table.Free;
    Screen.Cursor := crDefault;
    CreateBtn.Enabled := False;
    FillBtn.Enabled := True;
    MessageDlg(
      'Table Created Successfully', mtInformation, [mbOk], 0);
  end;

  procedure TForm2.FillBtnClick(Sender: TObject);
  var
    Table      : TTable;
    Datasource : TDataSource;
    Lines      : TStringList;
    S, S2      : string;
    I, P       : Integer;
  begin
    { Create a TTable. }
    Table := TTable.Create(Self);
    Table.DatabaseName := 'MyDatabase';
    Table.TableName := 'MyTable.db';

    { Create a data source and hook it up to the Table. }
    { Then hook the DBGrid to the datasource. }
    Datasource := TDataSource.Create(Self);
    Datasource.DataSet := Table;
    DBGrid.Datasource := Datasource;

    { Open the Table and the text file. }
    Table.Active := True;
    Lines := TStringList.Create;
    Lines.LoadFromFile('customer.txt');
```

```
{ Put the Table in edit mode. }
Table.Edit;

{ Process the Lines. }
for I := 0 to Pred(Lines.Count) do begin

  { Append a record to the end of the file. }
  Table.Append;

  { Parse the string and get the first value. }
  S := Lines[I];
  P := Pos(',', S);
  S2 := Copy(S, 1, P - 1);
  Delete(S, 1, P);

  { Write the value to the CustNo field. }
  Table.FieldByName('CustNo').Value := StrToInt(S2);

  { Continue for each of the fields. }
  P := Pos(',', S);
  S2 := Copy(S, 1, P - 1);
  Delete(S, 1, P);
  Table.FieldByName('Customer').Value := S2;

  P := Pos(',', S);
  S2 := Copy(S, 1, P - 1);
  Delete(S, 1, P);
  Table.FieldByName('Addr1').Value := S2;

  P := Pos(',', S);
  S2 := Copy(S, 1, P - 1);
  Delete(S, 1, P);
  Table.FieldByName('Addr2').Value := S2;

  P := Pos(',', S);
  S2 := Copy(S, 1, P - 1);
  Delete(S, 1, P);
  Table.FieldByName('City').Value := S2;

  P := Pos(',', S);
  S2 := Copy(S, 1, P - 1);
  Delete(S, 1, P);
  Table.FieldByName('State').Value := S2;

  P := Pos(',', S);
  S2 := Copy(S, 1, P - 1);
  Delete(S, 1, P);
  Table.FieldByName('Zip').Value := S2;
```

nastavlja se

LISTING 18.2: MakeTblU.pas *nastavak*

```
    P := Pos(',', S);
    S2 := Copy(S, 1, P - 1);
    Delete(S, 1, P);
    Table.FieldByName('Phone').Value := S2;

    P := Pos(',', S);
    S2 := Copy(S, 1, P - 1);
    Delete(S, 1, P);
    Table.FieldByName('FAX').Value := S2;

    { We might get a key violation exception if we try to add a }
    { record with a duplicate customer number. If that happens, }
    { we need to inform the user, cancel the edits, put the     }
    { put the Table back in edit mode, and continue processing. }
    try
      Table.Post;
    except
      on EDBEngineError do begin
        MessageBox(Handle,
          'Duplicate Customer Number', 'Key Violation', 0);
        Table.Cancel;
        Table.Edit;
        Continue;
      end;
    end;
  end;

  { All done with the TStringList so it. }
  Lines.Free;

  { We won't the Table so that the Table's data shows }
  { in the DBGrid. VCL will the Table and Datasource  }
  { for us. }
  {Table.Free; }
  {Datasource.Free; }
end;

end.
```

Korišćenje modula za rad sa podacima

Kao što znate, postavljanje komponenti za rad sa bazama podataka u Delphi-ju je jednostavno. Ipak, potrebno je neko vreme, čak i za najjednostavnije primere koje ste pisali.

Morate postaviti `Table`, ili `Query` komponentu na formu i izabrati ime baze podataka. Zatim morate postaviti ime tabele (za `Table`), ili `SQL` osobinu (za `Query`). Možda ćete morati da postavite i druge osobine što zavisi od načina na koji koristite bazu podataka. Možda ćete morati da odgovorite na neke događaje. Sledeće što morate da uradite je da postavite `DataSource` komponentu na formu i da je vežete za tabelu, ili upit. Ukoliko Vaš program koristi i `Database` komponentu, moraćete da postavite i njene osobine i da odgovorite i na njene događaje. Ovo nije teško, ali zar ne bi bilo lepo kada biste mogli da uradite taj posao samo jednom u toku pravljenja programa? Moduli za rad sa podacima (engl. Data Modules) Vam omogućavaju upravo to.

U osnovi, moduli za rad sa podacima su specijalizovane forme. Da biste napravili modul za rad sa podacima, otvorite Object Repository i dva puta kliknite na Data Module ikonu. Delphi će napraviti modul za rad sa podacima i odgovarajuću celinu, kao i prilikom pravljenja nove forme. Možete postaviti `Name` osobinu modula za rad sa podacima, kao što možete i pri radu sa formama.

Pošto ste napravili modul za rad sa podacima, na njega postavite komponente za rad sa podacima. Zatim postavite sve potrebne osobine ovih komponenti. Možete čak kreirati i procedure za obradu događaja za bilo koju komponentu na modulu. Kada ste postavili sve što želite, sačuvajte modul. Od sada, svaka forma u programu može pristupiti ovom modulu za rad sa podacima.

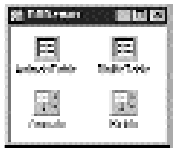
Postavljanje jednostavnog modula za rad sa podacima

Ova jednostavna vežba će Vam pomoći da bolje razumete module za rad sa podacima. Prvo ćete postaviti jedan modul, a zatim ćete ga isprobati.

1. Kreirajte novi program. Izmenite `Name` osobinu glavne forme u `MainForm`. Sačuvajte projekat. Snimite glavnu formu kao `DSExMain.pas` a projekat kao `DSExampl.dpr`.
2. Izaberite File➥New. Kada se otvori Object Repository, dva puta kliknite na Data Module ikonu da biste kreirali novi modul. Modul se prikazuje na Form Designer-u. Izmenite `Name` osobinu u `DBDemos`.
3. Kliknite na Data Access jezičak na paleti za komponente. Postavite `Table` komponentu na modul za rad sa podacima. Izmenite `DatabaseName` osobinu u `DBDEMOS` i `TableName` osobinu u `ANIMALS.DBF`. Izmenite `Name` osobinu u `AnimalsTable`.
4. Postavite drugu `Table` komponentu na modul. Postavite `DatabaseName` osobinu na `DBDEMOS` i `TableName` osobinu na `BIOLIFE.DB`. Izmenite Name osobinu u `BiolifeTable`.
5. Postavite `DataSource` komponentu na modul. Izmenite `Name` osobinu u `Animals` i `DataSet` osobinu u `AnimalsTable`.

6. Postavite drugu `DataSource` komponentu na modul. Izmenite njenu Name osobinu u `Biolife` i njenu `DataSet` osobinu u `BiolifeTable`. Vaš modul za rad sa podacima bi sada trebao da izgleda kao na slici 18.2.

Slika 18.2
Završeni modul za rad sa podacima



7. Dva puta kliknite na pozadinu modula. Biće kreirana procedura za obradu `OnCreate` događaja. Unesite sledeći kod:

```
AnimalsTable.Open;
BiolifeTable.Open;
```

8. Sačuvajte projekat. Snimite modul za rad sa podacima kao `DataMod.pas`.

Dodavanje modula za rad sa podacima na formu

Hajde da sada upotrebimo napravljeni modul. Napravićete dva tastera u okviru glavne forme programa. Jedan taster će prikazivati formu sa `Animals` tabelom, a drugi formu sa `Biolife` tabelom. Uradite sledeće:

1. Kreirajte novu formu. Izmenite njenu `Caption` osobinu u `Animals Form` i `Name` osobinu u `AnimalsForm`.
2. Izaberite File➥Use Unit. Izaberite `DataMod` celinu iz Use Unit dijalog-prozora i kliknite na OK. Sada se modulu za rad sa podacima može pristupiti sa glavne forme.
3. Postavite `DBGrid` i `DBNavigator` komponente na formu. Izaberite obe komponente. Pronađite `DataSource` osobinu u Object Inspector-u i kliknite na taster sa strelicom. Videćete sledeće stavke u listi mogućih izvora:

```
DBDemos.Animals
DBDemos.Biolife
```

Izaberite `DBDemos.Animals`.

4. Snimite celinu kao `DSExU2.pas` (ili, smislite neko bolje ime, ukoliko želite).
5. Ponovo, kreirajte novu formu. Ponovite korake 1-3, ali ovaj put izaberite `DBDemos.Biolife` za izvor i izmenite `Caption` osobinu u `BioLife Form`. Izmenite `Name` osobinu forme u `BioLifeForm`. Sačuvajte formu kao `DSExU3.pas`. Slika 18.3 prikazuje izgled Delphi-jevog IDE-a po završetku ove vežbe.

Pokretanje modula za rad sa podacima

Sledeći koraci pokazuju na koji način možete koristiti komponente sa modula za rad sa podacima bilo gde u svom programu. Sve što treba da uradite je da upotrebite celinu modula za rad sa podacima, a nakon toga će sve komponente za rad sa podacima moći da mu pristupe. Hajde da završimo program, kako biste mogli da ga isprobate:

1. Postavite taster na glavnu formu. Izmenite njegovu `Caption` komponentu u `Show Animals`.
2. Dva puta kliknite na taster da biste kreirali proceduru za obradu `OnClick` događaja. Unesite ovaj kod u proceduru:

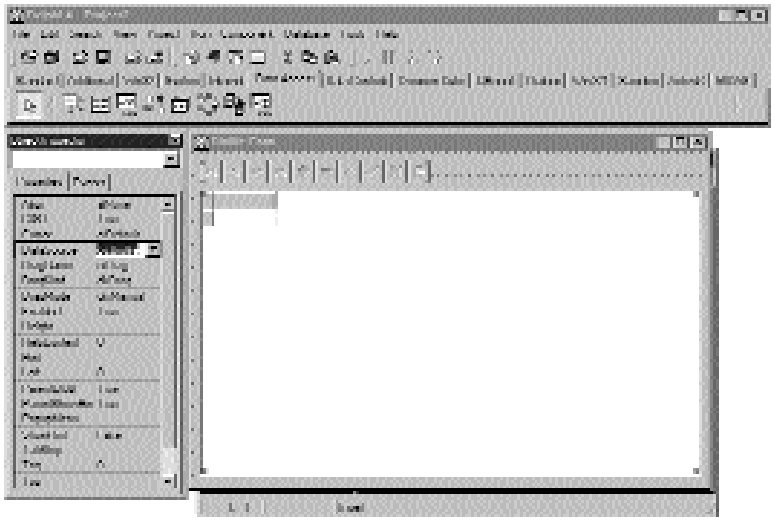
```
AnimalsForm.Show;
```

3. Postavite novi taster na formu. Izmenite njegovu `Caption` komponentu na `Show Biolife`.
4. Kreirajte proceduru za obradu OnClick događaja za ovaj taster i dodajte sledeći kod:

```
BiolifeForm.Show
```

5. Izaberite File➥Use Unit. Izaberite `DSExU2` celinu i kliiknite na OK.
6. Ponovite korak 5 da biste iskoristili i `DSExU3` celinu.

Sada možete pokrenuti program. Kada kliknete na neki od tastera, pojaviće se odgovarajuća forma. Slika 18.4 prikazuje program u fazi izvršavanja.



Slika 18.3
Završena druga forma

Slika 18.4
Program u fazi izvršavavanja, pri čemu su otvorene obe forme

Moduli za rad sa podacima omogućavaju jednostavno postavljanje komponenti za rad sa podacima i njihovo ponovno korišćenje. Pošto kreirate modul za rad sa podacima, možete ga postaviti u Object Repository, odakle ga možete uvek koristiti.

Generisanje izveštaja

Program za rad sa bazama podataka ne možemo nazvati kompletnim, dokle god u njemu ne postoji mogućnost pregleda i štampanja podataka. To se radi pomoću izveštaja. Do sada ste koristili mogućnosti prikaza jednog, ili više zapisa pomoću `DBGrid` komponente, što može biti dovoljno za neke programe. Međutim, ubrzo će Vam zatrebati mogućnost bolje kontrole pregleda podataka.

Pored pregleda zapisa na ekranu, gotovo uvek će Vam trebati i mogućnost štampanja podataka. Program za rad sa bazama podataka koji ne može da štampa i nije mnogo koristan. Delphi-jeve QuickReport komponente Vam omogućavaju jednostavan pregled i štampanje podataka.

Pregled QuickReport komponenti

Pre generisanja samih izveštaja morate znati na koji način QuickReport komponente rade.

`QuickRep` komponenta

Osnovna QuickReport komponenta je `QuickRep`. Ova komponenta igra ulogu podloge na koju dodajete potrebne elemente Vašeg izveštaja (mogući elementi će biti prikazani kasnije).

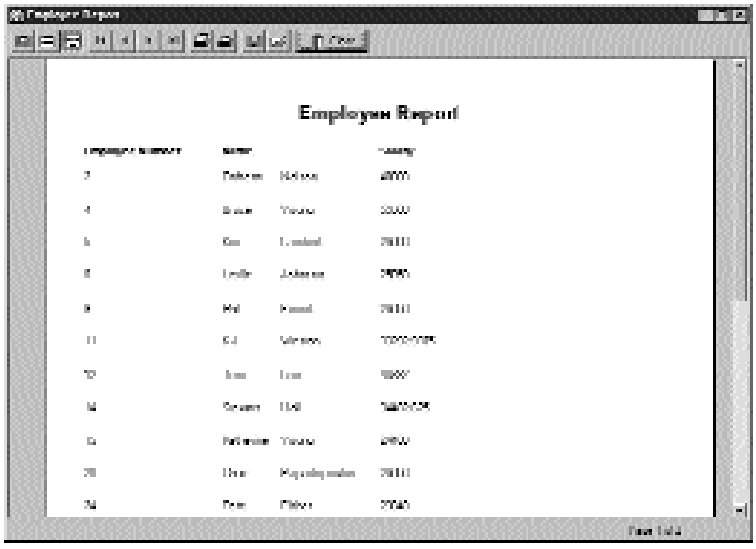
`QuickRep` komponenta sadrži osobine pomoću kojih menjate izgled izveštaja prilikom štampanja. Na primer, Page osobina je klasa koja sadrži osobine `TopMargin`,

`BottomMargin`, `LeftMargin`, `RightMargin`, `Columns`, `Orientation`, `PaperSize` i tako dalje.

`PrinterSettings` je takođe klasa. Ova osobina ima sopstvene osobine sa sledećim nazivima: `Copies`, `Duplex`, `FirstPage`, `LastPage` i `OutputBin`. `ReportTitle` osobina se koristi za prikazivanje obaveštenja u Windows-ovom Print Manager-u i u naslovnoj traci QuickReport-a, prilikom pregleda podataka. `Units` osobina kontroliše da li će se margine obračunavati u milimetrima, inčima, picas-ima, ili u nekom drugom mernom sistemu. `DataSet` osobina se koristi za postavljanje dataset-a iz kojeg će izveštaj koristiti podatke.

`DataSet` osobina mora biti postavljena i izabrani dataset mora biti aktivan, pre nego što izveštaj prikaže podatke.

Primarne `QuickRep` metode sadrže `Preview` i `Print`. `Print` metoda, kao što i njeno ime kaže, štampa izveštaj. `Preview` metoda prikazuje modalni prozor za pregled izveštaja. Prozor za pregled sadrži tastere za pregled opcija, tastere za prelazak na prvu stranicu, poslednju stranicu, prethodnu stranicu, sledeću stranicu, taster za štampanje, taster za pokretanje Print Setup-a, tastere za otvaranje i čuvanje izveštaja kao i taster za zatvaranje prozora za pregled. Slika 18.5 prikazuje izgled QuickReport-ovog prozora za pregled u toku izvršavanja programa.



Slika 18.5
QuickReport-ov prozor za pregled podataka

Bitni događaji `QuickRep` komponente su `OnPreview` i `OnNeedData`. Možete koristiti `OnPreview` za prikazivanje sopstvenog prozora za pregled podataka, umesto podrazumevanog. Kada se koristi izvor podataka koji nije jedna od VCL baza podataka, možete koristiti `OnNeedData` događaj. Na primer, možete praviti izveštaj iz liste stringova, niza podataka, ili iz tekstualnog fajla.

Celine izveštaja

QuickReport se sastoji od celina. Postoji nekoliko vrsta celina. Osnovni izveštaj sadrži barem tri celine: naslovnu, celinu sa naslovima kolona i celinu sa podacima (engl. detail band). Naslovna celina sadrži naslov koji se prikazuje samo na prvoj strani izveštaja. Celina sa naslovima kolona se koristi za prikazivanje naslova polja u dataset-u. Celina sa naslovima kolona se pojavljuje na vrhu svake stranice. Neki izveštaji, kao što je izveštaj za štampanje poštanskih nalepnica, ne sadrže celinu sa naslovima kolona.

U celini sa podacima se prikazuju stvarni podaci. Na nju možete postaviti sve podatke koje želite da prikažete u izveštaju. Možete definisati sadržaj ove celine i QuickReport će ga ponoviti za svaki zapis u dataset-u. Za par minuta ćete uraditi vežbu kroz koju ćete videti kako rade različite celine.

Druge često korišćene celine su: gornje zaglavlje strane (engl. page header), donje zaglavlje strane (engl. page footer), grupno zaglavlje i zaključna celina. `QRBand` komponenta predstavlja celine izveštaja. `BandType` osobina se koristi za preciziranje tipa celine (naslov, celina sa podacima, gornje zaglavlje, donje zaglavlje i tako dalje).

Celine se same raspoređuju na `QuickRep` komponenti, u zavisnosti od tipa celine. Na primer, ako na `QuickRep` postavite `QRBand` komponentu i izmenite njenu `BandType` osobinu na `rbPageFooter`, celina će se postaviti ispod ostalih celina. Analogno, gornje zaglavlje će se postaviti iznad ostalih celina.

Sastavni elementi QuickReport izveštaja

Sastavni elementi QuickReport izveštaja se nalaze u tri grupe. Prva grupa sadrži labele, slike, oblike, gornja i donja zaglavlja i tako dalje. Ovi elementi se koriste za prikazivanje statičkih delova izveštaja. Na primer, naslov izveštaja se postavlja jednom i više se ne menja. Drugi primer su grafički elementi koji se koriste za prikazivanje logotipa kompanije na izveštaju. `QRLabel` komponenta je slična klasičnoj `Label` komponenti, `QRImage` je slična `Image` komponenti, `QRShape` je slična `Shape` komponenti i tako dalje. Koristite ove komponente za pravljenje statičkih delova izveštaja.

Druga kategorija sadrži QuickReport varijante standardnih VCL komponenti za rad sa podacima. Ove komponente se postavljaju na celinu sa podacima u okviru izveštaja. Neke od komponenti u ovoj grupi su: `QRDBText`, `QRDBRichEdit` i `QRDBImage`. Podaci se uzimaju iz dataset-a i sa njima se popunjava telo izveštaja.

U trećoj grupi se nalaze `QRSysData` i `QRExpr` komponente. `QRSysData` se se koristi za prikazivanje broja strane, datuma i vremena generisanja izveštaja, naslova izveštaja i tako dalje. `QRExpr` komponenta se koristi za prikazivanje rezultata izračunavanja. `Expression` osobina definiše izraz pomoću kojeg se vrši izračunavanje. `Expression` komponenta sadrži editor koji se zove `Expression` builder i

koji se koristi za pravljenje jednostavnih izraza. Izraz može biti jednostavan, kao što je množenje sadržaja dva polja, ali može biti i komplikovaniji korišćenjem `AVERAGE`, `COUNT` i `SUM` formula.

Ručno generisanje izveštaja

Ručno generisanje je svakako najbolji način za pravljenje izveštaja. Možda će Vam zazvučati komplikovano, ali QuickReport komponente u mnogome olakšavaju posao. Najbolji način za objašnjavanje ovoga je primer. Kroz ovaj primer ćete napraviti jednostavan program koji prikazuje i štampa izveštaj u tabelarnom obliku. Neću objašnjavati svaki korak. Na primer, neću Vam govoriti kako da sačuvate projekat, ili kako da postavljate imena fajlovima - to možete i sami. Takođe, za sada se ne morate previše truditi oko izgleda izveštaja. Možete se vratiti kasnije i ulepšati Vaš izveštaj.

Prvi korak je kreiranje glavne forme programa. Kada to završite, morate napraviti osnovni izgled izveštaja. Uradite sledeće:

1. Napravite novi program. Postavite dva tastera na glavnu formu. Izmenite Caption osobinu prvog tastera u `Preview Report`, a drugog tastera u `Print Report`.
2. Izaberite File➥New. Dva puta kliknite na Report ikonu u Object Repository-ju. Delphi kreira novu QuickReport formu.
3. Postavite `Table` komponentu na QuickReport formu. Izmenite `DatabaseName` osobinu u `DBDEMOS` i `TableName` osobinu u `EMPLOYEE.DB`. Postavite `Active` osobinu na `True`.
4. Izaberite QuickReport formu. Izmenite `DataSet` osobinu u `Table1` i izmenite `ReportTitle` osobinu u `Employee Report`.
5. Vratite se na glavnu formu. Dva puta kliknite na `Preview Report` taster. Unesite sledeći kod u proceduru za obradu `OnClick` događaja:

```
QuickReport2.Preview;
```

6. Dva puta kliknite na `Print Report` taster i unesite sledeći kod u proceduru za obradu `OnClick` događaja:

```
QuickReport2.Print;
```

7. Izaberite File➥Use Unit da biste koristili QuickReport formu.

Sada imate prazan izveštaj. Potrebno je da dodate naslovnu celinu, celinu sa naslovima kolona i celinu sa podacima. U toku rada ćete verovatno želeti da pogledate slika 18.6 na kojoj se nalazi prikazan završeni izveštaj. Izvršite sledeće operacije:

1. Izaberite `QRBand` komponentu iz QReport jezička iz palete sa komponentama i postavite je na izveštaj. Podrazumeva se da je tip celine naslov.

2. Izaberite `QRLabel` komponentu i postavite je na naslovnu celinu. Izmenite `Caption` osobinu u `Employee Report` i izmenite `Font` osobinu u svoj omiljeni font (ja koristim Arial, 18 tačaka, Bold). Centrirajte komponentu u okviru naslovne celine.

3. Postavite još jednu celinu na izveštaj. Izmenite `BandType` osobinu u `rbColumnHeader` i izmenite `Font` osobinu u Bold i Underlined.

4. Postavite `QRLabel` na levu stranu celine sa naslovima kolona i izmenite njenu `Caption` osobinu u `Employee Number`. Postavite drugu `QRLabel` komponentu na celinu sa desne strane od prethodne i izmenite njenu `Caption` komponentu u `Name`. Postavite i treću `QRLabel` komponentu na celinu i to sa desne strane od prethodne i izmenite njenu `Caption` osobinu u `Salary`.

5. Postavite još jednu celinu na izveštaj i izmenite njenu `BandType` osobinu u `rbDetail`. Primetite da se posle ove izmene celina pomera ispod ostalih.

6. Postavite `QRDBText` komponentu uz levu ivicu celine sa podacima (poravnajte je sa `Employee Number` labelom iz celine sa naslovima kolona). Izmenite njenu `DataSet` osobinu u `Table1` i `DataField` osobinu u `EmpNo`.

7. Postavite sledeću `QRDBText` komponentu na celinu sa podacima i poravnajte je sa `Name` komponentom na celini sa naslovima kolona. Izmenite njenu `DataSet` osobinu u `Table1` i `DataField` osobinu u `FirstName`. Postavite još jednu `QRDBText` komponentu sa desne strane prethodne (pogledajte sliku 18.6). Povežite je sa `LastName` poljem tabele.

8. Dodajte poslednju `QRDBText` komponentu na celinu sa podacima. Postavite je ispod `Salary` komponente na celini sa naslovima kolona i povežite je sa `Salary` poljem u tabeli. Vaša forma bi sada trebala da izgleda kao i ona koja je prikazana na slici 18.6.

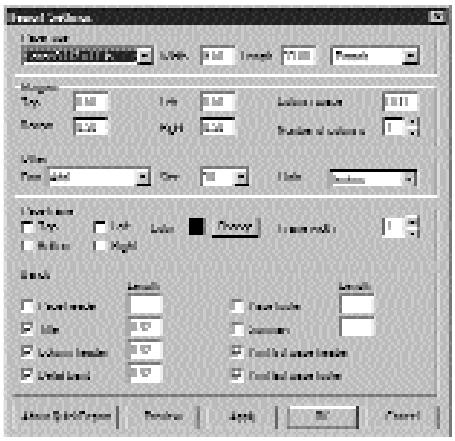
Slika 18.6
Vaša QuickReport forma

Verovatno se pitate kako će izveštaj izgledati na papiru. Ne morate da čekate da biste videli. Klinite desnim tasterom na QuickReport formu i izaberite Preview iz konteksnog menija. Prikazuje se prozor za pregled izveštaja tako da sada možete videti kako će izveštaj izgledati na papiru.

Da biste odštampali izveštaj, kliknite na Print taster. Kada završite sa pregledom izveštaja, kliknite na Close taster. Sada možete pokrenuti program i isprobati `Preview Report` i `Print Report` tastere.

NAPOMENA Vaš izveštaj je generisan sa dvostrukim razmakom između redova. Da biste to promenili, promenite visinu celine sa podacima.

Pre nego što završimo ovu priču o izveštajima, dozvolite mi da Vam pokažem još jednu lepu osobinu QuickReport-a. Kliknite desnim tasterom na QuickReport formu i izaberite Report Settings iz konteksnog menija. Prikazuje se Report Settings dijalog-prozor, kao što je prikazano na slici 18.7. Kroz ovaj dijalog-prozor možete vizuelno postaviti primarne osobine `QuickRep` komponente umesto korišćenja Object Inspector-a.



Slika 18.7
Report Settings dijalog-prozor QuickReport-a

Jednostavnije kreiranje izveštaja

Delphi se isporučuje sa tri ugrađene QuickReport forme koje se nalaze na Forms jezičku Object Repository-a. Forme imaju nazive: Quick Report Labels, Quick Report List i Quick Report Master➥Detail. Ove forme odrađuju početni deo posla prilikom generisanja izveštaja. Možete uzeti jednu od ovih formi iz Object Repository-ja i zatim je prilagoditi svojim potrebama.

Isporučivanje programa za rad sa bazama podataka napravljenog u Delphi-ju

Kao što sam rekao u danu 16, Borland Database Engine (BDE) je skup DLL-ova i drajvera koji omogućavaju programima napravljenim u Delphi-ju da komuniciraju sa različitim tipovima baza podataka. Kada isporučujete program koji koristi BDE morate isporučiti i odgovarajuće BDE fajlove i ispravno ih registrovati na korisnikovom računaru.

Najpogodniji način za to je korišćenje jednog od instalacionih programa koje je odobrio Borland. U TurboPower Software kompaniji koristimo Wise Install System firme Great Lakes Buisiness Solutions (`http://www.glbs.com`). Drugi program koji je na raspolaganju je InstallShield, zajedno sa svojim mlađim bratom InstallShield Express-om. InstallShield Express se isporučuje uz Professional i Client➥Server verzije Delphi-ja, tako da ukoliko ste korisnik jedne od ovih verzija, ne morate se brinuti u instalacionom programu.

Možda se pitate zašto Borland diktira način na koji se BDE mora isporučiti. Razlog je jednostavan. Postoji mnogo verzija BDE-a koje su u upotrebi. Neki programeri u svojim programima koriste BDE koji se isporučuje uz Delphi 1. Neki možda koriste BDE koji se isporučuje uz C++Builder 1, a neki možda onaj koji se isporučuje uz Delphi 4.

Važna činjenica je i ta da je svaka nova verzija BDE-a kompatibilna sa prethodnim verzijama. Programi pisani za rad sa starijim verzijama BDE-a će raditi i sa novijim. Ne biste smeli svojevoljno da brišete postojeće BDE fajlove. To je deo pravila koje se mora poštovati da bi sistem funkcionisao. Zbog toga, instalacioni programi koje je odobrio Borland, moraju da provere verziju svakog fajla iz BDE-a. Ukoliko je postojeći fajl noviji, instalacioni program ga mora ostaviti na disku.

Na ovaj način korisnici uvek imaju najnovije BDE fajlove na računaru. Druga stvar koju ovi instalacioni programi rade je određivanje neophodnih BDE fajlova koji se moraju isporučiti uz program. Pročitajte `DELPOY.TXT` fajl u Delphi-jevom osnovnog direktorijumu da biste saznali više o neophodnim BDE fajlovima.

Zaključak

Nema sumnje. Pravljenje programa za rad sa bazama podataka zahteva dosta truda. Delphi taj posao čini lakšim od drugih razvojnih okruženja. Danas ste saznali nešto o programiranju baza podataka bez vizuelnih komponenti. Takođe ste se upoznali sa modulima za rad sa podacima i načinima njihovog korišćenja. Dan ste završili pregledom QuickReport-a. QuickReport olakšava proces generisanja izveštaja za Vaše programe. Takođe sam objasnio i šta je sve potrebno za uspešno isporučivanje programa koji radi sa bazama podataka.

Radionica

Radionica sadrži test pitanja koja Vam pomažu da učvrstite svoje razumevanje izložene materije i vežbe koje Vam pomažu da steknete iskustvo u onome što ste naučili. Možete pronaći odgovore na test pitanja u Dodatku A "Odgovori na test pitanja".

Pitanja i odgovori

P **Pokušao sam da napravim bazu podataka u toku izvršavanja programa. Napravio sam BDE alias i postavio sve definicije polja, ali se tabela ne kreira na mom disku. Gde sam pogrešio?**

O Verovatno niste pozvali metodu `CreateTable`. Ova metoda mora biti pozvana, kako bi se tabela fizički kreirala na disku.

P **Da li mogu tokom pravljenja izveštaja, sve njegove osobine postaviti odjednom?**

O Da. Kliknite desnim tasterom na `QuickRep` komponentu i izaberite Report Settings iz konteksnog menija. Prikazuje se Report Settings dijalog-prozor. Odatle možete vizuelno postaviti veliki broj osobina.

P **Da li modul za rad sa podacima može pored komponenti sadržati i kod?**

O Da. Modul za rad sa podacima može sadržati kod koji je potreban da bi se uspešno obavile operacije sa tim modulom. Takođe može sadržati i procedure za obradu događaja i metode koje ste sami napravili. Ove metode mogu bili javne, ili privatne (samo za korišćenje unutar modula za rad sa podacima).

P **Kada zatražim pregled izveštaja, on je prazan. Gde grešim?**

O Verovatno niste postavili `Active` osobinu dataset-a na `True`. Dataset se mora otvoriti pre nego što se izveštaj počne sa radom.

P **Mogu li koristiti više od jedne celine za podatke u okviru izveštaja?**

O Ne. Možete postaviti više od jedne celine na izveštaj, ali prilikom njegovog pokretanja, popuniće se samo prva.

P **Zbog čega moram da koristim instalacioni program koji je odobrio Borland da bih instalirao moje programe za rad sa bazama podataka?**

O BDE je komplikovan za instalaciju. Instalacioni program koji je odobrio Borland će sigurno ispravno izvršiti instalaciju BDE-a.

Kviz

1. Koje je metode potrebno pozvati da bi se kreirala baza podataka u toku izvršavanja programa?
2. Čemu služi `Edit` metoda `TTable` komponente?
3. Koju ćete metodu pozvati kada želite da upišete izmene u zapis?
4. Kako se pravi novi modul za rad sa podacima?
5. Da li je modul za rad sa podacima obična forma?
6. Koji metod je potrebno pozvati da bi se odštampao QuickReport?
7. Koji tip QuickReport-ove celine prikazuje podatke iz dataset-a?
8. Koja komponenta se koristi za prikazivanje broja strane na izveštaju?
9. Kako možete pregledati izveštaj u toku pravljenja programa?
10. Za šta se koristi `QRExpr` komponenta?

Vežbe

1. Kreirajte bazu podataka (BDE alias) i tabelu za tu bazu podataka bilo kroz kod, bilo korišćenjem Delphi-jevih alata za rad sa bazama podataka.
2. Kreirajte modul za rad sa podacima koji će sadržati tabelu iz baze podataka kreirane u prethodnoj vežbi.
3. Generišite izveštaj koji prikazuje poštanske nalepnice. (Savet: Počnite sa QuickReport Labels objektom iz Forms strane Object Repository-ja).
4. Izmenite QuickReport koji ste generisali u ovom poglavlju, tako da se ime i prezime zaposlenog prikazuju preko `QRExpr komponente.`
5. Pročitajte `DEPLOY.TXT` fajl iz `Delphi-jevog` osnovnog direktorijuma da biste razumeli šta je sve potrebno za uspešno isporučivanje programa za rad sa bazama podataka napisanog u Delphi-ju.

Dan 19

Pravljenje i korišćenje DLL-ova

Ukoliko koristite Windows, ne možete pobeći od DLL-ova. Bacite pogled u svoje ➥`Windows` i ➥`Windows➥System` direktorijume. Ja koristim Windows NT 4 i prebrojao sam više od 650 DLL-ova u ➥`Winnt` direktorijumima na mom računaru. Pa, da se ne raspravljamo oko činjenice - DLL-ovi su deo svakodnevnog života sa Windows-om.

Pitanje je, da li je potrebno da znate kako se prave i koriste DLL-ovi. To ćete saznati danas. Konkretno, bavićemo se sledećim temama:

- Šta su DLL-ovi i zbog čega ih vi koristite
- Pravljenje DLL-ova
- Pozivanje funkcija i procedura iz DLL-ova
- Korišćenje formi iz DLL-ova
- Korišćenje resursa iz DLL-ova

Pri kraju dana ćete imati sasvim solidno znanje o DLL-ovima i videćete da li su Vam potrebni, ili ne. Čini mi se da Vaši programerski planovi ipak uključuju i DLL-ove.

Upoznavanje sa DLL-ovima

Jednostavno rečeno, DLL-ovi su veoma korisni. Objasniću prednosti korišćenja DLL-ova i kako se to odnosi na Vas, Delphi programera. Prvo, hajde da vidimo šta su to tačno DLL-ovi.

Šta je DLL?

NOVI TERMIN *DLL* (engl. Dynamic Link Library) je jedan, ili više delova koda koji se nalaze u fajlu sa `.dll` ekstenzijom.

DLL kod se može pozvati iz izvršnog programa, ali DLL, sam za sebe, nije izvršni program. Možete prihvatiti DLL kao pomoćni fajl izvršnog programa. Programi koji koriste kod iz DLL-ova su pozivajući programi zato što pozivaju funkcije i procedure koje se nalaze u DLL-ovima.

Postoje dva tipa DLL-ova: DLL-ovi sa kodom i DLL-ovi sa resursima. Ovo nije stroga podela. Možete, bez ikakvih problema, imati i kod i resurse u istom DLL-u. Ponekad je, ipak, jednostavnije imati jedan DLL sa kodom a drugi sa resursima. Razlog za ovo ću dati u odeljku "Korišćenje resursa u DLL-ovima".

Kod se u DLL-ovima može nalaziti u jednom od dva oblika. Prvi oblik je nezavisna funkcija koju pozivate iz svog glavnog programa. To ste već uradili nekoliko puta tokom rada sa ovom knjigom. Setite se koda iz dana 16 "Napredno programiranje" koji je glasio:

```
Screen.Cursors[myCursor] := LoadCursor(HInstance, 'MYCURSOR');
```

ili sledećeg koda, takođe iz dana 14:

```
  -1, Temp, DT_CENTER or DT_VCENTER or DT_SINGLELINE);
```

I `LoadCursor` i `DrawText` su Windows API funkcije. To ste verovatno znali. Da li ste znali da se ove dve funkcije pozivaju iz DLL-a? Kada malo bolje razmislite, ove dve funkcije se moraju negde nalaziti. Ali gde? U ovom slučaju, i `LoadCursor` i `DrawText` se nalaze u jednom od Windows-ovih DLL-ova pod imenom `user32.dll`.

A šta je sa onim uobičajenim dijalog-prozorima koje ste koristili, kao što su File Open, Print, ili Printer Setup? Oni se, takođe, moraju negde nalaziti. Konkretno, ovi dijalog-prozori se nalaze u fajlu pod imenom `Comctl32.dll`. Dakle, iako to niste znali, vi ste do sada koristili DLL-ove.

Drugi oblik koda koji se nalazi u DLL-ovima se sastoji od procedura i funkcija koje se koriste u samom DLL-u. Ove funkcije i procedure izvršavaju neke zadatke u samim DLL-ovima i nisu vidljive spoljašnjem svetu (programima koji koriste DLL-ove).

Korišćenje i pravljenje sopstvenih DLL-ova su dve potpuno različite stvari, zar ne? Pa ne baš. Pošto ste napravili DLL, možete pozivati funkcije i procedure iz njega, kao što pozivate Windows API funkcije. Pravljenje DLL-ova nije teško, pa ne postoji ništa što bi Vas sprečilo da to naučite.

Zašto biste trebali da koristite DLL-ove?

To je dobro pitanje. Na kraju krajeva, šta Vam to DLL pruža? Pruža Vam sledeće pogodnosti:

- Efektivno korišćenje postojećeg koda
- Mogućnost jednostavnog deljenja koda između više programa
- Mogućnost boljeg organizovanja koda
- Internacionalizaciju Vašeg programa
- Efektivno korišćenje Windows-ovih resursa

Hajde da pogledamo pojedinačno svaku od ovih stavki i sasvim sigurno ću Vas ubediti u to da su DLL-ovi dobra stvar.

Efektivno korišćenje postojećeg koda

Korišćenje postojećeg koda je veliki deo objektno-orijentisanog programiranja. Na kraju krajeva, zašto ponovo pronalaziti točak? DLL puno pomažu prilikom korišćenja postojećeg koda. Recimo da imate veliku količinu koda koju ste napisali da biste uradili jedan određeni zadatak u Windows okruženju. Vi ste naporno radili da biste napisali taj deo koda, pa zar ne bi bilo lepo kada biste mogli da ga iskoristite u svakom programu u kom Vam je potreban? DLL-ovi Vam omogućavaju da uradite baš to.

Sve što treba da uradite je da iskompajlirate ceo kod u DLL. Zatim je potrebno da jednostavno pozovete DLL iz bilo kog programa, gde je potreban, i da iskoristite kod. Zar može jednostavnije? (Precizno govoreći, postoje još neke operacije koje treba da uradite, ali su one u principu jednostavne, tako da se sada nećemo zadržavati na njima).

Sada imate gore pomenuti kod na raspolaganju, kad god poželite. I još nešto, sada taj DLL možete dati svojim kolegama, pa i oni mogu koristiti kod iz njega. Ovo počinje da zvuči dobro. Takođe, možete i prodate Vaš DLL drugim programerima. Pošto ste u njega stavili sve što je potrebno, takođe, možete iz njega i izvaditi sve što želite.

Možete pisati DLL-ove koji će se pozivati iz programa pisanih u C++Builder-u, Visual Basic-u, Visual C++-u i drugim okruženjima. Naravno, ovo zavisi od toga kakav se kod nalazi u DLL-u i na koji se način poziva, ali se, u svakom slučaju, to može uraditi.

Dakle, da li vidite gde ovo vodi? Pošto napišete DLL, možete ga koristiti gde god i kada god želite. Korišćenje svih lepota iz Vašeg DLL-a je samo nekoliko pritisaka na taster miša od Vas.

Deljenje koda između programa

Deljenje koda je u vezi sa korišćenjem postojećeg koda, ali zahteva jedan korak više. Recimo da ste Vi programer i da radite za veliku firmu. Imate nekoliko stotina korisnika i svaki od njih ima svoj sopstveni računar. (U ovom primeru ću zanemariti mogućnost umrežavanja računara.) Recimo da ste za ove korisnike napisali pet programa. Recimo, dalje, da svaki od ovih pet programa koristi isti deo koda čija je veličina, u prevedenom obliku, oko 100KB (slučajna vrednost). Ukoliko ne koristite DLL, moraćete da ponovite tih 100KB koda u svakom programu, što ukupno iznosi 500KB. Gospodo, to je uzaludno bačen kod.

Bolji pristup je stavljanje klasa u DLL-ove. Svaki od pet programa može koristiti isti DLL da bi obavio zajedničke operacije. To je jedna od lepota DLL-ova: pošto ste napisali DLL, svi Vaši programi ga mogu koristiti. Svaki korisnik će na svom računaru dobiti još 100KB, ali će se veličina svakog programa smanjiti za po 100KB. To znači da će svaki korisnik sačuvati 400KB prostora na disku. Ušteda 400KB prostora ne izgleda kao nešto naročito bitno, ali ukoliko tu cifru pomnožite sa stotinama korisnika, ušteda postaje značajna. Ovo je samo jednostavan primer. U realnim situacijama možete jednostavno uštedeti nekoliko megabajta korišćenjem DLL-ova.

Šta ukoliko tri od ovih pet programa rade istovremeno. Nema problema. Svaki program koristi kod iz DLL-a po potrebi i ne dolazi do konflikta. Windows se brine o tome kada i ko koristi koji deo koda tako da ceo sistem funkcioniše. Sve što treba da uradite je da napišete DLL i da počnete da ga koristite. (Ukoliko Vam prethodna priča deluje poznato, to je verovatno zato što sam sličan pristup koristio prilikom objašnjavanja fajlova potrebnih prilikom izvršavanja programa u danu 8, "Pravljenje programa u Delphi-ju".)

Organizovanje koda

Razdeljivanjem i organizovanjem Vašeg koda olakšavate proces njegove izmene. Ne sugerišem stavljanje svakog dela koda u poseban DLL, ali Vi treba da odlučite kada i gde je razumno podeliti Vaš program u logične celine. Nema mnogo prednosti ukoliko delite program u DLL-ove, samo da bi bio bolje organizovan. Sa druge strane, ukoliko se Vaš program sastoji iz više nezavisnih biblioteka, onda ima smisla postaviti svaku od njih u poseban DLL. To je, takođe, mnogo očiglednije u situacijama kada pravite biblioteke koje će koristiti nekoliko Vaših programa.

Jedna prednost ovakvog pristupa je i što možete jednostavnije unapređivati Vaše programe. Suočimo se sa tim. Čak i najbolji programi se isporučuju na tržište sa greškama. Ispravnije je pitati koliko i kakvih grešaka ima program, nego da li ima uopšte grešaka. Ukoliko otkrijete grešku u Vašem DLL-u, možete je ispraviti i poslati novi DLL Vašim korisnicima, umesto da ponovo prevodite i isporučujete ceo program.

Ovo, opet, ne izgleda kao nekakva prednost kod programa koje ste Vi do sada pisali. Ipak, možda ćete nekada pisati programe od nekoliko megabajta koji se sastoje iz puno celina. U tom slučaju, deljenje koda radi bolje organizacije ima smisla.

Internacionalizacija programa

Do pre nekoliko godina ste mogli da pišete programe i da uopšte ne razmišljate o stranim tržištima. Mogli ste da pravite menije, dijalog-prozore, pomoćne labele, poruke o greškama i ostalo na svom matičnom jeziku i to je bilo to. Pustili biste program u prodaju i više ne biste razmišljali o njemu.

Svet, izgleda, postaje sve manji i manji. Sa eksplozijom Internet-a i World Wide Web-a, stvari su potpuno drugačije nego što su bile pre nekoliko godina. Možete napraviti demo, ili čak shareware verziju Vašeg programa i staviti je na Internet. Za nekoliko sati, ili čak minuta, ljudi iz svih krajeva sveta će imati pristup Vašem programu. To je i uzbudljivo i zastrašujuće u isto vreme. To znači da morate planirati unapred i spremiti se za prevođenje Vašeg programa na druge jezike.

Jedan od načina na koji to možete uraditi je pravljenje DLL-ova sa resursima na različitim jezicima. Možete imati poseban DLL za svaki jezik, ili možete imati sve tekstualne resurse u istom DLL-u, a zatim po potrebi učitavati posebnu verziju teksta u trenutku izvršavanja. U toku izvršavanja programa, možete koristiti Windows API funkciju LoadString da biste učitali teksualne resurse i po potrebi ih dodelili različitim komponentama.

Jedan od mana Delphi-jevog modela programiranja je ta što ne koristi uobičajene resurse, kao ostala razvojna okruženja. Resursi, kao što su meniji, se ne učitavaju kao resursi nego se nalaze u okviru resursa forme. Ovo čini internacionalizaciju težom i to je jedan od nekoliko slučajeva da ovaj model programiranja radi protiv Vas.

Planiranje unapred Vam može sačuvati dosta vremena. Ukoliko, u startu, planirate program koji će biti internacionalizovan, kasnije je mnogo lakše prevesti Vaš program na neki drugi jezik.

Efektivno korišćenje Windows-ovih resursa

Današnji računari su brži, imaju više RAM-a i imaju više prostora na diskovima nego ikada. Ipak, moguće je da Vaš korisnik kaže: "Ja koristim samo 2MB sistemskog RAM-a. Pa šta?" Istina je, da uvek morate znati sa koliko resursa raspolažu računari Vaših korisnika. I ovde Vam DLL-ovi mogu pomoći. Ovo je, u stvari, nastavak priče o deljenju koda između programa.

Vratimo se na jedan od prethodnih primera. (Setite se, imate pet programa koji koriste jedan deo zajedničkog koda.) Ukoliko ne koristite DLL-ove, i ukoliko se neki od ovih programa izvršavaju u isto vreme, svi će u memoriju učitati isti kod. Bespotrebno trošite memoriju, pošto svaki od programa učitava u memoriju

sopstvenu kopiju potpuno istog koda. Umesto da svaki program učitava isti kod, koristite DLL i učitajte zajednički kod samo jednom. Svi vaši programi mogu koristiti isti kod u memoriji tako da se smanjuje opterećenost sistema.

Sastav celine DLL-a

Kao i bilo koja druga Pascal celina, i DLL celina se nalazi u posebnom obliku. Listing 19.1 prikazuje mininalnu DLL celinu.

Listing 19.1: Minimalna DLL celina

```
library TestDLL;

uses
  SysUtils,
  Classes,
  Forms,
  Windows;

procedure SayHello(AForm : TForm);
begin
  MessageBox(AForm.Handle, 'Hello From a DLL!',
    'DLL Message Box', MB_OK or MB_ICONEXCLAMATION);
end;

exports
  SayHello;

begin
end.
```

Prvo, uočite ključnu reč `library` na početku celine. Ova ključna reč pokazuje da je ova celina u stvari DLL celina. Ovaj DLL sadrži samo jednu proceduru, pod imenom `SayHello`. `SayHello` procedura se ne razlikuje od drugih Object Pascal procedura.

Sada usmerite svoju pažnju na ključnu reč `export` koja se nalazi blizu kraja celine. Sve procedure i funkcije koje se nalaze u export delu se izvoze iz DLL-a. U ovom slučaju, `SayHello` procedura se izvozi. Izvoženje funkcija i procedura je detaljnije objašnjeno u delu "Ključna reč `exports`."

Konačno, na samom kraju DLL celine se nalaze `begin` i `end` ključne reči. Ovaj deo koda je glavni deo koda DLL-a i u njega stavljate bilo koji kod koji se izvršava pošto se DLL učita u memoriju. U mnogim slučajevima (kao što je ovaj), ne morate da pišete inicijalizacioni kod, tako da ovaj deo koda ostaje prazan.

Osnove pisanja DLL-ova

Pisanje DLL-ova nije teško. Postoji nekoliko stvari o kojima morate paziti, ali uglavnom se radi o običnom Object Pascal programiranju. Počnimo priču o osnovama pisanja DLL-va. Posle toga ćete napraviti svoj prvi DLL.

Funkcije i procedure u DLL-ovima

Funkcije i procedure u DLL-ovima spadaju u jednu od dve grupe:

- Funkcije i procedure koje su lokalne za DLL
- Funkcije i procedure koje se izvoze iz DLL-a

DLL takođe može sadržati i klase koje, naravno, mogu imati metode. Sada neću govoriti o metodama klasa koje se nalaze u DLL-ovima, ali ću govoriti o ostala dva tipa procedura i funkcija.

Funkcije i procedure koje su lokalne za DLL

Funkcije i procedure koje se pozivaju iz samog DLL-a ne zahtevaju posebnu pažnju. Ovaj tip procedura i funkcija se deklariše kao i bilo koja druga procedura, ili funkcija. Druge procedure, ili funkcije koje se nalaze u samom DLL-u mogu pozivati ove procedure i funkcije, ali one ne mogu biti pozvane izvan DLL-a. Drugim rečima, pozivajući program neće imati pristup ovim procedurama i funkcijama. One se mogu smatrati privatnim za DLL, kao što su privatne metode klase privatne za klasu u kojoj se nalaze. U stvari, pozivajući program neće moći da "vidi" ove procedure i funkcije, tako da neće znati da one uopšte postoje.

Pored procedura i funkcija, DLL može sadržati i globalne podatke kojima mogu pristupiti sve funkcije i procedure iz DLL-a. U 16-bitnom Windows-u globalni podaci se dele između svih kopija DLL-a koje se nalaze u memoriji (tj. između svih instanci DLL-a). Drugim rečima, ukoliko jedan program postavi globalnu promenljivu `x` na `100`, `x` će imati vrednost `100` i za sve druge programe koji koriste isti DLL. U 32-bitnom Windows-u to nije slučaj pošto se prave zasebne kopije globalnih podataka za svaki program koji koristi DLL.

Funkcije i procedure izvezene iz DLL-a

Druga kategorija funkcija i procedura predstavlja one koje možete pozivati izvan DLL-a. Ove funkcije i procedure postaju javne pomoću izvoženja iz DLL-a. Njih mogu pozivati druge procedure i funkcije iz samog DLL-a, ili drugi programi izvan DLL-a.

Funkcije i procedure u DLL-u mogu pozivati izvršni programi, ali i drugi DLL-ovi. Drugim rečima, jedan DLL može pozivati procedure i funkcije iz drugog DLL-a.

Pošto se procedura, ili funkcija izveze, možete je pozvati iz svog programa.

Ključna reč `exports`

Da biste izvezli funkciju, ili proceduru, koristite ključnu reč exports u okviru DLL-a. Pogledajte listing 19.1 koji predstavlja primer DLL-a koji izvozi proceduru pod imenom `SayHello`. Pošto je procedura `SayHello` izvezena, može biti pozvana iz bilo kog drugog Delphi programa.

Izvoženje po imenu

Najčešće korišćeni način izvoženja funkcija i procedura je po imenu - na primer:

```
exports
  SayHello,
  DoSomething,
  DoSomethingReallyCool;
```

Ove procedure se izvoze po njihovim imenima. Možda ste primetili da `exports` deo ima istu sintaksu kao i `uses` lista. Svaka stavka se razdvaja od ostavlih pomoću zareza. Tačka-zarez se stavlja iza poslednje stavke u listi.

Izvoženje po rednom broju

Možete, takođe, izvoziti procedure i funkcije po rednom broju. Procedure i funkcije se izvoze prema rednom broju korišćenjem ključne reči index. Na primer:

```
exports
  SayHello index 1,
  DoSomething index 2,
  DoSomethingReallyCool index 3;
```

Kada uvezete funkciju u pozivajući program, morate naznačiti redni broj (objasniću uvoženje funkcija i procedura kasnije, u odeljku "Pozivanje korišćenjem statičkog učitavanja"). U najvećem broju slučajeva ćete izvoziti funkcije i procedure po imenu, tako da Vas neću zamarati sa izvoženjem po rednom broju.

Delphi automatski dodeljuje redne brojeve svakoj izveženoj proceduri, ili funkciji, bez obzira da li ste vi naznačili indeks, ili ne. Zadavanje indeksa Vam omogućava kontrolu nad rednim brojevima izveženih procedura, ili funkcija.

Izvoženje procedure, ili funkcije je samo polovina posla. Kada generišete program koji poziva proceduru, ili funkciju, morate uvesti funkciju, ili proceduru koju želite da pozovete iz DLL-a. Objasniću uvoženje funkcija i procedura kasnije, u odeljku "Pozivanje funkcija i procedura iz DLL-a".

Globalna promenljiva HInstance će, ukoliko se koristi u DLL-u, sadržati pokazivač na kopiju DLL-a u memoriji.

SAVET Da biste saznali da li se Vaš kod izvršava u programu, ili u DLL-u, ispitajte vrednost globalne promenljive `IsLibrary`. `IsLibrary` je `True` kada se poziva iz DLL-a i `False` kada se poziva iz programa.

SAVET Ukoliko imate problema prilikom izvoženja funkcija, ili procedura, pokrenite `TDUMP` program nad DLL-om. `TDUMP` prikazuje informacije o simbolima koji su izvezeni iz DLL-a. Proučavanje tih informacija će Vam dati ideju o tome gde se problem nalazi. Da biste videli samo izvezene simbole pozovite TDUMP sa `—ee` prekidačem - na primer,

```
tdump —ee mydll.dlll
```

Zapamtite i to da možete preusmeriti izlaz programa sa komandne linije u tekstualni fajl pomoću simbola >:

```
tdump —ee mydll.dll > dump.txtt
```

Korišćenje `DLLProc`

Kao što sam ranije rekao, bilo koji inicijalizacioni kod DLL-a koji treba da se izvrši se može staviti u glavni deo koda. To je jednostavno, ali šta raditi sa kodom koji treba da se izvrši pre nego što se DLL izbaci iz memorije? DLL-ovi nemaju `initialization` i `finalization` delove koda kao što to imaju druge celine. Vi, na primer, možete dinamički alocirati deo memorije u glavnom delu koda DLL-a, ali gde ćete je onda osloboditi? Odgovor je `DLLProc`. `DLLProc` je procedura koja se poziva nekoliko puta u toku rada DLL-a. Objasniću kako se koristi `DLLProc`, ali mi pre toga dozvolite da objasnim zbog čega ona uopšte postoji.

DLL prima poruke od Windows-a kada se učita u memoriju i neposredno pre njegovog izbacivanja iz memorije. On, takođe, prima poruke kada neki program počne da ga koristi i onda kada program prestane sa njegovim korišćenjem (naravno, samo ukoliko se DLL već nalazi u memoriji, kao što je slučaj kada više programa koriste isti DLL u isto vreme). Da biste primili te poruke, Vi ćete napraviti proceduru sa specijalnim potpisom i dodeliti njenu adresu globalnoj promenljivoj `DLLProc`. Tipična `DLLProc` procedura može izgledati ovako:

```
procedure MyDLLProc(Reason: Integer);
begin
  if Reason = DLL_PROCESS_DETACH then
    { DLL is unloading. Cleanup code here. }
end;
```

Jednostavno deklarisanje `DLLProc` procedure nije dovoljno da bi se ta procedura koristila. Sada morate dodeliti njenu adresu globalnoj promenljivoj `DLLProc`. To radite u glavnom delu koda DLL-a. Na primer:

```
begin
  DLLProc := @MyDLLProc;
  { More initialization code. }
end.
```

Ovaj kod će se izvršiti kada se DLL učita. `DLLProc` je sada instalirana i biće pozvana automatski kada procesi počnu i završe sa korišćenjem DLL-a, ili neposredno pre izbacivanja DLL-a iz memorije. Listing 19.2 predstavlja izvorni kod DLL-a koji sadrži `DLLProc`.

Listing 19.2: DLL celina koja koristi `DLLProc`

```
library TestDLL;

uses
  SysUtils,
  Classes,
  Forms,
  Windows;

var
  SomeBuffer : Pointer;

procedure MyDLLProc(Reason: Integer);
begin
  if Reason = DLL_PROCESS_DETACH then
    { DLL is unloading. Cleanup code here. }
    FreeMem(SomeBuffer);
end;

procedure SayHello(AForm : TForm);
begin
  MessageBox(AForm.Handle, 'Hello From a DLL!',
    'DLL Message Box', MB_OK or MB_ICONEXCLAMATION);
end;

{ More DLL code here that uses SomeBuffer. }

exports
  SayHello;

begin
  { Assign our DLLProc to the DLLProc global variable. }
  DLLProc := @MyDLLProc;
  SomeBuffer := AllocMem(1024);
end.
```

Kao što ste verovatno već zaključili iz listinga, Reason parametar `DLLProc` procedure sadrži vrednost koja predstavlja razlog zbog kojeg je `DLLProc` pozvana. Tabela 19.1 prikazuje sve moguće vrednosti `Reason` parametra.

Tabela 19.1: Vrednosti `Reason` parametra `DLLProc`

Vrednost	Opis
`DLL_PROCESS_DETACH`	DLL će biti izbačen iz memorije
`DLL_THREAD_ATTACH`	Proces počinje sa korišćenjem DLL-a
`DLL_THREAD_DETACH`	Proces završava sa korišćenjem DLL-a

Ranije sam rekao da DLL prima poruku od Windows-a kada se DLL učita u memoriju. Razumno je očekivati da se tada `DLLProc` pozove sa Reason parametrom koji ima vrednost `DLL_PROCESS_ATTACH`. To se, ipak, ne dešava. Windows definiše `DLL_PROCESS_ATTACH` poruku, ali je Object Pascal ne prenosi do `DLLProc`. Umesto toga, Object Pascal poziva glavni deo koda DLL-a kada primi `DLL_PROCESS_ATTACH` poruku. Pošto se glavni deo koda poziva kada se primi `DLL_PROCESS_ATTACH` poruka, nije neophodno da Object Pascal prenese ovu poruku do `DLLProc`.

`DLL_PROCESS_DETACH` poruka se prima samo jednom i to neposredno pre nego što se DLL izbaci iz memorije. (`DLL_THREAD_ATTACH` i `DLL_THREAD_DETACH` poruke se mogu primiti više puta, ukoliko više programa u isto vreme koristi isti DLL. Procesi mogu biti programi, višestruke niti istog programa, ili drugi DLL-ovi.) Možete koristiti `DLLProc` da biste izvršili proces čišćenja memorije neposredno pre izbacivanja DLL-a iz memorije.

Učitavanje DLL-ova

Pre nego što budete mogli da koristite funkciju, ili proceduru iz nekog DLL-a, morate taj DLL učitati u memoriju. Učitavanje DLL-a za vreme izvršavanja programa se može uraditi na dva načina. To su:

- Statičko učitavanje
- Dinamičko učitavanje

Oba načina imaju svojih prednosti i mana. Sada ću objasniti razlike između statičkog i dinamičkog učitavanja.

Statičko učitavanje

Statičko učitavanje znači da se DLL učitava automatski kada program, koji ga koristi, počne sa izvršavanjem. Da biste koristili statičko učitavanje, morate deklarisati funkciju, ili proceduru koja se nalazi u DLL-u sa ključnom reči `external` (više o tome u odeljku "Pozivanje korišćenjem statičkog učitavanja"). DLL se učitava automatski, pošto se učitao i program koji ga koristi, tako da možete pozivati bilo koju funkciju, ili proceduru koja je izvezena iz DLL-a, kao da pozivate bilo koju drugu funkciju, ili proceduru. Ovo je daleko najjednostavniji način za korišćenje koda koji se nalazi u DLL-u. Mana ovakvog načina rada je što, ukoliko se DLL koji koristi Vaš program ne nalazi na disku, program neće moći da se učita.

Dinamičko učitavanje

Dinamičko učitavanje znači da Vi učitavate DLL kada Vam je potreban i da ga Vi izbacujete iz memorije kada završite sa njegovim korišćenjem. Ovaj način učitavanja DLL-ova takođe ima svojih prednosti i svojih mana. Jedna prednost dinamičkog učitavanja je da se DLL nalazi u memoriji samo onda kada Vam je potreban, tako da efikasnije koristite memoriju. Druga prednost je što se Vaš program pokreće brže, premda ne postoji dodatni kod koji se učitava svaki put kada se i Vaš program pokrene.

Osnovna mana dinamičkog učitavanja je što zahteva više rada. Prvo je potrebno da učitate DLL korišćenjem Windows API funkcije `LoadLibrary`. Zatim, kada završite sa korišćenjem DLL-a, morate ga izbaciti iz memorije, korišćenjem funkcije `FreeLibrary`. Kasnije (sada dolazi pravi posao) morate koristiti funkciju `GetProcAddress` da biste dobili pokazivač na svaku funkciju, ili proceduru iz DLL-a koju želite da pozovete. Nepotrebno je reći koliko ovo može biti zbunjujuće. Sledeći odeljak pokazuje kako treba pozivati funkcije i procedure iz DLL-a korišćenjem i statičkog i dinamičkog učitavanja.

Pozivanje funkcija i procedura koje se nalaze u DLL-u

Način na koji ćete pozivati funkcije, ili procedure iz DLL-a zavisi od načina na koji je DLL učitan u memoriju.

Pozivanje korišćenjem statičkog učitavanja

Pozivanje funkcija i procedura iz DLL-a koji je statički učitan je jednostavno. Prvo, pozivajući program mora sadržati deklaraciju funkcije, ili procedure. Posle toga možete pozivati funkciju, ili proceduru kao i bilo koju drugu. Da biste uvezli funkciju, ili proceduru koja se nalazi u DLL-u, koristite ključnu reč `external` u deklaraciji funkcije, ili procedure. Na primer, da biste pozvali ranije definisanu proceduru `SayHello`, deklaracija u pozivajućem programu bi trebala da izgleda ovako:

```
procedure SayHello(AForm : TForm); external 'testdll.dll';
```

Ključna reč external govori prevodiocu da tu proceduru može naći u DLL-u (u ovom slučaju, u `TESTDLL.DLL`). Poziv procedure izgleda kao i poziv bilo koje druge procedure:

```
SayHello(Self);
```

Pošto ste ispravno uvezli funkciju, ili proceduru, možete je pozvati kao i bilo koju drugu funkciju, ili proceduru. Ovaj način rada podrazumeva da je funkcija, ili procedura prethodno izvezena iz DLL-a na ranije opisani način.

Kada deklarišete funkcije, ili procedure koje se nalaze u DLL-u, obratite pažnju na velika i mala slova. U ovom slučaju, Object Pascal pravi razliku između velikih i malih slova. Ukoliko navedete pogrešno ime funkcije, ili procedure, u toku izvršavanja programa ćete dobiti izuzetak i program neće moći da se učita.

Korišćenje ključne reči `external`

Postoje tri načina za korišćenje ključne reči `external`. Kada koristite ovu ključnu reč, možete uvesti proceduru, ili funkciju na jedan od sledeća tri načina:

- Po stvarnom imenu
- Po rednom broju
- Korišćenjem promene imena

Prvi način uvoženja, po stvarnom imenu, je onaj koji ste viđali do sada. Potrebno je jednostavno deklarisati funkciju, ili proceduru pod istim imenom pod kojim se nalazi u DLL-u - na primer:

```
procedure SayHello(AForm : TForm); external 'testdll.dll';
```

Drugi način uvoženja, po rednom broju, zahteva od Vas da navedete onaj redni broj procedure, ili funkcije pod kojim je izvezena iz DLL-a:

```
procedure SomeOrdinalProcedure;  external 'testdll.dll' index 99;
```

U ovom slučaju, ja uvozim proceduru koja je izvezena iz DLL-a sa indeksom `99`. Proceduru mogu nazvati kako god želim u pozivajućem programu, dokle god je indeks isti kao i redni broj procedure pod kojim je izvezena iz DLL-a. Ovo je moguće, pošto procedura nije izvezena po imenu nego po rednom broju.

Treći način, korišćenjem promene imena, mi dozvoljava da uvezem proceduru po njenom stvarnom imenu ali mi, takođe, dozvoljava da proceduri dam novo ime u pozivajućem programu. To izgleda ovako:

```
procedure CoolProcedure;
  external 'testdll.dll' name 'DoSomethingReallyCool';
```

Sada uvozim proceduru pod imenom `DoSomethingReallyCool` i menjam joj ime u `CoolProcedure`.

Od ova tri načina, prvi način (uvoženje po stvarnom imenu) se najčešće koristi.

Očigledno je da je ceo trik kod pravljenja i korišćenja DLL-ova u ispravnom izvoženju i uvoženju procedura i funkcija. Dokle god Vam je potrebna fleksibilnost koju pružaju DLL-ovi, trebali biste da koristite statičko učitavanje.

Pozivanje funkcija i procedura korišćenjem dinamičkog učitavanja

Pozivanje funkcija i procedura iz DLL-a koji je dinamički učitan nije mnogo zabavno. Morate deklarisati pokazivač na funkciju, ili proceduru u DLL-u, a pokazivači na funkcije mogu biti zbunjujući. Ilustracije radi, recimo da imate proceduru u DLL-u koja se zove `SayHello` (`SayHello` dolazi do izražaja u ovom odeljku). Ona bi trebala da izgleda ovako u izvornom kodu DLL-a:

```
procedure SayHello(AForm : TForm);
begin
  MessageBox(AForm.Handle, 'Hello From a DLL!',
    'DLL Message Box', MB_OK or MB_ICONEXCLAMATION);
end;
```

Da biste pozvali ovu proceduru iz svog programa, morate deklarisati tip koji opisuje proceduru:

```
type
  TSayHello = procedure(AForm : TForm);
```

Kada ste to uradili, morate učitati DLL, upotrebiti `GetProcAddress` da biste dobili pokazivač na proceduru, pozvati proceduru i, konačno, izbaciti DLL iz memorije. Ovako izgleda cela operacija:

```
var
  DLLInstance : THandle;
  SayHello : TSayHello;
begin
  { Load the DLL. }
  DLLInstance := LoadLibrary('testdll.dll');
  { Get the address of the procedure. }
  @SayHello := GetProcAddress(DLLInstance, 'SayHello');
  { Call the procedure. }
  SayHello(Self);
  { Unload the DLL. }
  FreeLibrary(DLLInstance);
end;
```

Kao što sam rekao, dinamičko učitavanje DLL-a zahteva malo više rada. Ipak, kada je potrebno da učitavate DLL u toku izvršavanja programa, ovo je način da se to uradi. Primetićete da je ovaj kod malo skraćen zbog jednostavnosti. Gotovo uvek ćete dodati i deo koda koji se bavi proverom grešaka da biste bili sigurni da je DLL ispravno učitan i da `GetProcAddress` vraća ispravnu adresu. Kompletan listing, sa kodom za proveru grešaka, izgleda ovako:

```
procedure TForm1.DynamicLoadBtnClick(Sender: TObject);
type
  TSayHello = procedure(AForm : TForm);
var
```

```
  DLLInstance : THandle;
  SayHello : TSayHello;
begin
  DLLInstance := LoadLibrary('testdll.dll');
  if DLLInstance = 0 then begin
    MessageDlg('Unable to load DLL.', mtError, [mbOK], 0);
    Exit;
  end;
  @SayHello := GetProcAddress(DLLInstance, 'SayHello');
  if @SayHello <> nil then
    SayHello(Self)
  else
    MessageDlg('Unable to locate procedure.', mtError, [mbOK], 0);
  FreeLibrary(DLLInstance);
end;
```

Kao što možete videti, verovatno nećete koristiti dinamičko učitavanje, dokle god Vam ne bude apsolutno neophodno.

Pravljenje DLL projekta sa Object Repository-jem

Pravljenje DLL-a u Delphi-ju se vrši kroz Object Repository. (Sam Object Repository je pokriven u danu 8 "Pravljenje programa u Delphi-ju".). Da biste napravili DLL projekat postupite na sledeći način:

1. Izaberite File➥New da biste prikazali Object Repository.
2. Dva puta kliknite na DLL ikonu.

Mislili ste da će biti komplikovanije? Delphi će napraviti DLL projekat i prikazati editor koda. Fajl u editoru izgleda ovako:

```
library Project2;


{ Important note about DLL memory management: ShareMem must be the
  first unit in your library's USES clause AND your project's (select
  View-Project Source) USES clause if your DLL exports any procedures
or
  functions that pass strings as parameters or function results. This
  applies to all strings passed to and from your DLL--even those that
  are nested in records and classes. ShareMem is the interface unit
to
  the DELPHIMM.DLL shared memory manager, which must be deployed
along
  with your DLL. To avoid using DELPHIMM.DLL, pass string information
  using PChar or ShortString parameters. }

uses
  SysUtils,
```

nastavlja se

```
  Classes;                                                    nastavak

begin
end.
```

Sada možete početi sa dodavanjem koda u DLL. Obavezno dodajte svaku proceduru i funkciju koju želite da izvezete u `exports` deo. Napišite sve samostalne funkcije, ili procedure koje će Va biti potrebne u DLL-u. Kada ste završili sa dodavanjem koda izaberite Compile, ili Build iz Project menija da biste generisali sam DLL.

Komentarisanje celine DLL-a

Želim da Vam za trenutak pojasnim veliki komentar koji se nalazi na početku celine DLL-a. Ova poruka Vam govori da ukoliko Vaš DLL izvozi funkcije, ili procedure koje kao parametar primaju dugački string, ili vraćaju dugački string, morate uraditi sledeće:

- Da stavite `ShareMem` na početak uses liste i u DLL-u i u glavnoj celini pozivajućeg programa. Obavezno stavite `ShareMem` pre svih drugih celina u uses listi.
- Da isporučite `Borlndmm.dll` fajl sa svojim DLL-om. Primetite da sam napisao `Borlndmm.dll`, a ne Delphimm.dll, kao što piše u komentaru celine DLL-a. Ljudi iz Borland-a su promenili ime ovog fajla (menadžera memorije), ali su zaboravili da izmene komentare koji se generišu kada pravite novu DLL celinu. Ipak, ovaj komentar ima smisla, premda se uz Delphi 4 isporučuju i `Delphimm.dll` i `Borlndmm.dll`.

Da biste ovo izbegli, budite sigurni da Vaše procedure i funkcije u DLL-u nemaju ni jedan parametar tipa dugački string i da Vaše funkcije ne vraćaju podatak tipa dugački string. Umesto dugačkog stringa možete koristiti `PChar`, ili kratki string. Na primer, umesto korišćenja:

```
procedure MyProcedure(var S : string);
begin
  { Procedure code here. }
end;
```

koristite:

```
procedure MyFunction(S : PChar);
begin
  { Procedure code here. }
end;
```

Lako je izbeći pomenutu situaciju, tako da ne biste trebali da isporučujete i `Borlndmm.dll`. Jednostavno, treba da budete svesni ograničenja koja postoje kada se koriste dugački stringovi sa DLL procedurama i funkcijama. Zapazite i to da možete slobodno koristiti dugačke stringove sa funkcijama i procedurama koje su lokalne za DLL. Ograničenje se odnosi samo na procedure i funkcije koje su izvezene iz DLL-a.

NAPOMENA Ja uvek obrišem komentar koji se odnosi na `Borlndmm.dll`, kada napravim novu DLL celinu. Vi ih, naravno, možete ostaviti u izvornom kodu svog DLL-a. Pošto jednom razumete to što Vam komentar govori, ni Vama više neće biti potreban, tako da ga možete slobodno obrisati.

Sledeća tri listinga sadrže kod koji ilustruje koncept koji je obrađen do sada. Listing 19.3 sadrži DLL koji će se statički učitavati iz pozivajućeg programa. Listing 19.4 sadrži DLL koji će se dinamički učitavati iz pozivajućeg programa i koji sadrži `DLLProc`. Konačno, listing 19.5 sadrži program koji poziva ova dva DLL-a. Ovaj program sadrži formu na kojoj se nalaze četiri tastera pomoću kojih se pozivaju razne funkcije iz ova dva DLL-a. Izvorni kod iz knjige (nalazi se na `http://www.mcp.com/info`) sadrži primere za projekte koji se nalaze u ova tri listinga.

Listing 19.3: `TestDll.dp`

```
library TestDLL;

uses
  SysUtils,
  Classes,
  Forms,
  Windows;

procedure SayHello(AForm : TForm);
begin
  MessageBox(AForm.Handle, 'Hello From a DLL!',
    'DLL Message Box', MB_OK or MB_ICONEXCLAMATION);
end;

procedure DoSomething;
begin
  MessageBox(0, 'This procedure was exported by ordinal.',
    'DLL Message Box', MB_OK or MB_ICONEXCLAMATION);
end;

procedure DoSomethingReallyCool;
begin
  MessageBox(0, 'Something really cool.',
    'DLL Message Box', MB_OK or MB_ICONEXCLAMATION);
end;

exports
  SayHello,
  DoSomething index 99,
  DoSomethingReallyCool;

begin
end.
```

Listing 19.4: DynLoad.dpr

```
library TestDLL;

uses
  SysUtils,
  Classes,
  Forms,
  Windows;

procedure MyDLLProc(Reason: Integer);
begin
  if Reason = DLL_PROCESS_DETACH then
    { DLL is unloading. Cleanup code here. }
  MessageBox(0, 'DLL is unloading!',
    'DLL Message', MB_OK or MB_ICONEXCLAMATION);
end;

procedure SayHelloDyn(AForm : TForm);
begin
  MessageBox(AForm.Handle, 'Hello From a DLL!' + #13 +
    'This DLL was loaded dynamically',
    'DLL Message', MB_OK or MB_ICONEXCLAMATION);
end;

exports
  SayHelloDyn;

begin
  DLLProc := @MyDLLProc;
end.
```

Listing 19.5: CallDllU.pas

```
unit CallDLLU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    HelloBtn: TButton;
    OrdBtn: TButton;
    DynamicLoadBtn: TButton;
    NamedBtn: TButton;
    procedure HelloBtnClick(Sender: TObject);
    procedure OrdBtnClick(Sender: TObject);
    procedure DynamicLoadBtnClick(Sender: TObject);
```

```
    procedure NamedBtnClick(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;

var
  Form1: TForm1;

{ A procedure imported by name. }
procedure SayHello(AForm : TForm);
  external 'testdll.dll';

{ A procedure imported by ordinal. }
procedure OrdinalProcedure;
  external 'testdll.dll' index 99;

{ A procedure imported and renamed. }
procedure CoolProcedure;
  external 'testdll.dll' name 'DoSomethingReallyCool';

implementation

{$R *.DFM}

procedure TForm1.HelloBtnClick(Sender: TObject);
begin
  SayHello(Self);
end;

procedure TForm1.OrdBtnClick(Sender: TObject);
begin
  OrdinalProcedure;
end;

procedure TForm1.NamedBtnClick(Sender: TObject);
begin
  CoolProcedure;
end;

procedure TForm1.DynamicLoadBtnClick(Sender: TObject);
type
  TSayHello = procedure(AForm : TForm);
var
  DLLInstance : THandle;
  SayHello    : TSayHello;
begin
  { Load the DLL. }
  DLLInstance := LoadLibrary('DynLoad.dll');
```

nastavlja se

Listing 19.5: `CallDllU.pas` *nastavak*

```
  { If loading fails then bail out. }
  if DLLInstance = 0 then begin
    MessageDlg('Unable to load DLL.', mtError, [mbOK], 0);
    Exit;
  end;
  { Assign the procedure pointer. }
  @SayHello := GetProcAddress(DLLInstance, 'SayHelloDyn');
  { If the procedure was found then call it. }
  if @SayHello <> nil then
    SayHello(Self)
  else
    MessageDlg('Unable to locate procedure.', mtError, [mbOK], 0);
  { Unload the DLL. }
  FreeLibrary(DLLInstance);
end;

end.
```

Korišćenje formi u DLL-ovima

Vaši DLL-ovi, pored koda, mogu sadržati i forme. Proces pravljenja formi koje će se nalaziti u DLL-ovima se ne razlikuje mnogo od uobičajenog procesa pravljenja formi. Prvo ću objasniti kako treba pisati DLL koji će sadržati formu. Posle toga, govoriću o specijalnom slučaju korišćenja MDI (engl. Multiple Document Interface) formi u DLL-u.

Pravljenje DLL-a koji sadrži formu

Pravljenje DLL-a koji sadrži forme nije mnogo teže od pravljenja DLL-a koji sadrži samo kod. Ja verujem u učenje na osnovu primera, tako da ćete u ovom odeljku generisati DLL koji sadrži formu. Uradite sledeće:

1. Napravite novi DLL projekat (obrišite komentare sa početka DLL-a, ukoliko želite). Sačuvajte ovaj projekat pod imenom `MyForms.dpr`.
2. Izaberite File➥New Form da biste napravili novu formu za DLL projekat. Postavite koje god želite komponente na formu.
3. Izmenite `Name` osobinu nove forme u `DLLForm`. Sačuvajte celinu forme pod imenom `DLLFormU.pas`.
4. Vratite se na izvorni kod celine DLL-a. Dodajte funkciju pod imenom `ShowForm` koja kreira i prikazuje formu. Koristite sledeći kod:

```
function ShowForm : Integer; stdcall;
var
  Form : TDLLForm;
begin
  Form := TDLLForm.Create(Application);

  Form.Free;
end;
```

5. Dodajte `exports` deo u DLL i izvezite funkciju `ShowForm`. `Exports` deo bi trebao da izgleda ovako:

```
exports
  ShowForm;
```

6. Izaberite Project➥Build MyForms da biste generisali DLL. Sačuvajte DLL projekat.

To je sve što se tiče pravljenja DLL-a. Primetite da je `ShowForm` funkcija deklarisana sa ključnom reči `stdcall`. Ova ključna reč govori prevodiocu da izveze funkciju koristeći standardnu konvenciju pozivanja. Izvoženje ove funkcije kao `stdcall` omogućava korišćenje ovog DLL-a i iz drugih razvojnih okruženja (a ne samo iz Delphi-ja).

NOVI TERMIN Konvencija pozivanja *(engl. calling convention)* određuje na koji način prevodilac treba da prenosi parametre prilikom poziva procedure, ili funkcije. Pet osnovnih konvencija pozivanja su: `stdcall`, `cdecl`, `pascal`, `register` i `safecall`. Pogledajte temu "Calling Conventions" u Delphi-jevom help-u da biste naučili više o konvencijama pozivanja.

Primetite, takođe, da je povratna vrednost DLL funkcije `ShowForm` u stvari vrednost koju vraća funkcija `ShowModal`. Ovo Vam dozvoljava da vratite informaciju o stanju forme pozivajućem programu. Listing 19.6 prikazuje kod DLL-a.

Listing 19.6: Kompletan DLL

```
library MyForms;

uses
  SysUtils,
  Classes,
  Forms,
  DLLFormU in 'DLLFormU.pas' {DLLForm};

function ShowForm : Integer; stdcall;
var
  Form : TDLLForm;
begin
  Form := TDLLForm.Create(Application);
  Result := Form.ShowModal;
```

nastavlja se

Listing 19.6: Kompletan DLL *nastavak*

```
  Form.Free;
end;

exports
  ShowForm;

begin
end.
```

Sada pozivajući program može da deklariše `ShowForm` funkciju i da je pozove. Listing 19.7 prikazuje Delphi program koji poziva `MyForms` DLL.

Listing 19.7: Program koji poziva `MyForms` DLL

```
unit TestAppU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TForm1 = class(TForm)
    Button1: TButton;
    procedure Button1Click(Sender: TObject);
  private
    { Private declarations }
  public
    { Public declarations }
  end;
```

Primetite da sam ponovo koristio ključnu reč `stdcall` prilikom deklarisanja funkcije u pozivajućem programu. Pošto je funkcija izvezena iz DLL-a korišćenjem `stdcall`, takođe mora biti i uvezena korišćenjem `stdcall`. Uvek morate uvoziti proceduru, ili funkciju iz DLL-a koristeći istu konvenciju pozivanja koju ste koristili prilikom izvoženja.

Ukoliko ćete koristiti DLL-ove, napravljene u Delphi-ju, samo iz Delphi programa, ne morate koristiti stdcall prilikom izvoženja funkcija i procedura. Ukoliko, ipak, postoji mogućnost da će se Vaš DLL pozivati i iz drugih programa trebali biste da izvozite svoje funkcije i procedure korišćenjem `stdcall`.

Pozivanje MDI forme iz DLL-a

DLL koji sadrži MDI child formu je specijalan slučaj. (MDI forme su objašnjene u danu 4, "Istraživanje Delphi-jevog IDE-a.") Recimo da imate Delphi program čija je glavna forma u stvari MDI forma. Ukoliko pokušate da koristite MDI child formu koja se nalazi u DLL-u, dobićete izuzetak od VCL-a koji kaže: "No `MDI forms are currently active`". Šta? Ali Vi imate MDI formu u svom programu. VCL ne misli tako. Sledi objašnjenje onoga što se desilo.

Kada pokušate da prikažete svoju MDI child formu, VCL proverava da li je `MainForm` osobine `Application` objekta u redu. Ukoliko `MainForm` osobina nije u redu, dolazi do izuzetka. Dakle, šta nije u redu? `MainForm` je u redu, zar ne? Problem je u tome što i sam DLL ima `Application` objekat i VCL proverava `MainForm` osobinu tog objekta, a ne `Application` objekta glavnog programa. Pošto DLL nema glavnu formu, ova provera će uvek biti neuspešna.

Rešenje ovog problema leži u dodeljivanju `Application` objekta glavnog programa `Application` objektu DLL-a. Naravno, ovo će funkcionisati samo ako je pozivajući program VCL program. Ali, to nije sve. Pre nego što se DLL izbaci iz memorije, morate vratiti njegov `Application` objekat u početno stanje. Ovo je potrebno da bi menadžer memorije mogao da izvrši čišćenje memorijskog prostora koji je DLL koristio. To znači da morate DLL-ov `Application` objekat dodeliti nekoj globalnoj promenljivoj u DLL-u kako biste, kasnije, mogli da ga vratite u početno stanje.

1. Napravite globalni `TApplication` pokazivač u DLL-u.
2. Sačuvajte DLL-ov `Application` objekat u globalnom `TApplication` pokazivaču.
3. Dodelite `Application` objekat pozivajućeg programa DLL-ovom `Application` objektu.
4 .Kreirajte i prikažite MDI child formu.
5. Postavite DLL-ov `Application` objekat na početnu vrednost, pre nego što se DLL izbaci iz memorije.

Prvi korak je jednostavan. Samo postavite sledeći kod blizu vrha izvornog koda Vaše DLL celine:

```
var
  DllApp : TApplication
```

Obavezno postavite var ključnu reč ispod uses liste u izvornom kodu DLL celine.

Sledeće, napravite proceduru koja će dodeliti novu vrednost `TApplication-u` i kreirati formu. Ta procedura bi trebala da izgleda otprilike ovako:

```
procedure ShowMDIChild(MainApp : TApplication);
var
  Child : TMDIChild;
begin
  if not Assigned(DllApp) then begin
    DllApp := Application;
    Application := MainApp;
  end;
  Child := TMDIChild.Create(Application.MainForm);
  Child.Show;
end;
```

Zastanite za momenat i analizirajte ovaj kod. Kada pozovete ovu proceduru, prenećete joj Application objekat pozivajućeg programa. Ukoliko `DllApp` pokazivaču još uvek nije ništa dodeljeno, dodeljujete mu vrednost DLL-ovog `Application` objekta. Zatim dodeljujete vrednost `Application` objekta pozivajućeg programa DLL-ovom `Application` objektu. Gornja provera obezbeđuje da se `Application` objekat postavlja samo jednom. Posle toga se kreira MDI child forma čiji vlasnik postaje glavna forma pozivajućeg programa (`MainForm` osobine Application objekta pozivajućeg programa). Konačno, prikazuje se sama forma.

Sledeće što je potrebno uraditi je vraćanje DLL-ovog Application pokazivača, pre izbacivanja DLL-a iz memorije. Možete koristiti `DLLProc` da biste vratili vrednost DLL-ovog Application pokazivača:

```
procedure MyDLLProc(Reason: Integer);
begin
  if Reason = DLL_PROCESS_DETACH then
    { DLL is unloading. Restore the Application pointer. }
    if Assigned(DllApp) then
      Application := DllApp;
end;
```

Zapamtite da ste DLL-ov `Application` pokazivač sačuvali ranije i da ga sada samo vraćate.

Kao što možete videti, postavljanje MDI child forme iz DLL-a na ekran zahteva malo dodatnog rada, ali je izvodljivo. Kod iz ove knjige sadrži projekat programa pod imenom `MDIApp` i DLL projekat `MyForms`. Ova dva projekta ilustruju korišćenje MDI forme iz DLL-a.

Prikazivanje DLL forme iz programa koji nije pisan u Delphi-ju

Pozivanje forme iz programa koji nije pisan pomoću VCL-a zahteva nešto drugačiji pristup od prethodnog. Potrebno je da napravite samostalnu funkciju u DLL-u koju će pozivati pozivajući program. Ovu funkciju će moći da pozove svaki program, samo ako je deklarišete preko `stdcall`. U okviru tela ove funkcije kreirate formu i izvršavate je. Funkcija bi trebala da izgleda ovako:

```
function ShowForm : Integer; stdcall;
var
  Form : TMyForm;
begin
  Form := TMyForm.Create(Application);
  Result := Form.ShowModal;
  Form.Free;
end;
```

Primetite da se `Application` prenosi iz forme koja je roditeljska ovoj formi. Ovo je DLL-ov `Application` objekat i ponaša se kao vlasnik forme. Iako ja eksplicitno izbacujem iz memorije `TMyForm` objekat, to nije neophodno, premda će DLL-ov `Application` objekat sam izbaciti formu iz memorije ukoliko ja to zaboravim.

Korišćenje resursa iz DLL-ova

Ponekad je korisno imati resurse u DLL-ovima. Već sam govorio o internacionalizaciji i o tome kako možete koristiti DLL-ove da biste Vaš program jednostavnije preveli na druge jezike. Recimo da Vaš program ima prozor sa instrukcijama za korisnika i da se te instrukcije nalaze u okviru pet stringova u DLL-u. Ovi stringovi mogu imati imena: `IDS_INSTRUCTION1`, `IDS_INSTRUCTION2`, i tako dalje. Mogli biste da učitate i prikažete stringove na sledeći način:

```
LoadString(DllInstance, IDS_INSTRUCTION1, Buff, SizeOf(Buff));
InstructionLabel1.Caption := Buff;
```

Prvi parametar `LoadString` funkcije sadrži pokazivač na kopiju DLL-a sa stringovima u memoriji. Drugi parametar sadrži ID broj resursa koga je potrebno učitati. Mogli biste napraviti DLL-ove u kojima se nalaze string resursi na više različitih jezika koje biste jednostavno učitavali na osnovu korisnikovog izbora. Kod bi mogao da izgleda ovako:

```
var
  DLLName : String;
begin
  case Language of
    laFrench : DllName := 'french.dll';
    laGerman : DllName := 'german.dll';;
    laSpanish : DllName := 'spanish.dll';
    laEnglish : DllName := 'english.dll';
  end;
  DllInstance := LoadLibrary(PChar(dllName));
end;
```

Sve što treba da uradite je da učitate ispravni DLL, ostatak koda ostaje neizmenjen (naravno, pod pretpostavkom da ste za stringove koristili iste identifikatore u svakom od DLL-ova). Ovo je samo jedan primer korišćenja resursa iz DLL-ova. Pronaćićete mnogo primena ove tehnike.

Pravljenje DLL-a sa resursima

Možete napraviti DLL koji sadrži samo resurse, ili možete mešati resurse i kod u okviru istog DLL-a. Postavljanje resursa u DLL je praktično isto kao i postavljanje resursa u program. Da biste napravili DLL sa resursima, napravite novi DLL projekat, a zatim dodajte sledeću liniju u DLL da biste povezali resurse:

```
{$R RESOURC.RES}
```

To je sve što je potrebno da bi se napravio DLL sa resursima. Pravljenje fajlova sa resursima sam objasnio u danu 8, tako da se na to neću ponovo vraćati.

Korišćenje DLL-a sa resursima

Pre nego što budete mogli da pristupite resursima koji se nalaze u DLL-u morate imati pokazivač na kopiju DLL-a u memoriji. Ukoliko Vaš DLL sadrži samo resurse, učitavaćete ga dinamički. Ukoliko Vaš DLL sadrži i kod i resurse, radije ćete ga učitavati statički. Čak i ukoliko budete statički učitavali DLL, moraćete da pozivate funkciju `LoadLibrary`, kako biste dobili pokazivač na kopiju DLL-a u memoriji:

```
DllInstance := LoadLibrary('resource.dll');
```

Sada možete koristiti ovaj pokazivač kad god je to potrebno. Sledeći kod učitava bit-mapirani resurs koji se nalazi u DLL-u u `Image` komponentu:

```
procedure TMainForm.FormCreate(Sender: TObject);
begin
  DLLInstance := LoadLibrary('resource.dll');
  if DLLInstance <> 0 then begin
    Image.Picture.Bitmap.
      LoadFromResourceName(DLLInstance, 'ID_BITMAP1');
    FreeLibrary(DLLInstance);
  end else
    MessageDlg('Error loading resource DLL.',
      mtError, [mbOk], 0);
end;
```

U stvari, i nema više šta da se kaže. Vi znate da možete učitati DLL sa resursima, bilo statički, bilo dinamički. Koji god način da izaberete, morate pozvati `LoadLibrary` funkciju da biste dobili pokazivač na kopiju DLL-a u memoriji. Nemojte zaboraviti da pozovete `FreeLibrary` kada završite sa korišćenjem DLL-a, ili pre nego što Vaš program završi sa radom.

Korišćenje dinamičkog učitavanja ima tu prednost što dozvoljava Vašem programu da se brže učita. U većini slučajeva ćete koristiti DLL sa resursima, samo onda kada Vam je potreban i izbacivaćete ga iz memorije, kada Vam više ne bude potreban. To znači da će Vaš program zauzimati manje memorije nego kada se resursi nalaze u samom fajlu programa. Mana je možda što će Vaši korisnici primetiti malu pauzu u radu programa, dok se DLL sa resursima učitava. Predvidite ovakve situacije i učitavajte DLL sa resursima, onda kada će usporenje biti najmanje primetno.

Sećate li se `JumpingJack` programa iz osmog dana? Izvorni kod iz ove knjige sadrži verziju ovog programa koji učitava bit- mape, zvuke i string resurse iz DLL-a. Pogledajte taj program kao primer korišćenja resursa iz DLL-a.

Zaključak

Korišćenje DLL-ova i nije tako teško kao što se čini na prvi pogled. DLL-ovi su odličan način za ponovno korišćenje postojećeg koda. Pošto napravite DLL, može ga koristiti svaki program koji zahteva funkcionalnost koju obezbeđuje kod iz tog DLL-a. Postavljanje VCL formi u DLL i kasnije korišćenje tih formi iz programa koji nisu pravljeni u Delphi-ju je moćna osobina. To znači da možete praviti forme koje ostali mogu pozivati iz praktično bilo kog tipa Windows programa, bez obzira da li je taj program pisan pomoću OWL-a, MFC-a ili u čistom C-u, Visual Basic-u i tako dalje. Korišćenje resursa u DLL-ovima je efikasno ukoliko imate puno resursa u svom programu i ukoliko želite da kontrolišete kada i gde se ti resursi učitavaju.

Radionica

Radionica sadrži test pitanja koja Vam pomažu da učvrstite svoje razumevanje izložene materije i vežbe koje Vam pomažu da steknete iskustvo u onome što ste naučili. Možete pronaći odgovore na test pitanja u Dodatku A "Odgovori na test pitanja".

Pitanja i odgovori

P **Imam mali program i ne vidim potrebu za korišćenjem DLL-ova. Da li bih trebao da drugačije struktuiram program?**

O Verovatno ne. DLL-ovi obično nisu korisni za male programe. Kada god imate klase koje možete ponovo upotrebiti, možete takođe koristiti i DLL-ove, ali nemojte menjati svoju osnovnu ideju samo da biste koristili DLL-ove i u malim programima.

P **Pokušavam da pozovem funkciju iz DLL-a, ali stalno dobijam grešku vezanu za deklaraciju funkcije. Ona glasi: "`Unsatisfied forward or external declaration`". Gde grešim?**

O Zaboravili ste da stvarite ključnu reč `external` u deklaraciju funkcije.

P **Ova `GetProcAddress` funkcija je malo čudna. Da li moram da je koristim da bih pozivao funkcije i procedure iz mog DLL-a?**

O Ne. Jednostavno koristite statičko učitavanje i nećete morati da koristite `LoadLibrary`, `GetProcAddress` i `FreeLibrary` funkcije.

P **Moj program se ispravno prevodi, ali dobijam grešku za vreme izvršavanja koja glasi: "`procedure entry point XXX cannot be found in the dynamic link library XXX.DLL`". Šta nije u redu?**

O Postoje dva moguća uzroka ove greške. Prvi je pogrešno napisano ime funkcije u deklaraciji funkcije u pozivajućem programu. Drugi mogući uzrok je da ste možda zaboravili da unesete ime te funkcije u exports deo DLL-a.

P **Imam formu koja se nalazi u DLL-u i moj pozivajući program je u stvari Delphi program. DLL je poprilično veliki. Postoji li način da ga nekako smanjim?**

O Na žalost, ne. Ovde moć programiranja u Delphi-ju ponovo radi malo protiv Vas. I DLL i pozivajući program sadrže po malo istog VCL koda. Drugim rečima, isti kod postoji i u `.exe` i u `.dll` fajlu. Morate prihvatiti činjenicu da će DLL biti veći, ukoliko se u njemu nalaze forme. Problem možete rešiti korišćenjem paketa u toku izvršavanja. Tada i pozivajući program i DLL mogu koristiti kod iz ovih paketa.

P **Imam puno fajlova sa zvučnim zapisima koje koristi moj program. Trenutno se svi nalaze u odvojenim fajlovima, ali bih ja voleo da ih stavim na jedno mesto. Da li je DLL sa resursima dobra ideja?**

O Apsolutno. Jedina mana je što, ukoliko Vam je potreban jedan zvučni zapis, morate učitavati ceo DLL. Ipak, korišćenjem dinamičkog učitavanja možete učitavati i izbacivati DLL iz memorije po potrebi. `PlaySound` funkcija olakšava puštanje zvučnih zapisa iz DLL-ova.

P **Živim u zemlji u kojoj se govori francuski. Zašto bih se zamarao sa internacionalizacijom mog programa?**

O To u potpunosti zavisi od Vašeg ciljnog tržišta. Ukoliko ste sigurni da će se Vaš program koristiti samo u zemljama u kojima se govori franucski, ne morate se brinuti o internacionalizaciji. Ukoliko, ipak, postoji i najmanja mogućnost da će se program koristiti i u drugim zamljama, trebali biste od starta razmišljati o internacionalnoj podršci. Mnogo je lakše napraviti pravu stvar na početku, nego se kasnije vraćati i ispravljati je.

Kviz

1. Kako se učitava DLL korišćenjem statičkog učitavanja?
2. Kako se učitava DLL korišćenjem dinamičkog učitavanja?
3. Kako se poziva procedura, ili funkcija iz DLL-a koji je statički učitan?
4. Šta treba da uradite da biste bili sigurni da će procedure i funkcije iz Vašeg DLL-a moći da se pozivaju i izvan DLL-a?

5. U slučaju da je DLL dinamički učitan, da li ga možete izbaciti iz memorije bilo kad, ili samo onda kada pozivajući program prestaje sa radom?
6. Šta treba da uradite da biste prikazali Delphi-jevu formu koja se nalazi u DLL-u u nekom programu koji nije pisan u Delphi-ju.
7. Kako se zove ključna reč koja se koristi da bi se deklarisale procedure i funkcije koje su uvežene iz DLL-a?
8. Kako dodajete resurse u DLL?
9. Da li DLL sa resursima mora sadržati i kod?
10. Da li se DLL koji sadrži resurse može statički učitavati (tj. kad pozivajući program počne sa radom)?

Vežbe

1. Napravite DLL koji sadrži proceduru koja, kada se pozove, prikazuje prozor sa porukom (message-box).
2. Napravite pozivajući program koji će pozvati DLL koji ste napravili u prvoj vežbi.
3. Napravite DLL koji sadrži formu i pozivajući program koji će prikazati tu formu.
4. Napravite DLL koji sadrži dva bit-mapirana resursa.
5. Napravite program koji će na zahtev korisnika prikazati bilo koju od bit mapa iz DLL-a koje ste napravili u četvrtoj vežbi. (Savet: Koristite `TImage` komponentu i njenu `LoadFormResourceId` metodu.)
6. **Posebna vežba**: Napravite pet DLL-ova od kojih svaki sadrži iste stringova, ali na pet različitih jezika.
7. **Posebna vežba**: Napravite pozivajući program koji prikazuje strngove koje ste napravili u šestoj vežbi. Dozvolite korisniku da izabere jezik koji će se koristiti u programu. Prikažite stringove u izabranom jeziku.

Dan 20

Pravljenje komponenti

Delphi sadrži veliki broj komponenti koje možete koristiti u svojim programima. Ove komponente pokrivaju najveći broj Windows-ovih kontrola, ali takođe i ponešto što nije podržano samim Windows-om. Ipak, moguće je da ćete želeti da sami napravite neke komponente, kako biste mogli da uradite nešto što standardne komponente ne mogu. Pravljenje komponenti se sastoji u sledećem:

1. Koristite New Component dijalog-prozor da biste započeli proces.
2. Dodajte osobine, metode i događaje klasi nove komponente.
3. Istestirajte komponentu.
4. Postavite komponentu na paletu sa komponentama.

Danas ćete naučiti kako da pravite komponente. Kao kod svih ostalih stvari koje su relativno teške, ne bi bilo loše da ovaj dan pređete nekoliko puta. Ovom tehnikom ćete savladati pravljenje `TFlashingLabel` komponente. `TFlashingLabel` je klasična `Label` komponenta čiji tekst svetli. Na kraju dana ćete znati sve što je potrebno za pravljenje jednostavnih komponenti.

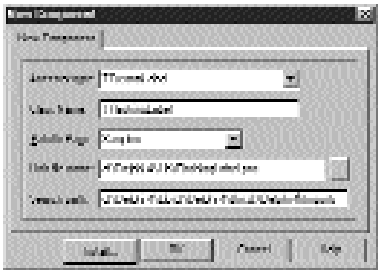
Pravljenje nove komponente

Pravljenje komponenti zahteva viši nivo poznavanja programiranja od onoga koji ste koristili do sada. Prvo, morate da napravite klasu za svoju novu komponentu. Klasa mora biti napravljena tako da pristup nekoj od njenih komponenti bude moguć kroz Object Inspector, dok će se druge koristiti samo kroz kod, u toku izvršavanja

programa. Dalje, verovatno ćete imati i metode u okviru komponente. Neke će biti privatne za komponentu, dok će ostalima moći da se pristupi i izvan same komponente. Konačno, možda ćete morati da realizujete i događaje. Očigledno, potrebno je malo više rada. Na žalost, Delphi-jevo programsko okruženje Vam ovde neće mnogo pomoći. Pravljenje komponenti je čisto programiranje.

New Component dijalog-prozor

New Component dijalog-prozor Vam pruža pomoć na samom početku pravljenja komponente. Da biste ga aktivirali, izaberite File➥New da biste prikazali Object Repository, a zatim dva puta kliknite na Component ikonu. Slika 20.1 prikazuje New Component dijalog-prozor koji vidite prilikom pravljenja nove komponente.



Slika 20.1
New Component dijalog-prozor

U `Ancestor type` polje upisujete ime roditeljske klase za klasu Vaše nove komponente. U Ancestor type kombo-listi listi su prikazane klase svih instaliranih komponenti. Izaberite klasu one komponente koja je najbliža po funkcionalnosti komponenti koju želite da napravite.

Na primer, `FlashingLabel` komponenta je klasična Label komponenta koja svetli. U ovom slučaju, `Label` komponenta sadrži sve što Vam je potrebno za početak. Dakle, možete koristiti `TCustomLabel` kao roditeljsku klasu. Ako biste, recimo, želeli da napravite komponentu koja pravi prečice na Windows desktop-u, morali biste za roditeljsku klasu da izaberete `TComponent`, premda u VCL-u ne postoji ni jedna komponenta sa sličnom funkcijom. TComponent je osnovna roditeljska klasa svim komponentama.

Delphi sadrži nekoliko klasa koje možete koristiti kao roditeljske za Vašu komponentu. Imena ovih klasa počinju sa `TCustom`. Na primer, roditeljska klasa za `TLabel` je `TCustomLabel`. Možete nasleđivati iz jedne od ovih klasa, kada god pravite novu komponentu. Ove klase već sadrže neke osobine koje će Vam najčešće trebati, ali one nisu objavljene (objavljene osobine su one kojima možete pristupiti iz Object Inspector-a). Sve što treba da uradite da biste ove osobine proglasili objavljenima je da promenite njihovu deklaraciju u published delu deklaracije Vaše klase. Ovo je važno, premda se jednom objavljena komponenta više ne može zaštititi. Nasleđivanje iz jedne od ovih klasa Vam pruža mogućnost da tačno izaberete koje osobine želite da objavite.

Kada izvedete novu komponentu iz već postojeće, koristite osobinu Object Pascal-a koja se naziva *nasleđivanje*. Prošlo je nekoliko dana od kada sam govorio o nasleđivanju, pa biste trebali da se vratite na dan 3 "Klase i objektno orijentisano programiranje", ukoliko želite da se podsetite. Nasleđivanje iz komponente u stvari znači da preuzimate sve ono što komponenta već ima, pri čemu dodajete ono što Vam je potrebno. Klasa iz koje izvodite je roditeljska klasa, a nova klasa je izvedena klasa. U prethodnom primeru, `TCustomLabel` je roditeljska klasa, dok će `TFlashingLabel` biti izvedena klasa.

Pošto ste odredili roditeljsku klasu, unesite ime Vaše nove komponente u `Class Name` polje. Ime klase bi trebalo da počinje sa *T* i trebalo bi da opisuje šta klasa predstavlja. Komponenta koju danas pravite će se zvati `TFlashingLabel` (uskoro ćete početi sa pravljenjem komponente). Ukoliko izaberete ime već postojeće klase, New Component dijalog-prozor će prijaviti poruku o grešci kada kliknete na OK taster. Moraćete da izaberete jedinstveno ime klase, pre nego što budete mogli da nastavite dalje.

Ne morate počinjati ime klase sa T. Ipak, to je uobičajeno za sve Borland-ove klase. (Korišćenje T na početku imena Borland-ovih klasa je tradicija koja postoji još od vremena Turbo Pascal-a. Ovakav način imenovanja klasa se koristio u Turbo Vision-u i OWL-u, a koristi se i sada, u VCL-u.) Neki ljudi koriste T kada izvode klasu iz nekih Borland-ovih klasa, ali ne i kada prave nezavisnu klasu. Izbor je na Vama.

Ljudi koji se profesionalno bave pravljenjem komponenti su odavno naučili da treba da daju jedinstvena imena klasama. Zamislite, recimo, da dva proizvođača komponenti daju svojim komponentama ime `TFancyLabel`. U kompaniji TurboPower, gde ja radim, imena naših Async Professional komponenti počinju sa `TApd`, Orpheus komponenti sa `TOr`, Abbrevia komponenti sa TAb i tako dalje. Iako ovo ne garantuje da se imena naših komponenti neće sudariti sa nekim drugim imenima, ovo je dobar način izbora imena.

`Palette Page` polje Vam dozvoljava da odredite stranicu na paleti sa komponentama na kojoj će se Vaša komponenta nalaziti. (Ikona komponente se neće pojaviti na stranici dokle god ne instalirate dizajn-paket za Vašu komponentu.) Možete izabrati postojeću stranicu na paleti, ili uneti ime nove stranice koju će Delphi napraviti za Vašu komponentu.

`Unit file` name polje se koristi za određivanje imena fajla u kome će se nalaziti izvorni kod komponente. Delphi automatski pravi fajl na osnovu imena komponente, ali Vi to možete promeniti unoseći novo ime u ovo polje. U `Search path` polje se unosi putanja po kojoj će se Delphi kretati u potrazi za komponentama. Uglavnom nećete morati da menjate sadržaj ovog polja.

Install taster se koristi za instaliranje nove komponente direktno u paket. Ne morate, za sada, brinuti o ovome, premda ćete koristiti podrazumevani Delphi-jev paket u koji se postavljaju razne komponente.

Pravljenje `FlashingLabel` komponente

Sada ste spremni da napravite `TFlashingLabel` komponentu. Kao što sam i ranije rekao, ova komponenta je klasična `Label` komponenta kod koje tekst svetli. Zapamtite to i počnite sa pravljenjem komponente:

1. Izaberite File➥New da biste aktivirali Object Repository.
2. Dva puta kliknite na Component ikonu u Object Repository-ju. Prikazuje se New Component dijalog-prozor.
3. Iz Ancestor type kombo-liste izaberite `TCustomLabel`. Ona će biti roditeljska klasa Vaše komponente.
4. Unesite `TFlashingLabel` u polje `Class Name`.
5. `Pallete Page` polje sadrži tekst `Samples`. To nemojte menjati. Nova komponenta će, pošto je instalirate, biti postavljena na Samples stranicu palete sa komponentama.
6. Kliknite na taster OK da biste zatvorili New Component dijalog-prozor. Pojavljuje se editor koda i prikazuje se nova celina sa izvornim kodom.
7. Sačuvajte celinu pod imenom `FlashingLabel.pas`

Listing 20.1: `FlashingLabel.pas`

```
unit FlashingLabel;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
  Controls, Forms, Dialogs, StdCtrls;

type
  TFlashingLabel = class(TCustomLabel)
  private
    { Private declarations }
  protected
    { Protected declarations }
  public
    { Public declarations }
  published
    { Published declarations }
  end;

procedure Register;

implementation
```

```
procedure Register;
begin
  RegisterComponents('Samples', [TFlashingLabel]);
end;

end.
```

Kao što možete videti, `TFlashingLabel` je izvedena iz klase `TCustomLabel`. Deklaracija klase je prazna, a dodate su samo ključne reči za pristup (`private`, `public`, `protected` i `published`). Kasnije ćete popuniti praznine, pošto naučite koji sve elementi čine komponentu.

Kao što možete videti, New Component dijalog-prozor Vam pomaže na početku tako što popunjava neke elemente celine Vaše nove komponente. Vi i dalje morate da obavite teži deo posla, ali su napisane barem Register procedura i deklaracija klase. Dozvolite mi da sada malo skrenem sa puta i kažem nešto o `Register` proceduri.

`Register` procedura

Registrovanje komponenti je obavezno da bi Delphi znao koje komponente se nalaze u biblioteci i na kojoj stranici bi svaka trebala da se pojavi. Tipična `Register` procedura izgleda ovako:

```
procedure Register;
begin
  RegisterComponents('Samples', [TMyComponent]);
end;
```

`Register` procedura poziva `RegisterComponents` da bi registrovala komponentu. `RegisterComponents` prima dva parametra. Prvi parametar je ime stranice na paleti sa komponentama na kojoj će se Vaša komponenta pojaviti neposredno po instalaciji. Drugi parametar je niz komponenti koje trebaju da budu registrovane. Ukoliko pravite biblioteku komponenti možete ih sve registrovati od jednom, jednostavnim dodavanjem imena svake od njih u niz. Na primer:

```
procedure Register;
begin
  RegisterComponents('Essentials 1',
    [TEsLabel, TEsScrollingMarquee, TEsCalendar,
     TEsCalculator, TEsDateEdit, TEsNumberEdit, TEsMenuButton,
     TEsColorComboBox, TEsTile, TEsGradient, TEsRollUp]);
end;
```

Ova `Register` procedura registruje 11 komponenti i postavlja ih na stranicu pod nazivom "Essentials 1". Uglavnom ćete pisati jednu po jednu komponentu, ali treba da znate i za mogućnost registrovanja više komponenti od jednom.

NAPOMENA `Register` procedura se takođe koristi i za registrovanje editora komponenti i editora osobina. Editori komponenti i osobina su specijalni editori, najčešće dijalog-prozori, koji dozvoljavaju izmenu komponente, ili njenih osobina u toku pravljenja programa. Ne bih želeo da sada komentarišem editore komponenti i osobina pošto ta tema prevazilazi okvire ove knjige.

U ovom trenutku, komponenta ne radi ništa korisno, ali pre nego što pređete na samo pravljenje komponenti, moram da Vam objasnim šta sve sačinjava jednu komponentu. Posle toga, napravićete ostatak `TFlashingLabel` komponente. Ostavite komponentu otvorenu u razvojnom okruženju, pošto će Vam biti potrebna malo kasnije.

Osobine i metode komponente

Veliki deo pravljenja komponenti je pravljenje njenih osobina i metoda. Događaji su, takođe, veliki deo pravljenja komponenti, ali hajde da prvo govorimo o osobinama i metodama a događaje ćemo objasniti kasnije.

Osobine

Do sada ste puno puta koristili osobine. Sa stanovišta korisnika, Vi znate šta su to osobine. Sada je potrebno da razumete osobine sa stanovišta osobe koja pravi komponente. Pre nego što počnete da pravite komponente, morate razumeti šta osobine jesu, a šta nisu.

Osobine nisu članovi klase. Logično bi bilo smatrati osobine, zapravo podacima klase kojoj pripadaju. Na kraju krajeva, sa osobinama radite na isti način kao i sa podacima iz klase. Na primer:

```
var
  W : Integer;
begin
  W := Width;
  Height := W * 2;
```

Ali, osobine nisu članovi klase i to morate imati na umu prilikom pravljenja komponenti. Osobine se mnogo razlikuju od podataka iz klase, ali imaju jednu zajedničku osobinu: određene su tipom podatka. Tip osobine može biti jedan od uobičajenih tipova podataka (`Integer`, `Word`, `Double`, `string` i tako dalje), klasa (`TCanvas`, `TFont` i tako dalje), ili striktura (na primer `TRect`).

Osobine su ustvari, specijalni tipovi objekata koji zadovoljavaju sledeće kriterijume:

- određene su podatkom iz klase koji se koristi za skladištenje vrednosti sobine.
- mogu implementirati `write` metodu.
- mogu implementirati `read` metodu.
- može im se direktno pristupati, nezavisno od `read` i `write` metoda.

- mogu biti određene samo za čitanje, ili samo za upisivanje.
- mogu imati podrazumevane vrednosti.
- mogu biti objavljene, ili ne.

Da bi ovo bilo jasnije, hajde da sada kažemo nešto o svakom od ovih kriterijuma.

Osobine su određene podatkom iz klase

Svaka osobina je određena jednim podatkom iz klase. Taj podatak sadrži trenutnu vrednost osobine. Za primer, uzmite obično dodeljivanje:

```
Label.Caption := 'Pat McGarry';
```

Ova naredba dodeljuje string `Caption` osobini `Label` komponente. Ono što se zaista dešava je nešto više od same dodele vrednosti. Pošto je `Caption` osobina tipa `string`, određena je jednim podatkom tipa `string` koji je član klase. Kada se izvrši ovakva dodela, vrednost se zapravo dodeljuje tom podatku preko koga je osobina određena. Korišćenje ovih podataka je neophodno pošto sama osobina nema mogućnost skladištenja podataka.

Ovaj podatak možete nazvati kako god želite, ali tradicija nalaže da ovi podaci imaju ista imena kao i same osobine sa početnim slovom *F*. Na primer, podatak iz klase za `Caption` osobinu se zove `FCaption`.

Veza između podatka iz klase i same osobine može biti izvor grešaka prilikom pravljenja komponenti. Nije problem u tome što je tu vezu teško shvatiti, nego u tome što često dolazi do zamene. Na primer, slučajno ste napisali:

```
Left := 20;
```

a hteli ste da napišete:

```
FLeft := 20;
```

Ovakva greška dovodi do čudnog ponašanja Vaše komponente, a znaćete i zašto, kada objasnim `write` metodu u sledećem odeljku.

Podatak iz klase se gotovo uvek deklariše kao private. Ovo je zato što Vi želite da korisnici menjaju njegovu vrednost kroz osobinu, ili kroz metod, ali nikada direktno. Ovakav način rada Vam daje maksimalnu kontrolu nad podacima iz klase, a to Vas vodi do sledeće mogućnosti osobina.

Osobine mogu imati write metode

Kada izvršite dodelu vrednosti osobini, mogu se desiti mnoge stvari. A šta će se tačno desiti, zavisi od tipa osobine. Na primer, sledeći kod izgleda jednostavno:

```
Left := 20;
```

Međutim, kada se ovaj kod izvrši, dešava se nekoliko stvari. Prvo, podatak iz klase, za koji je vezana `Left` osobina (FLeft), dobija novu vrednost. Dalje, (pretpostavljamo da se ovaj kod izvršava iz forme) forma se pomera ulevo na novo mesto pomoću Windows API funkcije `MoveWindow`. Konačno, sadržaj forme se osvežava pozivanjem `Invalidate` funkcije za formu.

Kako se sve to dešava? Dešava se kroz `write` metodu osobine `Left`. `Write` metod se poziva kada se osobini dodeli vrednost. Možete koristiti write metodu da biste proverili ispravnost podataka, ili da biste izvršili neku specijalnu operaciju.

`Write` metod osobine deklarišete onda kada deklarišete i samu osobinu. Sledi primer uobičajene deklaracije osobine:

```
property FlashRate : Integer
  read FFlashRate write SetFlashRate;
```

Ova deklaracija sadrži sintaksu koju do sada niste videli, pošto je ona specifična za osobine. Prvo, primetite da se osobina deklariše pomoću ključne reči `property` i da tip osobine dolazi iza imena osobine. Druga linija koda nalaže prevodiocu da se vrednost osobine čita direktno iz `FFlashRate` polja (govoriću više o read metodama za nekoliko trenutaka) i da osobina koristi `write` metodu pod imenom `SetFlashRate`. `Write` metodu možete nazvati kako god želite, ali tradicionalno, ime `write` metode je isto kao i ime osobine ispred koga se nalazi reč *Set*.

Kada se osobini dodeljuje vrednost, automatski se poziva write metoda te osobine. Write metoda mora biti procedura i mora imati jedan parametar. Ovaj parametar mora biti istog tipa kao i sama osobina. Na primer, deklaracija write metode FlashRate osobine glasi:

```
procedure SetFlashRate(AFlashRate : Integer);
```

Vrednost prenesena `write` metodi je vrednost koja je dodeljena osobini. Dakle, kada se izvrši sledeća naredba:

```
FlashingLabel.FlashRate := 1000;
```

dolazi do prenošenja vrednosti 1000 metodi `SetFlashRate`. Šta ćete uraditi sa tom vrednošću, zavisi od mnogo faktora. Barem ćete dodati tu vrednost podatku u klasi. Drugim rečima, `write` metoda bi trebala da izgleda ovako:

```
procedure TFlashingLabel.SetFlashRate(AFlashRate : Integer);
begin
  FFlashRate := AFlashRate;
  { Do some other stuff. }
end;
```

Korišćenje A kao prvog slova imena parametra `write` metoda je još jedna od Delphi tradicija.

Gotovo uvek ćete raditi nešto više, u okviru write metode, od prostog dodeljivanja vrednosti podatku u klasi. Ukoliko želite da samo dodelite vrednost podatku u klasi,

onda Vam `write` metoda neće ni trebati. Objasniću ovo za par trenutaka. Pre toga, obratimo pažnju na `read` metode.

Osobine mogu imati `read` metode

`Read` metoda radi na potpuno isti način kao i `write` metoda (osim, naravno, očigledne razlike između čitanja i pisanja). Kada se iščitava vrednost osobine, izvršava se read metoda i vraća se vrednost podatka iz klase koji je vezan za tu osobinu.

Ime `read` metode je isto kao i ime osobine ispred kojeg stoji reč *Get*. `Read` metoda ne prima nikakve parametre i vraća tip osobine. Na primer, ukoliko biste pisali `read` metodu `FlashRate` osobine, ona bi bila deklarisana na sledeći način:

```
function GetFlashRate : Integer;
```

`Read` metoda može prvo izvršiti neke operacije, pa zatim vratiti vrednost podatka iz klase (u ovom slučaju `FFlashRate`). Iščitavanje vrednosti osobine se dešava iz više razloga. Ponekad je to rezultat dodele vrednosti nekoj promenljivoj:

```
Rate := FlashingLabel.FlashRate;
```

A ponekad se koristi u okviru složenijih naredbi:

```
case FlashingLabel.FlashRate of
  1000 : SpeedUp;
  { etc. }
end;
```

Bez obzira na to kako se čitanje vrši, `read` metoda se uvek poziva.

Korišćenje `read` metoda nije obavezno. Uobičajeno je da se umesto `read` metoda koristi direktan pristup podatku iz klase. Hajde da sada pogledamo kako se koristi direktan pristup podatku iz klase.

Osobinama se može nezavisno pristupati

Ne morate koristiti `read` i `write` metode da biste radili sa osobinama. Ukoliko dodeljujete vrednost direktno podatku iz klase, ili ga čitate iz njega, možete koristite direktni pristup. Deklaracija osobine kod koje se koristi direktan pristup izgleda ovako:

```
property FlashRate : Integer
  read FFlashRate write FFlashRate;
```

Ova deklaracija govori kompajleru da se za pisanje i čitanje iz ove osobine, umesto odgovarajućih read i write metoda, koristi sam podatak (`FFlashRate`). Kada se u osobinu upisuje, upisuje se u podatak iz klase. Takođe, kada se osobina iščitava, iščitava se sam podatak. Ništa više.

Uobičajeno je da se čitanje iz osobine obavlja direktno, dok se upisivanje vrši kroz write metodu. Pogledajte prethodnu deklaraciju ove osobine:

```
property FlashRate : Integer
  read FFlashRate write SetFlashRate;
```

Osobina koristi direktan pristup za čitanje, dok za upisivanje koristi write metodu. Upisivanje u osobinu obično proizvodi sporedne efekte, kao što sam objasnio u prethodnom odeljku. U stvari, mogućnost da se anuliraju sporedni efekti je jedna od najjačih strana osobina. Da biste anulirali sporedne efekte, koristite `write` metodu prilikom upisivanja u osobinu. Čitanje, sa druge strane, zahteva samo iščitavanje podatka iz klase, tako da direktan pristup ima više smisla.

Osobine mogu biti postavljene samo za čitanje, ili samo za upisivanje

Možete postaviti osobinu samo za čitanje, ili samo za upisivanje. Postavljanje osobine tako da se iz nje može samo čitati je korisna mogućnost (VCL ima dosta takvih osobina). Na primer, možete imati osobinu čiju će vrednost korisnik moći da pročita, ali neće moći da je promeni. Može se desiti da izmena vrednosti jedne od osobina izazove nestabilan rad cele komponente, a to se ne sme dozvoliti.

Postavljanje osobine tako da se iz nje može samo čitati je jednostavno. Samo izostavite `write` deo u deklaraciji osobine:

```
property FlashRate : Integer
  read FFlashRate;
```

Ukoliko korisnik pokuša da dodeli vrednost ovakvoj osobini, prilikom prevođenja programa će dobiti grešku koja glasi: "`Cannot assign to a read-only property`". Kao što možete videti, postavljanje osobine tako da se iz nje može samo čitati, je veoma jednostavno.

Na sličan način možete postaviti osobinu tako da se u nju može samo upisivati. To se radi tako što se u deklaraciji osobine izostavi read deo. Teško je zamisliti čemu bi takva osobina služila, ali ukoliko Vam je potrebna, možete je napraviti.

Osobine mogu imati podrazumevane vrednosti

Podrazumevana vrednost je još jedna od korisnih mogućnosti osobina. Primetićete da, kada postavite komponentu na formu, mnoge osobine koje su prikazane u Object Inspector-u, već imaju neke vrednosti. To su podrazumevane vrednosti koje je postavila osoba koja je pravila komponentu. Postavljanje podrazumevanih vrednosti znatno olakšava rad korisnicima komponenti. Ukoliko je moguće, sve osobine bi trebale da imaju podrazumevane vrednosti. To omogućava korisniku komponente da izmeni samo neke osobine, dok druge može ostaviti onakvima kakve jesu. Neki tipovi osobina (kao, na primer, string osobine) ne mogu imati podrazumevane vrednosti, ali većina tipova može.

NAPOMENA String osobine ne mogu imati podrazumevane vrednosti.

Kao i `read` i `write` metode, i podrazumevane vrednosti se navode prilikom deklaracije osobine. Vratimo se na `FlashRate` osobinu. Deklarisanje `FlashRate` sa podrazumevanom vrednošću bi izgledalo ovako:

```
property FlashRate : Integer
  read FFlashRate write SetFlashRate default 800;
```

Kada se `FlashingLabel` komponenta prikaže u Object Inspector-u, vrednost `800` (konkretno, u milisekundama) će već biti prikazana pored `FlashRate` osobine.

NAPOMENA Postavljanje podrazumevane vrednosti za osobinu postavlja ovu vrednost samo u Object Inspector. Ne menja se vrednost samog podatka u klasi. Dakle, Vi morate dodeliti vrednost podatku u klasi kroz konstruktor komponente. Na primer, konstruktor za `FlashingLabel` komponentu bi trebao da izgleda ovako:

```
constructor TFlashingLabel.Create(AOwner : TComponent);
begin
  inherited;
  FFlashRate := 800;
  { Other things here. }
end;
```

Obavezno postavite vrednosti za sve podatke u klasi koji odgovaraju osobinama sa podrazumevanim vrednostima.

Ako ne želite da koristite podrazumevane vrednosti za neku osobinu, izostavite `default` deo u njenoj deklaraciji.

Osobine mogu biti objavljene, javne, ili privatne

Neke osobine su na raspolaganju u toku pravljenja programa. Vrednost ovih osobina, u toku pravljenja programa, možete menjati kroz Object Inspector, a u toku izvršavanja programa, kroz kod. Ovakve osobine su objavljene. Prosto rečeno, *objavljene* su one osobine koje se pojavljuju u Object Inspector-u u toku pravljenja programa. Sve osobine koje su navedene u `published` odeljku deklaracije klase komponente će biti prikazane u Object Inspector-u u toku pravljenja programa.

Drugim osobinama, takozvanim *javnim osobinama*, se može pristupiti samo u toku izvršavanja programa. Ovim osobinama se ne može pristupiti u toku pravljenja programa (one se ne pojavljuju u Object Inspector-u). Osobine ovog tipa se navode u `public` delu klase komponente.

Privatne osobine su one osobine koje koristi sama komponenta i nisu na raspolaganju korisniku komponente. Ove osobine se navode u `private`, ili `protected` delu deklaracije klase komponente.

Pisanje metoda za komponente

Pisanje metoda za komponente se ne razlikuje od pisanja metoda za bilo koju drugu Object Pascal klasu. Metode Vaše komponente mogu biti privatne, zaštićene, ili javne. Bitno je da imate u vidu ove nivoe pristupa u toku pravljenja komponenti.

Određivanje metoda koje bi trebale da imaju javni pristup je jednostavno. *Javna metoda* je ona koju korisnik Vaše komponente može pozvati kako bi izvršio određenu operaciju. Izbor između zaštićenih i privatnih metoda je nešto teži. Pošto steknete iskustvo u programiranju, biće Vam jednostavnije da odredite kada bi trebalo koristiti zaštićeni pristup, umesto privatnog pristupa.

Generalno govoreći, koristite privatne metode za operacije koje izvršava sama komponenta i kojima ne bi trebalo pristupati iz izvedenih klasa. Koristite zaštićene metode za operacije koje takođe izvršava sama komponenta, ali koje bi izvedene klase mogle da izmene kako bi omogućile dodatnu funkcionalnost.

`Read` i `write` metode za osobine su obično zaštićene. Dakle, Vi dozvoljavate klasama koje su izvedene iz klase Vaše komponente da izmene ponašanje `read` i `write` metoda.

Kao što sam rekao i ranije, metode komponenti su samo metode i u najvećem broju slučajeva se mogu tretirati kao i članice običnih klasa.

Dodavanje funkcionalnosti `TFlashingLabel` komponenti

Kasnije ću govoriti o događajima i o tome na koji način ih treba pisati. Sada imamo dovoljno informacija da bismo mogli da napišemo prvu komponentu. `FlashingLabel` komponenta ima sledeće:

- Osobinu `FlashRate` koja kontroliše period blinkanja
- Osobinu `FlashEnabled` koja uključuje i isključuje blinkanje
- Write metode za `FlashRate` i `FlashEnabled` osobine
- Podrazumevane vrednosti za `FlashRate` i `FlashEnabled` osobine
- Privatnu članicu klase (instanca klase `TTimer`) koja kontroliše vreme blinkanja
- Sve osobine klasične `Label` komponente

Prvo, pokazaću Vam kako da dovršite `FlashingLabel` celinu. Posle toga, prokomentarisaćemo svaki bitniji deo koda. Listing 20.2 prikazuje `FlashingLabel` celinu.

Listing 20.2: FlashingLabel.pas

```
unit FlashingLabel;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls;

type
  TFlashingLabel = class(TCustomLabel)
  private
    { Private declarations }
    FFlashEnabled : Boolean;
    FFlashRate    : Integer;
    Timer         : TTimer;
  protected
    { Protected declarations }

    { Protected write methods for the properties. }
    procedure SetFlashEnabled(AFlashEnabled : Boolean);
    procedure SetFlashRate(AFlashRate : Integer);

    { OnTimer event handler. }
    procedure OnTimer(Sender : TObject); virtual;
  public
    { Public declarations }
    constructor Create(AOwner : TComponent); override;
  published
    { Published declarations }

    { The component's properties. }
    property FlashEnabled : Boolean
      read FFlashEnabled write SetFlashEnabled default True;
    property FlashRate : Integer
      read FFlashRate write SetFlashRate default 800;

    { All the properties of TCustomLabel redeclared. }
    property Align;
    property Alignment;
    property AutoSize;
    property BiDiMode;
    property Caption;
    property Color;
    property Constraints;
    property DragCursor;
    property DragKind;
    property DragMode;
    property Enabled;
```

nastavlja se

Listing 20.2: FlashingLabel.pas *nastavak*

```
    property FocusControl;
    property Font;
    property ParentBiDiMode;
    property ParentColor;
    property ParentFont;
    property ParentShowHint;
    property PopupMenu;
    property ShowAccelChar;
    property ShowHint;
    property Transparent;
    property Layout;
    property Visible;
    property WordWrap;
    property OnClick;
    property OnDblClick;
    property OnDragDrop;
    property OnDragOver;
    property OnEndDock;
    property OnEndDrag;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
    property OnStartDock;
    property OnStartDrag;
  end;

procedure Register;

implementation
continues

constructor TFlashingLabel.Create(AOwner : TComponent);
begin
  inherited;

  { Set the data fields to their default values. }
  FFlashEnabled := True;
  FFlashRate    := 800;

  { Initialize the timer object. }
  Timer := TTimer.Create(Self);

  { Set the timer interval using the flash rate. }
  Timer.Interval := FFlashRate;

  { Assign our own OnTimer event handler to the
  { TTimer OnTimer event. }
  Timer.OnTimer := OnTimer;
```

```
end;

procedure TFlashingLabel.SetFlashEnabled(AFlashEnabled : Boolean);
begin
  { Set FFlashEnabled data field. }
  FFlashEnabled := AFlashEnabled;

  { Don't start the timer if the component is on a form
  { in design mode. Instead, just return. }

  if csDesigning in ComponentState then
    Exit;

  { Start the timer. }
  Timer.Enabled := FFlashEnabled;

  { If flashing was turned off, be sure that the label
  { is visible. }
  if not FFlashEnabled then
    Visible := True;
end;

procedure TFlashingLabel.SetFlashRate(AFlashRate : Integer);
begin
  { Set the FFlashRate data field and the timer interval. }
  FFlashRate := AFlashRate;
  Timer.Interval := AFlashRate;
end;

procedure TFlashingLabel.OnTimer(Sender : TObject);
begin
  { If the component is on a form in design mode,
  { stop the timer and return. }
  if csDesigning in ComponentState then begin
    Timer.Enabled := False;
    Exit;
  end;

  { Toggle the Visible property each time the timer
  { event occurs. }
  Visible := not Visible;
end;

procedure Register;
begin
  RegisterComponents('Samples', [TFlashingLabel]);
end;

end.
```

Deklaracija klase

ANALIZA Prvo, pogledajmo deklaraciju klase. Primetite da `private` deo deklariše tri podatka. Prva dva, `FFlashEnabled` i `FFlashRate` su podaci povezani sa `FlashEnabled` i `FlashRate` osobinama. Treća deklaracija izgleda ovako:

```
Timer : TTimer;
```

Ovaj kod deklariše pokazivač na TTimer objekat. TTimer objekat se koristi za regulaciju vremena blinkanja.

NAPOMENA Do sada nisam govorio o tajmerima. Tajmer se pokreće posle određenog intervala vremena (datog u milisekundama). Kada se tajmer pokrene, `WM_TIMER` poruka se šalje prozoru koji sadrži taj tajmer i, kao rezultat, poziva se procedura za obradu `OnTimer` događaja. Na primer, ako postavite interval vremena na 1000 milisekundi, procedura za obradu `OnTimer` događaja će biti pozvana svake sekunde. Obratite pažnju na činjenicu da je `WM_TIMER` poruka niskog prioriteta i može biti obrisana, ukoliko je sistem prezauzet. Iz ovog razloga, ne možete koristiti klasičan tajmer za operacije koje jako zavise od vremena. Ipak, za operacije kod kojih precizno računanje vremena nije toliko bitno, `TTimer` radi sasvim dobro. (Da biste naučili više o tajmerima, potražite stranice `TTimer` i `WM_TIMER` u Delphi-jevom Help-u.)

`Protected` deo deklaracije klase sadrži deklaracije `write` metoda za `FlashRate` i `FlashEnabled` osobine. `Protected` deo, takođe, deklariše i `OnTimer` funkciju. Ova funkcija se poziva svaki put kada se generiše događaj tajmera. Ona je zaštićena, ali je deklarisana kao `virtuelna`, tako da je izvedene klase mogu izmeniti, kako bi izvršile drugačiju obradu vremena. Na primer, izvedena klasa možda želi da promeni boju teksta svaki put kada tekst blinkne. Deklarisanje ove metode kao `virtuelne` omogućava izvedenoj klasi davanje funkcionalnosti.

Published deo

Konačno, pronađite `published` deo. Ovaj deo sadrži deklaracije `FlashEnabled` i `FlashRate` osobina. U ovom slučaju, `FlashEnabled` osobina koristi direktan pristup podatku u klasi prilikom čitanja, dok write deo pokazuje na `SetFlashEnabled` metod. Podrazumevana vrednost ove osobine je postavljena na `True`. `FlashRate` osobina koristi sličnu konstrukciju.

Primetite da sam pored ovih osobina deklarisao i sve osobine komponente `TCustomLabel` koje želim da objavim. Ukoliko ne izvršite ovaj korak, uobičajene osobine `Label` komponente neće biti na raspolaganju Vašoj komponenti ni u toku pravljenja programa (pošto instalirate komponentu na paletu sa komponentama), niti u toku izvršavanja programa.

Implementation deo

Sada usmerite Vašu pažnju na implementation deo. Prvo što ćete videti jeste konstruktor klase TFlashingLabel pod imenom Create. Ovde možete videti da su postavljene podrazumevane vrednosti za podatke iz klase koji predstavljaju osobine FlashEnabled i FlashRate. Zapamtite da ste već deklarisali podrazumevane vrednosti za ove osobine, ali da je efekat vidljiv samo u Object Inspector-u. Dakle, morate izvršiti dodelu vrednosti u konstruktoru.

Obratite pažnju na sledeće tri linije koda (komentari nisu navedeni):

```
Timer := TTimer.Create(Self);
Timer.Interval := FFlashRate;
Timer.OnTimer := OnTimer;
```

Prva linija pravi instancu TTimer objekta. Druga linija dodeljuje vrednost podatka FFlashRate TTimer-ovoj osobini Interval i na taj način postavlja interval vremena koji će se koristiti za blinkanje teksta na labeli. Konačno, poslednja linija dodeljuje OnTimer događaju Timer objekta funkciju OnTimer. Sada je sigurno da će komponenta moći da odreaguje kada se generiše OnTimer događaj.

SetFlashEnabled procedura je u stvari write metoda FlashEnabled osobine. Ova procedura prvo dodeljuje vrednost AFlashEnabled podatku FFlashEnabled. Dalje, Enabled osobina Timer objekta se postavlja u odnosu na vrednost AFlashEnabled. Pisanjem u FlashEnabled osobinu, korisnik komponente može da uključi, ili isključi blinkanje. Ova funkcija, takođe, sadrži kod koji postavlja vrednost Visible osobine na True i to onda kada je blinkanje isključeno. Ovaj korak onemogućava da se blinkanje ukine u trenutku kada tekst nestaje sa ekrana. Ukoliko bi se ovo desilo, tekst bi ostao sakriven.

SetFlashRate procedura

A sada, usmerite Vašu pažnju na SetFlashRate proceduru. Ovo je write metoda za FlashRate osobinu. Korisnik komponente dodeljuje vrednost osobini FlashRate kako bi kontrolisao brzinu kojom komponenta blinka. (Vrednost FlashRate od 1200 znači sporo blinkanje, dok vrednost od 150 znači veoma brzo blinkanje.) Ova funkcija sadrži sledeće linije koda:

```
FFlashRate := AFlashRate;
Timer.Interval := AFlashRate;
```

Kroz ovaj postupak se podatku FFlashRate i Interval osobini Timer objekta jednostavno dodeljuje vrednost parametra AFlashRate. Na ovaj način će se interval blinkanja promeniti i onda kada korisnik promeni vrednost osobine FlashRate u toku izvršavanja programa.

OnTimer metoda ne zahteva mnogo objašnjenja. Ovaj metod se poziva kao odgovor na OnTimer događaj. Metoda jednostavno postavlja određenu vrednost kompo-

nentinoj Visible osobini, svaki put kada se generiše `OnTimer` događaj. Drugim rečima, ukoliko je `FlashRate` postavljena na 1000, `OnTimer` događaj će se generisati približno jednom u toku svake sekunde. Kada se generiše prvi `OnTimer` događaj, komponenta se sakriva. Posle jedne sekunde, događaj se ponovo generiše i komponenta se ponovo prikazuje. Ovaj proces se nastavlja dokle god se `FlashEnabled` osobina ne postavi na `False`, ili dok program ne završi sa radom. Konačno, na kraju celine se nalazi kod za registraciju komponente.

Unesite kod iz listinga 20.2 u `FlashingLabel.pas` fajl koji je ranije za Vas napravio Delphi (onda kada ste koristili New Component dijalog-prozor). Ne morate unositi linije sa komentarima, ukoliko ne želite. Nešto kasnije ću Vam pokazati kako da proverite da li komponenta radi na odgovarajući način.

Delphi-jev sistem za automatsko popunjavanje deklaracije klase Vam pomaže prilikom deklarisanja `read` i `write` metoda za osobine. Recimo da ste deklarisali osobinu na sledeći način:

```
FFlashRate := AFlashRate;
Timer.Interval := AFlashRate;
```

Sledeći korak bi verovatno bio deklarisanje i definisanje `SetFlashRate` metode. Delphi će to uraditi za Vas! Potrebno je samo da pritisnete Ctrl+Shift+C i Delphi će napraviti sve `read` i `write` metode koje još uvek niste definisali i dodati deklaraciju podatka koji je vezan za tu osobinu (u ovom slučaju, `FFlashRate`).

Ukoliko imate Professional, ili Client/Server verziju Delphi-ja, možete iskopirati sve ponovne deklaracije direktno iz izvornog koda VCL-a. Otvorite `\Delphi4\Soruce\Vcl\StdCtrls.pas`, iskopirajte ponovne deklaracije iz deklaracije klase `TLabel` i vratite ih u deklaraciju Vaše klase.

`ComponentState` osobina

Možda ću sada malo požuriti, ali želim da ukažem na neke delove koda koje nisam objasnio u analizi listinga 20.2. Postoji jedna važna linija u `SetFlashEnabled` funkciji koja zahteva objašnjenje. Ovde je ponovo prikazana ta linija kao i linija koja se u listingu nalazi posle nje:

```
if csDesigning in ComponentState then
  Exit;
```

Ovaj deo koda proverava da li se komponenta koristi na formi u toku pravljenja programa. Ukoliko se koristi, morate onemogućiti pokretanje tajmera. Kada se komponenta postavi na formu u Form Designer-u, ne mora obavezno sadržati svu funkcionalnost - Form Designer ne može potpuno simulirati pokrenuti program. U ovom slučaju, Vi ne želite da tajmer radi dok se komponenta koristi u toku pravljenja programa.

NAPOMENA Mogli ste napraviti `TFlashingLabel` komponentu tako da ona može da blinka i u toku pravljenja programa kao i u toku izvršavanja. Jednostavnije je da to sada ne radite. U isto vreme mi omogućavate da kažem nešto i o `ComponentState` osobini.

Sve komponente imaju osobinu pod imenom `ComponentState`. `ComponentState` osobina je skup vrednosti koji pokazuje, između ostalog, da li se komponenta koristi u toku pravljenja programa. Ukoliko ovaj skup sadrži vrednost `csDesigning`, znate da se komponenta koristi na formi u Form Designer-u. Kada utvrdite da se komponenta koristi u toku pravljenja programa, možete bezuslovno izaći iz `OnTimer` funkcije i na taj način onemogućiti pokretanje tajmera.

Ovaj deo koda, dakle, isključuje tajmer ukoliko se komponenta koristi na Form Designer-u i to tako što momentalno izlazi iz OnTimer metode, pre nego što se izvrši ostatak koda. Tajmer se, inače, pokreće iz konstruktora `TFlashingLabel` komponente, tako da odmah mora biti isključen ukoliko se komponenta koristi u toku pravljenja programa.

Testiranje komponente

Sigurno ćete dodati svoju novu komponentu na paletu sa komponentama. Ipak, prvo morate da testirate komponentu da bi ste videli da li se prevodi na odgovarajući način i da li se ponaša onako kako biste Vi želeli. Ovo je presudan korak tokom pravljenja komponenti i mnogi programeri koji prave komponente ga ignorišu. Nema razloga za žurbu. Komponentu možete dodati na paletu kad god to želite. Bitno je da se uverite da ona zaista radi na odgovarajući način.

Da biste testirali komponentu, napišite program koji će biti okruženje za testiranje. Pošto ne možete da postavite komponentu direktno sa palete, moraćete ručno da je napravite. U ovom slučaju, pošto Vaša `FlashingLabel` komponenta ima samo dve osobine, želite da se uverite da svaka osobina radi.

Da biste to uradili, Vaš test program mora da uključuje i isključuje blinkanje. Dalje, Vaš test program mora da dozvoli postavljanje nekoliko intervala blinkanja, da biste mogli da vidite da li `FlashingRate` osobina radi na odgovarajući način. Slika 20.2 prikazuje test program u fazi izvršavanja. Ova slika će Vam dati ideju kako možete da napravite svoj test program.

Slika 20.2
Test program u fazi izvršavanja

Pošto ste bacili pogled na test program, hajde da ga napravimo. Kao i uvek, počnite od prazne forme. Prvo dodajte opciju i radio-grupu kao što je prikazano na slici 20.2:

1. Izmenite `Name` osobinu forme u `MainForm` i njenu `Caption` osobinu u `FlashingLabel Test Program`.

2. Koristeću sliku 20.2 kao primer, dodajte `CheckBox` komponentu na formu. Izmenite njenu `Name` osobinu u `FlashBox`, `Caption` osobinu u `Flash` i `Checked` osobinu na `True`.

3. Dva puta kliknite na opciju da biste napravili proceduru za obradu `OnClick` događaja. Kada se prikaže Code Editor koji prikazuje proceduru za obradu `OnClick` događaja, na mesto na kome se nalazi kurzor dodajte sledeću liniju koda (ime `FlashingLabel` komponente će biti `Flasher`):

   ```
   Flasher.FlashEnabled := FlashBox.Checked;
   ```

   Na ovaj način se blinkanje uključuje, ili isključuje na osnovu izbora opcije.

4. Postavite `RadioGroup` komponentu na formu. Izmenite njenu `Name` osobinu u `Group` i njenu `Caption` osobinu u `Flash Speed`.

5. Dva puta kliknite na Value kolonu koja se nalazi pored `Items` osobine. Kada se prikaže String Editor, unesite sledeće linije:

   ```
   Slow
   Medium
   Fast
   Light Speed
   ```

   Kliknite na OK taster da biste zatvorili String Editor. Stringovi, koje ste upravo uneli, će se prikazati kao radio-tasteri u okviru radio-grupe.

6. Postavite `ItemIndex` osobinu na 1. Biće izabran Medium radio-taster.

7. Dva puta kliknite na radio-grupu. Prikazaće se Code Editor sa procedurom za obradu `OnClick` događaja radio-grupe. Na mesto na kome se nalazi kurzor unesite sledeće linije koda:

   ```
   case Group.ItemIndex of
   ```

```
    0 :  Flasher.FlashRate := 1200;
    1 :  Flasher.FlashRate := 800;
    2 :  Flasher.FlashRate := 400;
    3 :  Flasher.FlashRate := 150;
  end;
```

Na ovaj način ćete postaviti vrednost `FlashRate` osobine `FlashingLabel` komponente na osnovu izabranog radio-tastera.

8. Sačuvajte projekat u istom direktorijumu u kome se nalazi Vaša `TFlashingLabel` komponenta. Sačuvajte glavnu formu pod imenom `FlshTstU.pas` (obratite pažnju da ja koristim kratka imena fajlova u izvornim kodovima iz knjige. Ukoliko želite, Vi možete koristiti dugačka imena fajlova).

U redu, sada dodajte samu komponentu. Pošto komponenta još uvek nije vizuelna (ne možete je dodati sa palete sa komponentama), moraćete ručno da je dodate.

1. Kliknite na Add to Project taster, (iz toolbar-a, glavnog menija, ili iz kontekstnog menija Project Manager-a). Kada se prikaže Add to Project dijalog-prozor, izaberite `FlashingLabel.pas` i kliknite na taster OK.
2. Pomerite se na početak celine i dodajte `FlashingLabel` u njenu uses listu.
3. Dodajte sledeću deklaraciju u `private` deo klase `TMainForm`:

```
Flasher : TFlashingLabel;
```

4. Dva puta kliknite na pozadinu forme da biste napravili proceduru za obradu `OnCreate` događaja forme. Unesite sledeći kod u proceduru:

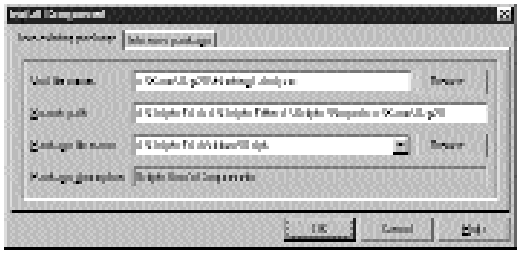
```
Flasher := TFlashingLabel.Create(Self);
Flasher.Parent := Self;
Flasher.SetBounds(20, 20, 200, 20);
Flasher.Font.Size := 16;
Flasher.Caption := 'This is a test';
Flasher.FlashRate := 800;
```

Sada ste spremni za testiranje komponente. Kliknite na Run taster da biste preveli i pokrenuli test program. Ukoliko dobijete grešku u toku prevođenja, pažljivo proverite da li ste ispravno uneli kod i ispravite svaku grešku na koju ukaže prevodilac.

Kada se program pokrene, kliknite na Flash opciju da biste uključili, ili isključili blinkanje. Izmenite interval blinkanja izborom jednog od radio-tastera. Hej, radi! Čestitamo, upravo ste napravili svoju prvu komponentu.

Postavljanje komponente na paletu sa komponentama

Pošto ste se uverili da komponenta radi na odgovarajući način, možete je postaviti na paletu sa komponentama. Da biste dodali `FlashingLabel` komponentu na paletu, izaberite Component➥Install Component. Pojaviće se Install Component dijalog prozor. Pomoću ovog dijalog-prozora dodajete komponente u odgovarajuće pakete. Slika 20.3 prikazuje Install Component dijalog-prozor.



Slika 20.3
Install Component dijalog-prozor

U redu, sada ste spremni da dodate `FlashingLabel` komponentu na paletu. Izvršite sledeće operacije:

1. Izaberite Component➥Install Component iz Delphi-jevog glavnog menija. Prikazuje se Install Component dijalog-prozor.
2. Kliknite na taster Browse koji se nalazi sa desne strane `Unit file name` polja. Kada se prikaže dijalog-prozor sa imenima fajlova, pronađite `FlashingLabel.pas` i kliknite na taster OK.
3. Sada pogledajte `Package file name` polje. Trebalo bi da sadrži tekst `DCLUSR40.DPK`. Ukoliko to nije slučaj, klinkite na taster sa strelicom na dole i iz liste izaberite `DCLUSR40.DPK`. Ukoliko se ovaj paket ne nalazi u listi, kliknite na taster Browse da biste pronašli ovaj fajl (nalazi se u `\Delphi 4\Lib` direktorijumu).
4. Kliknite na taster OK da biste zatvorili Install Component dijalog-prozor. Delphi prikazuje poruku koja Vam kaže da će morati da prekompajlira paket kako bi Vaša komponenta mogla da bude instalirana. Kliknite na taster Yes.
5. Delphi kompajlira paket i postavlja komponentu. Kada se proces završi Delphi će Vas, preko prozora sa porukom, obavestiti da je `TFlashingLabel` komponenta registrovana.

Vaša komponenta će se sada naći na Samples stranici palete sa komponentama. Aktivirajte ovu stranicu i uverite se da se na njoj nalazi nova komponenta koja je predstavljena tasterom na kome se nalazi Delphi-jeva sličica. Ukoliko zadržite miša na ovom tasteru, pojaviće se balon za pomoć na kome će pisati `FlashingLabel`.

Pokrenite novi projekat i istestirajte `FlashingLabel` komponentu, tako što ćete je spustiti na formu. Primetite da se u Object Inspector-u nalaze sve osobine koje sadrži i klasnična `Label` komponenta. Pored njih su prisutne i `FlashRate` i `FlashEnabled` osobine. Primetite da se i podrazumevane vrednosti, koje ste naveli za ove osobine, takođe nalaze u Object Inspector-u.

Želim da objasnim ono što ste uradili u koraku 3 prethodnog procesa. Delphi sadrži podrazumevani paket pod nazivom `DCLUSR40`. U njega možete stavljati pojedinačne komponente. Rekao sam Vam da instalirate `TFlashingLabel` u ovaj paket zato što je ona pojedinačna komponenta (a ne deo nekog skupa komponenti), a `DCLUSR40` služi za instaliranje baš takvih komponenti. Mogli ste da napravite i novi paket, pa da komponentu instalirate u njega, umesto u `DLCUSR40`. Ipak, jednostavnije je instalirati komponentu u već postojeći paket.

Postavljanje proizvoljne bitne mape na taster komponente

Primetili ste da je Vaša nova komponenta predstavljena tasterom na kome se nalazi Delphi-jeva sličica. Ovo ne biste smeli da dozvolite. Srećom, postoji mogućnost postavljanje proizvoljne bit mape za Vaše komponente. Morate napraviti bit mapu i postaviti je u fajl sa prevedenim resursima (`.dcr` fajl).

Ponekad će Vam biti zgodno da iskoristite bit mapu roditeljske klase, koju ćete malo izmeniti, tako da odgovara Vašim potrebama. U tom slučaju, pokrenite Image Editor i otvorite jedan od `.dcr` fajlova koji se nalaze u `\Delphi 4\Lib\Obj` direktorijumu. Moraćete da pronađete odgovarajuću bit mapu. Na primer, bit mapa za klasičnu `Label` komponentu se nalazi u fajlu Stdreg.dcr. Otvorite taj fajl i iskopirajte `TLABEL` bit mapu u Clipboard. Zativ, otvorite novu bit mapu, vratite sadržaj iz Clipboard-a i nazovite tu bit mapu imenom `TFLASHINGLABEL`. Izmenite bit mapu, tako da odgovara Vašim potrebama a posle toga, sačuvajte projekat sa resursima.

Bit mapa koja će se nalaziti na tasteru komponente mora imati dimenzije 24x24 piksela. U najvećem broju slučajeva će 16 boja biti sasvim dovoljno. Zapamtite da Delphi koristi niži-levi piksel u bit mapi za transparentnu boju. (Delphi-jeve bit mape za komponente koriste tamno žutu boju kao transparentnu tako da, ukoliko želite da poštujete konvenciju, možete postupiti na isti način.)

Pazite da obavezno napravite bit mapu, a ne ikonu. Tasteri na paleti sa komponentama se često nazivaju ikonama, ali njihove sličice su bit mape, a ne ikone.

Pošto ste napravili fajl sa resursima, Delphi automatski dodaje komponentinu bit mapu na paletu, kada instalirate odgovarajući paket. Da bi ova procedura bila moguća, morate poštovati konvenciju o davanju imena bit mapama.

Konkretno, fajl sa resursima mora imati bit mapu čije je ime identično punom nazivu klase komponente. Na primer, da biste napravili bit mapu za taster `FlashingLabel` komponente, koristite Image Editor da biste napravili fajl sa resursima koji sadrži bit

mapu pod imenom `TFLASHINGLABEL`. Sa druge strane, fajl sa resursima možete nazvati kako god želite.

Pošto ste napravili fajl sa resursima, morate reći Delphi-ju da ga poveže sa komponentinim kodom. Da biste to uradili, dodajte ovakvu liniju koda u izvorni kod Vaše komponente:

```
{$R Flashing.res}
```

Direktiva prevodiocu `$R` nalaže uključivanje sadržaja fajla sa resursima u prevedeni kod celine. Sada, prekompajlirajte paket. Ukoliko ste sve uradili na odgovarajući način, bit mapa, koju ste napravili za taster, će se pojaviti na paleti sa komponentama.

NAPOMENA Primetite da je ekstenzija fajla sa resursima u prethodnoj liniji koda `.res`. Ekstenzija `.res` se koristi ravnopravno sa `.dcr` ekstenzijom. Neki proizvođači komponenti koriste jedinstvenu konvenciju o zadavanju imena fajlovima za svoje prevedene resurse, dok drugi koriste, ili `.res`, ili `.dcr`. Ekstenzija fajla nije bitna prilikom korišćenja `$R` direktive prevodiocu. Važno je da taj fajl sadrži ispravne resurse.

Takođe, možete dodati `$R` direktivu direktno u izvorni kod paketa. Ipak, u najvećem broju slučajeva, to nije potrebno.

UPOZORENJE Možete dodati resurse samo na jednom mestu. Postavite `$R` direktivu prevodiocu, ili u izvorni kod celine komponente, ili u izvorni kod paketa, ali nikako na u obadva. Ukoliko više puta dodate iste resurse, dobićete poruku o grešci i Vaša komponenta neće biti instalirana.

SAVET Ukoliko imate biblioteku koja se sastoji od više komponenti, možete koristiti jedan fajl sa resursima u koji ćete staviti bit mape za sve komponente iz te biblioteke. Korišćenje odvojenih fajlova sa resursima za svaku komponentu nije neophodno.

Pravljenje događaja za komponente

Pravljenje događaja zahteva planiranje. Kada govorimo o događajima, u stvari govorimo o dve mogućnosti. Ponekad se događaj generiše kao odgovor na neku Windows-ovu poruku, a ponekad i kao rezultat neke izmene u komponenti. Događaj, koji je generisala Windows-ova poruka, je manje, ili više pod Vašom kontrolom. Možete odgovoriti na ovaj tip događaja ali, u principu, ne možete sami generisati ovakav događaj. Drugi tip događaja generiše sama komponenta. Drugim rečima, Vi kao autor komponente možete kontrolisati kada će se generisati ovakav događaj.

Rad sa događajima na ovom nivou može biti zbunjujući. Probaću da zaobiđem sve zbunjujuće aspekte i da Vam predstavim rad sa komponentama na čisto praktičnom nivou. Da bih to mogao da uradim, trebali bismo da napravimo jedan događaj za `TFlashingLabel` klasu. Na početku ćemo ipak nešto reći neke osnovne stvari o događajima.

Pregled događaja

Za početak biste trebali da razumete da su događaji, u stvari, osobine i da kao takvi imaju sve mogućnosti koje imaju i klasične osobine. U slučaju događaja, podatak iz klase, sa kojim je on povezan, je privatan i sadrži adresu funkcije koja će biti pozvana kada se generiše taj događaj. Kao i osobine, i događaji mogu biti objavljeni, ili neobjavljeni. Objavljeni događaji se pojavljuju u Object Inspector-u kao i objavljene osobine.

Događaji su *pokazivači na metode*. Pokazivači na metode su slični pokazivačima na funkcije. Mogu pokazivati ne samo na funkciju iz klase kojoj pripadaju, već i na funkciju koja pripada nekoj drugoj klasi. Dokle god se deklaracije funkcija poklapaju (to znači da im je povratna vrednost istog tipa i da je im lista parametara identična), pokazivač na metode može uspešno pozivati funkcije, bez obzira na to gde se one nalaze. Na primer, `OnClick` događaj `TLabel` komponente može pokazivati na funkciju za obradu događaja iz `TEdit` objekta, `TForm` objekta, `TListBox` objekta i tako dalje. Pokazivači na metode su, inače, komplikovaniji, ali neću ulaziti u nepotrebne detalje.

Procedure za obradu događaja moraju biti baš procedure. Događaj bi mogao da prosledi jedan, ili više parametara, u zavisnosti od tipa događaja, ali nikada ne bi mogao da vrati vrednost. Ipak, postoji način na koji možete dobiti povratnu informaciju od dagađaja. Možete deklarisati jedan, ili više parametara kao `Var` i dozvoliti korisniku da menja ove parametre kako bi određena aktivnost bila zabeležena.

Možete raditi sa događajima na jednom od nekoliko nivoa. Na primer, možete zameniti proceduru za obradu nekog događaja iz osnovne klase i na taj način dodati određenu funkcionalnost. Recimo da želite da se nešto specijalno dogodi kada korisnik klikne mišem na Vašu komponentu. Nema potrebe da pravite novi događaj iz početka, jer je to veoma komplikovano, kada taj događaj već postoji kao `OnClick` događaj u okviru osnovne klase. Jednostavno se oslonite na taj događaj, kako biste mogli da reagujete na kliktanje mišem.

Drugi nivo rada je pravljenje događaja koji možete pokrenuti iz same komponente. Prvo ću opisati ovaj tip događaja. Kao što sam rekao, dodaćete jedan događaj `TFlashingLabel` komponenti koju ste napravili ranije. Dodavanje ovog događaja zahteva i dodavanje nove osobine. Ime događaja će biti `OnLimitReached`, dok će se nova osobina zvati `FlashLimit`. Ovaj događaj će biti pokrenut kada komponenta blinkne onoliko puta koliko je to određeno vrednošću osobine `FlashLimit`. Ukoliko je vrednost `FlashLimit 0` (podrazumevana vrednost), `OnLimitReached` događaj nikada neće biti generisan.

Pravljenje korisnički definisanog događaja se može podeliti na pet zadataka:

1. Određivanje tipa događaja.
2. Deklarisanje podatka iz klase sa kojim će događaj biti povezan.

3. Deklarisanje događaja.
4. Pravljenje virtuelnog metoda koji pokreće događaj.
5. Pisanje koda koji pokreće događaj.

Prođimo sada kroz sve ove zadatke kako biste bolje razumeli šta je sve potrebno uraditi.

Određivanje tipa događaja

Kada sam ranije govorio o osobinama, rekao sam da svaka osobina ima određeni tip. Isto važi i za događaje. U slučaju događaja, taj tip je pokazivač na metode, koji sadrži opis parametara procedure za obradu događaja. Juče, u danu 19 "Pravljenje i korišćenje DLL-ova", sam prilikom objašnjavanje rada DLL-ova govorio i o pokazivačima na funkcije.

Kao što sam rekao već nekoliko puta, postoja dva osnovna tipa događaja. Jedan je takozvani *notofikacioni događaj*. Ovaj događaj Vam govori da se nešto zaista dogodilo, ali Vam ne daje nikakve detalje. Deklaracija funkcije procedure za obradu ovakvog događaja izgleda ovako:

```
procedure Clicked(Sender : TObject);
```

Jedina informacija koju dobijate od notifikacionog događaja jeste objekat koji je pokrenuo događaj. Delphi obezbeđuje `TNotifyEvent` tip za notifikacione događaje. Svaki događaj koji pravite, a koji treba da bude notifikacioni, bi trebali da daklarišete kao događaj tipa `TNotifyEvent`.

Drugi tip događaja je onaj koji prima više od jednog parametra i koji zaista prenosi neke informacije proceduri za obradu. Možete, ako želite, da napravite ovakav događaj za Vašu komponentu. Recimo da želite da koristite proceduru za obradu događaja čiji prototip izgleda ovako:

```
procedure LimitReached(Sender : TObject; var Stop : Boolean);
```

Korišćenjem ovog tipa procedure za obradu događaja, dozvoljavate korisniku da izmeni vrednost parametra `Stop` i da na taj način vrati podatak komponenti. Ukoliko želite da pravite događaje koji imaju parametre, moraćete da definišete sopstveni tip metode.

Recimo da želite da definišete tip događaja za prethodnu deklaraciju metode i da će se taj tip događaja zvati `TLimitReachedEvent`. Trebao bi da izgleda ovako:

```
TLimitReachedEvent =
  procedure(Sender : TObject; var Stop : Boolean) of object;
```

Iako izgleda malo zbunjujuće, sve što treba da uradite je da iskopirate ovu definiciju i da po potrebi ubacite i izabacite parametre. Pošto ste definisali tip događaja, možete deklarisati događaj koji treba da bude tipa `TLimitReachedEvent`. (Verovatno je da

ovo niste razumeli, što je i normalno. Situacija će biti jasnija kada prođemo kroz ceo proces. Zato, nastavite dalje.)

Postavite deklaraciju novog tipa događaja u type deo celine i to iznad deklaracije klase:

```
unit FlashingLabel;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics,
Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls;

type

  TLimitReachedEvent =
    procedure(Sender : TObject; var Stop : Boolean) of
object;

  TFlashingLabel = class(TCustomLabel)
  private
{ Rest of unit follows. }
```

Pokušajte da do maksimuma istrenirate svoju sposobnost određivanja tipa potrebnih događaja za Vašu komponentu (naravno, ukoliko su Vam događaji uopšte potrebni). Ukoliko su Vam potrebni samo notifikacioni događaji, pravićete događaje koji su tipa `TNotifyEvent`. Ukoliko će Vaši događaji prosleđivati parametre do procedure za obradu, morađete da definišete sopstveni tip događaja.

Deklarisanje podatka iz klase sa kojim će događaj biti povezan

Deklarisanje podatka iz klase je jednostavan zadatak. Sve što treba da uradite je da deklarišete privatni podatak koji je istog tipa kao i Vaš događaj. To izgleda, otprilike, ovako:

```
FOnLimitReached : TLimitReachedEvent;
```

Kao i kod osobina, ime podatka iz klase je isto kao i ime događaja pri čemu je na početku dodato slovo *F*.

Deklarisanje događaja

Pošto ste odredili tip događaja, možete deklarisati događaj za komponentu. Deklaracija događaja izgleda isto kao i deklaracija bilo koje druge osobine. Sledi tipična deklaracija događaja:

```
property OnSomeEvent : TNotifyEvent
  read FOnSomeEvent write FOnSomeEvent;
```

Ova deklaracija liči na one sa kojima ste se sretali do sada. Primetite da ne postoji `default` deo. Primetite da postoje `read` i `write` delovi, ali da oba pokazuju na `FSomeEvent` podatak iz klase. Ovo znači da događaji koriste direktan pristup podacima iz klase umesto korišćenje `read` i `write` metoda. Na kraju, primetite da je tip ovog događaja `TNotifyEvent`.

Događaj `OnLimitReached` će prenositi parametre, tako da morate definisati sopstveni tip događaja. Dakle, deklaracija Vašeg `OnLimitReached` događaja treba da izgleda ovako:

```
property OnLimitReached : TLimitReachedEvent
  read FOnLimitReached write FOnLimitReached;
```

Uskoro ću Vam prikazati kompletnu celinu, tako da ćete moći da vidite kako sve funkcioniše. (Ukoliko želite da radite unapred, pogledajte listing 20.3.)

Pravljenje virtuelnog metoda koji pokreće događaj

Pravljenje virtuelnog metoda koji pokreće događaj zahteva prethodno objašnjenje. Događaj ćete pokretati kao rezultat neke promene u komponenti. U ovom slučaju, pokretaćete događaj onda kada komponenta blinkne onoliki broj puta koji je određen vrednošću osobine `FlashLimit`. Događaj pokrećete njegovim pozivanjem:

```
var
  Stop : Boolean;
begin
  Stop := False;
  FOnLimitReached(Self, Stop);
```

Događaj pokrećete sa jednog od nekoliko mesta u komponenti, u zavisnosti od različitih faktora. Da biste pokretali događaj na isti način, bez obzira odakle, napišite virtuelnu metodu koja će pokretati događaj. Virtuelna metoda će imati isto ime kao i sam događaj, pri čemu se na početku imena umesto slova *On* nalaze slova *Do*.

Postoje dve popularne konvencije o zadavanju imena metodama koje generišu događaje. Prva konvencija nalaže da se sa početka imena izbace slova On i postave slova Do. U slučaju `OnLimitReached`, virtuelna metoda koja generiše ovaj događaj bi se zvala `DoLimitReached`. Druga konvencija nalaže da se samo izbrišu slova On. Poštovao sam obe konvencije, ali mi se malo više sviđa prva.

Prvo, deklarišite metodu u deklaraciji klase:

```
procedure DoLimitReached; virtual;
```

Zatim, napravite metodu koja zaista pokreće događaj. Ova metoda bi trebala da izgleda, otprilike, ovako:

```
procedure TFlashingLabel.DoLimitReached;
var
  Stop : Boolean;
```

```
begin
  Stop := False;
  if Assigned(FOnLimitReached) then
    FOnLimitReached(Self, Stop);
  FlashEnabled := not Stop;
end;
```

Postoji nekoliko stvari koje zahtevaju komentar. Prvo, primetite da ste postavili podrazumevanu vrednost `Stop` parametra. Ukoliko korisnik ne promeni parametar `Stop`, njegova vrednost će biti `False`. Vrednost `Stop` parametra određuje da li treba zaustaviti blinkanje kada se dostigne vrednost osobine `FlashLimit`. U proceduri za obradu događaja (u programu koji koristi komponentu), korisnik može postaviti vrednost parametra `Stop` na `True`, kako bi prouzrokovao prestanak blinkanja, ili na `False` kako bi omogućio nastavak blinkanja. Primetite da se `Self` prenosi kao prvi parametar, pri čemu se parametar Sender postavlja na pokazivač na komponentu.

Sada, obratite pažnju na naredbu koja pokreće događaj:

```
if Assigned(FOnLimitReached) then
    FOnLimitReached(Self, Stop);
```

Ukoliko je korisnik komponente napravio proceduru za obradu događaja, pozovite nju. Ukoliko procedura za obradu ne postoji, pozovite podrazumevanu proceduru za rad sa događajem. (U ovom slučaju, ne postoji ni ta, podrazumevana procedura.) Važno je da omogućite korisniku da ignoriše događaj, ukoliko to želi.

Jedna od stvari koju je teže razumeti na početku je i da događaji, sami po sebi, ne obezbeđuju proceduru za njihovu obradu. Program koji koristi komponentu mora da obezbedi proceduru za obradu događaja, dok Vi obezbeđujete pozivanje te procedure, kada se događaj zaista i desi.

Ukoliko korisnik nije definisao proceduru za obradu događaja, događaj će biti `nil` (vrenost nije dodeljena). Nemojte nikada pozivati proceduru za obradu bez provere da li je događaju dodeljena neka vrednost. Pokušaj da se pokrene događaj kome nije dodeljena procedura će dovesti do prekoračenja prava pristupa (engl. Access Violation) u Vašoj komponenti.

Kao što sam i ranije pomenuo, `DoLimitReached` je virtuelna metoda. Ona je virtuelna zato da bi izvedene klase mogle da je ponovo definišu i na taj način odrede novo ponašanje događaja. Pošto ste Vi bili dovoljno uviđavni i deklarisali metodu kao virtuelnu, sve klase izvedene iz klase Vaše komponente mogu da prepišu `DoLimitReached` funkciju i promene ponašanje događaja. Na ovaj način je promena ponašanja događaja jednostavna i ne zahteva pronalaženje svakog mesta u kodu odakle je događaj pokrenut. Sam događaj se pokreće samo iz `DoLimitReached` metode.

Pisanje koda koji pokreće događaj

Negde u komponenti se mora nalaziti kod koji poziva `DoLimitReached` metodu (koja, u stvari, pokreće događaj). U slučaju `TFlashingLabel` komponente, pozivate `DoLimitReached` iz `OnTimer` procedure. Posle dodavanja mogućnosti pokretanja događaja, ova funkcija izgleda ovako:

```
procedure TFlashingLabel.OnTimer(Sender : TObject);
begin
  { If the component is on a form in design mode,
  { stop the timer and return. }
  if csDesigning in ComponentState then begin
    Timer.Enabled := False;
    Exit;
  end;

  { Toggle the Visible property each time }
  { the timer event occurs. }
  Visible := not Visible;

  { Trigger the event if needed. Only increment }
  { the counter when the label is visible. }
  if (FFlashLimit <> 0) and Visible then begin

    { Increment the FlashCount data field. }
    Inc(FlashCount);

    { If the FlashCount is greater than or equal to   }
    { the value of the FlashLimit property, reset the }
    { FlashCount value to 0 and trigger the event.    }
    if FlashCount >= FFlashLimit then begin
      FlashCount := 0;
      DoLimitReached;
    end;
  end;
end;
```

Kao što možete videti, kada se dostigne vrednost osobine `FlashLimit`, poziva se `DoLimitReached` metod i pokreće se događaj. Usput, treba da brojite svaki drugi `OnTimer` događaj. Ukoliko budete brojali svaki `OnTimer` događaj, dobićete neodgovarajuću vrednost, pošto se `OnTimer` poziva dva puta za svako blinkanje (prikazivanje i brisanje sa ekrana). Promenljiva `Count` je podatak iz klase, a `FlashLimit` je osobina.

Prepisivanje događaja bazne klase

Prethodni odeljak govori o nečemu što bih ukratko želeo da objasnim. Ukoliko želite da prepišete podrazumevano ponašanje jednog od događaja iz bazne klase, sve što

treba da uradite je da prepišete procedure koje pokreću te događaje, na način koji sam već opisao. Recimo da želite da prepišete podrazumevano ponašanje `OnClick` događaja, kako biste naterali zvučnik da se oglasi svaki put kada se klikne na komponentu. Treba samo da prepišete `Click` funkciju bazne klase i to na sledeći način:

```
procedure TFlashingLabel.Click;
begin
  { Beep the speaker and then call the base class }
  { Click method for the default handling. }
  MessageBeep(-1);
  inherited;
end;
```

Pošto je ova funkcija u okviru bazne klase deklarisana dinamički, biće pozvana svaki put kada se klikne mišem na komponentu. Funkcionisaće samo ukoliko je komponenta trenutno vidljiva na ekranu. Imajte to na umu, ako želite da kliknete na komponentu koja blinka. Poziv funkcije `MessageBeep` je tu samo da bi dokazao da ovaj princip funkcioniše.

Sastavljanje celine

Do sada ste radili sa delovima `TFlashingLabel` komponente, a sada ćemo da sastavimo celu komponentu. Listing 20.3 prikazuje izvorni kod završene TFlashingLabel komponente. Proučite implementaciju `OnLimitReached` događaja da biste saznali kako treba da realizujete događaje u svojim komponentama. Listing 20.4 sadrži glavnu celinu nešto izmenjenog test programa. Metoda `MainFormLimitReached` demonstrira korišćenje `OnLimitReached` događaja.

Listing 20.3: `FlashingLabel.pas` **(NOV I POBOLJŠAN)**

```
unit FlashingLabel;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls,
  Forms, Dialogs, StdCtrls, ExtCtrls;

type

  TLimitReachedEvent =
    procedure(Sender : TObject; var Stop : Boolean) of object;
 TFlashingLabel = class(TCustomLabel)
  private
    { Private declarations }
    FFlashEnabled   : Boolean;
    FFlashLimit     : Integer;
    FFlashRate      : Integer;
    FOnLimitReached : TLimitReachedEvent;
```

nastavlja se

Listing 20.3: `FlashingLabel.pas` (NOV I POBOLJŠAN) *nastavak*

```
    FlashCount      : Integer;
    Timer           : TTimer;
  protected
    { Protected declarations }

    { Protected write methods for the properties. }
    procedure SetFlashEnabled(AFlashEnabled : Boolean);
    procedure SetFlashRate(AFlashRate : Integer);
    procedure DoLimitReached; virtual;
    procedure Click; override;

    { OnTimer event handler. }
    procedure OnTimer(Sender : TObject); virtual;
  public
    { Public declarations }
    constructor Create(AOwner : TComponent); override;
  published
    { Published declarations }

    { The component's properties. }
    property FlashEnabled : Boolean
      read FFlashEnabled write SetFlashEnabled default True;
    property FlashRate : Integer
      read FFlashRate write SetFlashRate default 800;
    property FlashLimit : Integer
      read FFlashLimit write FFlashLimit default 0;

    property OnLimitReached : TLimitReachedEvent
      read FOnLimitReached write FOnLimitReached;

    { All the properties of TCustomLabel redeclared. }
    property Align;
    property Alignment;
    property AutoSize;
    property BiDiMode;
    property Caption;
    property Color;
    property Constraints;
    property DragCursor;
    property DragKind;
    property DragMode;
    property Enabled;
    property FocusControl;
    property Font;
    property ParentBiDiMode;
    property ParentColor;
    property ParentFont;
```

```
    property ParentShowHint;
    property PopupMenu;
    property ShowAccelChar;
    property ShowHint;
    property Transparent;
    property Layout;
    property Visible;
    property WordWrap;
    property OnClick;
    property OnDblClick;
    property OnDragDrop;
    property OnDragOver;
    property OnEndDock;
    property OnEndDrag;
    property OnMouseDown;
    property OnMouseMove;
    property OnMouseUp;
    property OnStartDock;
    property OnStartDrag;
  end;

procedure Register;

implementation

constructor TFlashingLabel.Create(AOwner : TComponent);
begin
  inherited;

  { Set the data fields to their default values. }
  FFlashEnabled := True;
  FFlashRate    := 800;
  FFlashLimit   := 0;
  FlashCount    := 0;

  { Initialize the timer object. }
  Timer := TTimer.Create(Self);

  { Set the timer interval using the flash rate. }
  Timer.Interval := FFlashRate;
continues

  { Assign our own OnTimer event handler to the
  { TTimer OnTimer event. }
  Timer.OnTimer := OnTimer;
end;

procedure TFlashingLabel.SetFlashEnabled(AFlashEnabled : Boolean);
begin
  { Set FFlashEnabled data field. }
```

nastavlja se

Listing 20.3: `FlashingLabel.pas` (NOV I POBOLJŠAN) *nastavak*

```
    FFlashEnabled := AFlashEnabled;

    { Don't start the timer if the component is on a form
    { in design mode. Instead, just return. }

    if csDesigning in ComponentState then
      Exit;

    { Start the timer. }
    Timer.Enabled := FFlashEnabled;

    { If flashing was turned off, be sure that the label
    { is visible. }
    if not FFlashEnabled then
      Visible := True;
  end;

  procedure TFlashingLabel.SetFlashRate(AFlashRate : Integer);
  begin
    { Set the FFlashRate data field and the timer interval. }
    FFlashRate := AFlashRate;
    Timer.Interval := AFlashRate;
  end;

  procedure TFlashingLabel.OnTimer(Sender : TObject);
  begin
    { If the component is on a form in design mode,
    { stop the timer and return. }
    if csDesigning in ComponentState then begin
      Timer.Enabled := False;
      Exit;
    end;

    { Toggle the Visible property each time }
    { the timer event occurs. }
    Visible := not Visible;

    { Trigger the event if needed. Only increment }
    { the counter when the label is visible. }
    if (FFlashLimit <> 0) and Visible then begin

      { Increment the FlashCount data field. }
      Inc(FlashCount);

      { If the FlashCount is greater than or equal to   }
      { the value of the FlashLimit property, reset the }
      { FlashCount value to 0 and trigger the event.    }
      if FlashCount >= FFlashLimit then begin
```

```
      FlashCount := 0;
      DoLimitReached;
    end;
  end;
end;

procedure TFlashingLabel.DoLimitReached;
var
  Stop : Boolean;
begin
  Stop := False;
  if Assigned(FOnLimitReached) then
    FOnLimitReached(Self, Stop);
  FlashEnabled := not Stop;
end;

procedure TFlashingLabel.Click;
begin
  { Beep the speaker and then call the base class }
  { Click method for the default handling. }
  MessageBeep(-1);
  inherited;
end;

procedure Register;
begin
  RegisterComponents('Samples', [TFlashingLabel]);
end;
end.
```

Listing 20.4: FlshTstU.pas

```
type
  TMainForm = class(TForm)
    FlashBox: TCheckBox;
    Group: TRadioGroup;
    procedure FormCreate(Sender: TObject);
    procedure GroupClick(Sender: TObject);
    procedure FlashBoxClick(Sender: TObject);
  private
    { Private declarations }
    Flasher : TFlashingLabel;
    procedure OnLimitReached(Sender : TObject; var Stop : Boolean);
public
    { Public declarations }
  end;

var
  MainForm: TMainForm;
```

nastavlja se

Listing 20.4: `FlshTstU.pas` *nastavak*

```
implementation

{$R *.DFM}

procedure TMainForm.FormCreate(Sender: TObject);
begin
  Flasher := TFlashingLabel.Create(Self);
  Flasher.Parent := Self;
  Flasher.SetBounds(20, 20, 200, 20);
  Flasher.Font.Size := 16;
  Flasher.Caption := 'This is a test';
  Flasher.FlashRate := 800;
  Flasher.FlashLimit := 5;
  Flasher.OnLimitReached := OnLimitReached;
end;

procedure TMainForm.OnLimitReached(Sender : TObject; var Stop :
Boolean);
begin
  { The OnLimitReached event handler. Set Stop to }
  { true to stop flashing or leave as-is to }
  { continue flashing.}
  Stop := True;
end;

procedure TMainForm.GroupClick(Sender: TObject);
begin
  case Group.ItemIndex of
    0 :  Flasher.FlashRate := 1200;
    1 :  Flasher.FlashRate := 800;
    2 :  Flasher.FlashRate := 400;
    3 :  Flasher.FlashRate := 150;
  end;
end;

procedure TMainForm.FlashBoxClick(Sender: TObject);
begin
  Flasher.FlashEnabled := FlashBox.Checked;
end;

end.
```

I komponenta i `FlashTst` test program se nalaze u izvornom kodu iz knjige. Pri tome, izvorni kod celine komponente se nalazi u fajlu `Flashing.pas`.

Pokrenite test program da biste se uverili da komponenta radi na predviđeni način. Eksperimentišite sa programom da biste osetili kako rade događaji i procedure za njihovu obradu. Primetite da se zvučnik oglašava svaki put kada kliknete na kompo-

nentu (ako je vidljiva na ekranu u trenutku klikanja). To je zbog dinamičke metode `Click` koju smo prepisali. Dodao sam ovu funkciju da bih pokazao kako treba prepisivati događaje bazne klase.

NAPOMENA Ukoliko želite da instalirate novu i poboljšanu `FlashingLabel` komponentu na paletu sa komponentama, otvorite `DCLUSR40` paket i kliknite na taster Compile Package. Stara verzija `FlashingLabel` komponente će biti unapređena.

Potrebno je dosta vremena da biste naučili kako da dobro pravite događaje. Morate poljubiti mnogo žaba pre nego što budete dobili princa (ili princezu) kao nagradu. Drugim rečima, ne postoji zamena za iskustvo kada treba napraviti događaje. Jednostavno, treba da sednete i da to uradite. Verovatno ćete pri tome nailaziti na dosta poteškoća, ali će to biti dobra škola i zbog toga ćete sebe smatrati boljim programerom.

Zaključak

Pa, sada se nalazite u prvoj ligi. Pravljenje komponenti ne mora da bude jednostavno, ali sada, kada imate predstavu kako se to radi, svet je pod Vašim prstima. Ukoliko imate makar malo mojih osobina, naučićete da uživate u nevizuelnom programiranju isto onoliko koliko uživate u vizuelnom programiranju, koje je Vaš osnovni posao. Ukoliko Vam ovo poglavlje nije najjasnije, ne očajavajte. Možda biste trebali da ostavite knjigu na par dana i da se zatim vratite na ovo poglavlje. Možda biste trebali da pročitate nekoliko drugih publikacija koje se odnose na pravljenje komponenti, pre nego što Vam sve postane jasno. Budite uporni i videćete da to ima smisla.

Radionica

Radionica sadrži test pitanja koja Vam pomažu da učvrstite svoje razumevanje izložene materije i vežbe koje Vam pomažu da steknete iskustvo u onome što ste naučili. Možete pronaći odgovore na test pitanja u Dodatku A "Odgovori na test pitanja".

Pitanja i odgovori

P **Postoji li neki razlog zbog kojeg bih morao da znam kako se prave komponente?**

O Ne. U svojim programima možete koristiti samo komponente koje se isporučuju uz Delphi. Možda nećete nikada morati da pravite komponente.

P **Mogu li da kupim komponente?**

O Da. Postoji više različitih izvora iz kojih se mogu nabaviti biblioteke komponenti. Ove biblioteke proizvode i prodaju kompanije koju su specijalizovane za

pravljenje VCL komponenti. Takođe, postoji veliki broj besplatnih i shareware komponenti koje možete nabaviti preko Internet-a. Potražite ih.

P **Da li mogu koristiti moje komponente iz Delphi-ja u C++Builder-u?**

O Da. C++Builder ima mogućnost prevođenja i korišćenja komponenti iz Delphi-ja.

P **Da li moram koristiti `read` i `write` metoda da bih pristupao mojim osobinama?**

O Ne. Možete koristiti direktan pristup i skladištiti vrednosti osobine direktno u podatak iz klase koji je povezan sa tom osobinom.

P **Koja je prednost korišćenja `write` metoda za moje osobine?**

O Korišćenje `write` metoda dozvoljava izvršavanje drugih operacija prilikom promene vrednosti osobine. Promena vrednosti komponente obično zahteva od komponente da izvrši specifične operacije. Korišćenje `write` metoda Vam omogućava da izvršite te operacije.

P **Zbog čega bih trebao da definišem osobinu kao javnu, a ne kao objavljenu?**

O Objavljena osobina se pojavljuje u Object Inspector-u u toku pravljenja programa. Nekim osobinama ne bi trebalo pristupati u toku pravljenja programa. Te osobine treba da budu javne kako bi korisnik mogao da ih menja u toku izvršavanja, ali ne i u toku pravljenja programa.

P **Kako mogu da testiram komponentu da bih bio siguran da ona radi, pre nego što je instaliram?**

O Napravite test program i dodajte izvorni kod komponente u projekat. Napravite instancu komponente u konstruktoru glavne forme. Postavite sve osobine koje je potrebno postaviti, pre nego što komponenta postane vidljiva. U test programu biste trebali da menjate vrednosti javnih osobina da biste bili sigurni da sve radi kako treba.

P **Da li moram da pravim događaje za moje komponente?**

O To nije neophodno. Neke komponente koriste događaje, a neke ne. Nemojte praviti događaje ukoliko Vam nisu potrebni, ali se nemojte ni plašiti od pravljenja događaja, ukoliko su Vam potrebni.

Kviz

1. Da li osobina mora koristiti `write` metodu? Zašto i zašto ne?
2. Mora li osobina imati podatak iz klase sa kojim je povezana? Zašto i zašto ne?
3. Možete li napraviti komponentu proširivanjem postojeće komponente?

4. Šta se dešava ukoliko ne navede `write` deo u deklaraciji osobine (bilo `write` metodu, bilo direktan pristup)?
5. Šta znači "direktan pristup"?
6. Da li osobine moraju imati podrazumevane vrednosti? Zašto i zašto ne?
7. Da li postavljanje podrazumevane vrednosti osobini znači i automatsko postavljanje vrednosti podatka iz klase?
8. Kako se postavlja komponenta na paletu sa komponentama?
9. Kako se postavlja sličica na tasteru koji predstavlja komponentu na paleti sa komponentama?
10. Kako se pokreće događaj koji je definisao korisnik?

Vežbe

1. Pregledajte izvorni kod `FlashingLabel` komponente (Listing 20.3.) Proučite ga da biste shvatili šta se zaista dešava.
2. Izbacite `FlashingLabel` komponentu sa palete. Ponovo postavite istu komponentu na paletu.
3. Napravite test program koji koristi tri `FlashingLabel` komponente sa različitim intervalima blinkanja.
4. Izmenite sličicu koja se nalazi na tasteru koji predstavlja `FlashingLabel` komponentu na paleti u neku koju ćete sami napraviti.
5. Napišite `write` metodu za `FlashLimit` osobinu `FlashingLabel` komponente, kako korisnik ne bi mogao da unese vrednost veću od 100.
6. Izmenite `OnLimitReached` događaj `FlashingLabel` komponente tako da postane notifikacioni događaj. (Savet: Koristite `TNotifyEvent`.)
7. **Posebna vežba:** Samostalno napravite jednu komponentu.
8. **Posebna vežba:** Testirajte Vašu novu komponentu i postavite je na paletu sa komponentama.

Dan 21

Delphi i C++Builder

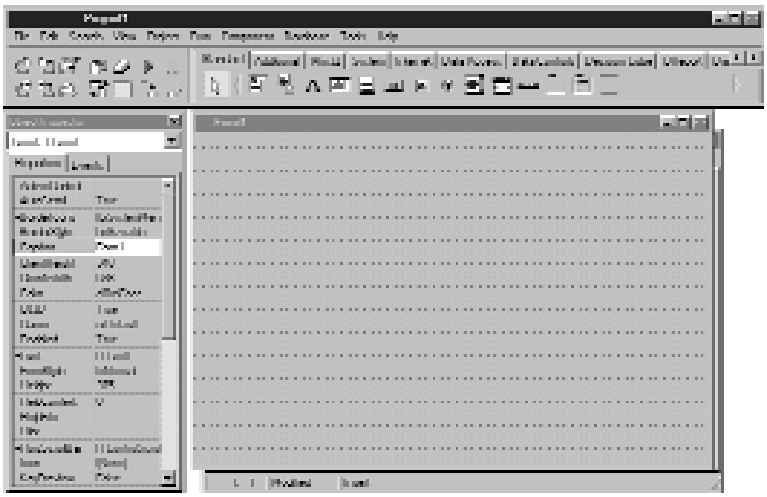
Kada je, pre par godina, Borland predstavio Delphi 1, to je bio veliki uspeh. I sada, u verziji 4, Delphi predstavlja veliki uspeh. Nešto posle predstavljanja Delphi-ja 2, ljudi iz kompanije Borland su odlučili da iskoriste uspeh Delphi-ja u pravljenju RAD razvojnog okruženja za C++. Borland je, dakle, iskoristio ovaj uspeh u više pravaca. Jedan od načina, na koji su to i uspeli da urade je korišćenje nekih delova Delphi-ja u C++Builder-u. U ovom poglavlju ćete naučiti koliko su Delphi i C++Builder slični i koliko se razlikuju. Naučićete i kako da razmenjujete kod između Delphi-ja i C++Builder-a i kako da konvertujete kod iz C++Builder-a u Delphi.

Sličnosti između Delphi-ja i C++Builder-a

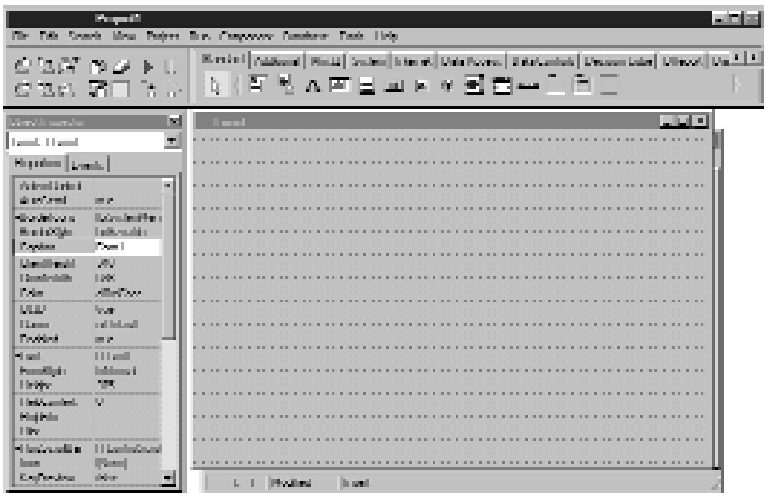
Delphi i C++Builder su više slični nego što se razlikuju. U ovom odeljku ćemo navesti te sličnosti.

Razvojna okruženja (IDE-ovi)

Ukoliko ste koristili i Delphi i C++Builder, vertovatno ste bili iznenađeni koliko su njihova razvojna okruženja slična. Vi, koji niste koristili i Delphi i C++Builder (čak, iako jeste), pogledajte slike 21.1 i 21.2. Jedna od ovih slika prikazuje izgled IDE-a Delphi-ja 3, a druga IDE-a C++Builder-a 3. Sklonio sam ikone i imena programa iz naslovne trake prozora, tako da ne možete baš odmah uočiti koji je koji. Možete li da nađete neku razliku?



Slika 21.1
Delphi ili
C++Builder?



Sliks 21.2
Delphi ili
C++Builder?

Predajete se? Slika 21.1 prikazuje Delphi-jev IDE, dok slika 21.2 prikazuje IDE C++Builder-a. Ako ste koristili i Delphi i C++Builder, mogli ste da primetite da Delphi-jev Object Inspector prikazuje vrednosti osobina kao True i False, dok Object Inspector C++Builder-a prikazuje vrednosti osobina sa `true` i `false`.

Smisao ove vežbe je bio da uočite da su razvojna okruženja Delphi-ja 3 i C++Builder-a 3 praktično identična. (Primetili ste da sam za potrebe ove vežbe koristio Delphi 3, a ne Delphi 4. Razlog je što sam želeo da uporedim Delphi sa C++Builder-om 3 koji je poslednja verzija C++Builder-a u trenutku pisanja ove knjige i to je verzija koja odgovara Delphi-ju 3).

Kada počnete sa istraživanjem menija, primetićete neke razlike. To je i logično, ali ova dva okruženja su toliko slična, da ukoliko znate kako da radite sa jednim, znate i sa drugim. To je očigledna prednost. Recimo da je Vaša kompanija do sada koristila Delphi i da sada želi da doda C++ u svoj skup alata. Korišćenjem C++Builder-a kao C++ razvojnog okruženja štedite vreme i novac zato što vaši programeri ne moraju da uče kako se radi sa novim razvojnim okruženjem. To nas vodi do sledeće sličnosti između Delphi-ja i C++Builder-a: do VCL-a.

VCL (Visual Component Library)

Delphi i C++Builder nisu samo slični po pitanju razvojnih okruženja, već i po tome što dele istu biblioteku komponenti u VCL-u. VCL je napisan u Object Pascal-u, ali i Delphi i C++Builder koriste isti VCL (sa nekim manjim izmenama na koje ću ukazati kasnije, u odeljku "Unapređenja VCL-a").

Pre nekog vremena se u Borland-ovim konferencijama vodila rasprava između C++ programera na temu biblioteke komponenti koja je napisana u Pascal-u, a koristi se u C++ razvojnom okruženju. Neki C++ programeri, jednostavno, ne mogu da prežale činjenicu da je VCL napisan u Object Pascal-u. Većina programera, ipak, koristi razuman pristup problemu i ne insistira na programskom jeziku u kome je napisan VCL. Oni jednostavno žele da obave svoj posao u okviru raspoloživog vremena i budžeta. Ovim programerima je C++Builder toliko povećao produktivnost da i ne razmišljaju o programskom jeziku u kome je napisan VCL.

Činjenica da i Delphi i C++Builder koriste isti princip razvoja je takođe ogromna prednost, ukoliko koristite i Delphi i C++Builder. Ako zanemarimo razlike u sintaksi između Pascal-a i C++-a, VCL komponente se i u jednom i u drugom okruženju koriste na isti način. Pogledajte sledeći Delphi kod:

```
if OpenDialog1.Execute then
  Memo1.Lines.LoadFromFile(OpenDialog1.FileName);
```

Sada pogledajte ekvivalentni C++ kod:

```
if (OpenDialog1->Execute())
  Memo1->Lines->LoadFromFile(OpenDialog1->FileName);
```

Kao što možete videti, ne postoji razlika u korišćenju VCL-a. Jedina razlika koju možete videti je, u stvari, razlika između sintakse C++-a i sintakse Object Pascal-a. To znači da ne morate učiti novi princip programiranja, ukoliko želite da pređete sa Delphi-ja na C++Builder i obrnuto. To za kompanije sa velikim brojem programera znači da mogu maksimalno iskoristiti programersko znanje i iskustvo. Čak iako radite sami, poznavanje i Delphi-ja i C++Builder-a je veoma korisno.

Fajlovi sa formama

U Delphi-ju i C++Builder-u se koriste isti fajlovi sa formama. Možete napraviti formu u Delphi-ju i zatim je ponovo iskoristiti u C++Builder-u. Kasnije, u odeljku "Ponovno korišćenje formi", ću objasniti kako to da uradite. Iako je pravljenje formi i u Delphi-ju i u C++Builder-u jednostavno, zahteva neko vreme, posebno u slučaju komplikovanih formi. Ponovnim korišćenjem formi ne morate ponavljati posao koji je već odrađen.

Paketi

I Delphi i C++Buider koriste pakete. U gotovo svim slučajevima se paketi napravljeni u Delphi-ju mogu koristiti u C++Builder-u. Ti paketi se, verovatno, moraju ponovo prevesti u C++Builder-u, ali je to trivijalan zadatak. Mogućnost C++Builder-a da može da koristi Delphi-jeve pakete znači da Vam na raspolaganju stoji puno komercijalnih, shareware i besplatnih VCL komponenti. VCL komponente su, uglavnom, pravljene u Delphi-ju, tako da se mogu koristiti i u Delphi-ju i u C++Builder-u. Za sada nije moguće koristiti pakete pravljene za C++Builder u Delphi-ju, ali će se to možda promeniti u nekoj od sledećih verzija ova dva proizvoda.

Razlike između Delphi-ja i C++Builder-a

Iako su Delphi i C++Builder veoma slični, postoje i razlike. Najveći broj tih razlika je rezultat razlike između C++-a i Object Pascal-a.

Ostale razlike su rezultat onoga što ja nazivam "efektom preskakanja". U početku je postojao Delphi 1. Posle toga je, naravno, došao Delphi 2. Pošto je Delphi 2 već izašao, Borland je počeo da razvija C++Builder. C++Builder 1 je izašao posle Delphi-ja 2 (koji je izašao godinu dana pre C++Builder-a 1). C++Builder 1 je imao veliki deo mogućnosti Delphi-ja 2. Praktično, to je bio plagijat, pošto C++Builder 1 nije ponudio ništa novo u odnosu na Delphi 2. Ubrzo posle C++Builder-a 1, pojavio se i C++Builder 3.

C++Builder 3 (Borland je preskočio verziju 2 C++Builder-a kako bi ga uskladio sa Delphi-jem 3). C++Builder 3 je izašao posle Delphi-ja 3, dakle, isto kao što je C++Builder 1 izašao posle Delphi-ja 2. Ipak, C++Builder 3 je imao neke mogućnosti koje nisu postojale u Delphi-ju 3 (na primer, Project Manager). Delphi 4 je nasledio neke mogućnosti C++Builder-a 3 ali je, naravno, dodao i veliki broj novih. C++Builder 4 će verovatno sadržati najveći broj mogućnosti Delphi-ja 4, ali će imati i neke nove i tako dalje. Ovo kašnjenje će se verovatno nastaviti još neko vreme i tako će svaka nova verzija (bez obzira da li se radi o Delphi-ju, ili o C++Builder-u) imati sve mogućnosti prethodne, ali će imati i neke nove.

Hajde da se sada koncentrišemo na razlike između Delphi-ja i C++Builder-a.

Programski jezik

Najočiglednija razlika između Delphi-ja i C++Builder-a je što C++Builder generiše C++ kod, dok Delphi generiše Object Pascal kod što, neizbežno, dovodi do nekih razlika i u samim razvojnim okruženjima. Na primer, Delphi ima stavku u File meniju koja se zove Use Unit. C++ koristi fajlove zaglavlja (.H, ili .HPP fajlove) za svaku celinu, tako da se ova stavka menija u C++Builder-u zove Include Unit Hdr. Ovakve razlike u razvojnim okruženjima postoje iz očiglednih razloga.

Ekstenzije fajlova

Delphi i C++Builder koriste različite ekstenzije za projektne fajlove i za fajlove podrške. Ovo je delom rezultat razlike između programskih jezika. Različite ekstenzije fajlova Vam pružaju mogućnost da lako razlikujete fajlove iz Delphi-ja i iz C++Builder-a. Tabela 21.1 prikazuje različite elemente projekta i analogne ekstenzije fajlova u Delphi-ju i C++Builder-u.

Tabela 21.1: Ekstenzije fajlova u Delphi-ju i C++Builder-u

Element	Delphi	C++Builder
Fajl projekta	.DPR	.BPR
Fajl grupe projekata	.BPG	.BPG
Fajl sa izvornim kodom	.PAS	.CPP
Fajl sa zaglavljem	Nema	.H ili .HPP
Fajl sa formom	.DFM	.DFM
Prevedeni binarni fajl	.DCU	.OBJ
Prevedeni resursi	.RES ili .DCR	.RES
Parametri razvojnog okruženja	.DSK	.DSK
Opcije projekta	.DOF	Nema
Paketi izvornih kodova	.DPK	.BPK
Prevedeni paketi	.BPL	.BPL

Primetite da u nekim slučajevima i Delphi i C++Builder koriste iste ekstenzije fajlova.

Razvojno okruženje (IDE)

Razvojno okruženje Delphija 4 je potpuno novo. U stvari, sledeće osobine razvojnog okruženja postoje u Delphi-ju 4, ali ne i u C++Builder-u 3:

- Meni, paleta sa alatkama i paleta sa komponentama se mogu pomerati uz ivice prozora,
- Postoje nove sličice na tasterima u paleti sa alatkama razvojnog okruženja,

- Postoje bit mape na najbitnijim stavkama u menijima,
- Prozori sa alatima se mogu pomerati uz ivice prozora.

Neka od ovih unapređenja su kozmetičke prirode (pa, ne boli ukoliko se posle nekog vremena da novi izgled programima). Ostala unapređenja, kao što su meniji, palete sa alatkama i palete sa komponentama, koje se mogu pomerati uz ivice prozora, omogućavaju i povećanje produktivnosti.

Editor koda (Code Editor)

U editoru koda Delphi-ja 4 postoje neke mogućnosti koje se ne nalaze u C++Builder-u. U suštini, sledeće mogućnosti se mogu naći u Delphi-ju 4:

- Kretanje kroz module,
- Kompletiranje koda,
- Kompletiranje deklaracija klasa,
- Prikazivanje parametara u kodu.

Kretanje kroz module je osobina koja je specifična za Object Pascal, tako da i nije čudno što se ne nalazi u C++Builder-u, ali zašto C++Builder nema kompletiranje koda i prikazivanje parametara u kodu? Delphi ponovo prevodi aktivnu celinu svaki put kada se pozovu, bilo kompletiranje koda, bilo prikazivanje parametara u kodu. Ovo je moguće zato što je Object Pascal prevodilac neverovatno brz, dok je C++ kodu potrebno mnogo duže vreme za prevođenje. Iz tog razloga nije baš jednostavno napraviti kompletiranje koda i prikazivanje parametara u kodu korišćenjem te tehnike i u C++Builder-u. Postoje i drugi načini da se to uradi i verovatno je da će sledeće verzije C++Builder-a imati i kompletiranje koda i prikazivanje parametara u kodu.

C++Builder nema kompletiranje deklaracija klasa zato što je to novost u Delphi-ju 4. Sledeća verzija C++Builder-a će verovatno imati kompletiranje deklaracija klasa.

Code Explorer

Code Explorer je, takođe, novost u Delphi-ju 4, pa se zbog toga ne nalazi u C++Builder-u 3. To je još jedna od stvari koju očekujem da ću videti u sledećoj verziji C++Builder-a.

Unapređenja VCL-a

Delphi 4 sadrži neka unapređenja u VCL-u koja, još uvek, ne postoje u C++Builder-u. U suštini, sledeće VCL klase su nove u Delphi-ju 4:

- `TActionList`
- `TControlbar`
- `TDateTimePicker`
- `TPageScroller`
- `TSimpleMail`
- Dodatne QuickReport komponente: `TQRTextFilter`, `TQRCSVFilter`, `TQRHTMLFilter`.
- Dodatne klijent/server MIDAS komponente za DCOM konekcije, CORBA konekcije, OLE Enterprise konekcije i druge.

Kao i mnoge druge prednosti Delphi-ja 4, i ove će se, gotovo sigurno, naći u sledećoj verziji C++Builder-a.

C++Builder može da prevodi Pascal celine

C++Builder može da prevodi Pascal celine, isto kao što može da prevodi i C++ izvorni kod. To znači da možete dodati fajl sa Pascal izvornim kodom direktno u projekat u C++Builder-u i C++Builder će ga prevesti i povezati sa ostalim celinama u .EXE fajl. Možete dodati Delphi-jevu formu i celinu u projekat u C++Builder-u i ta forma će biti pozivana kao i obična C++Builder forma. C++Builder može da prevodi Pascal celine, ali Delphi ne može da prevodi fajlove sa izvornim kodom iz C++Builder-a. Delphi 4, međutim, može da napravi C++ objektne fajlove (.obj) iz Object Pascal izvornog koda.

Podrška za ActiveX

Podrška pravljenju ActiveX kontrola u C++Builder-u je nešto drugačija od one u Delphi-ju. Konkretno, C++Builder ActiveX kontrole mogu koristiti ActiveX Template Library (ATL), a to je posebna biblioteka za razvoj ActiveX kontrola.

Delphi prevodi brže i proizvodi manje EXE fajlove

Delphi programi će se uvek brže prevoditi od C++Builder programa. Pascal se prevodi mnogo brže od C++-a zato što nije toliko komplikovan kao C++. Pomoću prekompajliranih zaglavlja i inkrementalnog povezivanja se i proces generisanja C++Builder programa može ubrzati, ali je prevođenje Delphi programa i dalje neuporedivo brže.

Takođe, Delphi programi su uvek manji od C++Builder programa. To je posledica mnogih faktora, ali najviše zbog toga što je VCL napisan u Object Pascal-u. Zbog toga, C++Builder programi sadrže i C++ izvršnu biblioteku i Object Pascal izvršnu

biblioteku. C++ i Pascal drugačije rade sa izuzecima i imaju različite tipove informacija u izvršnom kodu što, sve zajedno, rezultuje većim C++Builder programima.

Konvertovanje iz Delphi-ja u C++Builder

U jednom trenutku ćete, možda, morati da vršite konverziju programa iz Delphi-ja u C++Builder i obrnuto. Ja sam do sada uradio puno takvih konverzija i iako zahtevaju dosta vremena, uopšte nisu komplikovane. (Usput, kada vršim konverziju, paralelno koristim i Delphi i C++Builder.) U sledećem odeljku ću objasniti samo konverziju iz Delphi-ja u C++Builder. Obrnuti proces je, u stvari, samo varijacija ovog procesa.

Zbog čega objašnjavam samo konverziju iz Delphi-ja u C++Builder, a ne i obrnuti proces? Uglavnom zato što Delphi ima ve"u bazu korisnika (duže se nalazi na tržištu) i zato što se konverzije uglavnom i rade iz Delphi-ja u C++Builder. To ne mora da znači da programeri napuštaju Delphi i prelaze na C++Builder. Ja sam do sada uradio preko 200 ovakvih konverzija za TurboPower Sotfware kompaniju (tamo ja radim). U tom slučaju sam konvertovao primere naših proizvoda koji su bili napisani u Delphi-ju u C++Builder. To je bilo neophodno kada smo počeli da dajemo podršku za korišćenje naših komponenti u C++Builder-u.

Konverzija projekta iz Delphi-ja u C++Builder zahteva dva osnovna koraka. Prvi je kopiranje formi iz Delphi programa u C++Builder projekat. Drugi korak je konverzija Delphi koda. Sada ćemo detaljnije pogledati ova dva koraka.

Kopiranje Delphi-jevih formi

Prva stvar koju treba da uradite je da iskopirate Delphi-jeve forme u C++Builder program. U osnovi, formati fajlova sa formama u Delphi-ju i C++Builder-u su isti. Međutim, postoji barem jedna značajna razlika na koju biste trebali da obratite pažnju.

Očigledno je da fajl sa formom sadrži poziciju i veličinu same forme, ali i svake od komponenti. Ono što možda i nije toliko očigledno je da fajl sa formom sadrži i informacije o događajima i procedurama za njihovu obradu. Konkretno, u fajlovima sa formama postoje opisi svih procedura za obradu svih događaja napravljenih za samu formu i za svaku od komponenti na formi. U Delphi-jevim fajlovima sa formama pos toje pokazivači na procedure za obrade događaja u Pascal-u.

U fajlovima sa formama u C++Builder-u postoje pokazivači na procedure za obrade događaja u C++-u. Naravno, moraćete da obrišete Pascal reference, kako biste mogli da koristite forme u C++Builder-u. Nije neophodno da detaljno razumete ovaj proces, ali ćete se sa ovime sretati prilikom konverzije formi iz Delphi-ja u C++Builder.

Potrebno je uraditi sledeće da bi se iskopirala glavna forma Delphi programa:

1. Otvorite projekat u Delphi-ju i zabeležite ime fajla u kome se nalazi glavna forma, kao i sadržaj njene `Name` osobine.

2. Prebacite se u C++Builder i aktivirajte glavnu formu. Izmenite njenu `Name` osobinu u ime glavne forme u Delphi programu.

3. Sačuvajte C++Builder projekat pod istim imenom kao i u Delphi-ju. Glavnu formu sačuvajte u fajlu koji će imati isto ime kao i u Delphi-ju (naravno, bez `.PAS` ekstenzije).

4. Prebacite se u Windows Explorer. Iskopirajte fajl sa glavnom formom (.DFM fajl) iz direktorijuma gde se nalazi Delphi projekat u direktorijum gde se nalazi C++Builder projekat. Pazite da fajl samo iskopirate, a ne da ga prebacite. Explorer će Vas upozoriti da fajl sa takvim imenom već postoji u odredišnom direktorijumu. Kliknite na Yes da biste prepisali fajl u direktorijumu sa C++Builder projektom. Ukoliko ne dobijete upozorenje, znači da ste uradili nešto pogrešno prilikom snimanja celine u C++Builder-u.

5. Vratite se u C++Builder. Prikazaće se dijalog-prozor na kome će pisati: `Module XXX.CPP's time/date has changed. Reload?` Kliknite na Yes, kako biste ponovo učitali formu. Kada se forma učita, sadržaće komponente koje su se nalazile na formi u Delphi-ju.

6. Aktivirajte formu i izberite Edit➥Select All iz glavnog menija C++Builder-a da biste izabrali sve komponente na formi. Sada izaberite Edit➥Cut iz glavnog menija a zatim odmah izaberite Edit➥Paste. Na ovaj način ste osigurali da će se deklaracije pojedinačnih komponenti naći u fajlu zaglavlja glavne forme.

U ovom trenutku morate obrisati sve reference na Delphi-jeve procedure za obradu događaja iz fajla sa formom C++Builder-a. To je jednostavno:

1. Izaberite File➥Save iz glavnog menija C++Builder-a, ili kliknite na Save File taster na paleti sa alatkama.

2. C++Builder će prikazati poruku za svaku proceduru za obradu događaja koja je vezana za formu, kao što se vidi na slici 21.3. Kliknite na taster Yes, svaki put kada se pojavi poruka, da biste obrisali sve reference na procedure za obradu događaja. Možda ćete morati da puno puta kliknete na Yes da biste obrisali sve reference. (Ja sam jednom morao da obrišem 100 procedura za obradu događaja iz samo jedne forme!)

Slika 21.3
Brisanje procedure za obradu događaja iz forme

U ovom trenutku ste završili sa kopiranjem forme. Sada se možete posvetiti konverziji koda.

NAPOMENA Morate ponoviti postupak kopiranja forme za svaku formu koja se nalazi u Delphi programu.

Konverzija koda

Konverzija koda iz Delphi-ja u C++Builder je mnogo teža od kopiranja forme. Postoji mnogo načina da se to uradi, ali ću objasniti onaj koji ja koristim. Prvo, pronađite sve promenljive i metode koje je napravio programer, a ne sam Delphi i to na sledeći način:

1. Pronađite deklaraciju klase glavne forme u Delphi celini.
2. Zabeležite sve promenljive i metode koje se nalaze u `private` i `public` delovima.
3. Iskopirajte svaku od ovih deklaracija u Clipboard. Na primer, deo Delphi-jeve deklaracije klase bi mogao da izgleda ovako:

```
private
  { Private declarations }
  DrawColor : TColor;
  procedure DoDrawing;
  function  SetDrawColor : Integer;
```

   U ovom slučaju biste iskopirali deklaracije za `DrawColor`, `DoDrawing` i `SetDrawColor`.
4. Vratite se u C++Builder. Prikažite fajl zaglavlja glavne forme u editoru koda. Pronađite `private` deo i vratite kod iz Clipboard-a.
5. Konvertujte deklaracije koje ste vratili u sintaksu C++-a. Na primer, deklaracije iz primera u koraku 3 bi trebale da izgledaju ovako:

```
private:  // User declarations
  TColor DrawColor;
  void DoDrawing();
  int SetDrawColor();
```

6. Vratite se u Delphi. U `implementation` delu pronađite definicije svih metoda koje su deklarisane u deklaraciji klase (u ovom slučaju, `DoDrawing` i `SetDrawColor`). Iskopirajte sve metode u Clipboard.
7. Vratite se u C++Builder. Vratite iz Clipboard-a metode koje ste iskopirali u koraku 6, u fajl sa izvornim kodom forme. Konvertujte prve linije svih metoda (zaglavlja metoda) iz Pascal u C++ sintaksu. Za sada ne morate brinuti o telima funkcija i begin i end naredbama. To ćete popraviti kasnije.

Kopiranje procedura za obradu događaja

Sledeći korak je kopiranje koda iz procedura za obradu događaja iz Delphi celine u C++Builder celinu. Najbolji način koji sam pronašao za rešavanje ovog zadatka je pozicioniranje na početak `implementation` dela Delphi celine i pretraživanje na dole. Evo šta treba da uradite kada naiđete na proceduru za obradu događaja u Delphi kodu:

1. Zabeležite ime te procedure. Ukoliko je njeno ime `Button1Click`, znate da se radi o proceduri za obradu `OnClick` događaja komponente pod imenom `Button1`.
2. Iskopirajte kod iz procedure u Clipboard.
3. Vratite se u C++Builder. Napravite proceduru za obradu tog događaja.
4. Vratite kod iz Clipboard-a u telo procedure.

Ponovite korake od 1 do 4 za svaku proceduru za obradu događaja u Delphi celini. Kada završite sa tim, imaćete nekoliko procedura za obradu događaja u svom C++Builder projektu.

Korišćenje Replace Text dijalog-prozora

Procedure za obradu događaja u Delphi-ju sadrže Pascal kod, tako da ćete morati da ga konvertujete u C++ kod. Srećom, dobar deo konverzije možete uraditi pomoću Replace Text dijalog-prozora u C++Builder-u. Tabela 21.2 prikazuje delove Pascal sintakse koje tražite i delove C++ sintakse u koje ćete menjati pronađeni tekst. U većini slučajeva ćete želeti da vršite Find i Replace operacije nad tekstom, koristeći redosled prikazan u tabeli 21.2.

Tabela 21.2: Tekst koji treba zameniti prilikom konverzije iz Delphi-ja u C++Builder

Opis	Find tekst	Replace text
Operator jednakosti	`·=·`	`·==·`
Operator dodele	`:=`	`=`
Operator nejednakosti	`<>`	`!=`
Operator pripadnosti	`.`	`->`
Navodnik string-a	`'`	`"`
Početak komentara	`{`	`//`
Kraj komentara	`}`	(Ništa)
Pascal ključna reč `True`	`True`	`true`
Pascal ključna reč `False`	`False`	`false`
`if` struktura	`if·`	`if·(`
Početak bloka	`begin`	`{`
Kraj bloka	`end;`	`}`

nastavlja se

Tabela 21.2: Tekst koji treba zameniti prilikom konverzije iz Delphi-ja u C++Builder *nastavak*

Opis	Find tekst	Replace text
Kraj bloka (druga mogućnost)	`end`	`}`
Pascal struktura `then`	`then`	`)·`
Pascal struktura `do`	`·do·`	(Ništa)
Pascal struktura `not`	`not·`	`!`
Pascal ključna reč `nil`	`nil`	`NULL`
Pascal struktura `case`	`case·`	`switch·(`
Pascal struktura `case`	`·of·`	`)·{`
Pascal ključna reč `Self`	`Self`	`this`

Kada koristite Find i Replace, trebali biste da koristite Replace All opciju, ali biste, takođe, trebali da budete i vrlo oprezni. Na primer, Vi ne želite da zamenite svako pojavljivanje niza znakova `->`, gledano od početka fajla, zato što se prvih nekoliko linija svakog C++Builder fajla sa izvornim kodom sastoje od `include` direktiva sa imenima fajlova.

Pazite da zamenu Pascal komentara (koji počinju sa { i završavaju se sa }) uradite pre zamene `begin` i `end` naredbi. Takođe, kada zamenjujete reči kao što je end, trebali biste da uključite opciju Whole Words Only iz Replace Text dijalog prozora. Sada ste sigurni da se pojedinačni karakteri u okviru dužih reči neće obrisati. Imajte u vidu da pojedine Find i Replace operacije mogu imati neželjenih efekata (kao što je zamena tačke koja razdvaja ime fajla i ekstenziju sa `->`).

Razmislite o pravljenju Microsoft Word makroa koji će za Vas obaviti Find i Replace operacije navedene u tabeli 21.2. Ukoliko morate da prevedete puno celina, to je način da sačuvate vreme.

Pošto ste izvršili sve Find i Replace operacije, imaćete fajl koji je mešavina Pascal-a i C++-a. Lakši deo je završen i ostatak koda ćete morati ručno da konvertujete. Morate dobro poznavati i jedan i drugi jezik da biste mogli dobro da konvertujete iz Pascal sintakse u C++ sintaksu. Odavde radite sami, ali ću Vam ukazati na nekoliko potencijalnih problema na koje možete naići.

Pascal struktura `with`

Za početak, u C++-u ne postoji struktura koja bi bila ekvivalent Pascal-ovoj `with` strukturi. Za primer uzmite sledeći kod:

```
with MyForm do begin
  Width   := 200;
  Height  := 500;
  Caption := 'Hello there';
end;
```

Kada budete konvertovali ovakav kod u C++ moraćete da referencirate svaku osobinu:

```
MyForm->Width = 200;
MyForm->Height = 500;
MyForm->Caption = "Hello there";
```

Pascal struktura as

As je još jedna Pascal struktura koja se mora posebno konvertovati. Često viđate kod sličan ovome u Delphi programima:

```
with Sender as TButton do
  Click;
```

U C++Builder-u, kod bi trebao da izgleda ovako:

```
TButton* button = dynamic_cast<TButton*>(Sender);
if (button)
  button->Click();
```

Rad sa stringovima

Stringovi su još jedan oblast koja zahteva posebnu pažnju. Pascal ima funkcije za manipulaciju stringovima koje rade sa `string` tipom podataka. C++Builder, sa druge strane, ima `AnsiString` klasu, koja sadrži sopstvene funkcije za rad sa stringovima. Pogledajte, na primer, ovaj Pascal kod:

```
X := StrToInt(Edit.Text);
```

U C++Builder-u, kod bi trebao da izgleda ovako:

```
X = Edit->Text.ToInt();
```

Konverzija skupova

Kao i kod stringova, C++Builder-ov ekvivalent Pascal-ovih skupova su C++ klase. U slučaju skupova, C++ klasa se naziva `Set`. Sledeći kod prikazuje konverziju Pascal-ove sintakse skupova u C++Builder `Set` klasu. Koristio sam `Style` član Font osobine u ovoj ilustraciji. Prvo, pogledajte Pascal kod:

```
{Clear the set.}
Font.Style := [];
{ Add the bold and italics styles. }
Font.Style := Font.Style + [fsBold, fsItalic];
{ Remove the italics style. }
Font.Style := Font.Style - [fsItalic];
{ See if the set contains the fsBold style. }
if fsBold in Font.Style then
  { Code here if the font is bold. }
```

Zatim, pogledajte ekvivalentni C++Builder kod:

```
// Clear the set.
Font->Style.Clear();
// Add the bold and italics styles.
Font->Style = Font->Style << fsBold << fsItalic;
// Remove the italics style.
Font->Style = Font->Style >> fsItalic;
// See if the set contains the fsBold style.
if (Font->Style.Contains(fsBold))
  // Code here if the font is bold.
```

Ne mogu objasniti baš svaku razliku između Object Pascal-a i C++-a u ovom poglavlju, ali će Vam ovih nekoliko primera pomoći na samom početku. Ukoliko koristite C++Builder, mogli biste da pogledate moju knjigu za C++Builder koju je izdao SAMS. Knjiga se zove *Teach Yourself* C++*Builder 3 in 21 Days* (ISBN 0-672-31266-2). Ova knjiga, takođe, detaljno objašnjava neke razlike između Object Pascal-a i C++-a.

Ponovno korišćenje formi

Ukoliko ne želite, ne morate konvertovati Delphi forme u C++. Možete koristiti Delphi forme u C++Builder-u onakve kakve jesu. Jednostavno dodajte `.PAS` fajl sa formom u svoj C++Builder projekat. C++Builder će napraviti fajl zaglavlja za Delphi celinu koji možete koristiti u svakoj C++Builder celini koja koristi tu Delphi formu.

Iako možete dodati Delphi formu u C++Builder projekat, ne možete je menjati pomoću C++Builder Form Designer-a. Bilo koja izmena, koju želite da napravite, se mora izvršiti iz Delphi razvojnog okruženja. Međutim, možete menjati formu u tekstualnom formatu i iz C++Builder razvojnog okruženja. Izaberite View As Text iz kontekstnog menija C++Builder Form Designer-a da biste mogli da menjate formu u tekstualnom formatu.

Kratak pregled

Delphi i C++Builder nisu toliko konkurentski proizvodi koliko se međusobno dopunjuju. Ukoliko znate da programirate sa C++Builder-om, učenje Delphi-ja je relativno jednostavno. Prelazak sa Delphi-ja na C++Builder je komplikovaniji zbog kompleksnosti samog C++-a. Međutim, ukoliko odlučite da pređete sa Delphi-ja na C++Builder, možete biti sigurni da nećete morati da učite razvojno okruženje. Bez sumnje, ukoliko budete stručnjak i za Delphi i za C++Builder, bićete mnogo vredniji programer.

Radionica

Radionica sadrži test pitanja koja Vam pomažu da učvrstite svoje razumevanje izložene materije i vežbe koje Vam pomažu da steknete iskustvo u onome što ste naučili. Možete pronaći odgovore na test pitanja u Dodatku A "Odgovori na test pitanja".

Pitanja i odgovori

P **Mogu li koristiti Pascal celine u C++Builder projektu?**

O Da. Dodajte Pascal celinu u projekat, kao što biste dodali i C++ celinu. Morate staviti Pascal celine u Project Manager-u iznad C++ celina koje koriste kod iz njih.

P **Mogu li koristiti C++ celine u Delphi projektima?**

O Ne. Možete koristiti Pascal celine u C++Builder-u, ali ne i obrnuto.

P **Kao programer, radoznao sam po pitanju nečega. Da li su Delphi i C++Builder razvojna okruženja napravljena korišćenjem istog koda?**

O Da. Iako Delphi i C++Builder imaju očiglednih razlika, imaju i mnogo sličnosti, tako da Borland koristi jedinstvenu osnovu za pravljenje oba razvojna okruženja.

P **Zbog čega se moji Delphi projekti toliko brže prevode nego moji C++Builder projekti?**

O Zato što je Object Pascal jednostavniji od C++-a, pa je potrebno manje vremena za prevođenje.

P **Čuo sam da ukoliko znam C++Builder, mogu lako da naučim i Delphi. Da li je to tačno?**

O Pa, ne baš. U odnosu na stari Pascal, Delphi-jev Object Pascal je dosta komplikovaniji. Iako je prelaz sa C++Builder-a na Delphi jednostavniji nego obrnuti proces, to i dalje nije nešto čemu bi trebalo olako prilaziti.

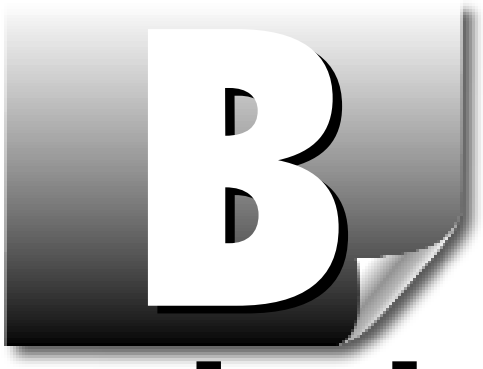
Kviz

1. Da li Delphi i C++Builder projektni fajlovi imaju iste ekstenzije?
2. Da li Delphi i C++Builder fajlovi sa formama imaju iste ekstenzije?
3. Da li možete koristiti pakete od nezavisnih proizvođača komponenti i u Delphi-ju i u C++Builder-u?
4. Da li možete otvoriti Delphi-jevu formu u C++Builder-u?
5. Možete li menjati Delphi-jevu formu pomoću C++Builder Form Designer-a?

6. Da li možete koristiti C++Builder izvornu celinu u Delphi-ju?
7. Šta je bolje? Delphi, ili C++Builder?

Vežbe

1. Ukoliko imate C++Builder, uzmite jedan primer iz Delphi-jevog Demos direktorijuma i konvertujte ga u C++Builder.
2. Uzmite jedan primer iz C++Builder-ovog `Examples` direktorijuma i konvertujte ga u Delphi.
3. Napravite pauzu. Upravo ste završili 21. dan.



Dodatak

Sadržaji na Internet-u vezani za Delphi

Delphi se na tržištu nalazi već nekoliko godina, tako da postoji nekoliko veoma dobrih mesta na Internet-u koja sadrže informacije o njemu. Ovi izvori se mogu svrstati u četiri kategorije: komercijalni Web sajtovi (uključujući i Borland-ov Web sajt), korisnički Web sajtovi, News grupe i publikacije. Sledi kratak opis ovih kategorija.

Obavezno posetite moju stranicu na kojoj se nalaze ispravke teksta iz ove knjige. Ona se nalazi na `www.home.turbopower.com/~kentr/delphi`. Posetite i stranicu na kojoj se nalaze izvorni kodovi iz knjige i neki primeri (`http://www.mcp.com/info`).

Kompanija INPRISE

Kompanija INPRISE (nekadašnji Borland International) je najbolji izvor podataka za Borland-ove proizvode. INPRISE Web sajt (`www.inprise.com`) sadrži najsvežije informacije o Delphi-ju, najnovije zakrpe, FAQ stranice, kao i listu proizvođača koji prodaju komponente za Delphi i pomoćne programe. Obavezno pregledajte ovaj Web sajt s vremena na vreme, kako biste bili u toku sa najnovijim informacijama vezanim za Delphi.

Komercijalni Web sajtovi

Komercijalni Web sajtovi pripadaju kompanijama koje prodaju komponente za Delphi. Tabela B.1 sadrži delimičnu listu Web sajtova (bez posebnog reda) koje biste, možda, želeli da posetite.

Tabela B.1: Komercijalni Web sajtovi

Ime kompanije (Web adresa)	Opis/sadržaj
TurboPower Software	TurboPower je najstarija kompanija koja proizvodi alate za Borland-(www.turbopower.com) ove proizvode. U poslu se nalazi već 10 godina. Turbo Power je specijalizovan za Delphi i C++Builder komponente pomoćne programe. Njihov AsyncProfessional je najbolji paket komponenti za serijsku komunikaciju.
Raize Software Solutions (www.raize.com)	VCL komponente opšte namene.
DevSoft (www.dev-soft.com)	Kompanija koja proizvodi IP*Works, Internet komponente za C++Builder i Delphi.
Out and About Productions (www.o2a.com)	DTalk komponente za upravljanje glasom.
Skyline Tools (www.imagelib.com)	Kompanija koja proizvodi ImageLib, biblioteku komponenti za rad sa slikama.
QuSoft AS (www.qusoft.com)	Proizvođači QuickReport-a.
TeeChartPro (www.teemach.com)	Proizvođači TeeChart komponente.
SCT Associates. Inc. (www.sct-associates.com)	Proizvođači ACE Reporter-a.
Enginnering Objects International (www.inconresearch.com/eoi)	Komponente za inženjersku i naučnu primenu.
Luxent Development Corp. (www.luxent.com)	Success Ware, Light Lib i AS/400 Middleware.

Naravno da se ne može napraviti kompletna lista komercijalnih Web sajtova. Da biste videli kompletniju listu, pogledajte INPRISE Delphi Tools stranicu na www.inprise.com/delphi/deltools.html.

Korisnički Web sajtovi

Sledeće sajtove su napravili korisnici Delphi-ja, za korisnike Delphi-ja. Sajtovi su navedeni bez posebnog reda. Ove strane sadrže linkove na druge strane, tako da kada počnete, pred Vama su sati pretraživanja. Pogledajte Delphi Deli stranicu da biste videli veliku listu linkova.

Delphi32.com

`www.delphi32.com`

Dr. Bob's JBuilder, C++Builder and Delphi Clinic

`www.drbob42.com`

The Delphi Deli

`www.delphideli.com`

The Delphi Super Page

`sunsite.icm.edu.pl/delphi`

Torry's Delphi Pages

`www.torry.ru`

The Delphi Games Creator

`www.users.dircon.co.uk/~zeus`

News grupe

INPRISE sponzoriše News grupe, da bi korisnici Borland-ovih proizvoda mogli da postavljaju pitanja jedni drugima. To su privatne News grupe i INPRISE dozvoljava pristup svojim korisnicima. Sve što treba da uradite je da usmerite svoj program za čitanje News poruka na `forums.inprise.com` i pronaćićete hiljade poruka koje su svrstane u nekoliko News grupa. Da biste dobili kompletne informacije, pogledajte Web stranu kompanije INPRISE (`www.inprise.com/newsgroups`).

Publikacije

Postoji nekoliko publikacija o Delphi-ju koje izlaze nekoliko puta godišnje. Neke od glavnih publikacija su prikazane u listi. Najveći broj ovih Web sajtova Vam pruža mogućnost besplatnog probnog pristupa:

Delphi Developer's Journal (The Cobb Group)

`www.cobb.com/ddj`

Visual Developer (Coriolis Group)

`www.coriolis.com`

Delphi Developer (Pinnacle Publishing)

www.pinpub.com/delphi

Delphi Informant (Informant Communications Group)

www.informant.com/delphi

Delphi Magazine

www.itecuk.com/delmag

Dodatak

Odgovori na test pitanja

Ovaj dodatak sadrži odgovore na kviz pitanja koja se nalaze iza svakog poglavlja.

Dan 1

1. Koja je ekstenzija Pascal-ovog junita?

   `.pas`

2. Koji je naziv ključne reči koja markira odeljak u kom se deklarišu promenljive?

   `var`

3. Šta radi funkcija `IntToStr`?

   Funkcija `IntToStr` konvertuje celobrojnu vrednost u Pascal string.

4. Koja je svrha liste uses u Pascal-ovom junitu?

   `Uses` lista u Pascal-ovom junitu sadrži spisak svih junita od kojih taj junit zavisi. Prevodilac mora da ima uvid u tu listu, kako bi mogao da prevede junit.

5. Da li su sledeće dve deklaracije različite? Objasnite zašto da, odnosno zašto ne?

   ```
   var
     top : Integer;
     Top : Integer;
   ```

   Ove dve deklaracije su identične, pošto Pascal ne pravi razliku između velikih i malih slova.

6. Kako grupišete Pascal stringove?

   Korišćenjem operatora `+`. (Možete koristiti i `StrCat` funkciju za stringove koji se završavaju nulom)

7. Kako možete ubaciti kontrolne karaktere u string?

   Korišćenjem simbola # iza koga se nalazi ASCII kod karaktera koji želite da ubacite.

8. Koja je maksimalna dužina kratkog stringa?

   255 znakova.

9. Pogledajte ovu liniju koda:

   ```
   MyArray : array [0..10] of Byte;
   ```

   Koliko bajtova rezeviše ovaj niz?

   11 bajtova (od 0 do 10).

10. Koji indeks označava prvi element niza; 0, ili 1?

    To zavisi od načina na koji je niz deklarisan. Indeks prvog elementa ovog niza je 0:

    ```
    Array1 : array [0..9] of Integer;
    ```

    Sa druge strane, indeks prvog elementa ovog niza je 1:

    ```
    Array2 : array [1..10] of Integer;
    ```

Dan 2

1. Koje naredbe će biti izvršene, ukoliko je izraz u okviru `if` naredbe `True`?

   Izvršiće se naredbe koje se nalaze neposredno iza `if` naredbe. Ukoliko se iza `if`naredbe nalazi blok naredbi, izvršiće se ceo blok.

2. Koliko vrednosti može vratiti funkcija?

   Jednu. Sa druge strane, Vi možete funkciji preneti nekoliko promenljivih parametra, tako da ona, efektivno, može vratiti više vrednosti.

3. Pored sintaksne, koja još razlike postoje između `while` i `repeat` petlji?

   `While` testira uslov na početku petlje. `Repeat` testira uslov na kraju petlje.

4. Šta rade procedure `Break` i `Continue`?

   `Break` procedura se koristi za iskakanje iz petlje. Posle izvršenja `Break` procedure će se izvršiti naredba koja sledi telo petlje. Procedura `Continue` prebacuje izvršavanje programa na početak petlje.

5. Šta je globalna promenljiva?

   Promenljiva čija je oblast definisanosti ceo program. Može joj pristupiti svaka funkcija u programu.

6. Može li struktura sadržati mešavinu tipova podataka (`Char`, `Integer`, `Word` i tako dalje)?

   Da, struktura može sadržati bilo koji broj promenljivih koje mogu biti proizvoljnog tipa.

7. Kako se pristupa članovima strukture?

   Pomoću operatora (`.`). Na primer:

   ```
   record.LastName = "Noble";
   ```

8. Koliko procedura i funkcija može postojati u programu?

   Ne postoji ograničenje broja procedura i funkcija koje program može sadržati.

9. Može li funkcija pozvati drugu funkciju, ili proceduru?

   Da, funkcije i procedure mogu pozivati druge funkcije i procedure. Ta osobina se često koristi.

10. Da li je moguće imati niz struktura?

    Da.Možete imati niz struktura, isto kao što možete imati i niz celih brojeva, bajtova, ili stringova.

Dan 3

1. Kako možete osloboditi skup svih vrednosti?

   Tako što ćete skupu dodeliti prazan skup - na primer:

   ```
   Font.Style := [];
   ```

2. Koja je svrha uvođenja privatnih polja i metoda?

   Privatna polja štite podatke od direktne izmene od strane korisnika klase. Privatna polja se mogu menjati kroz javne funkcije, ili osobine, ali ne i direktno.

3. Kako možete da zadržite privatnost polja, a da omogućite korisniku da čita i menja njihove vrednosti?

   Korišćenjem metoda, ili osobina.

4. Kada se poziva destruktor klase?

   Destruktor se poziva prilikom uništavanja objekta.

5. Šta znači preskakanje metoda osnovne klase?

   Preskakanje metoda znači zamenu metoda osnove klase metodom izvedene klase. Novi metod mora imati isto ime, parametre i povratnu vrednost kao i preskočeni metod.

6. Kako možete da preskočite metodu osnovne klase, a da još uvek imate koristi od operacija koje metode osnovne klase izvršavaju?

   Pozovite funkciju osnovne klase iz nove funkcije:

```
procedure MyClass.DoIt;
begin
  Inherited DoIt;
  { do some other stuff }
end;
```

7. Koji operator se koristi da ukloni referencu pointera?

   Operator pointera (`^`).

8. Da li klasa može da sadrži ostale slučajeve klasa kao svoja polja?

   Da. To je česta pojava.

9. Koja se ključna reč koristi da bi se definisalo da pointer nema vrednost?

   Ključna reč `nil`.

10. Za šta se koristi ključna reč `as`?

    Da bi se pointeru promenio tip iz izvedenog u osnovni i obrnuto.

Dan 4

1. Kako da pozovete okvir za dijalog Customize za glavni prozor?

   Kliknite desnim tasterom na bilo koju traku sa alatima i izaberite Customize iz kontekstnog menija.

2. Nakon što ste otvorili okvir za dijalog Customize, kako možete dodati dugmad na traku sa alatima?

   Prosto prevucite stavku sa Commands stranice na traku sa alatima i spustite je tamo gde biste želeli da se pojavi na traci sa alatima.

3. Kako da uklonite dugmad iz trake sa alatima?

   Prevucite dugmad koju više ne želite ispod dna trake za alatima i spustite je.

4. Koji je najlakši način da postavite više komponenti istog tipa na formu?

   Pridržite Shift taster prilikom klikanja mišem na komponentu na paleti sa komponentama. Svaki put kada kliknete mišem na formu, postaviće se nova komponenta.

5. Koji je najlakši način da postavite komponentu na središte forme?

   Dva puta kliknite na dugme sa komponentom na paleti sa komponentama.

6. Nabrojte tipove datoteka koji su potrebni za kreiranje Delphi aplikacije.

   To su `.dpr`, `.pas` i `.dfm` datoteke.

7. Koji VCL metod koristite da prikažete formu ne-modalnu?

   `Show` metod.

8. Koji VCL metod koristite da prikažete formu modalnu?

   `ShowModal` metod.

9. Kako možete prikačiti događaj na upravljač događajem koji je prethodno bio definisan?

   Prebacite se na Events stranu Object Inspector-a. U koloni vrednosti pored događaja, klinite na dugme sa strelicom na dole. Prikazaće se lista kompatibilnih upravljača događajima. Izaberite jedan.

10. Kada koristite Object Inspector, kako možete da pregledate specifikaciju opcija za određenu karakteristiku?

    Dva puta kliknite na kolonu sa vrednostima pored imena karakteristike u Object Inspector-u. Svaki put kada dva puta kliknete, vrednost se menja na sledeću iz liste.

Dan 5

1. Da li su sve komponente vidljive u toku dizajniranja?

   Ne. Vidljive su samo vizuelne komponente.

2. Da li je komponenta `OpenDialog` vizuelna, ili nevizuelna komponenta?

   Ona je nevizuelna. Iako se u toku izvršavanja programa može videti, smatra se nevizuelnom, zato što nije vidljiva u toku dizajniranja.

3. Koji je naziv VCL klase koja predstavlja Delphi formu?

   `TForm`.

4. Da li se sve verzije Delphi-ja isporučuju sa istim skupom komponenti?

   Ne. Profesionalna verzija dolazi sa većim brojem komponenti od standardne verzije. Takođe, klijent/server verzija dolazi sa većim brojem komponenti od profesionalne.

5. Da li su sve VCL klase bez razlike izdvojene iz klase `TObject`?

   Da.

6. Navedite jednu nevizuelnu VCL komponentu.

`TOpenDialog`, `TSaveDialog`, `TColorDialog`, `TTimer`, `TImageList`, `TFontDialog` i mnoge druge su nevizuelne VCL komponente.

7. Da li sve komponente dele neke zajedničke karakteristike?

   Da. Sve komponente su izvedene iz `TComponent`, tako da imaju sve karakteristike koje se mogu naći u klasi `TComponent`, kao što su `Name` i `Owner`.

8. Navedite dve uobičajene karakteristike koje sve vizuelne komponente dele.

   Uobičajene karakteristike koje dele sve vizuelne komponente su: `Top`, `Left`, `Owner`, `Parent`, `Width`, `Height` i tako dalje.

9. Mogu li dve, odnosno više komponenti, deliti isti upravljač događajima?

   Da.

10. Koji je VCL termin za Windows kontekst uređaja? Koji je naziv VCL klase koja enkapsulira kontekst uređaja?

    Kanvas je Windows kontekst uređaja. VCL enkapsulira kontekst uređaja kroz `TCanvas` klasu.

Dan 6

1. Kada koristite kombinaciju Ctrl+prevlačenje mišem za odabiranje komponenti?

   Kada odabirate komponete koje se nalaze u okviru neke druge komponente (na primer, komponente na panelu).

2. Koji značaj ima izbor prve komponente, kada poravnavate grupu komponenti?

   To je osnovna komponenta. Njena pozicija se ne menja prilikom poravnavanja grupe komponenti.

3. Koji je najbrži način za izbor grupe komponenti?

   Uokvirite ih isprekidanim kvadratom.

4. Kako možete da postignete da sve komponente u okviru grupe imaju širinu najšire komponente?

   Izaberite sve komponente koje želite da promenite. Izaberite Edit➥Size iz glavnog menija i izaberite Grow to Largest opciju.

5. Šta će se dogoditi kada kliknete mišem dva puta na komponentu u okviru forme?

   U Code Editor-u se prikazuje podrazumevani upravljač događajima. U slučaju više komponenti prikazuje se upravljač `OnClick` događajem. U nekim specijalnim slučajevima (kao što je `Image` komponenta), prikazaće se dijalog.

6. Šta radi opcija `alClient` karakterisktike `Align`?

   Primorava komponentu da popuni celu korisničku oblast komponente čiji je član, bez obzira kolika je ta komponenta (obično forma).

7. Šta znače tri tačke iza naziva opcije menija?

   Znači da će se posle izbora te opcija, na ekranu, pojaviti dijalog.

8. Koja su dva načina za pomeranje opcija menija?

   U Menu Designer-u možete prevući opciju na drugo mesto, ili koristiti Cut i Paste.

9. Kako možete dodati akceleratore menija opcijama menija?

   Prilikom unošenja naziva stavke, koristite ampersend (`&`) pre slova koje želite da koristite kao prečicu. Na primer, naziv File➥Exit opcije bi bio `E&xit`.

10. Kako možete u početku deaktivirati opciju menija?

    Postavljanjem njene `Enabled` karakteristike na .

Dan 7

1. Da li možete da promenite karakteristiku `Name` u toku rada programa?

   Da, ali je to vrlo loša ideja.

2. Koja karakteristika se koristi za aktiviranje i deaktiviranje kontrola?

   Karakteristika `Enabled`.

3. Kako možete da u toku rada programa saopštite da je dugme deaktivirano?

   Tako što je njegov tekst zatamnjen.

4. Koja je razlika između dugog saveta i kratkog saveta?

   Dugi savet se koristi kao tekst na statusnoj traci, dok se kratak savet koristi kao tekst u balonu.

5. Navedite tri od četiri metode koje se mogu koristiti za ponovno iscrtavanje kontrole.

   `Invalidate, Repaint, Refresh i Update.`

6. Koliko tipova kombo okvira postoji?

   Tri: simple, drop-down i drop-down list.

7. Kako se karakteristika `ModalResult` koristi za komponente dugmad?

Kada se pritisne dugme kome je `ModalResult` karakteristika postavljena na ceo broj, zatvara se forma i vrednost `ModalResult` karakteristike postaje povratna vrednost `ShowModal` metode.

8. Koja se komponenta najviše koristi kao kontejner koji sadrži druge komponente?

   Komponenta `Panel`. Koriste se i druge komponente.

9. Koja je povratna vrednost metoda `Execute` za komponentu `OpenDialog`, ukoliko korisnik klikne mišem na dugme OK, kako bi zatvorio okvir za dijalog?

   `True`.

10. Kako da od komponente `SaveDialog` napravite okvir za dijalog Save As?

    Izmenite njenu `Title` karakteristiku u `Save As`.

Dan 8

1. Kada koristite opciju Inherit prilikom odabiranja objekta u okviru skladišta objekata?

   Onda kada želite da koristite sve osobine baznog objekta i ukoliko želite da promenite izvedeni objekat, ukoliko se promeni bazni objekat.

2. Koja je procedura za snimanje projekta u skladište objekata?

   Potrebno je izabrati opciju Project➥Add to Repository iz glavnog menija.

3. Šta se dešava sa nasleđenim formama kada promenite formu original?

   Kada promenite glavnu formu, sve izvedene forme se menjaju, kako bi prihvatile izmene.

4. Gde postavljate deklaracije korisničkih metoda u okviru deklaracije klasa forme?

   Deklaracije korisničkih metoda postavljate u `private`, ili `public` deo deklaracije klase. Nemojte postavljati ove deklaracije u deo kojim upravlja Delphi (barem ne, dok niste sigurni šta radite).

5. Gde postavljate definiciju metoda (samu metodu), kada dodajete sopstvene metode u Delphi kod?

   U `implementation` deo junita.

6. Kako možete odrediti ko je napisao određeni objekat koji se nalazi u skladištu objekata?

   Tako što ćete se prebaciti u Details pregled objekta u Object Repository-ju. Tamo je navedeno ime autora objekta.

7. Gde dodajete i brišete kartice skladišta objekata?

   Stranice možete dodavati i brisati iz Object Repository dijaloga za konfiguraciju (koji dobijate izborom Tools➥Repository opcije iz glavnog menija).

8. Da li je lakše kreirati osnovnu aplikaciju od početka, ili korišćenjem čarobnjaka aplikacija?

   U skoro svim slučajevima je lakše koristiti čarobnjaka aplikacija.

9. Šta je bolje za male aplikacije: statičko povezivanje, ili dinamičko povezivanje korišćenjem paketa?

   Za male aplikacije je bolji rad bez paketa (statičko povezivanje).

10. Da li možete da kreirate resursnu skript datoteku koja koristi tabelu stringova, koristeći editor teksta?

    Da, možete lako napraviti tabelu stringova, korišćenjem tekst editora. Sve što treba da znate je kao izgleda osnovna forma tabele stringova.

Dan 9

1. Kako možete brzo preći između forme junita i izvorne datoteke, kada radite u Delphi-ju?

   Korišćenjem tastera F12 brzo prelazite iz Form Designer-a u Code Editor i obnuto.

2. Ukoliko uklonite datoteku iz projekta, koristeći menadžer projekta, da li uklanjate i datoteku sa Vašeg tvrdog diska?

   Ne, datoteka se uklanja samo iz projekta.

3. Kako možete da podesite glavnu formu aplikacije?

   Aktivirajte Forms stranicu Project Options dijaloga i izaberite formu koju želite da koristite kao glavnu iz Main form kombo liste.

4. Šta znači ako nemate Delphi-jeve forme koje se automatski kreiraju?

   Moraćete da preuzmete odgovornost koju sa sobom nosi samostalno kreiranje formi pre njihovog korišćenja.

5. Kako možete dodavati nove elemente u Vaše junite koristeći prozor za ispitivanje koda?

   Kliknite desnim tasterom miša i izaberite New from the Code Explorer iz kontekstnog menija. Unesite deklaraciju novog elementa i pritisnite Enter.

6. Koji je značaj generisanja informacija za debagiranje u slučaju Vaše aplikacije?

Kada se generiše informacija za debagiranje, možete da se krećete kroz kod u toku procesa debagiranja.

7. Zašto se koristi opcija Find in Files?

   Da bi se pronašao određeni tekst u više datoteka.

8. Koja je prečica tastature za snimanje datoteke u editor koda?

   Ctrl+S (pod pretpostavkom da koristite podrazumevano mapiranje tastature).

9. Kako možete podesiti oznaku u prozoru editora? Koliko oznaka možete najviše imati?

   Oznaku postavljate korišćenjem kombinacija od Ctrl+K+0 do Ctrl+K+9. Na raspolaganju je 10 oznaka.

10. Kako možete podesiti da datoteka bude dostupna samo za čitanje (read-only) u okviru editora koda?

    Izaberite Read Only iz kontekstnog menija Code Editor-a.

Dan 10

1. Kako možete da postavite tačku prekida na određenu liniju koda?

   Kliknite na levu marginu uz tu liniju koda. Možete pritisnuti i taster F5, ili izabrati Toggle Breakpoint iz kontekstnog menija Code Editor-a.

2. Šta je pogrešna tačka prekida?

   Prekidna tačka postavljena na liniju za koju se ne generiše konkretan kod.

3. Kako postavljate uslovne tačke prekida?

   Postavite tačku prekida, izaberite View➥Debug Windows➥Breakpoints iz glavnog menija, kliknite na željenu tačku prekida u Breakpoint List prozoru i zatim izaberite Properties iz kontekstnog menija Breakpoint List prozora. Postavite uslov u Condition polje Edit Breakpoint dijaloga.

4. Kako možete promeniti karakteristike elementa liste za pregled?

   Dva puta kliknite na stavku u u Watch List prozoru. Prikazuje se Watch Properties dijalog. Izmenite karakteristike po potrebi.

5. Koji je najbrži način za dodavanje promenljive u listu za pregled?

   Kliknite na promenljivu i pritisnite Ctrl+F5 (ili, izaberite Add Watch at Cursor iz kontekstnog menija Code Editor-a).

6. Koji alat koristite da biste pregledali polja podataka i metode klase?

   Debug Inspector.

7. Kako možete ući u izvorni kod metoda kada prolazite kroz izvorni kod, koristeći debager?

   Koristite taster F7, ili Run➥Trace Into opciju iz glavnog menija.

8. Kako možete promeniti vrednost promenljive u toku rada programa?

   Kliknite na promenljivu i izaberite Evaluate➥Modify iz kontekstnog menija Code Editor-a (ili, izaberite Run➥Evaluate/Modify iz glavnog menija). Vrednost izmenite u Evaluate/Modify dijalogu.

9. Kako možete poslati sopstvene poruke u listu događaja?

   To možete uraditi korišćenjem Windows API funkcije `OutputDebugString`.

10. Šta radi opcija Integrated debugging u okviru za dijalog opcije debagera?

    Kada je ova opcija uključena i kada se program izvršava u okviru IDE, izvršava se pod kontrolom debagera. Kada je ova opcija isključena, program se izvršava bez korišćenja debagera.

Dan 11

1. Kako se transparentna boja koristi za ikone i kursore?

   Kroz sve elemente koji su obojeni transparentnom bojom se providi pozadina.

2. Kako možete da odaberete boju u editoru slika?

   Klikom na boju iz palete sa bojama.

3. Kako možete da odaberete oblast na bitmapi koju treba da isečete, ili kopirate?

   Izaberite Marquee alatku i zatim razvucite pravougaonik, korišćenjem miša. Možete koristiti Edit➥Select All, da biste izabrali celu bitmapu, ili Lasso alatku.

4. Koji je maksimalan broj boja dozvoljen u radu sa bitmapama u editoru slika?

   256.

5. Šta je tačka pokazivanja kursora (hot spot)?

   Tačka kurzora koja se koristi za obaveštavanje Windows-a o koordinatama tačke na ekranu, kada se klikne mišem.

6. Da li program WinSight može istražiti skrivene prozore?

   Da. WinSight istražuje i skrivene i vidljive prozore.

7. Koji je najbrži način da pronađete prozor u stablu prozora programa WinSight?

Izaberite Window➥Follow to Focus iz glavnog menija i zatim kliknite na prozor koji želite da istražite.

8. Kako možete da omogućite automatsko snimanje datoteka, svaki put kada program bude pokrenut preko debagera?

   U Autosave options delu koji se nalazi na Preferences stranici Environment Options dijaloga izaberite opciju Editor files.

9. Gde možete reorganizovati sadržaj palete komponenti?

   Koristite Pallete stranicu Environment Options dijaloga.

Dan 12

1. Koje komponente možete koristiti da biste crtali grafiku na formi?

   Iako možete crtati direktno po kanvasu forme, `PaintBox` komponenta Vam omogućava crtanje na određenom delu forme (na delu koji zauzima `PaintBox` komponenta).

2. Koja karakteristika klase `TCanvas` kontroliše boju koja popunjava kanvas?

   Karakteristika `Brush`.

3. Za koje operacije služi region za isecanje?

   Region za isecanje definiše deo kanvasa u kome će se crtati, što znači da se izvan tog dela ne može crtati. Sve što bude nacrtano izvan regiona za isecanje, neće biti prikazano.

4. Koje funkcije treba da koristite da biste nacrtali više linija teksta na kanvasu?

   `DrawText` funkciju, uz korišćenje `DT_WORDBREAK` flega.

5. Koje metode klase `TCanvas` se mogu koristiti za crtanje bitmape sa transparentnom pozadinom?

   `BrushCopy` metoda.

6. Koja metoda klase `TCanvas` se koristi za kopiranje kompletne bitmape na kanvas?

   Možete koristiti nekoliko, ali je najlakša za korišćenje i najbrža metoda `Draw`. Druge metode su: `BrushCopy`, `StretchDraw` i `CopyRect`.

7. Kako možete snimiti memorijsku bitmapu u datoteku?

   Korišćenjem `SaveToFile` metode.

8. Koju komponentu koristite da biste izvodili wave datoteke?

   `TMediaPlayer` komponentu. Da biste izveli wave detoteku korišćenjem Windows API-ja, koristite `PlaySound` funkciju.

9. Za šta se koristi karakteristika `TimeFormat` klase `TMediaPlayer`?

   Koristi se za postavljanje formata vremena na osnovu tipa medije koja će biti izvedena. Neki tipovi medija koriste nekoliko formata vremena (na primer CD Audio).

10. Da li možete da snimite wave audio datoteku, koristeći komponentu MediaPlayer?

    Da, ali prethodno morate naučiti više o tome.

Dan 13

1. Kako možete prikačiti upravljač događajem na događaj `OnClick` dugmeta trake sa alatima?

   U Object Inspectoru kliknite na Events stranicu. Kliknite na dugme sa strelicom na dole, pored `OnClick` događaja. Izaberite upravljač događajem iz liste.

2. Da li možete postaviti kontrole na traku sa alatima, a da to ne budu dugmad?

   Da. Možete postaviti bilo koji tip kontrole na traku sa alatima. Na trakama sa alatima se često koriste kombo liste.

3. Koji je naziv događaja u okviru klase `TActionList` na koji odgovarate prilikom aktiviranja komande?

   `OnUpdate`.

4. Šta karakteristika **`SimplePanel`** komponente `StatusBar` radi?

   Prebacuje statusnu traku u režim rada sa jednim panelom.

5. Kako možete da manuelno izmenite tekst statusne trake?

   Kod jednostavne statusne trake je dovoljno da uradite sledeće:

   ```
   StatusBar.SimpleText := 'Text';
   ```

6. Kako možete da aktivirate i deaktivirate opcije menija i dugmad?

   Kod pojedinačnih komponenti je dovoljno da postavite `Enabled` karakteristiku na `True`, da biste aktivirali komponentu, ili na `False`, da biste je deaktivirali. Kod više komponenti koje imaju zajednički zadatak, napravite `Action` za taj zadatak i postavite `Enabled` karakteristiku `Actiona` na potrebnu vrednost.

7. Kako možete pristupiti štampaču u okviru Delphi aplikacije?

   Kroz `Printer` funkciju.

8. Koju metodu pozivate kada počinjete štampanje koristeći klasu TPrinter?

   `BeginDoc` metodu.

9. Koju metodu klase `TPrinter` pozivate kada želite da počnete štampanje nove strane?

   `NewPage` metodu.

10. Kako možete da, u toku rada programa, promenite izgled kursora za određenu komponentu?

    Izmenite `Cursor` karakteristiku komponente.

Dan 14

1. Kako se postavlja ime fajla sa tekstom pomoći koji će koristiti Vaš program?

   Koristite Project Options dijalog-prozor (Application stranicu), ili postavite `HelpFile` osobinu `Application` objekta u toku izvršavanja programa.

2. Kako se pravi podrška za taster F1 u određenoj formi ili dijalog-prozoru?

   Dodelite vrednost koja je različita od nule `HelpContext` osobini. Proverite da li je tom objektu dodeljen odgovarajući identifikator konteksta u fajlu za pomoć i da li je taj fajl pripremljen za rad sa programom.

3. Koju metodu treba pozvati da bi se prikazao indeks tema koje se nalaze u Vašem fajlu sa tekstom pomoći.

   `HelpCommand` metodu.

4. Koje tipove objekata može generisati izuzetak?

   Može generisati objekat bilo koje klase koja je izvedena iz `Exception`.

5. Da li je dozvoljeno imati više od jednog `except` bloka u okviru istog `try` bloka?

   Ne. Može postojati samo jedan except blok.

6. Kako se generiše izuzetak?

   Pomoću ključne reči `raise` - na primer:

   ```
   raise EMyException.Create('An error occurred.');
   ```

7. Koja je podrazumevana vrednost RootKey osobine TRegistry

   klase?

   ```
   \HKEY_CURRENT_USER
   ```

8. Da li morate pozvati CloseKey, kada završite rad sa ključem?

   Ne. Destruktor klase `TRegistry` će sam zatvoriti ključ. Ipak, ne biste trebali da dugo držite ključ otvorenim.

9. Koja je razlika između `SendMessage` i `PostMessage`?

`PostMessage` šalje poruku u Windows-ov red za čekanje i vraća kontrolu programu. `SendMessage` šalje poruku, ali ne vraća kontrolu programu, dokle god se poruka ne obradi.

10. Kako se zove VCL metoda pomoću koje se poruka šalje direktno do komponente?

    `Perform` metoda.

Dan 15

1. Koji je osnovni interfejs za sve COM interfejse?

   Osnovni interfejs iz koga se izvode svi ostali interfejsi je `IUnknown`.

2. Šta je GUID?

   GUID je 128-bitna celobrojna vrednost koja jedinstveno određuje COM objekat (interfejs, klasu ili biblioteku tipova).

3. Šta se dešava kada brojač referenci COM objekta dostigne vrednost 0?

   Kada brojač referenci COM objekta dostigne vrednost 0, Windows oslobađa iz memorije taj COM objekat.

4. Kako se zove Delphi-jev alat za rad sa bibliotekama tipova?

   Alat za rad sa bibliotekama tipova se zove Library Type Editor.

5. Kako se kreiraju GUID-ovi prilikom pisanja COM objekata u Delphi-ju?

   Ne morate da samostalno pravite GUID-ove prilikom pisanja COM objekata u Delphi-ju. Delphi sam pravi GUID-ove umesto Vas (ovo je, u stvari, trik pitanje). Ukoliko, ipak, želite da sami pravite GUID-ove u okviru svog koda, možete pritisnuti Ctrl+Shift+G u trenutku kada se nalazite u Code Editor-u. Delphi će napraviti i ubaciti GUID u Vaš kod.

6. Šta je potrebno da izaberete iz Object Repository-ja, kada želite da kreirate ActiveX komponentu iz VCL komponente?

   Da biste napravili ActiveX komponentu iz postojeće VCL komponente, potrebno je da izaberete ActiveX Control stavku iz Object Repository-ja.

7. Da li možete da koristite ActiveX kontrole koje ste kreirali u Delphi-ju u okviru Visual Basic okruženja?

   Da. Možete koristiti Delphi ActiveX kontrole u Visual Basic-u. ActiveX kontrola koju želite da koristite na ovaj način mora sadržati informaciju o verziji.

8. Kada ste generisali i registrovali ActiveX kontrolu, šta je potrebno uraditi da bi se je postaviti na Delphi-jevu paletu sa komponentama?

Da biste instalirali komponentu na Delphi-jevu paletu sa komponentama, izaberite Component➥Import ActiveX Control iz Delphi-jevog glavnog menija.

9. Kako biste izbrisali ActiveX kontrolu koju ste sami napravili iz Registry-a?

   Da biste izbrisali kontrolu iz Registry-ja, izaberite Run➥Unregister ActiveX Server iz glavnog menija, ili pokrenite `TREGSVR` program uz korišćenje -u prekidača. ActiveX kontrolu možete izbrisati i iz Import ActiveX dijalog-prozora.

10. Da li možete da koristite ActiveX kontrole koje ste kreirali u Delphi-ju na Web strani?

    Da. ActiveX kontrole napravljene u Delphi-ju su pripremljene za korišćenje na Web stranama.

Dan 16

1. Šta je lokalna baza podataka?

   To je baza podataka koja se nalazi na korisnikovom računaru, umesto na serveru za baze podataka. Ovaj izraz se obično koristi kada se govori o Paradox i dBASE tabelama.

2. Koja je uloga BDE-a?

   BDE omogućava pristup bazama podataka programima pravljenim u Delphi-ju.

3. Da li su dataset i tabela jedno te isto? Ako nisu, objasnite razlike.

   Ne. Dataset i tabela nisu isto. Dataset može sadržati celokupan sadržaj tabele, ali može sadržati i samo jedan mali deo.

4. Navedite jednu prednost korišćenja keširanja izmena.

   Keširanja izmena smanjuju količinu informacija koja protiče kroz mrežu, omogućavaju Vam da izmenite dataset-ove koji su predviđeni samo za čitanje, dozvoljavaju Vam da napravite nekoliko izmena, a da ih zatim odjednom upišete, ili zanemarite.

5. Šta je stored procedura?

   Program koji radi sa bazama podataka i koji se obično nalazi na samom serveru za baze podataka.

6. Koja je uloga `SQL` osobine `TQuery` komponente?

   `SQL` osobina sadrži SQL naredbe koje se izvršavaju kada se pozove `Execute`, ili `Open` metoda.

7. Navedite jedan razlog zbog kojeg biste želeli da koristite sopstveni `TDatabase` objekat, umesto podrazumevanog.

Da biste omogućili automatsko logovanje na bazu podataka.

8. Zašto biste želeli da održavate vezu između udaljene baze podataka i Vašeg programa, iako je trenutno ne koristite?

   Da biste smanjili vreme koje je potrebno za ponovno uspostavljanje veze svaki put kada je potrebna neka usluga od baze podataka.

9. Šta radi `TBatchMove` komponente?

   `TBatchMove` Vam dozvoljava da napravite, ili izmenite jedan dataset sa sadržajem drugog.

10. Šta je BDE alias?

    BDE alias je skup parametara koji opisuju vezu sa bazom podataka.

Dan 17

1. Koji je najbrži i najjednostavniji način za pravljenje forme za rad sa bazama podataka?

   Korišćenje Database Form Wizard-a.

2. Na koji način možete da kontrolišete raspored i broj kolona koje će se prikazivati u `DBGrid` komponenti?

   Pomoću Colums Editor-a. (Kliknite desnim tasterom miša na `DBGrid` komponentu i zatim izaberite Columns Editor iz kontekstnog menija.)

3. Koje komponente Vam omogućavaju da prikažete dataset u tabelarnom obliku?

   `DBGrid` komponenta.

4. Kako možete dodati, ili ukloniti taster sa `DBNavigator-a`?

   Izmenom `VisibleButtons` osobine.

5. Koje komponente ćete koristiti za prikaz slika koje se nalaze u BLOB poljima?

   `DBImage` komponenta.

6. Koje osobine su zajedničke za sve komponente za rad sa podacima?

   Između ostalih, `DataSource` osobina.

7. Koja osobina se koristi za definisanje polja za koje će se vezati određena komponenta?

   `DataField` osobina.

8. Da li možete ponovo rasporediti kolone u `DBGrid` komponenti?

   Da. Koristite Columns Editor u toku pravljenja programa i drag and drog tehniku u toku izvršavanja.

9. Koja komponenta se koristi za prikaz i izmenu tekst polja u bazi podataka?

   `DBEdit` komponenta.

10. Šta znači BLOB?

    Veliki binarni objekat (engl. Binary Large Object).

Dan 18

1. Koje je metode potrebno pozvati da bi se kreirala baza podataka u toku izvršavanja programa?

   `CreateTable`.

2. Čemu služi `Edit` metoda `TTable` komponente?

   `Edit` metoda postavlja dataset u režim izmena, tako da je moguće izmeniti sadržaj zapisa.

3. Koju ćete metodu pozvati kada želite da upišete izmene u zapis?

   `Post` metodu.

4. Kako se pravi novi modul za rad sa podacima?

   Kroz Object Repository.

5. Da li je modul za rad sa podacima obična forma?

   Modul za rad sa podacima je veoma sličan običnoj formi, ali nije isto što i ona.

6. Koji metod je potrebno pozvati da bi se odštampao QuickReport?

   `Print` metod.

7. Koji tip QuickReport-ove celine prikazuje podatke iz dataset-a?

   Celina sa podacima.

8. Koja komponenta se koristi za prikazivanje broja strane na izveštaju?

   `QRSysData` komponenta se može koristiti za prikazivanje broja strane, aktuelnog datuma, trenutnog vremena i drugih stvari.

9. Kako možete pregledati izveštaj u toku pravljenja programa?

   Kliknite desnim tasterom miša na izveštaj i izaberite Preview iz kontekstnog menija.

10. Za šta se koristi QRExpr komponenta?

    `QRExpr` komponenta se koristi za prikaz rezultata izračunavanja nekog izraza.

Dan 19

1. Kako se učitava DLL korišćenjem statičkog učitavanja?

   U pozivajućem programu deklarišite sve funkcije i procedure koje se nalaze u DLL-u, pomoću ključne reči `extern`. DLL će se automatski učitati prilikom pokretanja programa.

2. Kako se učitava DLL korišćenjem dinamičkog učitavanja?

   Korišćenjem Windows API funkcije `LoadLibrary`.

3. Kako se poziva procedura, ili funkcija iz DLL-a koji je statički učitan?

   Proceduru, ili funkciju pozivate kao i bilo koju drugu proceduru, ili funkciju.

4. Šta treba da uradite da biste bili sigurni da će procedure i funkcije iz Vašeg DLL-a moći da se pozivaju i izvan DLL-a?

   Funkcije i procedure moraju biti izvezene iz DLL-a. Postavite ime funkcije, ili procedure u `exports` celinu DLL-a.

5. U slučaju da je DLL dinamički učitan, da li ga možete izbaciti iz memorije bilo kad, ili samo onda kada pozivajući program prestaje sa radom?

   Možete ga izbaciti iz memorije kad god želite (korišćenjem funkcije `FreeLibrary`).

6. Šta treba da uradite da biste prikazali Delphi-jevu formu koja se nalazi u DLL-u u nekom programu koji nije pisan u Delphi-ju.

   Napravite izvezenu funkciju koju će pozivajući program moći da pozove. U okviru ove funkcije, napravite formu i prikažite je. Proverite da li je funkcija deklarisana sa ključnom reči `stdcall`.

7. Kako se zove ključna reč koja se koristi da bi se deklarisale procedure i funkcije koje su uvezene iz DLL-a?

   Ključna reč `external`.

8. Kako dodajete resurse u DLL?

   Povežite fajl sa prevedenim resursima (`.res` ili `.dcr`) sa DLL-om korišćenjem `$R` direktive prevodiocu - na primer:

   ```
   {$R Resources.res}
   ```

9. Da li DLL sa resursima mora sadržati i kod?

   Ne.DLL-u sa resursima nije potreban nikakav kod.

10. Da li se DLL koji sadrži resurse može statički učitavati (tj. kad pozivajući program počne sa radom)?

To je moguće, ali se retko koristi. Vama je potrebna povratna vrednost funkcije `LoadLibrary`, kako biste mogli da pristupite resursima u DLL-u, tako da biste trebali da koristite dinamičko učitavanje DLL-ova sa resursima.

Dan 20

1. Da li osobina mora koristiti `write` metodu? Zašto i zašto ne?

   Ne. Možete koristiti direktan pristup.

2. Mora li osobina imati podatak iz klase sa kojim je povezana? Zašto i zašto ne?

   Ne. Osobina ne mora imati podatak iz klase sa kojim je povezana, ali većina osobina ga ima. Osobina čija se vrednost ne upisuje nije retka pojava.

3. Možete li napraviti komponentu proširivanjem postojeće komponente?

   Naravno. To je najlakši način za pravljenje novih komponenti.

4. Šta se dešava ukoliko ne navede `write` deo u deklaraciji osobine (bilo `write` metodu bilo direktan pristup)?

   Osobina će moći da se koristi samo za čitanje.

5. Šta znači "direktan pristup"?

   To znači da se vrednost podatka iz klase sa kojim je osobina povezana može čitati i menjati direktno.

6. Da li osobine moraju imati podrazumevane vrednosti? Zašto i zašto ne?

   Ne. Podrazumevane vrednosti nisu obavezne. Objavljene osobine bi trebale da imaju podrazumevane vrednosti. Osobine koje su tipa string ne mogu imati podrazumevane vrednosti (kao i neke druge osobine).

7. Da li postavljanje podrazumevane vrednosti osobini znači i automatsko postavljanje vrednosti podatka iz klase?

   Ne, podrazumevana vrednost služi prikazivanju podatka u Object Inspector-u i koristi se tokom pravljenja programa. Podrazumevanu vrednost podatka iz klase morate postaviti u konstruktoru komponente.

8. Kako se postavlja komponenta na paletu sa komponentama?

   Izaberite Component➥Install iz glavnog menija.

9. Kako se postavlja sličica na tasteru koji predstavlja komponentu na paleti sa komponentama?

   Napravite .dcr fajl sa istim imenom kao i fajl sa izvornim kodom komponente. Ovaj fajl bi trebao da sadrži 24x24 bitnu mapu koja ima isto ime kao i klasa

komponente. Proverite da li se .dcr fajl nalazi u istom direktorijumu kao i .pas fajl komponente.

10. Kako se pokreće događaj koji je definisao korisnik?

    Pošto ustanovite da je za određeni događaj definisana procedura za obradu, pozovite ga:

```
if Assigned(FOnMyEvent) then
FOnMyEvent(Self);
```

Dan 21

1. Da li Delphi i C++Builder projektni fajlovi imaju iste ekstenzije?

   Ne. Ekstenzija Delphi-jevog projektnog fajla je `.dpr` dok je ekstenzija C++Builder-ovog projektnog fajla `.bpr`.

2. Da li Delphi i C++Builder fajlovi sa formama imaju iste ekstenzije?

   Da. Za čuvanje formi i Delphi i C++Builder koriste fajlove sa ekstenzijom `.dfm`.

3. Da li možete koristiti pakete od nezavisnih proizvođača komponenti i u Delphi-ju i u C++Builder-u?

   U najvećem broju slučajeva ćete moći da koristite iste pakete i u Delphi-ju i u C++Builder-u. U ostalim slučajevima ćete morati ponovo da prevedete paket pomoću C++Builder-a. Zatražite od proizvođača verziju komponente koja je predviđena za korišćenje sa C++Builder-om.

4. Da li možete otvoriti Delphi-jevu formu u C++Builder-u?

   Da. Ne možete menjati formu (dodavati i brisati komponente), ali je možete pregledati.

5. Možete li menjati Delphi-jevu formu pomoću C++Builder Form Designer-a?

   Ne. Ne možete menjati Delphi-jevu formu korišćenjem C++Builder Form Designer-a. Ipak, Delphi-jevu formu možete menjati u tekstuelnom obliku izborom View As Text iz C++Builder-ovog kontekstnog menija.

6. Da li možete koristiti C++Builder izvornu celinu u Delphi-ju?

   Sa trenutno aktuelnom verzijom Delphi-ja nije moguće prevoditi C++Builder izvorne celine.

7. Šta je bolje? Delphi, ili C++Builder?

   Ni jedan ni drugi. Oba proizvoda imaju svoje prednosti. Izbor najviše zavisi od toga da li više volite Pascal, ili C++.

Dodatni dan

1. Koju kontrolu treba koristiti za prikazivanje Web strana?

   `THTML` kontrolu.

2. Koju kontrolu treba koristiti za vezivanje na News grupe?

   `TNMNNTP` kontrolu.

3. Kako se zove metoda koja se koristi za prikazivanje HTML dokumenta u okviru `THTML` kontrole?

   `RequestDoc`.

4. Koji događaj se generiše kada se HTML dokument učita u celini?

   `OnEndRetrieval` događaj.

5. Koju kontrolu treba koristiti za slanje elektronske pošte?

   `TNMSMTP` kontrolu.

6. Koju kontrolu treba koristiti za prijem elektronske pošte?

   `TNMPOP3` kontrolu.

7. Kako se zove metoda koja se koristi za slanje pošte putem `TNMSMTP` kontrole?

   `SendMail`.

8. Da li možete slobodno distribuirati Internet Explorer ActiveX kontrolu?

   Ne.Morate od Microsoft-a dobiti licencu za distribuiranje Internet Explorer ActiveX kontrole.

9. Kako se zove pomoćni program koji se koristi za registrovanje ActiveX kontrola?

   `TREGSVR.EXE`.

10. Koja kompanija proizvodi najveći deo Internet kontrola koje se isporučuju uz Delphi?

    NetMasters.

Dodatni dan

Pravljenje Internet programa

Da li ste i Vi jedan od onih ljudi koji misli da je Internet samo hir i da neće trajati još dugo? Pa, da Vam kažem nešto... niste u pravu. Internet je ogroman i svakim danom postaje sve veći. U stvari Internet čine Web i ljudi koji provode sate pretražujući ga. Ali Internet takođe čine prenos fajlova, elektronska pošta (e-mail) i elektronsko reklamiranje. Internet je veliki posao i on neće nestati, barem ne skoro. Verovatno biste želeli da malo ispolirate svoje poznavanje Internet programiranja. Srećom, Delphi čini i eksperimentisanje sa Internet programiranjem i pravljenje ozbiljnih programa jednostavnim.

Danas ćete se upoznati sa nekim aspektima Internet programiranja sa Delphi-jem. Tamo, na Internet-u, čeka ceo svet. Hajdemo i mi.

Delphi-jeve Internet komponente

Delphi-jeve Internet komponente se nalaze na Internet stranici palete sa komponentama i grupisane su u dve kategorije. U prvoj kategoriji se nalaze komponente firme NetMasters. Sa jednim izuzetkom, ovo su klasične VCL komponente. Izuzetak je `THTML` kontrola, koja je ActiveX. Tabela DD.1 prikazuje NetMasters kontrole i opis svake od njih. Komponente su raspoređene na isti način kao i na paleti sa komponentama.

Tabela DD.1: ActiveX Internet kontrole firme NetMasters

Kontrola	Opis
TNMDayTime	Prenosi informacije o datumu i vremenu sa specijalizovanih Internet servera.
TNMEcho	Prima i šalje tekst sa Internet echo servera.
TNMFinger	Prenosi informacije o korisniku sa Internet finger servera.
TNMFTP	Obavlja operacije prenosa fajlova sa umreženim sistemima koji koriste FTP (engl. File Transfer Protocol).
TNMHTTP	Obavlja prenos fajlova korišćenjem HTTP-a (engl. Hypertext Transport Protocol). Hipertekst dokumenti se, inače, pregledaju pomoću Web Browser-a. Koristite TNMHTTP da biste preneli dokumente koji ne treba da se pregledaju pomoću Web Browser-a.
TNMMsg	Šalje jednostavne ASCII teksualne poruke koristeći TCP/IP protokol.
TNMMSGServ	Prima poruke poslate pomoću TNMMsg kontrole.
TNMNNTP	Komunicira sa News serverima. Šalje i prima poruke iz News grupa na Internetu korišćenjem NNTMP (engl. Networking News Transfer Protocol).
TNMPOP3	Prenosi elektronsku poštu sa mail servera korišćenjem POP3 (engl. Post Office Protocol).
TNMUUProcessor	Kodira, ili dekodira MIME, ili uuencode fajlove.
TNMSMTP	Šalje elektronsku poštu kroz SMTP (engl. Simple Mail Transfer Protocol) mail servere.
TNMStrm	Šalje tokove podataka na mrežu, ili na Internet stream server.
TNMStrmServ	Prima tokove poslate pomoću TNMStrm kontrole.
TNMTime	Prenosi informaciju o vremenu sa specijalizovanoh Internet servera.
TNMUDP	Vrši prenos podataka kroz mrežu korišćenjem UDP-a (engl. User Datagram Protocol).
TPowersock	Omogućava vezu sa Winsock API-jima.
TNMGeneralServer	Koristi se za opšte TCP/IP servere.
THTML	Prikazuje HTML (engl. Hypertext Markup Language) fajlove. Ovo je komponenta za pravljenje Web Browser-a.
TNMURL	Konvertuje URL podatke u čitljivi string i string podatke u URL format.

U drugu kategoriju spadaju klasične VCL komponente koje je proizveo Borland. TClientSocket i TServerSocket komponente se isporučuju samo sa profesionalnom i klijent/server verzijom Delphi-ja. Web Broker komponente (TWebDispatcher, TPageProducer, TQueryTableProducer i TDataSetTableProducer) se isporučuju samo sa klijent/server verzijom Delphi-ja. VCL Internet kontrole su nabrojane u tabeli DD.2.

Tabela DD.2: Klasične VCL Internet komponente

Komponenta	Opis
`TClientSocket`	Upravlja TCP/IP klijent socket vezom.
`TServerSocket`	Upravlja TCP/IP server socket vezom.
`TWebDispatcher`	Konvertuje običan modul za rad sa podacima u Web modul za rad sa podacima.
`TPageProducer`	Omogućava pravljenje dinamičkih HTML strana.
`TQueryTableProducer`	Generiše HTML dokument iz rezultata `Query-a`.
`TDataSetTableProducer`	Generiše HTML dokument iz `TDataSet` zapisa.

Ove dve grupe kontrola Vam pružaju sve mogućnosti koju su Vam potrebne za pravljenje visoko-kvalitetnih Internet programa.

Pravljenje WebBrowser-a

Jedan od najočiglednijih primera pravljenja Internet programa je pravljenje Web Browser-a. Najzad, to je najglamuroznji posao. Dobra vest je ta da može biti i najjednostavniji.

Kome treba još jedan Web Browser?

Možda se pitate zbog čega bi bilo ko hteo da pravi Web Browser. Na kraju krajeva, svi koriste, ili Netscape Navigator, ili Internet Explorer, pa se postavlja, možda očigledno pitanje, kome treba još jedan Web Browser? Istini za volju, verovatno nećete pokušavati da napravite Web Browser koji bi mogao da se takmiči sa Netscape-om, ili Microsoft-om. Sa druge strane, uzmite u obzir kompaniju koja ima stotine, ili hiljade zaposlenih kojima je potreban pristup Web-u. Licenciranje hiljada kopija komercijalnog Web Browser-a može biti veoma skupo. Vi možete napraviti kvalitetan Web Browser pomoću Delphi-ja za samo nekoliko sati rada i na taj način uštedeti mnogo novca Vašoj kompaniji.

Drugi razlog zbog kojeg bi kompanije želele poseban Web Browser je ograničavanje pristupa Web-u. Na primer, postoje mesta na Internet-u koje zaposleni moraju da posete s vremena na vreme. Poseban Web Browser dozvoljava pristup tim, autorizovanim mestima na Web-u, ali ne i drugim, neautorizovanim mestima. U stvari, poseban Web Browser bi bio odličan za Vašu decu.

Konačno, jedan od najvažnijih razloga za pravljenje posebnog Web Browser-a je *intranet*. Intranet je Web lokacija koja je lokalna za određenu mrežu. Intranet može sadržati mnoštvo informacija koje se koriste unutar same kompanije (informacije o kompaniji, informacije o poslovanju kompanije, telefonski imenik zaposlenih, raspored zakazanih sastanaka ili, čak, informacije o košarkaškoj ligi kompanije).

Poseban Web Browser može omogućiti pristup intranet-u i zabraniti pristup Internet-u.

Na osnovu ovih zapažanja ćete napraviti jednostavan Web Browser. Verovatno ćete biti iznenađeni koliko je to jednostavan posao.

Prvi koraci u pravljenju Vašeg Web Browser-a

`THML` kontrola je Web Browser koji je spreman za upotrebu. Sve što treba da uradite je da postavite jednu od ovih kontrola na formu i da pozovete `RequestDoc` metodu. Ovo je malo previše pojednostavljen pristup problemu, ali Vi možete, bez ikakvih problema, na ovaj način pristupiti Web dokumentu koji se nalazi bilo gde na Internet-u. Na osnovu ovoga ću Vam pokazati kako da napravite Web Browser. Slede prvi koraci:

1. Napravite novi program. Izmenite `Name` osobinu forme u `WebMain` i `Caption` osobinu u `EZ Web Browser`.
2. Postavite `Panel` komponentu na formu i postavite njenu `Align` osobinu na `alTop` i njenu `Height` osobinu na `60`. Izbrišite vrednost `Caption` osobine.
3. Postavite `ComboBox` komponentu na panel. Pomerite je na vrh panela i proširite je do širine samog panela. Izmenite njenu `Name` osobinu u `URLComboBox`. Izmenite njenu `Text` osobinu u proizvoljni URL (probajte `www.turbopower.com`). Dva puta kliknite na `Constraints` osobinu i izmenite `AnchorHorz` u `akStretch`.
4. Postavite `StatusBar` komponentu na formu. Sama će se postaviti na dno forme. Izmenite njenu `Name` osobinu u `StatusBar` u njenu `SimplePanel` osobinu u `True`.
5. Postavite `THTML` kontrolu na sredinu forme. Izmenite njenu `Align` osobinu u `alClient`. `THTML` kontrola će ispuniti ekran. Izmenite njenu `Name` osobinu u `HTML`.

   Sada bi Vaša forma trebalo da liči na onu sa slike DD.1. Ukoliko Vaš program ne izgleda potpuno isto kao i slika DB.1, izmenite ga, ili ga ostavite onakvim kakav jeste. (Malo originalnosti nije na odmet.)

   Sada biste trebali da sačuvate projekat. Sačuvajte formu pod imenom `WebBrwsU.pas` i projekat pod imenom `WebBrows.dpr`. Sada ćete dodati još osobina koje će naterati Web Browser da radi nešto korisno.

Slika DD.1
Vaš novi Web Browser posle prvih koraka

6. Kliknite na URL kombo-listu. Napravite proceduru za obradu `OnClick` događaja. Unesite sledeći kod u proceduru:

```
if URLComboBox.Text <> '' then
  HTML.RequestDoc(URLComboBox.Text);
```

`RequestDoc` metoda učitava traženi dokument, pošto se utvrdi da kombo-lista sadrži neki tekst.

7. Sada treba da napravite proceduru za obradu `OnKeyPress` događaja. Unesite sledeći kod u proceduru:

```
if Key = Char(VK_RETURN) then begin
  Key := #0;
  if URLComboBox.Text = '' then
    Exit;
  URLComboBoxClick(Sender);
end;
```

Ovaj kod prvo proverava `Key` parametar da bi utvrdio da li je pritisnut taster Enter. Ukoliko jeste, vrednost parametra `Key` se postavlja na 0 i poziva se `URLComboBoxClick` metoda (napravili ste je u koraku 6). Postavljanjem vrednosti parametra `Key` na 0 se sprečava oglašavanje zvučnika kada se pritisne taster Enter. Poziv `URLComboBoxClick` metode učitava URL u Web Browser.

8. Prevedite i pokrenite Vaš program. Unesite URL i kombo-listu i pritisnite taster Enter. Ukoliko ste uneli ispravan URL, stranica će biti učitana u `HTML` kontrolu.

Pa to je Web Browser za samo 15 minuta! Primetite da se Vaš program ponaša kao i bilo koji drugi Web Browser, ili makar približno. Potrebno je da dodate još mnogo stvari, ali i ovo nije loše za početak.

Ukoliko ste jedan od srećnika koji ima stalni pristup Internet-u, Vaš Web Browser će odmah proraditi. Ukoliko koristite Dial-Up Networking sa uključenom auto-dial opcijom, program za biranje će Vas automatski povezati za Vaš ISP (engl. Internet Service Provider). Ukoliko nemate instaliran Dial-Up Networking, moraćete ručno da se vežete na Internet pre nego što pokrenete program.

Dodavanje pokazivača procesa

Sada imate dobru polaznu osnovu za pravljenje ozbiljnijeg Web Browser-a. Jedna od mogućnosti koja nedostaje je i informacija učitavanju stranice. Sada ćete napraviti proceduru koja osvežava pokazivač procesa u toku učitavanja stranice. Koristićete `OnUpdateRetrieval` i `OnEndRetrieval` događaje THTML kontrole da biste dobili informacije o procesu učitavanja stranice. Koristićete `GetBytesTotal` i `GetBytesDone` metode da biste izračunali procenat do kojeg je stranica učitana. Zatim ćete prikazali tu informaciju u statusnoj liniji. Da li ste spremni?

1. Kliknite na `HTML` kontrolu Vaše forme. Napravite proceduru za obradu `OnUpdateRetrieval` događaja. Dodajte sledeći kod u proceduru:

```
procedure TWebMain.HTMLUpdateRetrieval(Sender: TObject);
var
  Total   : Integer;
  Done    : Integer;
  Percent : Integer;
begin
  Total := HTML.RetrieveBytesTotal;
  Done  := HTML.RetrieveBytesDone;
  if (Total = 0) or (Done = 0) then
    Percent := 0
  else
    Percent := ((Done * 100) div Total);
  StatusBar.SimpleText := Format(
    'Getting Document: %d%% of %dK', [Percent, Total div 1024]);
end;
```

2. Napravite proceduru za obradu `OnEndRetreival` događaja. Unesite sledeći kod u proceduru:

```
StatusBar.SimpleText := 'Done';
```

Pogledajte malo bolje kod iz koraka 1. Nije posebno komplikovan. `GetBytesTotal` Vam pokazuje kolika je veličina dokumenta koji se trenutno učitava i objekata na njemu (u objekte spadaju i slike). `GetBytesDone` metoda Vam daje broj bajtova koji su do tog trenutka preneseni sa Internet-a. Sada je jednostavno izračunati koliki je procenat dokumenta učitan. Konačno, formatirate string sa informacijama dobijenim od `HTML` kontrole i šaljete ga statusnoj liniji. Kod iz koraka 2 jednostavno osvežava statusnu liniju, pošto se učita ceo dokument.

Ponovo pokrenite program i pratite šta se dešava tokom učitavanja stranice. Statusna linija prikazuje koliko je procenata dokumenta i objekata na njemu učitano.

Završne operacije

A sada slede završne operacije. Prvo ćete dodati nekoliko tastera ispod URL kombo-liste. (Bacite pogled na završeni program koji je prikazan na slici DD.2.) Uradite sledeće:

1. Postavite taster na panel ispod URL kombo-liste. Izmenite njegovu `Name` osobinu u `GoBtn` i njengovu `Caption` osobinu u `Go!`.
2. Napravite proceduru za obradu `OnClick` događaja novog tastera. Unesite sledeći kod u proceduru:

```
URLComboBoxClick(Self);
```

3. Postavite sledeći taster na panel, sa desne strane prethodnog. Izmenite njegovu `Name` osobinu u `StopBtn` i njegovu `Caption` osobinu u `Stop`.
4. Napravite proceduru za obradu `OnClick` događaja i ovog tastera. Unesite sledeći kod u proceduru:

```
HTML.Cancel(0);
StatusBar.SimpleText := 'Done';
```

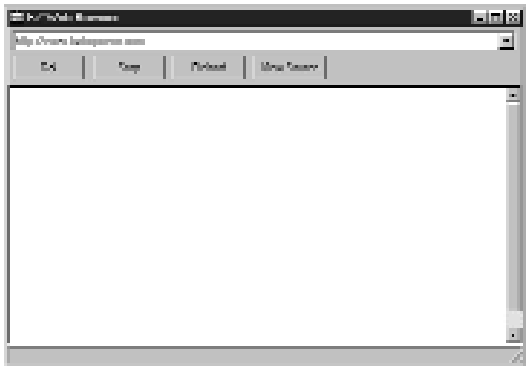
5. Postavite treći taster na panel i to sa desne strane od prethodna dva. Izmenite njegovu `Name` osobinu u `ReloadBtn` i njegovu `Caption` osobinu u `Reload`.
6. Napravite proceduru za obradu `OnClick` događaja i ovog tastera, a zatim u nju unesite isti kod kao i u koraku 2:

```
URLComboBoxClick(Self);
```

7. Postavite četvrti (i poslednji) taster na panel. Izmenite mu Name osobinu u `SoruceBtn` i Caption osobinu u `View Source`.
8. Napravite proceduru za obradu `OnClick` događaja i unesite sledeći kod:

```
HTML.ViewSource := not HTML.ViewSource;
if HTML.ViewSource then
  SourceBtn.Caption := 'View Document'
else
  SourceBtn.Caption := 'View Source';
```

Vaš forma bi sada trebala da izgleda kao i ona na slici DD.2.



Slika DD.2
EZ Web Browser sa postavljenim tasterima

Ovi koraci Vam predstavljaju nekoliko novih `THTML` elemenata. `Cancel` metoda prekida proces učitavanja dokumenta. `ViewSource` osobina se koristi za prelazak iz režima pregleda HTML dokumenta u režim pregleda HTML izvornog koda i obrnuto.

Ponovo pokrenite program. Isprobajte nove tastere i proverite da li rade na očekivani način. Posebnu pažnju obratite na `View Source` taster.

Dobro, još malo pa ste završili svoj Web Browser. Dodajmo još nekoliko osobina. Obradićete još dva događaja `THTML` kontrole, kako biste obezbedili više informacija o toku učitavanja stranice:

1. Napravite proceduru za obradu `OnDoRequestDoc` događaja `THTML` kontrole. Unesite sledeći kod u proceduru:

```
StatusBar.SimpleText := 'Connecting to ' + URL + '...';
```

2. Sada napravite proceduru za obradu `OnBeginRetrieval` događaja. Kada se procedura pojavi na ekranu, unesite sledeći kod:

```
StatusBar.SimpleText := 'Connected...';
URLComboBox.Items.Insert(0, URLComboBox.Text);
```

U ovom delu, korak 1 dodaje mogućnost korišćenja `OnDoRequestDoc` događaja, koji se generiše svaki put kada se zatraži neki dokument. `URL` parametar procedure za obradu `DoRequestDoc` događaja je URL mesto na koje se trenutno vezujete. Dokle god je URL parametar na raspolaganju, možete ga koristiti za prikazivanje stringa u statusnoj liniji. Korak 2 dodaje još jednu informaciju i to u trenutku kada dokument počne sa učitavanjem. On, takođe, dodaje URL u kombo-listu. Morate biti sigurni da ste se vezali za određeno mesto pre nego što dodate njegov URL u listu posećenih mesta.

Čestitamo. Upravo ste završili (ili skoro završili) svoj prvi Internet program. Slika DD.3 prikazuje Vaš EZ Web Browser u toku rada.

Slika DD.3 Završeni EZ Web Browser prikazuje stranicu

To je bio dobar posao. Postoje neke stvari koje Vaš Web Browser ne radi, ali je više onih koje radi, tako da možete biti ponosni. Zastanite za trenutak i divite se svom delu. Možete dodati neke nove osobine svom programu. Sigurno biste želeli da dodate tastere koji omogućavaju kretanje kroz posećena mesta (Back i Next). Takođe biste mogli i da zamenite standardne tastere sa paletom alatki i da na njene tastere dodate sličice. Ukoliko želite da načinite veliki korak unapred, pustite animaciju prilikom učitavanja stranice, tako da Vaši korisnici vide da Vaš program zaista nešto radi. To najlakše možete uraditi pomoću `TImageList` komponente. `TAnimate` komponenta će takođe odraditi posao.

`THTML` kontrola ima nekoliko osobina koje nisam spomenuo. Najveći deo tih osobina se odnosi na korisničke opcije kao što su: boja pozadine, boja linkova, boja posećenih linkova, različiti fontovi koji se koriste za različite veličine naslova i tako dalje. Neću objašnjavati ove osobine, pošto su vrlo jednostavne. Za najveći broj ovih osobina možete koristiti podrazumevane vrednosti. Ukoliko želite da dalje unapređujete svoj Web Browser, upoznajte se detaljnije sa svim osobina `THTML` kontrole.

Listing DD.1 prikazuje izvorni kod glavne celine Web Browser-a.

Listing DD.1: `WebBrwsU.pas`

```
unit WebBrwsU;

interface

uses
  Windows, Messages, SysUtils, Classes, Graphics, Controls, Forms,
Dialogs,
  StdCtrls, ExtCtrls, OleCtrls, NMHTML, ComCtrls;

type
```

nastavlja se

Listing DD.1: WebBrwsU.pas *nastavak*

```
    TWebMain = class(TForm)
      Panel1: TPanel;
      URLComboBox: TComboBox;
      StatusBar: TStatusBar;
      HTML: THTML;
      GoBtn: TButton;
      StopBtn: TButton;
      ReloadBtn: TButton;
      SourceBtn: TButton;
      procedure URLComboBoxClick(Sender: TObject);
      procedure URLComboBoxKeyPress(Sender: TObject; var Key: Char);
      procedure HTMLUpdateRetrieval(Sender: TObject);
      procedure HTMLEndRetrieval(Sender: TObject);
      procedure GoBtnClick(Sender: TObject);
      procedure StopBtnClick(Sender: TObject);
      procedure ReloadBtnClick(Sender: TObject);
      procedure SourceBtnClick(Sender: TObject);
      procedure HTMLDoRequestDoc(Sender: TObject; const URL:
WideString;
        Element: HTMLElement; DocInput: DocInput;
        var EnableDefault: WordBool);
      procedure HTMLBeginRetrieval(Sender: TObject);
    private
      { Private declarations }
    public
      { Public declarations }
    end;

var
  WebMain: TWebMain;

implementation

{$R *.DFM}

procedure TWebMain.URLComboBoxClick(Sender: TObject);
begin
  if URLComboBox.Text <> '' then
    HTML.RequestDoc(URLComboBox.Text);
end;

procedure TWebMain.URLComboBoxKeyPress(Sender: TObject; var Key:
Char);
begin
  if Key = Char(VK_RETURN) then begin
    Key := #0;
    if URLComboBox.Text = '' then
      Exit;
```

```
    URLComboBoxClick(Sender);
  end;
end;

procedure TWebMain.HTMLUpdateRetrieval(Sender: TObject);
var
  Total   : Integer;
  Done    : Integer;
  Percent : Integer;
begin
  Total := HTML.RetrieveBytesTotal;
  Done  := HTML.RetrieveBytesDone;
  if (Total = 0) or (Done = 0) then
    Percent := 0
  else
    Percent := ((Done * 100) div Total);
  StatusBar.SimpleText := Format(
    'Getting Document: %d%% of %dK', [Percent, Total div 1024]);
end;

procedure TWebMain.HTMLEndRetrieval(Sender: TObject);
begin
  StatusBar.SimpleText := 'Done';
end;

procedure TWebMain.GoBtnClick(Sender: TObject);
begin
  URLComboBoxClick(Self);
end;

procedure TWebMain.StopBtnClick(Sender: TObject);
begin
  HTML.Cancel(0);
  StatusBar.SimpleText := 'Done';
end;

procedure TWebMain.ReloadBtnClick(Sender: TObject);
begin
  URLComboBoxClick(Self);
end;

procedure TWebMain.SourceBtnClick(Sender: TObject);
begin
  HTML.ViewSource := not HTML.ViewSource;
  if HTML.ViewSource then
    SourceBtn.Caption := 'View Document'
  else
    SourceBtn.Caption := 'View Source';
end;
```

nastavlja se

Listing DD.1: `WebBrwsU.pas` *nastavak*

```
procedure TWebMain.HTMLDoRequestDoc(Sender: TObject; const URL:
WideString;
  Element: HTMLElement; DocInput: DocInput; var EnableDefault:
WordBool);
begin
  StatusBar.SimpleText := 'Connecting to ' + URL + '...';
end;

procedure TWebMain.HTMLBeginRetrieval(Sender: TObject);
begin
  StatusBar.SimpleText := 'Connected...';
  URLComboBox.Items.Insert(0, URLComboBox.Text);
end;

end.
```

Korišćenje Internet Explorer-a u obliku ActiveX kontrole

Ukoliko imate instaliran Microsoft Internet Explorer, na svom računaru, možete ga koristiti kao ActiveX kontrolu. Prvo što treba da uradite jeste da uvezete kontrolu u Delphi-jevu paletu sa komponentama. Posle toga je možete postaviti na formu, kao i bilo koju drugu kontrolu. Prvo, dozvolite mi da Vam pokažem kako da uvezete Internet Explorer. (Pokazao sam Vam kako se uvoze ActiveX kontrole u danu 15 "COM i ActiveX", ali nije na odmet ponoviti.) To se radi na sledeći način:

1. Izaberite Component/Import ActiveX Control iz Delphi-jevog glavnog menija. Prikazuje se Import ActiveX dijalog-prozor.
2. Pronađite `Microsoft Internet Controls` (`Version 1.x`) u listi raspoloživih ActiveX kontrola (pogledajte sliku DD.4). Ukoliko imate instaliran Internet Explorer 3, broj verzije će biti 1.0. Ukoliko imate instaliran Internet Explorer 4, broj verzije će biti 1.1. Primetite da polje `Class names` sadrži `TWebBrowser`, što je u stvari kontrola koja se nalazi u ovom fajlu. Ukoliko imate Internet Explorer 4, polje `Class names` sadrži i `TWebBrowser_V1` (originalna `WebBrowser` kontrola) i `TShellFolderViewOC`.

Slika DD.4
Import ActiveX dijalog-prozor

3. Kliknite na Install da biste instalirali kontrolu (ostatak polja u ovom dijalog-prozoru možete ostaviti sa podrazumevanim vrednostima).

4. Pojavljuje se Install dijalog-prozor koji zahteva da unesete ime paketa. Kliknite na `Into new package`, koji se nalazi na vrhu dijaloga i u polje `File name` unesite `IE`. (Možete uneti i opis, ali to nije neophodno.) Kliknite na taster OK da biste napravili paket.

5. Prikazuje se dijalog-prozor kroz koji potvrđujete da želite da napravite i instalirate paket. Kliknite na Yes da biste napravili paket.

Delphi pravi paket i prijazuje dijalog-prozor u kome obaveštava da je `TWebBrowser` kontrola instalirana. Sada možete isprobati kontrolu:

1. Prvo, izaberite File➥Close All da biste zatvorili sve prozore a zatim napravite novi program.

2. Kliknite na ActiveX stranicu na paleti sa komponentama. Izaberite `WebBrowser` kontrolu i spustite je na glavnu formu. Promenite veličinu kontrole, ukoliko to želite, ali ostavite dovoljno prostora za taster.

3. Postavite `Button` komponentu na formu. Dva puta kliknite na nju da biste napravili proceduru za obradu `OnClick` događaja. Unesite sledeći kod u proceduru (možete koristiti proizvoljni URL):

```
WebBrowser1.Navigate('http:\\www.turbopower.com',
  EmptyParam, EmptyParam, EmptyParam, EmptyParam););
```

   Kao što i pretpostavljate, `Navigate` metoda učitava dokument u Web Browser.

4. Kliknite na Run da biste pokrenuli program.

Kada se program pokrene, kliknite na taster koji se nalazi na formi. Učitaće se Web strana i njen sadržaj će se prikazati u `WebBrowser` kontroli.

Kada ste instalirali kontrolu, Delphi je napravio celinu pod nazivom `SHDocVw_TLB.pas`. Pregledajte ovu celinu da biste saznali koje osobine i metoda Vam stoje na raspolaganju za `TWebBrowser`.

SAVET Možete naći dokumentaciju za `WebBrowser` kontrolu na Microsoft-ovom Web sajtu. Pretražite sajt po tekstu: `Reusing the WebBrowser Control`.

Primetite da ne možete distribuirati `TWebBrowser` kontrolu, pre nego što dobijete licencu od Microsoft-a. Ipak, ukoliko Vaši korisnici imaju instaliran Internet Explorer, Vaš program će raditi, pošto je ActiveX kontrola već instalirana na njihovim računarima. Ipak, Vi morate registrovati kontrolu na računarima Vaših korisnika. Pogledajte odeljak "Isporučivanje Internet programa", koji se nalazi u ovom poglavlju, da biste dobili više informacija o registrovanju ActiveX kontrola.

Slanje elektronske pošte

Postoji više dobrih razloga zbog kojih biste želeli da šaljete elektronsku poštu iz Vaših programa. Dobra vest je da slanje elektronske pošte uopšte nije teško. Možda biste želeli da Vaši korisnici mogu da Vam pošalju poštu sa problemima na koje su naišli. U tom slučaju bi Vaš program mogao da sadrži formu sa `Memo` komponentom i Send tasterom. Vaši korisnici treba da unesu tekst poruke u `Memo` komponentu i da pritisnu Send taster. Tada će poruka biti poslata Vama. Možete ići i dalje pa, recimo, vezati log fajl Vašeg programa za poruku i na taj način otkriti probleme sa kojima se susreću Vaši korisnici.

`TNMSMTP` kontrola se koristi za slanje elektronske pošte kroz SMTP server. SMTP je jednostavan protokol, premda ne zahteva nikakvu autorizaciju prilikom logovanja na server (to važi za većinu SMTP servera). Možete se jednostavno vezati na bilo koji mail server, poslati poruku i isključiti se. `Host` osobina se koristi za određivanje imena servera na koji želite da se vežete. Uglavnom ćete moći da koristitite mail kao ime servera. Pošto ste naveli `mail` kao ime mail servera, `TNMSMTP` kontrola će Vas vezati za lokalni `mail` server, bez obzira da li je to mail server Vašeg ISP-a, ili Vaše kompanije.

Ukoliko želite, možete i eksplicitno navesti ime servera (na primer, `mail.mycompany.com`), ali to uglavnom nije neophodno. Ukoliko navedete pogrešno ime servera, generisaće se `ESockError` izuzetak. `Port` osobina se koristi za određivanje port-a na koji želite da se vežete. Podrazumevani `port` za SMTP je port `25`. Podrazuvana vrednost Port osobine je 25 tako da, verovatno, nećete morati da je menjate.

Svi podaci o samoj poruci se nalaze u `PostMessage` osobini. Ova osobina je klasa koja sadrži osobine kao što su: `ToAddress`, `FromAddress`, `Subject`, `Body` i tako dalje. Morate popuniti odgovarajuća polja u `PostMessage` osobini i tek onda poslati poruku.

Pre nego što pošaljete poruku, morate se vezati na SMTP server. To se radi pomoću `Connect` metode:

```
SMTP.Host := 'mail';
SMTP.Connect;
```

Pošto ste se vezali, možete poslati poruku. Procedura za obradu `OnConnect` događaja je dobro mesto za ovu operaciju, premda ste tada sigurni da ste se zaista i vezali na server. Na primer:

```
procedure TMainForm.SMTPConnect(Sender: TObject);
begin
  with SMTP.PostMessage do begin
    FromAddress := 'bilbo@baggins.com';
    ToAddress.Add('gandolf@baggins.com');
    Subject := 'Test';
    Body.Add('This is a test');
  end;
  SMTP.SendMail;
end;
```

Ovaj kod postavlja `FormAddress`, `ToAddress`, `Subject` i `Body` parametre `PostMessage` osobine i zatim šalje poruku korišćenjem `SendMail` metode. I, to je sve. Primetite da su `ToAddress` i `Body` osobine PostMessage-a podaci tipa `TStringList`. To je zato što sam tekst poruke može biti sastavljen od više linija i zato što se u `ToAddress` osobini može naći nekoliko primalaca kojima će biti poslata ista poruka.

U polja `FromAddress` i `ToAddress` morate uneti neke vrednosti. U sva ostala polja možete, ali i ne morate. Čak i sam tekst poruke može ostati prazan.

Pošto utvrdite da je poruka uspešno poslata, možete se isključiti sa SMTP servera. Prilikom uspešnog slanja poruke se generiše `OnSuccess` događaj. Vaša procedura za obradu ovog događaja može biti čak i ovoliko jednostavna:

```
void __fastcall TForm1.SMTPSuccess(TObject *Sender then
begin
  SMTP.Disconnect;
end;
```

Naravno, Vi možete poslati nekoliko poruka, dok ste vezani za SMTP server. Ukoliko to i želite, ne morate se vezivati i isključivati za svaku poruku. Jednostavno, vežite se jednom, pošaljite sve poruke, a zatim se isključite sa servera.

Vaša poruka može biti poslata bez ikakvih problema ali, takođe, do problema može doći. U svakom slučaju, morate biti spremni da se isključite sa servera. Ukoliko dođe do greške prilikom slanja poruke, generiše se `OnFailure` događaj. Možete koristiti, ili ovaj, ili `OnSuccess` događaj da biste se isključili sa servera. Izvorni kod iz ove knjige, koji se nalazi na `http://www.mcp.com/info` sadrži jednostavan program za rad sa elektronskom poštom koji demonstrira slanje poruka pomoću `TNMSMTP`.

Isporučivanje Internet programa

Ukoliko Vaši Internet programi koriste samo VCL komponente, prilikom isporučivanja programa nećete morati da radite ništa posebno, dokle god ne koriste pakete. Ukoliko koristite pakete, moraćete da isporučite i `INET40.BPL`, a ukoliko koristite i kontrole za pravljenje dinamičkih Web strana, moraćete da isporučite i `INETDB40.BPL`.

Isporučivanje programa koji koristi ActiveX kontrole zahteva nešto više rada. ActiveX kontrole moraju biti registrovane na računaru na kome će se koristiti Vaš program. Najlakši način za registrovanje ActiveX kontrola je korišćenje dobrog instalacionog programa (na primer, InstallShield Express, koji se isporučuje sa profesionalnom i klijent/server verzijom Delphi-ja). Trebali biste da koristite taj program. Drugi dobar program je Wise Install, koji je proizvod kompanije Great Lake Business Solutions. Dobar instalacioni program, prilikom instalacije, registruje sve ActiveX kontrole, koje Vaš program koristi.

Ukoliko ne koristite komercijalni instalacioni program, morate ručno da registrujete sve ActiveX kontrole koje koristi Vaš program. `TREGSRV.EXE` pomoćni program se koristi za registrovanje i izbacivanje ActiveX i OCX kontrola. Ovaj pomoćni program se nalazi u `Delphi 4\Bin` direktorijumu. Na primer, da biste instalirali EZ Web Browser program, koji ste napravili danas, isporučite sledeće fajlove:

```
HTML.OCX
NMOCOD.DLL
NMSCKN.DLL
NWM3VWN.DLL
NMORENU.DLL
WEBBROWS.EXE
```

Pošto ste instalirali ove fajlove, morate pokrenuti `TREGSVR.EXE` da biste registrovali `HTML.OCX` i `NMOCOD.CLL`. Uradite sledeće u komandnoj liniji:

```
TREGSVR HTML.OCX
```

Isto uradite i za `NMOCOD.DLL`. Sada je `THTML` kontrola registrovana na računaru Vašeg korisnika i Vaš program će raditi kao što je i očekivano.

Ukoliko ne registrujete ActiveX kontrole na odgovarajući način, prilikom pokretanja programa, Vaši korisnici će videti "`Class not registred`." poruku. Vaš program će zatim završiti sa radom i Vašim korisnicima neće biti jasno šta, u stvari, nije u redu.

Da biste izbacili kontrolu iz Registry baze, koristite prekidač /u i to na sledeći način:

```
TREGSVR /u HTML.OCX
```

Da napomenemo još jednom, dobar instalacioni program ima opciju za deinstaliranje programa, koja će to uraditi umesto Vas.

Kao što možete videti, ActiveX kontrole zahtevaju nešto više rada prilikom instaliranja. Ukoliko ne obratite pažnju na registrovanje ActiveX kontrola, možete zbuniti

i sebe i svoje korisnike. Usput, ukupna veličina fajlova koje je potrebno isporučiti sa programom koji koristi `THTML` kontrolu je oko 900kB. Može se zaključiti da korišćenje ActiveX kontrola zahteva dosta mesta na disku. Baš iz tog razloga, radije koristim klasične VCL kontrole, kada god je to moguće.

Zaključak

Danas ste naučili nešto o Internet komponentama koje se isporučuju uz Delphi. Napravili ste jednostavan, ali upotrebljiv Web Browser koji koristi `THTML` kontrolu i naučili ste kako da šaljete elektronsku poštu korišćenjem `TNMSMTP` kontrole. Internet programiranje je, trenutno, veliki posao. Sigurno je da Vam poznavanje Internet programiranja ne može smetati.

Radionica

Radionica sadrži test pitanja koja Vam pomažu da učvrstite svoje razumevanje izložene materije i vežbe koje Vam pomažu da steknete iskustvo u onome što ste naučili. Možete pronaći odgovore na test pitanja u Dodatku A "Odgovori na test pitanja".

Pitanja i odgovori

P **Koje komponente, ili kontrole bih trebao da koristim, da bih napravio TCP/IP klijent/server program?**

O Koristite `TClientSocket` i `TServerSocket` komponente.

P **Mogu li napraviti Web stranu iz mojih tabela koje se nalaze u bazi podataka?**

O Da. `TQueryTableProducer` i `TDataSetTableProducer` komponente prave HTML dokumente iz tabela koje se nalaze u bazi podataka. Naučićete više o ovim kontrolama, ako proučite primere koji se nalaze u Delphi `4\Demos\Webserv` direktorijumu.

P **`TNMSMTP` kontrola se koristi za slanje elektronske pošte, a `TNMPOP3` za prijem. Zbog čega postoje dve kontrole za rad sa elektronskom poštom?**

O Postoje dve kontrole zato što postoje i dva različita protokola za rad sa elektronskom poštom: jedan za slanje pošte (SMTP) a drugi za prijem (POP3).

P **Kada je definisan najveći deo Internet protokola (SMTP, POP3, FTP, UDP i tako dalje)?**

O Možda ćete biti iznenađeni, ali najveći broj protokola koji se koriste u Internet programiranju su definisani pre 20 godina, iako je sam Internet mnogo mlađi. Internet protokoli su prvenstveno bili napravljeni za korišćenje na UNIX platformi.

Kviz

1. Koju kontrolu treba koristiti za prikazivanje Web strana?
2. Koju kontrolu treba koristiti za vezivanje na News grupe?
3. Kako se zove metoda koja se koristi za prikazivanje HTML dokumenta u okviru `THTML` kontrole?
4. Koji događaj se generiše kada se HTML dokument učita u celini?
5. Koju kontrolu treba koristiti za slanje elektronske pošte?
6. Koju kontrolu treba koristiti za prijem elektronske pošte?
7. Kako se zove metoda koja se koristi za slanje pošte putem `TNMSMTP` kontrole?
8. Da li možete slobodno distribuirati Internet Explorer ActiveX kontrolu?
9. Kako se zove pomoćni program koji se koristi za registrovanje ActiveX kontrola?
10. Koja kompanija proizvodi najveći deo Internet kontrola koje se isporučuju uz Delphi?

Vežbe

1. Napravite program za slanje elektronske pošte. Glavna forma u programu bi trebala da sadrži polja za unos adrese sa koje se šalje, adrese na koju se šalje, teme poruke i samog teksta poruke.
2. Dodaje Forward i Back taster EZ Web Browser programu i načinite ih funkcionalnim.
3. **Posebna vežba:** Dodajte animaciju EZ Web Browser programu, tako da korisnik vidi da Web Browser trenutno učitava neki dokument.

Naučite

Delphi 4

Kent Reisdorph

kompjuter biblioteka

SAMS PUBLISHING

Izdavač:
"Kompjuter biblioteka"
Vladana Šićevića 19,
32000 Čačak
tel: 032/ 27-632, 32-322,
fax: 032/ 32-322
žiro-račun: 41300-601-6-15715
E-mail:
kombib@emi.yu
Internet:
www.kombib.co.yu

Urednik:
Mihailo J. Šolajić
Prevod:
Stevica Gološin
Dejan Pervulov
Recenzija i lektura
Radojka Krneta
Indeks:
Zora Radojević

Dizajn:
Zoran Pavlović
Slog:
Milutin Petković
Ana Pešić
Zora Radojević

Znak Kompjuter biblioteke:
Miloš Milosavljević

Štampa:
"Svetlost" Čačak

Godina izdanja:
1999.
Izdanje:
Prvo
ISBN:
86-7310-035-6

Sams Teach Yourself Delphi™ 4 in 21 Days

Kent Reisdorph

ISBN: 0-672-31286-7

Naučite za 21 dan Delphi 4

1
- Otkrijte kako funkcioniše Delphi-jevo okruženje i jezik Object Pascal; kratak pregled Delphi paketa i uvod u Object Pascal jezik.

2
- Nastavite da učite o jeziku Object Pascal, istražujući detaljnije osnove jezika Object Pascal, kao i važne ključne reči i iskaze.

3
- Naučite klase - suštinu jezika Object Pascal i osnovni deo objektno orjentisanog programiranja.

4
- Detaljnije pregledajte Delphi-jevo okruženje, osnovne strukture menija, kako opcije menija funkcionišu i kako u celini funkcioniše ikruženje.

5
- Naučite o modelu vizuelnih komponenti i Borlandovoj biblioteci vizuelnih komponenti (VCL), kao i o kostur klasa.

6
- Počnite rad sa dizajnerom formi i dizajnerom menija - oba alata su važna u Delphi-jevom okruženju za vizuelno programiranje.

7
- Naučite više o komponentama. Naučićete o komponentama koje se često koriste, a dodatno ćete naučiti i klase koje predstavljaju te komponente.

8
- Kreirajte aplikacije u Delphiju uz pomoć skladišta objekata, čarobnjaka za dijaloge i čarobnjaka za aplikacije, kao i uz pomoć mnogih drugih alata.

9
- Naučite sve o projektima, o korišćenju menadžera projekta i projektnim grupama. Projekti su smisao života Delphi-ja.

10
- Naučite kako da debagirate Vaše programe kori-steći debager iz okruženja. Poglavlje obuhvata razloge za debagiranje, kao i načine debagiranja programa.

11
- Naučite da koristite neke od Delphi-jevih nenadmašnih alata, editor slika, Windows sistem poruka, WinSight i alate komandne linije.

12
- Zavirite u programira-nje multimedije i grafike uz pomoć Delphi-ja, koristeći klase TCanvas i TBitmap. Naučite o Delphi-jevoj komponenti - TMediaPlayer.

13
- Zaronite u važne pojmove za programiranje, kao što su aktiviranje komandi, štampanje iz Delphi aplikacija i korišćenje kursora.

14
- Uživajte u elementima naprednog programiranja, kao što je implementiranje kontekst pomoći, upravljanje izuzecima i rad sa Windows Registry bazom.

15
- Počnite da učite o Microsoft Component Object Model arhitekturi. Lekcija sadrži uvod u COM, ActiveX kontrole i Delphi Type Library Editor.

16
- Pročitajte uvod u Delphi arhitekturu baza podataka, kao i opis Delphi komponenti za baze podataka.

17
- Istražite Delphi-jeve forme baza podataka i kreirajte forme baza podataka, koristeći Delphi-jevog čarobnjaka za forme baza podataka.

18
- Kreirajte i napunite bazu podataka u potpunosti, koristeći kod, a zatim naučite sve o modulima i QuickReport-u.

19
- Istražite DLL datoteke, šta su DLL datoteke i zašto treba da ih koristite, kako da ih pišete i kako da koristite forme u DLL datotekama.

20
- Kreirajte sopstvenu Delphi komponentu, testirajte komponentu i dodajte je u paletu komponenti.

21
- Naučite sličnosti i razlike između Delphi-ja i C++Builder-a. Naučite kako da razmenjujete kod između Delphi-ja i C++Builder-a.

1 SEDMICA

Pregled sadržaja

Ove nedelje ste obradili veliku oblast. Na neki način ovo je bila najteža nedelja. Object Pascal, iako nije tako zahtevan kao neki drugi programski jezici, još uvek zahteva dosta vremena da bi se naučio. Iako nema sumnje da ćete naučiti da programirate na Delphiju ukoliko natavite da radite. Nemojte zaboraviti da s vremena na vreme odmorite. Ova knjiga ima naslov: Sams, naučite Delphi za 21 dan, ali to ne znači da treba da učite 21 dan uzastopno! Ponekad je dobro da odmorite nekoliko dana da bi se sve sleglo.

Ukoliko ste zbunjeni sintaksom u okviru jezika Object Pascal, nemojte misliti da ste jedini. U početku se svi zbune. Ipak, nemojte brinuti, pošto ćete ubrzo uhvatiti priključak. Dok radite sa Delphijem, malo po malo sve počinje da dobija smisao. Ono što Vam u ovom trenutku verovatno nedostaje, je iskustvo u radu sa svakodnevnim programima. Sa takvim primerima ćete baš stvarno naučiti Delphi. Znanje stečeno iskustvom je ta vrsta veze. Moj savet bi bio da smislite nešto i da počnite sa pisanjem programa. Nećete moći da ga u potpunosti napišete u ovom trenutku, ali ćete moći da ga dobro počnete. Program ne mora da bude tako napredan kao što je Microsoft Word, Netscape Navigator, odnosno igra kao što je DOOM, ako tako mislite. Program bi mogao biti nešto kratko, što će Vam pomoći da povežete Vaše znanje sa iskustvom.

Prvi deo ove nedelje, radili ste sa jezikom Object Pascal, ključnim rečima i sintaksom. Elementi poput petlji i if iskaza su prilično laki za shvatanje. Nemojte se zabrinuti ukoliko se vratite i pogledate sintaksu jednom, ili

više puta. Treba mnogo stvari naučiti i od Vas se ne očekuje da zapamtite svaku ključnu reč i njenu sintaksu. Kasnije ćete biti u stanju, ali u ovom stadijumu igre, to se od Vas ne očekuje.

U lekciji dana 3, predstavljene su Vam klase jezika Object Pascal. Klase su veliki deo jezika Object Pascal i programiranju na Delphiju. Ponekad je potrebno vremena da usvojite gde se klase mogu koristiti u Vašim programima. Dugo vremena ćete možda raditi sa klasama koje postoje u okviru biblioteke vizuelnih komponenti (VCL) i nećete pisati samostalno svoje klase. Kasnije ćete verovatno pronaći situacije kada će se klasa dobro uklopiti u određeni zadatak koji treba da izvršite. Kada dođe vreme, bićete spremni da pređete na pisanje sopstvenih klasa. Kada napišete jednu ili dve, krenućete i raditi dalje.

U lekciji dana 4, učili ste o Delphi okruženju (IDE): kako da prilagodite Vaše povezivanje programa, kako radi paleta komponenti, šta je Object Inspector i čemu služi, itd. U ovam delu nedelje ste iskusili zabavne stvari. Sasvim je adekvatno da se koristi reč zabavno. Smatram sve vrste programiranja uglavnom zabavnim. To je način na koji ja radim. Nadam se da će i Vama, isto tako biti zabavno.

U lekciji dana 5, predstavio sam Vam biblioteke klasa poznate još kao kosturi programa (frameworks). VCL je kostur programa. Kosturi programa Vam olakšavaju život enkapsulirajući teške Windows zadatke za programiranje u klase sa kojima možete raditi na razumnijem nivou. Verujte mi, ponekad je čist Windows API daleko od toga da bude racionalan. VCL vodi računa o tome da radi sa ovakvim stvarima umesto Vas, i da Vam pruži viši nivo objekata za programiranje koje jednostavno možete uklopiti u Vaše aplikacije. Ne, VCL nije jednostavan, ali je puno jednostavniji od rada sa API-jem. Kao deo diskusije o kosturima programa, upoznali ste model komponenti. Naučili ste o karakteristikama, metodama i događajima, kako možete da ih koristite da biste napravili Windows program u Delphiju.

U lekciji dana 6, učili ste o dizajneru forme. Dizajner forme je mesto gde ćete dizajnirati većinu Vaših Delphi aplikacija - on je, ustvari, grafički deo aplikacija. Rad sa dizajnerom forme može biti isto tako zabavan. Korišćenjem dizajnera forme možete kreirati forme koje veoma lepo izgledaju. Zapamtite, forma predstavlja prozor u Vaše aplikacije. Većina aplikacija ima glavni prozor i nekoliko okvira za dijalog koji su prikazani u zavisnosti od interakcije korisnika sa programom. Ovaj program, ScratchPad predstavlja početak u pravljenju aplikacija na Delphiju. Program ScratchPad ćete još koristiti u toku rada sa ovom knjigom. Dok budete gradili svoje znanje programiranja dodaćete nove mogućnosti programu ScratchPad, da bi stekli praktično iskustvo sa tehnikama koje su predstavljene. Ukoliko samostalno razvijate aplikacije, ohrabrujem Vas da dodate nove mogućnosti u Vaš program, onako kako ih budete učili.

U lekciji dana 7, naučili ste nešto od VCL komponenti koje su Vam dostupne. Nisam obradio sve VCL komponente, ali sam obradio komponente koje se najčešće koriste

u Windows programiranju sa Delphijem. U nastavku knjige ćete se upoznati sa ostalim VCL komponentama.

Nadam se da Vas ova nedelja nije umorila. Ukoliko jeste, uzmite kratak predah, a zatim se vratite nazad u igru. Ukoliko smatrate da je ova nedelja bila nedelja koja Vas je uzdigla i napunila energijom, nastavite da okrećete stranice. Ja sam spreman, ukoliko ste i Vi spremni.

1 SEDMICA

Na prvi pogled

U nedelji 1 počećete da učite pisanje Windows programa u Delphiju. Prva tri dana ove nedelje ćete učiti osnove jezika Object Pascal. Dok budete prolazili kroz prva tri poglavlja napisaćete jednostavne test programe da biste učvrstili Vaše razumevanje određenih mogućnosti jezika Object Pascal. Upozoravam Vas takođe da programi u knjizi verovatno nisu tip programa koji ste želeli da pišete kada ste poručili Delphi. Njima nedostaju blještavilo i sjaj. Verovatno nećete biti previše impresionirani. Ovi programi će Vam svakako pomoći da uhvatite osnove jezika Object Pascal.

Počev od dana 4, počećete sa učenjem nekih stvari koje čine Delphi odličnim alatom za vizuelno programiranje, što ustvari jeste. Naučićete o Delphi IDE i kako se on koristi za kreiranje Windows programa. Do kraja dana 4 napravićete Vaš prvi pravi Windows program.

U danu 5, govorimo o kosturima (frameworks) i šta kosturi znače za Vas kao Windows programera. Istog dana ćete naučiti o modelu vizuelnih komponenti, uključujući karakteristike (properties), metode (methods) i događaje (events) koji su vitalni delovi programiranja na Delphiju. U danu 6, naučićete još više o Delphijevom okruženju (Delphi IDE), tako da ćete biti upoznati kako kompletno Delphi okruženje radi u celini, kako bi Vam olakšalo programiranje. Ovde stvari postaju mnogo interesantnije.

U danu 7 objašnjavamo neke komponente biblioteke vizuelnih komponenti (Visual Component Library - VCL) koje ćete najčešće koristiti u toku programiranja na Delphiju. Otkrićete posebne komponente i način kako da ih koristite.

Sigurno ćete provesti neko vreme tokom ove nedelje čitajući. Nadam se da ćete takođe provesti dosta vremena eksperimentišući, čitanje ove knjige nije trka. Prvi koji je završi ne dobija nagradu. Kada učite programiranje, mnogo je bolje biti kor-njača nego zec. Odvojite vreme za eksperimentisanje. Iskustvo je najbolji učitelj, zato ako ste spremni da počnete, okrenite stranu i započnite Vaš izlet u Delphi programiranje.

2 SEDMICA

8
9
10
11
12
13
14

Na prvi pogled

Da li ste spremni za zabavu? Ove nedelje ćete učiti o ozbiljnom Windows programiranju. Počećete sa učenjem novih stvari vezanih za kreiranje aplikacija u Delphiju. Naučićete o kreiranju aplikacija korišćenjem Delphijevih čarobnjaka (Wizards). Ovi čarobnjaci Vam pomažu da za kratko vreme napravite program.

Lekcija dana 9, obrađuje Delphijeve projekte, koji su ključ za upravljanje Vašim datotekama, kada kreirate aplikaciju. Učićete o Delphijevom menadžeru projekta, vitalnom alatu za administriranje projektom. Lekcija dana 9, takođe pokriva Delphijev editor koda (Code Editor). Kada budete uvežbali programiranje na Delphiju, većinu vremena ćete provoditi koristeći editor koda. Dobra ideja je naučiti neke mogućnosti editora koda, kako bi bili efikasniji u programiranju.

U lekciji dana 10, naučićete o uklanjanju grešaka (debagiranju) iz Vašeg programa. Da, Vaši programi će imati greške (bugs - bube). Nemojte ih ubijati. Samo treba da naučite kako da pronađete ova grozna stvorenja u Vašim programima i da ih zgnječite. Debager je vitalan alat za razvoj aplikacija, pa ćete morati da naučite kako da debagirate Vaše programe. Ukoliko znate kako da koristite debager, dugoročno gledajući, uštedećete sate i sate rada.

Lekcija dana 12 Vam predstavlja grafiku i multimedijalno programiranje. Naučićete osnove programiranja grafike, kao što je crtanje figura, prikazivanje bitmapa i rad sa paletama. Takođe ćete naučiti jednostavne multimedijalne operacije, kao što su puštanje audio datoteka i AVI video datoteka.

Na kraju nedelje ćete početi da učite napredne tehnike programiranja, kao što su statusna traka, trake sa alatima i štampanje. U lekciji a 14, ćete naučiti kako da koristite pomoć za određeni sadržaj i kako da snimite informacije vezane za Vaš program u Windows Registry datoteku. Mislim da će Vam se svideti ono što ćete pronaći. Do kraja nedelje bićete kao parni valjak koji se ne može zaustaviti.

3 SEDMICA

15
16
17
18
19
20
21

Na prvi pogled

U toku ove nedelje ćete se baviti malo drugačijim stvarima. Ona počinje opisom COM i ActiveX tehnologija. Naučićete kako da kreirate ActiveX komponente kao i posebnu vrstu ActiveX kontrola pod nazivom "Aktivne forme" (engl. ActiveForm). ActiveX komponente su slične komponentama iz VCL biblioteke, ali imaju potpuno drugačiju arhitekturu.

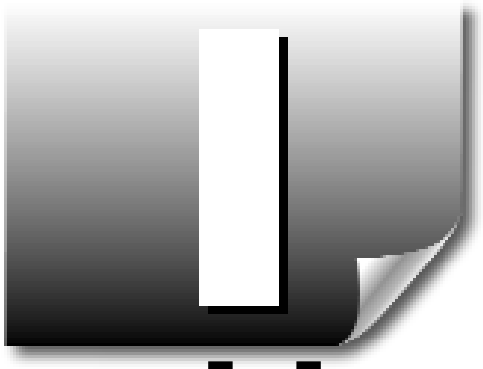
Posle opisa COM i ActiveX tehnologija, početećete sa programiranjem baza podataka. Učićete o arhitekturi baza podataka i o načinu na koji Delphi i VCL omogućavaju operacije nad njima. Počećete sa osnovnim tehnikama da biste kasnije stigli do naprednih tehnika programiranja baza podataka.

Posle rada sa bazama podataka, pomenućemo i DLL-ove (eng. Dynamic Link Libraries). DLL-ovi služe tome da iz njih nešto uzmete. Možete ih koristiti, ali i ne morate. Ipak, veoma je važno da, kada donosite takvu odluku, budete detaljno upoznati sa DLL-ovima. Da biste to postigli, morate znati šta su DLL-ovi i na koji način možete da ih koristite u svojim programima. Devetnaesti dan se upravo bavi tim problemom. Nakon toga, možete doneti odluku da li su DLL-ovi pravo rešenje za Vas. Sasvim sigurno, videćete da su DLL-ovi veoma korisni u nekim situacijama.

Dvadesetog dana bavićete se jednom vrlo naprednom temom - pisanjem komponenti. Pisanje komponenti zahteva temeljnije razumevanje osobina, metoda i dogadjaja, od onoga koga sada imate. Pisanje komponenti nije za osobe sa slabim srcem. To je ozbiljno programiranje. Možda će Vam se dopasti, a možda i neće. Ako odlučite da ne želite da se bavite pisanjem komponenti, to nije veliki problem. Na tržištu postoji veliki broj dostupnih

komponenti (shareware, freeware, ili komercijalnih) koje će za Vas uraditi praktično sve. Konačno, videćete koliko su Delphi i C++Builder slični, ali i koliko su različiti. Jedna od najboljih osobina Delphi-jevih formi je i što ih možete koristiti i u Delphi-ju i u C++Builder-u. Delphi i C++Builder nisu konkurentski proizvodi. Oni postoje da bi se međusobno dopunjavali.

Dakle, ostala je još jedna nedelja. Da li ste nestrpljivi? Onda, krenimo.



Indeks

A

B

C

Č

D-Dž

E

F

G

H

I

J

K

L

M

N

O

P

R

S

Š

T

U

V

W

Z

Ž